Святослав Куликов

Работа с MySQL, MS SQL Server и Oracle

В ПРИМЕРАХ

СВЯТОСЛАВ КУЛИКОВ

РАБОТА С MYSQL, MS SQL SERVER И ORACLE В ПРИМЕРАХ

ПРАКТИЧЕСКОЕ ПОСОБИЕ

МИНСК
УП «БОФФ»
2016

УДК 004.4’6
ББК 32.973.26-018.2
К90

Куликов, С. С.
К90 Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие. / С. С. Куликов. — Минск: БОФФ, 2016. — 556 с.
ISBN 978-985-430-054-1.

Три СУБД, 50 примеров, 129 задач, более 500 запросов с пояснениями и комментариями. От SELECT * до поиска кратчайшего пути в ориентированном графе; никакой теории, только схемы и код, много кода. Книга будет полезна тем, кто: когда-то изучал базы данных, но многое забыл; имеет опыт работы с одной СУБД, но хочет быстро переключиться на другую; хочет в предельно сжатые сроки научиться писать типичные SQL-запросы.

УДК 004.4’6
ББК 32.973.26-018.2

ISBN 978-985-430-054-1

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Выражаю огромную благодарность коллегам из EPAM Systems за ценные замечания и рекомендации в процессе подготовки материала.

Эта книга посвящена практике использования SQL для решения типичных задач. Здесь не рассматривается теория реляционных баз данных (предполагается, что вы с ней знакомы либо способны найти недостающую информацию), но приведено более 500 SQL-запросов: от элементарных выборок до использования представлений, триггеров, хранимых процедур и функций.

Все примеры представлены в виде постановки задачи и её решения с использованием MySQL, MS SQL Server и Oracle, а также снабжены пояснениями и разбором типичных ошибок.

Этот материал в первую очередь будет полезен тем, кто:

- когда-то изучал базы данных, но многое забыл;
- имеет опыт работы с одной СУБД, но хочет быстро переключиться на другую;
- хочет в предельно сжатые сроки научиться писать типичные SQL-запросы.

Все решения выполнены на MySQL Community Server 5.6, Microsoft SQL Server Express 2012, Oracle 11gR2 Express Edition и, скорее всего, успешно будут работать на более новых версиях этих СУБД, но **не на более старых**.

В большинстве решений со сложной логикой алгоритм пояснён на примере MySQL, а для двух других СУБД лишь приведён код с небольшими комментариями, потому желательно рассматривать решения для всех трёх СУБД, даже если вам интересна только одна из них.

Исходные материалы (схемы, скрипты и т.д.) можно получить по этой ссылке: http://svyatoslav.biz/database_book_download/src.zip

Условные обозначения, используемые в этой книге:

Постановка задачи. Сразу отметим, что почти в каждом рассмотренном примере приведено несколько задач. Поскольку решение задачи в некоторых случаях будет размещено далеко от её формулировки, рядом с номером задачи в фигурных скобках будет приведена {ссылка} на решение.

Ожидаемый результат. Сначала мы будем показывать, что должно получаться при правильном решении задачи, а потом — само решение.

Решение задачи. Решение всегда будет приведено для MySQL, MS SQL Server и Oracle. Иногда решения будут похожими, иногда очень разными. Поскольку решение задачи в некоторых случаях будет размещено далеко от её формулировки, рядом с номером решения в фигурных скобках будет приведена {ссылка} на формулировку задачи.

Исследование. Это — тоже задача, но уже не на получение неких данных, а на проверку того, как ведёт себя СУБД в некоторых условиях.

Предостережение. Мы будем рассматривать типичные ошибки и приводить пояснения их причин и последствий.

Задание для самостоятельной проработки. Настоятельно рекомендуется выполнять эти задания. Даже если вам кажется, что всё очень просто. Если, напротив, вам кажется, что всё очень сложно — сначала попробуйте самостоятельно решить уже разобранные примеры, обращаясь к готовому решению лишь для самопроверки.

МОДЕЛЬ, ГЕНЕРАЦИЯ И НАПОЛНЕНИЕ БАЗЫ ДАННЫХ

ОБЩЕЕ ОПИСАНИЕ МОДЕЛИ

Для решения задач и рассмотрения примеров мы будем использовать три базы данных: «Библиотека», «Большая библиотека» и «Исследование». База данных «Исследование» будет состоять из множества разрозненных таблиц, необходимых для демонстрации особенностей поведения СУБД, и мы будем формировать её постепенно по мере проведения экспериментов. Также в ней будет физически расположена база данных «Большая библиотека».

Модели баз данных «Библиотека» и «Большая библиотека» полностью идентичны (отличаются эти базы данных только количеством записей). Здесь всего семь таблиц:

- `genres` — описывает литературные жанры:
  - `g_id` — идентификатор жанра (число, первичный ключ);
  - `g_name` — имя жанра (строка);
- `books` — описывает книги в библиотеке:
  - `b_id` — идентификатор книги (число, первичный ключ);
  - `b_name` — название книги (строка);
  - `b_year` — год издания (число);
  - `b_quantity` — количество экземпляров книги в библиотеке (число);
- `authors` — описывает авторов книг:
  - `a_id` — идентификатор автора (число, первичный ключ);
  - `a_name` — имя автора (строка);
- `subscribers` — описывает читателей (подписчиков) библиотеки:
  - `s_id` — идентификатор читателя (число, первичный ключ);
  - `s_name` — имя читателя (строка);
- `subscriptions` — описывает факты выдачи/возврата книг (т.н. «подписки»):
  - `sb_id` — идентификатор подписки (число, первичный ключ);
  - `sb_subscriber` — идентификатор читателя (подписчика) (число, внешний ключ);
  - `sb_book` — идентификатор книги (число, внешний ключ);
  - `sb_start` — дата выдачи книги (дата);

    - `sb_finish` — запланированная дата возврата книги (дата);
    - `sb_is_active` — признак активности подписки (содержит значение `Y`, если книга ещё на руках у читателя, и `N`, если книга уже возвращена в библиотеку);
- `m2m_books_genres` — служебная таблица для организации связи «многие ко многим» между таблицами `books` и `genres`:
    - `b_id` — идентификатор книги (число, внешний ключ, часть составного первичного ключа);
    - `g_id` — идентификатор жанра (число, внешний ключ, часть составного первичного ключа);
- `m2m_books_authors` — служебная таблица для организации связи «многие ко многим» между таблицами `books` и `authors`:
    - `b_id` — идентификатор книги (число, внешний ключ, часть составного первичного ключа);
    - `a_id` — идентификатор автора (число, внешний ключ, часть составного первичного ключа).

В таблицах `m2m_books_genres` и `m2m_books_authors` оба внешних ключа входят в составной первичный ключ, чтобы исключить ситуацию, когда, например, принадлежность некоторой книги некоторому жанру будет указана более одного раза (такие ошибки приводят к неверной работе запросов вида «посчитать, к скольким жанрам относится книга»).

Общая схема базы данных представлена на рисунке 1.а. Скрипты генерации баз данных для каждой СУБД вы можете найти в исходном материале, ссылка на который приведена в предисловии[5].

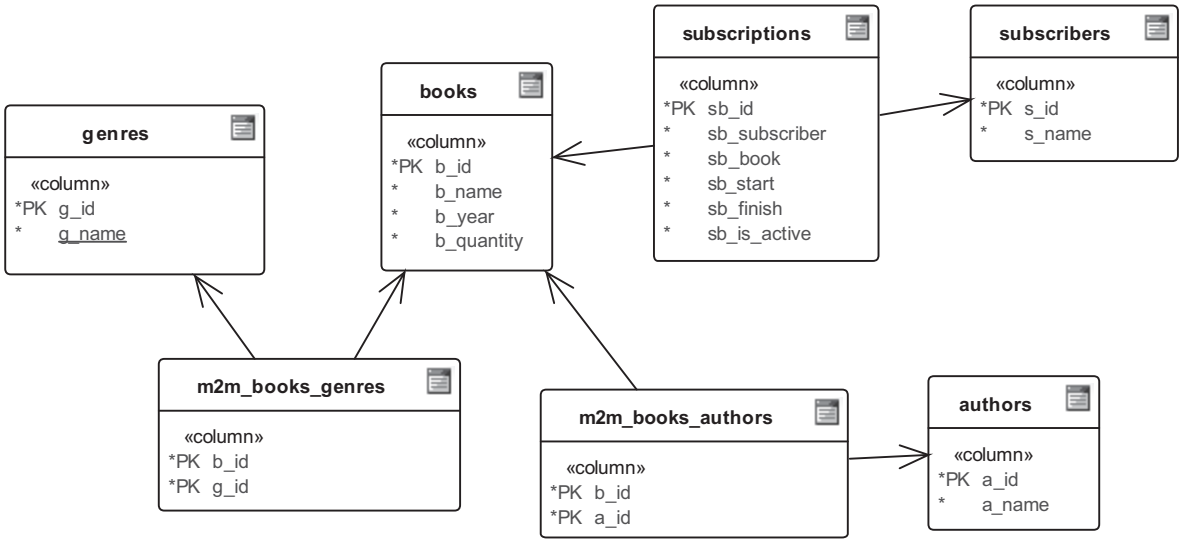


*Рисунок 1.а — **Общая схема базы данных***

1.2. МОДЕЛЬ ДЛЯ MYSQL

Модель базы данных для MySQL представлена на рисунках 1.b и 1.c. Обратите внимание на следующие важные моменты:

- Первичные ключи представлены беззнаковыми целыми числами для расширения максимального диапазона значений.
- Строки представлены типом **`varchar`** длиной до 150 символов (чтобы гарантированно уложиться в ограничение 767 байт на длину индекса; 150*4 = 600, MySQL выравнивает символы UTF-строк на длину в четыре байта на символ при операциях сравнения).
- Поле **`sb_is_active`** представлено характерным для MySQL типом данных **`enum`** (позволяющим выбрать одно из указанных значений и очень удобным для хранения заранее известного предопределённого набора значений — **`Y`** и **`N`** в нашем случае).
- Поле **`g_name`** сделано уникальным, т.к. существование одноимённых жанров недопустимо.
- Поля **`sb_start`** и **`sb_finish`** представлены типом **`date`** (а не более полными, например, **`datetime`**), т.к. мы храним дату выдачи и возврата книги с точностью до дня.

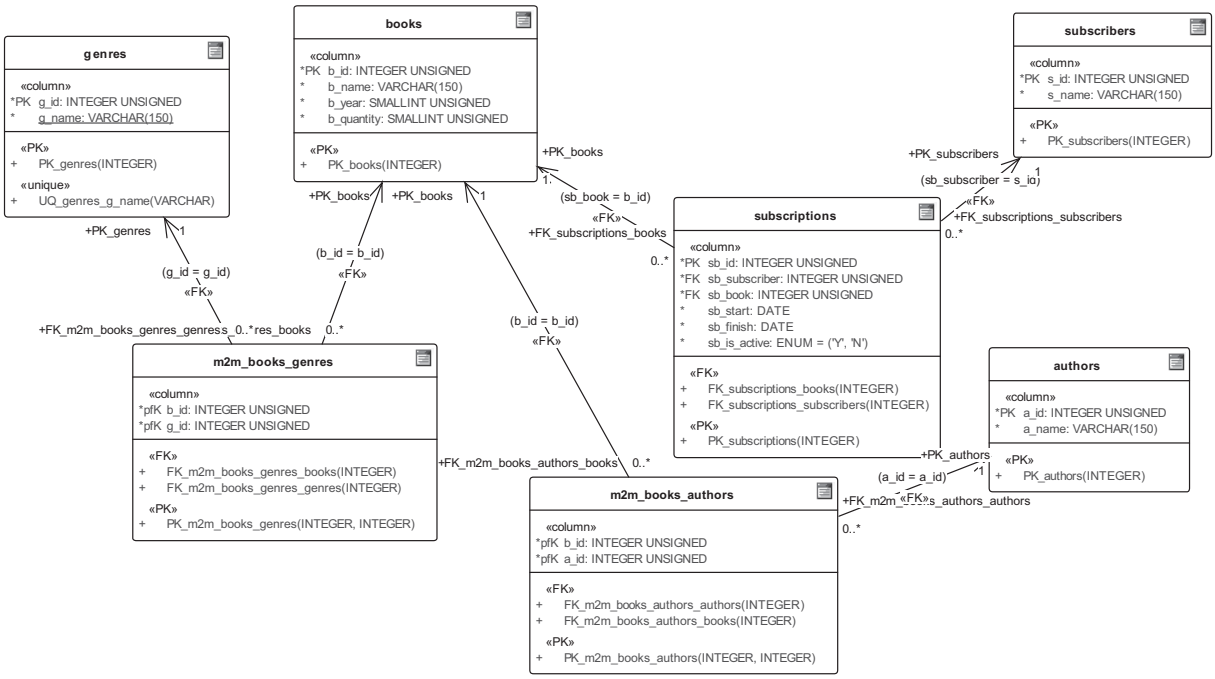


*Рисунок 1.b — **Модель базы данных для MySQL в Sparx EA***

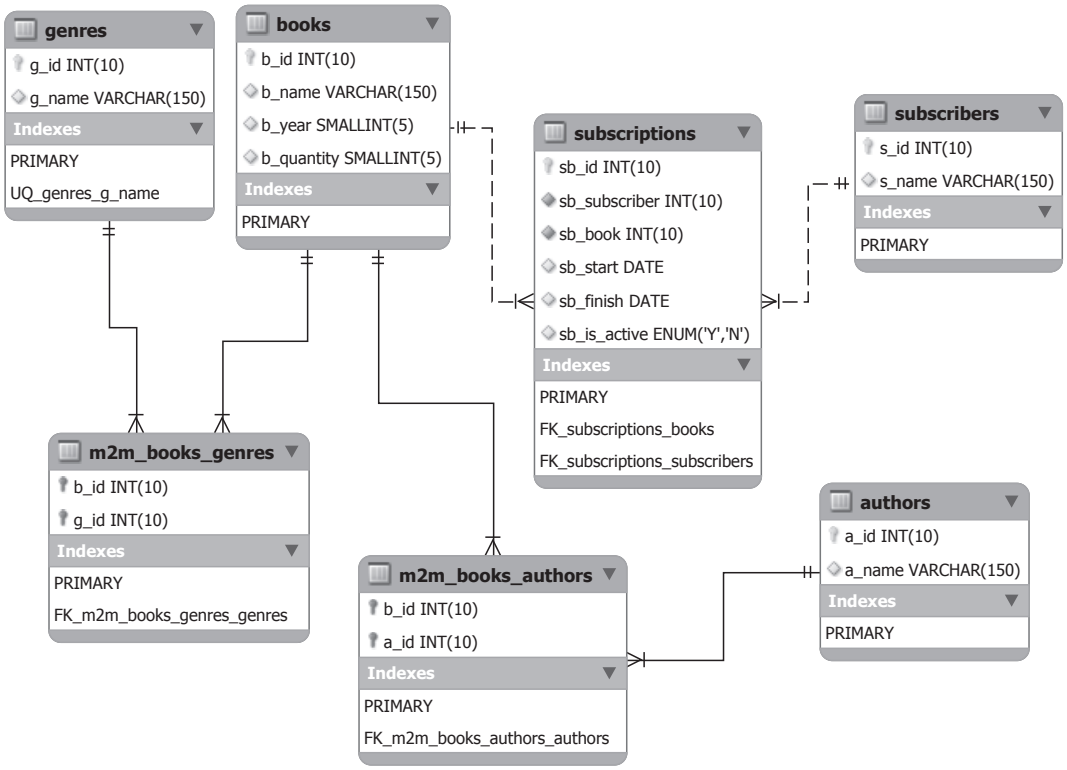


*Рисунок 1.c — **Модель базы данных для MySQL в MySQL Workbench***

1.3. МОДЕЛЬ ДЛЯ MS SQL SERVER

Модель базы данных для MS SQL Server представлена на рисунках 1.d и 1.e. Обратите внимание на следующие важные моменты:

- Первичные ключи представлены знаковыми целыми числами, т.к. в MS SQL Server нет возможности сделать `bigint`, `int` и `smallint` беззнаковыми (а `tinyint`, наоборот, бывает только беззнаковым).
- Строки представлены типом `nvarchar` длиной до 150 символов (как из соображений аналогии с MySQL, так и чтобы гарантированно уложиться в ограничение 900 байт на длину индекса; 150*2 = 300, MS SQL Server для хранения и сравнения символов в национальных кодировках использует два байта на символ).
- Поле `sb_is_active` представлено типом `char` длиной в один символ (т.к. в MS SQL Server нет типа данных `enum`), а для соблюдения аналогии с MySQL на это поле наложено ограничение `check` со значением `[sb_is_active] IN ('Y', 'N')`.
- Как и в MySQL, поля `sb_start` и `sb_finish` представлены типом `date` (а не более полными, например, `datetime`), т.к. мы храним дату выдачи и возврата книги с точностью до дня.

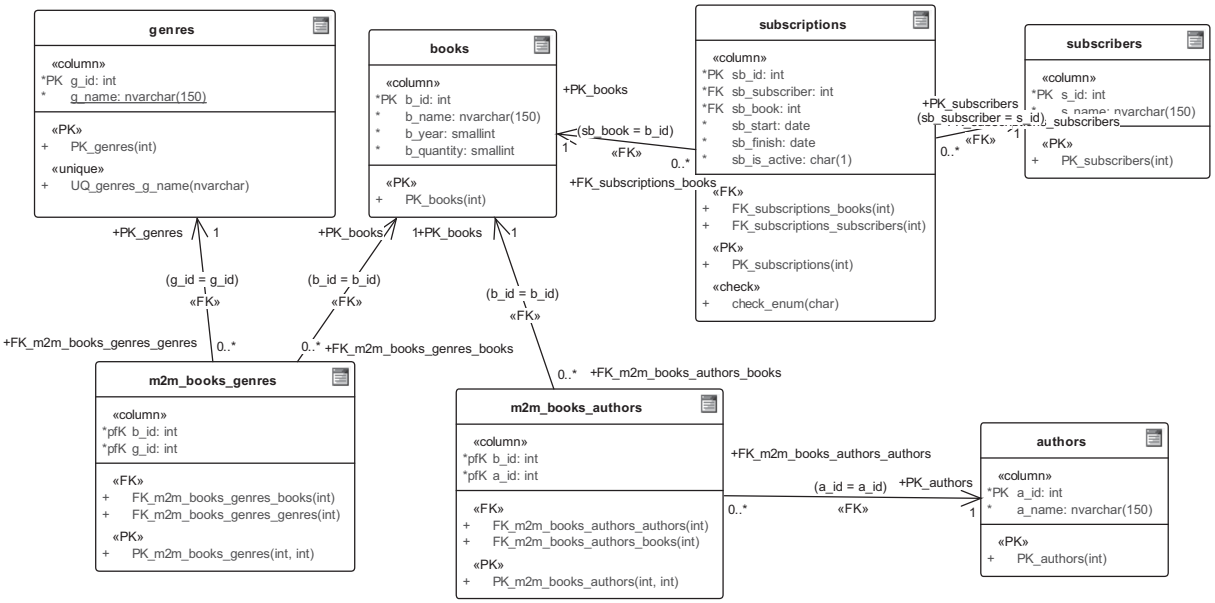


*Рисунок 1.d — **Модель базы данных для MS SQL Server в Sparx EA***

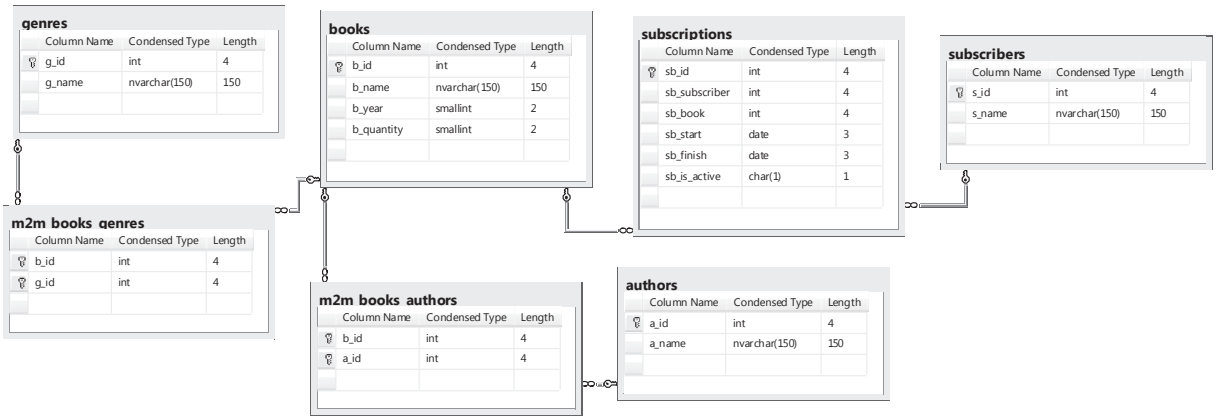


*Рисунок 1.e — **Модель базы данных для MS SQL Server в MS SQL Server Management Studio***

1.4. МОДЕЛЬ ДЛЯ ORACLE

Модель базы данных для Oracle представлена на рисунках 1.f и 1.g. Обратите внимание на следующие важные моменты:

- Целочисленные поля представлены типом **`number`** с указанием длины — это наиболее простой способ эмуляции целочисленных типов из других СУБД.
- Строки представлены типом **`nvarchar2`** длиной до 150 символов (как из соображений аналогии с MySQL и MS SQL Server, так и чтобы гарантированно уложиться в ограничение 758-6498 байт на длину индекса; 150*2 = 300, Oracle для хранения и сравнения символов в национальных кодировках использует два байта на символ, и максимальная длина индекса может зависеть от разных условий, но минимум — 758 байт).
- Поле **`sb_is_active`** представлено типом **`char`** длиной в один символ (т.к. в Oracle нет типа данных **`enum`**), а для соблюдения аналогии с MySQL применено то же решение, что и в случае с MS SQL Server: на это поле наложено ограничение check со значением **`"sb_is_active" IN ('Y', 'N')`**.
- Для полей **`sb_start`** и **`sb_finish`** выбран тип **`date`** как «самый простой» из имеющихся в Oracle типов хранения даты. Да, он всё равно сохраняет часы, минуты и секунды, но мы можем вписать туда нулевые значения.

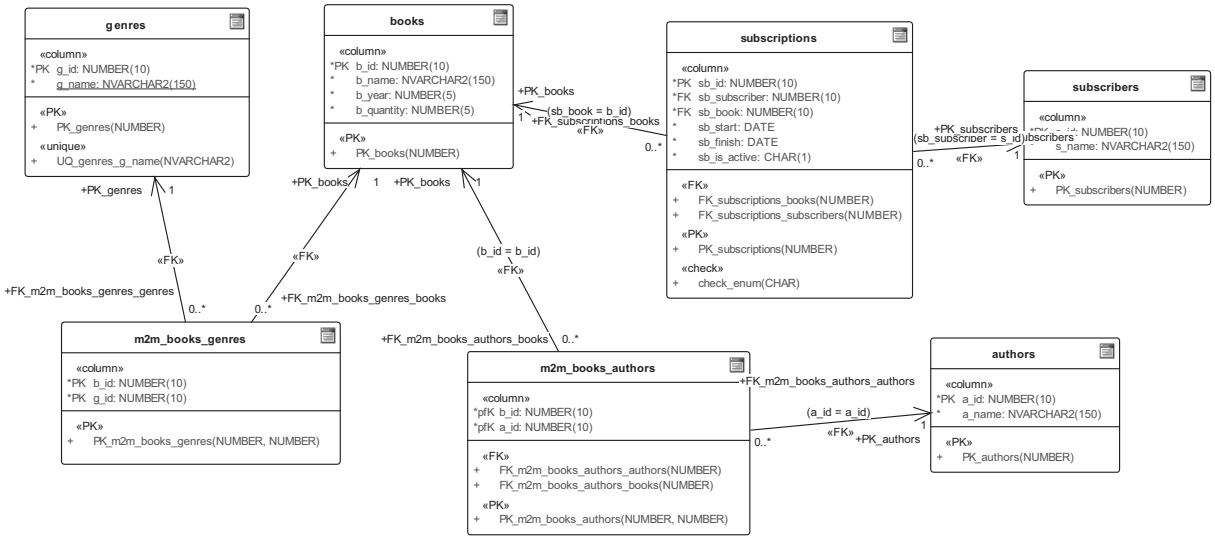


*Рисунок 1.f — **Модель базы данных для Oracle в Sparx EA***

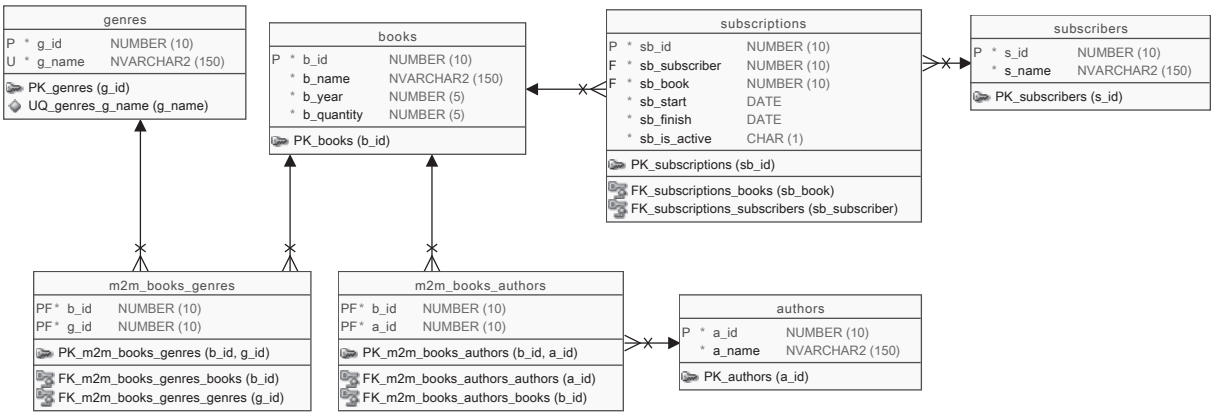


*Рисунок 1.g — **Модель базы данных для Oracle в Oracle SQL Developer Data Modeler***

ГЕНЕРАЦИЯ И НАПОЛНЕНИЕ БАЗЫ ДАННЫХ

На основе созданных в Sparx Enterprise Architect моделей получим DDL-скрипты{5} для каждой СУБД и выполним их, чтобы создать базы данных (обратитесь к документации по Sparx EA за информацией о том, как на основе модели базы данных получить скрипт её генерации).

Наполним имеющиеся базы данных следующими данными.

Таблица **`books`**:

b_id	b_name	b_year	b_quantity
1	Евгений Онегин	1985	2
2	Сказка о рыбаке и рыбке	1990	3
3	Основание и империя	2000	5
4	Психология программирования	1998	1
5	Язык программирования C++	1996	3
6	Курс теоретической физики	1981	12
7	Искусство программирования	1993	7

Таблица **`authors`**:

a_id	a_name
1	Д. Кнут
2	А. Азимов
3	Д. Карнеги
4	Л.Д. Ландау
5	Е.М. Лифшиц
6	Б. Страуструп
7	А.С. Пушкин

Таблица `genres`:

g_id	g_name
1	Поэзия
2	Программирование
3	Психология
4	Наука
5	Классика
6	Фантастика

Таблица `subscribers`:

s_id	s_name
1	Иванов И.И.
2	Петров П.П.
3	Сидоров С.С.
4	Сидоров С.С.

Присутствие двух подписчиков с именем «Сидоров С.С.» — не ошибка. Нам понадобится такой вариант полных тёзок для демонстрации нескольких типичных ошибок в запросах.

Таблица `m2m_books_authors`:

b_id	a_id
1	7
2	7
3	2
4	3
4	6
5	6
6	5
6	4
7	1

Таблица `m2m_books_genres`:

b_id	g_id
1	1
1	5
2	1
2	5
3	6
4	2
4	3
5	2
6	5
7	2
7	5

Таблица **`subscriptions`**:

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
100	1	3	2011-01-12	2011-02-12	N
2	1	1	2011-01-12	2011-02-12	N
3	3	3	2012-05-17	2012-07-17	Y
42	1	2	2012-06-11	2012-08-11	N
57	4	5	2012-06-11	2012-08-11	N
61	1	7	2014-08-03	2014-10-03	N
62	3	5	2014-08-03	2014-10-03	Y
86	3	1	2014-08-03	2014-09-03	Y
91	4	1	**2015-10-07**	**2015-03-07**	Y
95	1	4	2015-10-07	2015-11-07	N
99	4	4	**2015-10-08**	**2025-11-08**	Y

Странные комбинации дат выдачи и возврата книг (2015-10-07 / 2015-03-07 и 2015-10-08 / 2015-11-08) добавлены специально, чтобы продемонстрировать в дальнейшем особенности решения некоторых задач.

Также обратите внимание, что среди читателей, бравших книги, ни разу не встретился «Петров П.П.» (с идентификатором 2), это тоже понадобится нам в будущем.

Неупорядоченность записей по идентификаторам и «пропуски» в нумерации идентификаторов также сделаны осознанно для большей реалистичности.

После вставки данных в базы данных под управлением всех трёх СУБД стоит проверить, не ошиблись ли мы где-то. Выполним следующие запросы, которые позволят увидеть данные о книгах и работе библиотеки в человекочитаемом виде. Разбор этих запросов представлен в примере 11{68}, а пока — просто код:

MySQL

```
SELECT `b_name`,
       `a_name`,
       `g_name`
FROM   `books`
       JOIN `m2m_books_authors` USING(`b_id`)
       JOIN `authors` USING(`a_id`)
       JOIN `m2m_books_genres` USING(`b_id`)
       JOIN `genres` USING(`g_id`)
```

MS SQL

```
SELECT [b_name],
       [a_name],
       [g_name]
FROM   [books]
       JOIN [m2m_books_authors]
         ON [books].[b_id] = [m2m_books_authors].[b_id]
       JOIN [authors]
         ON [m2m_books_authors].[a_id] = [authors].[a_id]
       JOIN [m2m_books_genres]
         ON [books].[b_id] = [m2m_books_genres].[b_id]
       JOIN [genres]
         ON [m2m_books_genres].[g_id] = [genres].[g_id]
```

Oracle

```
SELECT "b_name",
       "a_name",
       "g_name"
FROM   "books"
       JOIN "m2m_books_authors"
         ON "books"."b_id" = "m2m_books_authors"."b_id"
       JOIN "authors"
         ON "m2m_books_authors"."a_id" = "authors"."a_id"
       JOIN "m2m_books_genres"
         ON "books"."b_id" = "m2m_books_genres"."b_id"
       JOIN "genres"
         ON "m2m_books_genres"."g_id" = "genres"."g_id"
```

После выполнения этих запросов получится результат, показывающий, как информация из разных таблиц объединяется в осмысленные наборы:

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Классика
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
Курс теоретической физики	Л.Д. Ландау	Классика
Курс теоретической физики	Е.М. Лифшиц	Классика
Искусство программирования	Д. Кнут	Классика
Евгений Онегин	А.С. Пушкин	Поэзия
Сказка о рыбаке и рыбке	А.С. Пушкин	Поэзия
Психология программирования	Д. Карнеги	Программирование
Психология программирования	Б. Страуструп	Программирование
Язык программирования C++	Б. Страуструп	Программирование
Искусство программирования	Д. Кнут	Программирование
Психология программирования	Д. Карнеги	Психология
Психология программирования	Б. Страуструп	Психология
Основание и империя	А. Азимов	Фантастика

Выполним ещё один запрос, чтобы посмотреть человекочитаемую информацию о том, кто и какие книги брал в библиотеке (подробнее об этом запросе сказано в примере 11{68}):

MySQL

```
SELECT `b_name`,
       `s_id`,
       `s_name`,
       `sb_start`,
       `sb_finish`
FROM   `books`
       JOIN `subscriptions`
         ON `b_id` = `sb_book`
       JOIN `subscribers`
         ON `sb_subscriber` = `s_id`
```

MS SQL

```
SELECT [b_name],
       [s_id],
       [s_name],
       [sb_start],
       [sb_finish]
FROM   [books]
       JOIN [subscriptions]
         ON [b_id] = [sb_book]
       JOIN [subscribers]
         ON [sb_subscriber] = [s_id]
```

Oracle

```
SELECT "b_name",
       "s_id",
       "s_name",
       "sb_start",
       "sb_finish"
FROM   "books"
       JOIN "subscriptions"
         ON "b_id" = "sb_book"
       JOIN "subscribers"
         ON "sb_subscriber" = "s_id"
```

В результате выполнения этих запросов получится такой результат:

b_name	s_id	s_name	sb_start	sb_finish
Евгений Онегин	1	Иванов И.И.	2011-01-12	2011-02-12
Сказка о рыбаке и рыбке	1	Иванов И.И.	2012-06-11	2012-08-11
Искусство программирования	1	Иванов И.И.	2014-08-03	2014-10-03
Психология программирования	1	Иванов И.И.	2015-10-07	2015-11-07
Основание и империя	1	Иванов И.И.	2011-01-12	2011-02-12
Основание и империя	3	Сидоров С.С.	2012-05-17	2012-07-17
Язык программирования C++	3	Сидоров С.С.	2014-08-03	2014-10-03
Евгений Онегин	3	Сидоров С.С.	2014-08-03	2014-09-03
Язык программирования C++	4	Сидоров С.С.	2012-06-11	2012-08-11
Евгений Онегин	4	Сидоров С.С.	2015-10-07	2015-03-07
Психология программирования	4	Сидоров С.С.	2015-10-08	2025-11-08

Обратите внимание, что «Сидоров С.С.» на самом деле — два разных человека (с идентификаторами 3 и 4).

В базу данных «Большая библиотека» поместим следующее количество автоматически сгенерированных записей:

- в таблицу **`authors`** — 10’000 имён авторов (допускается дублирование имён);
- в таблицу **`books`** — 100’000 названий книг (допускается дублирование названий);
- в таблицу **`genres`** — 100 названий жанров (все названия уникальны);
- в таблицу **`subscribers`** — 1’000’000 имён читателей (допускается дублирование имён);
- в таблицу **`m2m_books_authors`** — 1’000’000 связей «автор-книга» (допускается соавторство);

- в таблицу **m2m_books_genres** — 1’000’000 связей «книга-жанр» (допускается принадлежность книги к нескольким жанрам);
- в таблицу **subscriptions** — 10’000’000 записей о выдаче/возврате книг (возврат всегда наступает не раньше выдачи, допускаются ситуации, когда один и тот же читатель брал одну и ту же книгу много раз — даже не вернув предыдущий экземпляр)

При проведении экспериментов по оценке скорости выполнения запросов все операции с СУБД будут выполняться в небуферизируемом режиме (когда результаты выполнения запросов не передаются в пространство памяти приложения, их выполнявшего), а также перед каждой итерацией (кроме исследования 2.1.3.EXP.A[23]) кэш СУБД будет очищен следующим образом:

MySQL
```
1   RESET QUERY CACHE;
```

MS SQL
```
1   DBCC FREEPROCCACHE;
2   DBCC DROPCLEANBUFFERS;
```

Oracle
```
1   ALTER SYSTEM FLUSH BUFFER_CACHE;
2   ALTER SYSTEM FLUSH SHARED_POOL;
```

Итак, наполнение баз данных успешно завершено и можно переходить к решению задач.

ЗАПРОСЫ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ

ВЫБОРКА ИЗ ОДНОЙ ТАБЛИЦЫ

2.1.1. ПРИМЕР 1: ВЫБОРКА ВСЕХ ДАННЫХ

Задача 2.1.1.a[{18}]: показать всю информацию обо всех читателях.

Ожидаемый результат 2.1.1.a.

s_id	s_name
1	Иванов И.И.
2	Петров П.П.
3	Сидоров С.С.
4	Сидоров С.С.

Решение 2.1.1.a[{18}].

В данном случае достаточно самого простого запроса — классического **`SELECT *`**. Для всех трёх СУБД запросы совершенно одинаковы.

MySQL Решение 2.1.1.a

```
1  SELECT *
2  FROM `subscribers`
```

MS SQL Решение 2.1.1.a

```
1  SELECT *
2  FROM [subscribers]
```

Oracle Решение 2.1.1.a

```
1  SELECT *
2  FROM "subscribers"
```

Задание 2.1.1.TSK.A: показать всю информацию:

- об авторах;
- о жанрах.

2.1.2. ПРИМЕР 2: ВЫБОРКА ДАННЫХ БЕЗ ПОВТОРЕНИЯ

Задача 2.1.2.a[{20}]: показать без повторений идентификаторы читателей, бравших в библиотеке книги.

Задача 2.1.2.b[{21}]: показать поимённый список читателей с указанием количества полных тёзок по каждому имени.

Ожидаемый результат 2.1.2.a.

sb_subscriber
1
3
4

Ожидаемый результат 2.1.2.b.

s_name	people_count
Иванов И.И.	1
Петров П.П.	1
Сидоров С.С.	2

Решение 2.1.2.a[19].

Важным для получения правильного результата является использование ключевого слова **DISTINCT**, которое предписывает СУБД убрать из результирующей выборки все повторения.

MySQL Решение 2.1.2.a

```
1  SELECT DISTINCT `sb_subscriber`
2  FROM `subscriptions`
```

MS SQL Решение 2.1.2.a

```
1  SELECT DISTINCT [sb_subscriber]
2  FROM [subscriptions]
```

Oracle Решение 2.1.2.a

```
1  SELECT DISTINCT "sb_subscriber"
2  FROM "subscriptions"
```

Если ключевое слово **DISTINCT** удалить из запроса, результат выполнения будет таким (поскольку читатели брали книги по несколько раз каждый):

sb_subscriber
1
1
1
1
1
3
3
3
4
4
4

Важно понимать, что СУБД в **DISTINCT**-режиме выполнения запроса производит сравнение между собой не отдельных полей, а строк целиком, а потому, если хотя бы в одном поле между строками есть различие, строки не считаются идентичными.

Типичная ошибка — попытка использовать **DISTINCT** «как функцию». Допустим, мы хотим увидеть без повторений все имена читателей (напомним, «Сидоров С.С.» у нас встречается дважды), выбрав не только имя, но и идентификатор. Следующий запрос составлен **неверно**.

MySQL Пример неверно составленного запроса 2.1.2.ERR.A

```
1  SELECT DISTINCT(`s_name`),
2                  `s_id`
3  FROM   `subscribers`
```

MS SQL Пример неверно составленного запроса 2.1.2.ERR.A

```
1  SELECT DISTINCT([s_name]),
2                  [s_id]
3  FROM   [subscribers]
```

Oracle Пример неверно составленного запроса 2.1.2.ERR.A

```
1  SELECT DISTINCT("s_name"),
2                  "s_id"
3  FROM   "subscribers"
```

В результате выполнения такого запроса мы получим результат, в котором «Сидоров С.С.» представлен дважды (дублирование не устранено) из-за того, что у этих записей разные идентификаторы (3 и 4), которые и не позволяют СУБД посчитать две соответствующие строки выборки совпадающими. Результат выполнения запроса 2.1.2.ERR.A будет таким:

s_name	s_id
Иванов И.И.	1
Петров П.П.	2
Сидоров С.С.	3
Сидоров С.С.	4

Частый вопрос: а как тогда решить эту задачу, как показать записи без повторения строк, в которых есть поля с различными значениями? В общем случае никак, т.к. сама постановка задачи неверна (мы просто потеряем часть данных).

Если задачу сформулировать несколько иначе (например, «показать поимённый список читателей с указанием количества полных тёзок по каждому имени»), то нам помогут группировки и агрегирующие функции. Подробно мы рассмотрим этот вопрос далее (см. пример 10{63}), а пока решим только что упомянутую задачу, чтобы показать разницу в работе запросов.

Решение 2.1.2.b{19}.

Используем конструкцию **GROUP BY** для группировки рядов выборки и функцию **COUNT** для подсчёта количества записей в каждой группе.

MySQL Решение 2.1.2.b

```
1  SELECT `s_name`,
2         COUNT(*) AS `people_count`
3  FROM   `subscribers`
4  GROUP  BY `s_name`
```

MS SQL Решение 2.1.2.b

```
1  SELECT [s_name],
2         COUNT(*) AS [people_count]
3  FROM   [subscribers]
4  GROUP  BY [s_name]
```

Oracle Решение 2.1.2.b

```
1  SELECT "s_name",
2         COUNT(*) AS "people_count"
3  FROM   "subscribers"
4  GROUP  BY "s_name"
```

Агрегирующая функция **COUNT(*)** (представленная во второй строке всех запросов 2.1.2.b) производит подсчёт записей, сгруппированных по совпадению значения поля **s_name** (см. четвёртую строку в запросах 2.1.2.b).

Для быстрого понимания логики группировок можно использовать аналогию с объединением ячеек с одинаковыми значениями в таблице Word или Excel, после чего происходит анализ (чаще

всего — подсчёт или суммирование) ячеек, соответствующих каждой такой «объединённой ячейке». В только что рассмотренном примере 2.1.2.b эту аналогию можно выразить следующим образом:

s_id	s_name
1	Иванов И.И.
2	Петров П.П.
3	Сидоров С.С.
4	

Теперь СУБД считает строки таблицы в контексте выполненной группировки, и получает, что Иванова и Петрова у нас по одному человеку, а Сидоровых — два, что и отражено в ожидаемом результате 2.1.2.b.

Задание 2.1.2.TSK.A: показать без повторений идентификаторы книг, которые были взяты читателями.

Задание 2.1.2.TSK.B: показать по каждой книге, которую читатели брали в библиотеке, количество выдач этой книги читателям.

2.1.3. ПРИМЕР 3: ИСПОЛЬЗОВАНИЕ ФУНКЦИИ COUNT И ОЦЕНКА ЕЁ ПРОИЗВОДИТЕЛЬНОСТИ

Задача 2.1.3.a{22}: показать, сколько всего разных книг зарегистрировано в библиотеке.

Ожидаемый результат 2.1.3.a.

total_books
7

Решение 2.1.3.a{22}.

Если переформулировать задачу, она будет звучать как «показать, сколько всего записей есть в таблице `books`», и решение выглядит так.

MySQL Решение 2.1.3.a

```
1  SELECT COUNT(*) AS `total_books`
2  FROM   `books`
```

MS SQL Решение 2.1.3.a

```
1  SELECT COUNT(*) AS [total_books]
2  FROM   [books]
```

Oracle Решение 2.1.3.a

```
1  SELECT COUNT(*) AS "total_books"
2  FROM   "books"
```

Функция **COUNT** может быть использована в следующих пяти форматах:

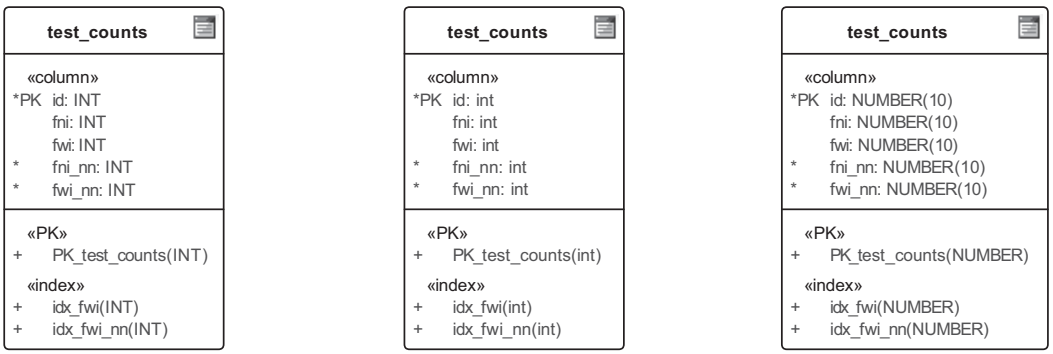
- **COUNT(*)** — классический вариант, используемый для подсчёта количества записей;
- **COUNT(1)** — альтернативная запись классического варианта;
- **COUNT(первичный_ключ)** — альтернативная запись классического варианта;
- **COUNT(поле)** — подсчёт записей, в указанном поле которых нет NULL-значений;
- **COUNT(DISTINCT поле)** — подсчёт без повторения записей, в указанном поле которых нет **NULL**-значений.

Одним из самых частых вопросов относительно разных вариантов **COUNT** является вопрос о производительности: какой вариант работает быстрее?

Исследование 2.1.3.EXP.A: оценка скорости работы различных вариантов **COUNT** в зависимости от объёма обрабатываемых данных. Это исследование — единственное, в котором кэш СУБД не будет сбрасываться перед выполнением каждого следующего запроса.

В базе данных «Исследование» создадим таблицу **test_counts**, содержащую такие поля:

- **id** — автоинкрементируемый первичный ключ (число);
- **fni** — поле без индекса («field, no index») (число или **NULL**);
- **fwi** — поле с индексом («field, with index») (число или **NULL**);
- **fni_nn** — поле без индекса и **NULL**'ов («field, no index, no nulls») (число);
- **fwi_nn** — поле с индексом без **NULL**'ов («field, with index, no nulls») (число).



MySQL MS SQL Server Oracle

*Рисунок 2.1.a — **Таблица test_counts во всех трёх СУБД***

Суть исследования состоит в последовательном добавлении в таблицу по тысяче записей (от нуля до десяти миллионов с шагом в тысячу) и выполнении следующих семи запросов с **COUNT** после добавления каждого десятка тысяч записей:

MySQL Исследование 2.1.3.EXP.A

```
1   -- Вариант 1: COUNT(*)
2   SELECT COUNT(*)
3   FROM   `test_counts`

1   -- Вариант 2: COUNT(первичный_ключ)
2   SELECT COUNT(`id`)
3   FROM   `test_counts`

1   -- Вариант 3: COUNT(1)
2   SELECT COUNT(1)
3   FROM   `test_counts`

1   -- Вариант 4: COUNT(поле_без_индекса)
2   SELECT COUNT(`fni`)
3   FROM   `test_counts`

1   -- Вариант 5: COUNT(поле_с_индексом)
2   SELECT COUNT(`fwi`)
3   FROM   `test_counts`

1   -- Вариант 6: COUNT(DISTINCT поле_без_индекса)
2   SELECT COUNT(DISTINCT `fni`)
3   FROM   `test_counts`

1   -- Вариант 7: COUNT(DISTINCT поле_с_индексом)
2   SELECT COUNT(DISTINCT `fwi`)
3   FROM   `test_counts`
```

MS SQL Исследование 2.1.3.EXP.A

```
1   -- Вариант 1: COUNT(*)
2   SELECT COUNT(*)
3   FROM   [test_counts]

1   -- Вариант 2: COUNT(первичный_ключ)
2   SELECT COUNT([id])
3   FROM   [test_counts]

1   -- Вариант 3: COUNT(1)
2   SELECT COUNT(1)
3   FROM   [test_counts]

1   -- Вариант 4: COUNT(поле_без_индекса)
2   SELECT COUNT([fni])
3   FROM   [test_counts]

1   -- Вариант 5: COUNT(поле_с_индексом)
2   SELECT COUNT([fwi])
3   FROM   [test_counts]
```

MS SQL Исследование 2.1.3.EXP.A (продолжение)

```
1   -- Вариант 6: COUNT(DISTINCT поле_без_индекса)
2   SELECT COUNT(DISTINCT [fni])
3   FROM   [test_counts]

1   -- Вариант 7: COUNT(DISTINCT поле_с_индексом)
2   SELECT COUNT(DISTINCT [fwi])
3   FROM   [test_counts]
```

Oracle Исследование 2.1.3.EXP.A

```
1   -- Вариант 1: COUNT(*)
2   SELECT COUNT(*)
3   FROM   "test_counts"

1   -- Вариант 2: COUNT(первичный_ключ)
2   SELECT COUNT("id")
3   FROM   "test_counts"

1   -- Вариант 3: COUNT(1)
2   SELECT COUNT(1)
3   FROM   "test_counts"

1   -- Вариант 4: COUNT(поле_без_индекса)
2   SELECT COUNT("fni")
3   FROM   "test_counts"

1   -- Вариант 5: COUNT(поле_с_индексом)
2   SELECT COUNT("fwi")
3   FROM   "test_counts"

1   -- Вариант 6: COUNT(DISTINCT поле_без_индекса)
2   SELECT COUNT(DISTINCT "fni")
3   FROM   "test_counts"

1   -- Вариант 7: COUNT(DISTINCT поле_с_индексом)
2   SELECT COUNT(DISTINCT "fwi")
3   FROM   "test_counts"
```

В первую очередь рассмотрим время, затраченное каждой СУБД на вставку тысячи записей, и зависимость этого времени от количества уже имеющихся в таблице записей. Соответствующие данные представлены на рисунке 2.1.b.

Как видно из графика, даже на таком относительно небольшом объёме данных все три СУБД показали падение производительности операции вставки ближе к концу эксперимента. Сильнее всего данный эффект проявился у Oracle. MySQL показал наименьшее время выполнения операции на протяжении всего эксперимента (также результаты MySQL оказались самыми стабильными).

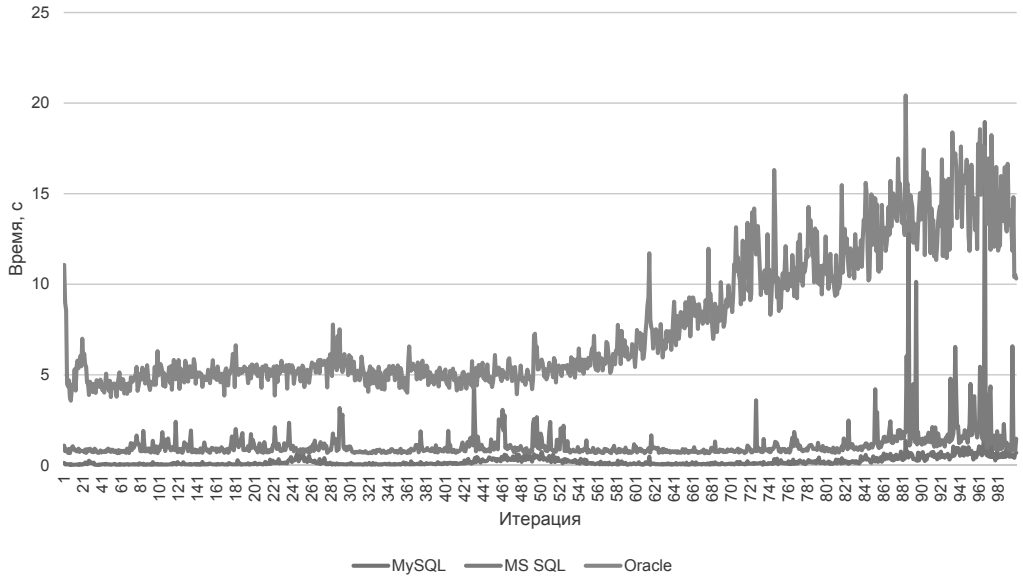


*Рисунок 2.1.b — **Время, затраченное каждой СУБД на вставку тысячи записей***

Теперь посмотрим на графики зависимости времени выполнения каждого из семи рассмотренных выше запросов 2.1.3.EXP.A от количества данных в таблице. На рисунках 2.1.c, 2.1.d, 2.1.e приведены графики для MySQL, MS SQL Server и Oracle соответственно.

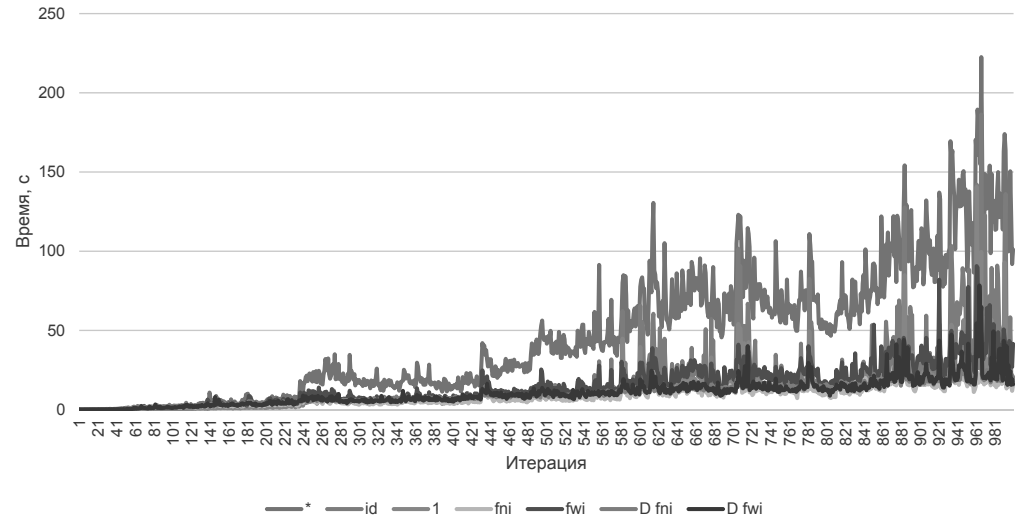


*Рисунок 2.1.c — **Время, затраченное MySQL на выполнение разных COUNT***

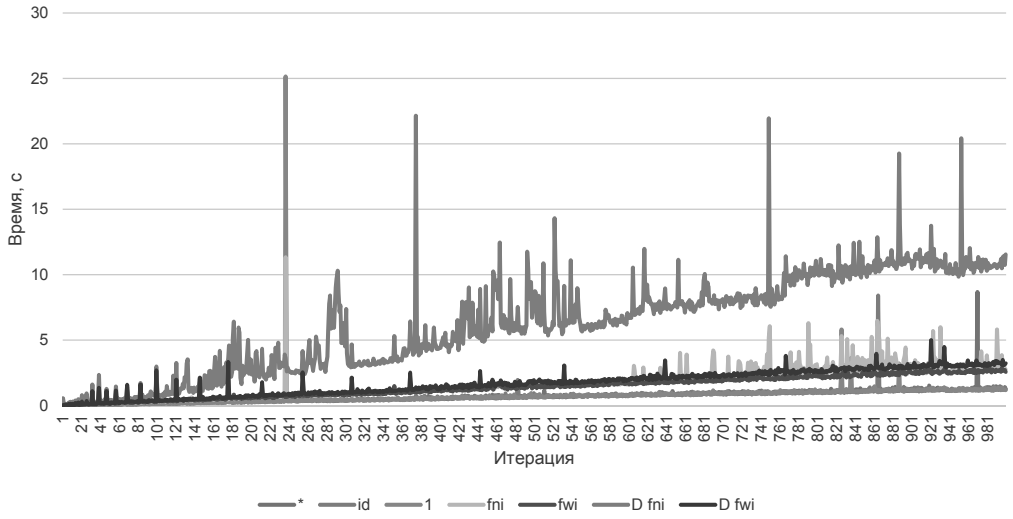


*Рисунок 2.1.d — **Время, затраченное MS SQL Server на выполнение разных COUNT***

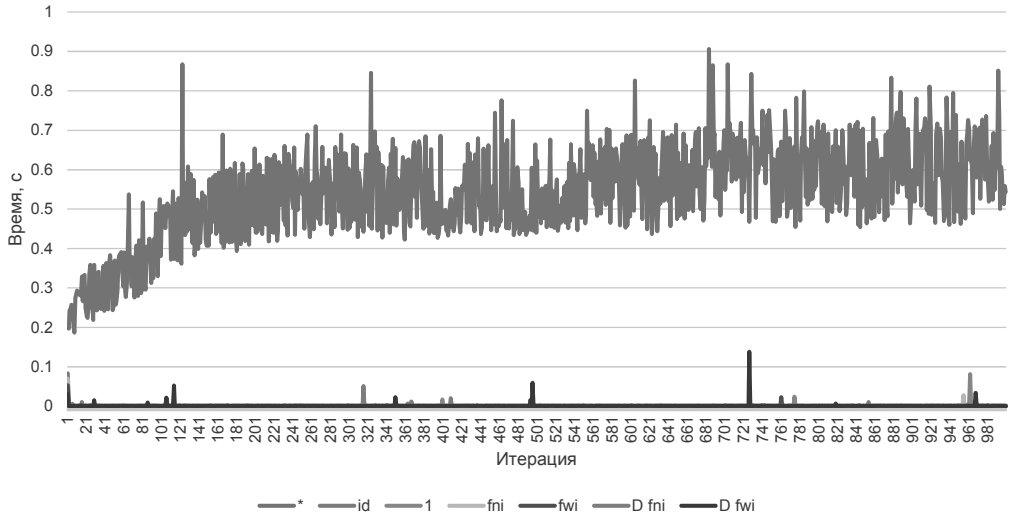


*Рисунок 2.1.e — **Время, затраченное Oracle на выполнение разных COUNT***

Чтобы увидеть всю картину целиком, представим в таблице значения медиан времени выполнения каждого запроса 2.1.3.EXP.A для каждой СУБД:

	MySQL	MS SQL Server	Oracle
COUNT(*)	36.662	0.702	0.541
COUNT(id)	11.120	0.679	0.001
COUNT(1)	9.995	0.681	0.001
COUNT(fni)	6.999	1.494	0.001
COUNT(fwi)	12.333	1.475	0.001
COUNT(DISTINCT fni)	9.252	6.300	0.001
COUNT(DISTINCT fwi)	9.626	1.743	0.001

Здесь можно увидеть несколько странную картину.

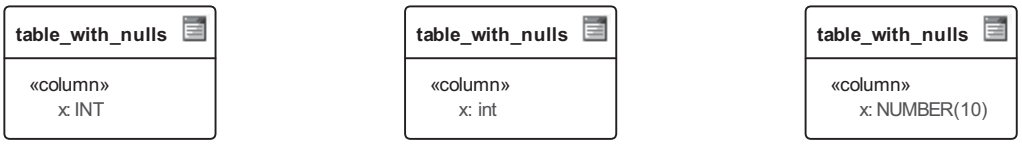
Для MySQL подтверждается бытующее мнение о том, что `COUNT(*)` работает медленнее всего, но неожиданно самым быстрым оказывается вариант с `COUNT(поле_без_индекса)`. Однако стоит отметить, что на меньшем объёме данных (до миллиона записей) ситуация оказывается совершенно иной — быстрее всего работает `COUNT(*)`.

MS SQL Server показал ожидаемые результаты: `COUNT(первичный_ключ)` оказался самым быстрым, `COUNT(DISTINCT поле_без_индекса)` — самым медленным. На меньших объёмах данных этот результат не меняется.

И теперь самое интересное — результаты Oracle: здесь снова подтвердились слухи о медленной работе `COUNT(*)`, но все остальные запросы показали потрясающий результат, очень высокую стабильность этого результата и полную независимость от объёма анализируемых данных. (Обратитесь к документации по Oracle, чтобы узнать, как этой СУБД удаётся так быстро выполнять некоторые виды `COUNT`).

Пусть данное исследование и не претендует на научный подход, но общая рекомендация состоит в том, чтобы использовать `COUNT(1)` как в среднем один из самых быстрых вариантов для разных СУБД и разных объёмов данных, т.к. остальные варианты иногда оказываются быстрее, но иногда и медленнее.

Исследование 2.1.3.EXP.B: рассмотрим, как функция `COUNT` реагирует на `NULL`-значения и на повторяющиеся значения в `DISTINCT`-режиме. Добавим в базу данных «Исследование» таблицу `table_with_nulls`.



MySQL MS SQL Server Oracle

*Рисунок 2.1.f — **Таблица table_with_nulls во всех трёх СУБД***

Поместим в созданную таблицу следующие данные (одинаковые для всех трёх СУБД):

x
1
1
2
NULL

Выполним следующие запросы на подсчёт количества записей в этой таблице:

MySQL Исследование 2.1.3.EXP.B

```
1   -- Вариант 1: COUNT(*)
2   SELECT COUNT(*) AS `total_records`
3   FROM   `table_with_nulls`

1   -- Вариант 2: COUNT(1)
2   SELECT COUNT(1) AS `total_records`
3   FROM   `table_with_nulls`
```

MS SQL Исследование 2.1.3.EXP.B

```
1   -- Вариант 1: COUNT(*)
2   SELECT COUNT(*) AS [total_records]
3   FROM   [table_with_nulls]

1   -- Вариант 2: COUNT(1)
2   SELECT COUNT(1) AS [total_records]
3   FROM   [table_with_nulls]
```

Oracle Исследование 2.1.3.EXP.B

```
-- Вариант 1: COUNT(*)
SELECT COUNT(*) AS "total_records"
FROM   "table_with_nulls"

-- Вариант 2: COUNT(1)
SELECT COUNT(1) AS "total_records"
FROM   "table_with_nulls"
```

Все шесть запросов во всех СУБД вернут одинаковый результат:

total_records
4

Теперь выполним запрос с **COUNT**, где аргументом будет являться имя поля:

MySQL Исследование 2.1.3.EXP.B

```
-- Вариант 3: COUNT(поле)
SELECT COUNT(`x`) AS `total_records`
FROM   `table_with_nulls`
```

MS SQL Исследование 2.1.3.EXP.B

```
-- Вариант 3: COUNT(поле)
SELECT COUNT([x]) AS [total_records]
FROM   [table_with_nulls]
```

Oracle Исследование 2.1.3.EXP.B

```
-- Вариант 3: COUNT(поле)
SELECT COUNT("x") AS "total_records"
FROM   "table_with_nulls"
```

Поскольку в таком случае подсчёт не учитывает **NULL**-значения, во всех трёх СУБД получится следующий результат:

total_records
3

И, наконец, выполним запрос с **COUNT** в **DISTINCT**-режиме:

MySQL Исследование 2.1.3.EXP.B

```
-- Вариант 4: COUNT(DISTINCT поле)
SELECT COUNT(DISTINCT `x`) AS `total_records`
FROM   `table_with_nulls`
```

MS SQL Исследование 2.1.3.EXP.B

```
-- Вариант 4: COUNT(DISTINCT поле)
SELECT COUNT(DISTINCT [x]) AS [total_records]
FROM   [table_with_nulls]
```

Oracle Исследование 2.1.3.EXP.B

```
-- Вариант 4: COUNT(DISTINCT поле)
SELECT COUNT(DISTINCT "x") AS "total_records"
FROM   "table_with_nulls"
```

В таком режиме **COUNT** не учитывает не только **NULL**-значения, но и все дубликаты значений (в наших исходных данных значение «1» было представлено дважды). Итого, результат выполнения запроса во всех трёх СУБД:

total_records
2

Осталось проверить, как **COUNT** работает на пустом множестве значений. Изменим запрос так, чтобы в выборку не попадал ни один ряд:

MySQL Исследование 2.1.3.EXP.B

```
1  -- Вариант 5: COUNT(поле_из_пустого_множества_записей)
2  SELECT COUNT(`x`) AS `negative_records`
3  FROM   `table_with_nulls`
4  WHERE  `x` < 0
```

MS SQL Исследование 2.1.3.EXP.B

```
1  -- Вариант 5: COUNT(поле_из_пустого_множества_записей)
2  SELECT COUNT([x]) AS [negative_records]
3  FROM   [table_with_nulls]
4  WHERE  [x] < 0
```

Oracle Исследование 2.1.3.EXP.B

```
1  -- Вариант 5: COUNT(поле_из_пустого_множества_записей)
2  SELECT COUNT("x") AS "negative_records"
3  FROM   "table_with_nulls"
4  WHERE  "x" < 0
```

Все три СУБД возвращают одинаковый легко предсказуемый результат:

negative_records
0

Задание 2.1.3.TSK.A: показать, сколько всего читателей зарегистрировано в библиотеке.

2.1.4. ПРИМЕР 4: ИСПОЛЬЗОВАНИЕ ФУНКЦИИ COUNT В ЗАПРОСЕ С УСЛОВИЕМ

Задача 2.1.4.a[{31}]: показать, сколько всего экземпляров книг выдано читателям.

Задача 2.1.4.b[{31}]: показать, сколько всего разных книг выдано читателям.

Ожидаемый результат 2.1.4.a.

in_use
5

Ожидаемый результат 2.1.4.b.

in_use
4

Решение 2.1.4.a[30].

Вся информация о выданных читателям книгах (фактах выдачи и возврата) хранится в таблице **subscriptions**, поле **sb_book** которой содержит идентификатор выданной книги. Поле **sb_is_active** содержит значение **Y** в случае, если книга сейчас находится у читателя (не возвращена в библиотеку).

Таким образом, нас будут интересовать только строки со значением поля **sb_is_active** равным **Y** (что отражено в секции **WHERE**, см. третью строку всех шести запросов 2.1.4.a-2.1.4.b), а разница между задачами 2.1.4.a и 2.1.4.b состоит в том, что в первом случае мы просто считаем все случаи «книга находится у читателя», а во втором случае мы не учитываем повторения (когда на руки выдано несколько экземпляров одной и той же книги) — и достигаем это за счёт **COUNT(DISTINCT поле)**.

MySQL Решение 2.1.4.a

```
1  SELECT COUNT(`sb_book`) AS `in_use`
2  FROM   `subscriptions`
3  WHERE  `sb_is_active` = 'Y'
```

MS SQL Решение 2.1.4.a

```
1  SELECT COUNT([sb_book]) AS [in_use]
2  FROM   [subscriptions]
3  WHERE  [sb_is_active] = 'Y'
```

Oracle Решение 2.1.4.a

```
1  SELECT COUNT("sb_book") AS "in_use"
2  FROM   "subscriptions"
3  WHERE  "sb_is_active" = 'Y'
```

Решение 2.1.4.b[30].

Здесь мы расширяем решение[30] задачи 2.1.4.a[30], добавляя ключевое слово **DISTINCT** в параметр функции **COUNT**, что обеспечивает подсчёт неповторяющихся значений поля **sb_book**.

MySQL Решение 2.1.4.b

```
1  SELECT COUNT(DISTINCT `sb_book`) AS `in_use`
2  FROM   `subscriptions`
3  WHERE  `sb_is_active` = 'Y'
```

MS SQL Решение 2.1.4.b

```
1  SELECT COUNT(DISTINCT [sb_book]) AS [in_use]
2  FROM   [subscriptions]
3  WHERE  [sb_is_active] = 'Y'
```

Oracle Решение 2.1.4.b

```
SELECT COUNT(DISTINCT "sb_book") AS "in_use"
FROM   "subscriptions"
WHERE  "sb_is_active" = 'Y'
```

Задание 2.1.4.TSK.A: показать, сколько всего раз читателям выдавались книги.

Задание 2.1.4.TSK.B: показать, сколько читателей брало книги в библиотеке.

2.1.5. ПРИМЕР 5: ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ SUM, MIN, MAX, AVG

Задача 2.1.5.a{32}: показать общее (сумму), минимальное, максимальное и среднее значение количества экземпляров книг в библиотеке.

Ожидаемый результат 2.1.5.a.

sum	min	max	avg
33	1	12	4.7143

Решение 2.1.5.a{32}.

В такой простой формулировке эта задача решается запросом, в котором достаточно перечислить соответствующие функции, передав им в качестве параметра поле **`b_quantity`** (и только в MS SQL Server придётся сделать небольшую доработку).

MySQL Решение 2.1.5.a

```
SELECT SUM(`b_quantity`) AS `sum`,
       MIN(`b_quantity`) AS `min`,
       MAX(`b_quantity`) AS `max`,
       AVG(`b_quantity`) AS `avg`
FROM   `books`
```

MS SQL Решение 2.1.5.a

```
SELECT SUM([b_quantity]) AS [sum],
       MIN([b_quantity]) AS [min],
       MAX([b_quantity]) AS [max],
       AVG(CAST([b_quantity] AS FLOAT)) AS [avg]
FROM   [books]
```

Oracle Решение 2.1.5.a

```
1  SELECT SUM("b_quantity") AS "sum",
2         MIN("b_quantity") AS "min",
3         MAX("b_quantity") AS "max",
4         AVG("b_quantity") AS "avg"
5  FROM   "books"
```

Обратите внимание на 4-ю строку в запросе 2.1.5.a для MS SQL Server: без приведения функцией **CAST** значения количества книг к дроби, итоговый результат работы функции **AVG** будет некорректным (это будет целое число), т.к. MS SQL Server выбирает тип данных результата на основе типа данных входного параметра. Продемонстрируем это.

MS SQL Решение 2.1.5.a (пример запроса с ошибкой)

```
1  SELECT SUM([b_quantity]) AS [sum],
2         MIN([b_quantity]) AS [min],
3         MAX([b_quantity]) AS [max],
4         AVG([b_quantity]) AS [avg]
5  FROM   [books]
```

Получится:

sum	min	max	avg
33	1	12	4

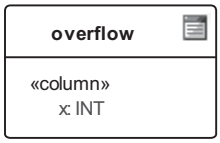
А должно быть:

sum	min	max	avg
33	1	12	4.7143

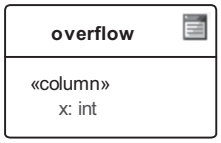
Стоит упомянуть ещё одну опасную ошибку: очень легко забыть, что при вычислении суммы и среднего значения (которое определяется как сумма, поделённая на количество значений) может произойти переполнение разрядной сетки.

Исследование 2.1.5.EXP.A: рассмотрим реакцию различных СУБД на ситуацию переполнения разрядной сетки.

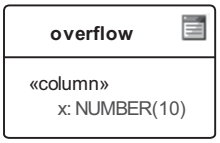
Создадим в БД «Исследование» таблицу **overflow** с одним числовым целочисленным полем:



MySQL



MS SQL Server



Oracle

*Рисунок 2.1.g — **Таблица overflow во всех трёх СУБД***

Поместим в созданную таблицу три максимальных значения для её поля **x**:

x
2147483647
2147483647
2147483647

MySQL

x
2147483647
2147483647
2147483647

MS SQL Server

x
9999999999
9999999999
9999999999

Oracle

Теперь выполним для каждой СУБД запросы на получение суммы значений из поля **x** и среднего значения в поле **x**:

MySQL Исследование 2.1.5.EXP.A

```
1  -- Запрос 1: SUM
2  SELECT SUM(`x`) AS `sum`
3  FROM   `overflow`

1  -- Запрос 2: AVG
2  SELECT AVG(`x`) AS `avg`
3  FROM   `overflow`
```

MS SQL Исследование 2.1.5.EXP.A

```
1  -- Запрос 1: SUM
2  SELECT SUM([x]) AS [sum]
3  FROM   [overflow]

1  -- Запрос 2: AVG
2  SELECT AVG([x]) AS [avg]
3  FROM   [overflow]
```

Oracle Исследование 2.1.5.EXP.A

```
1  -- Запрос 1: SUM
2  SELECT SUM("x") AS "sum"
3  FROM   "overflow"

1  -- Запрос 2: AVG
2  SELECT AVG("x") AS "avg"
3  FROM   "overflow"
```

MySQL и Oracle выполнят запросы 2.1.5.EXP.A и вернут корректные данные, а в MS SQL Server мы получим сообщение об ошибке:

```
Msg 8115, Level 16, State 2, Line 1. Arithmetic overflow error converting
expression to data type int.
```

Это вполне логично, т.к. при попытке разместить в переменной типа **INT** трёхкратное максимальное значение типа **INT**, возникает переполнение разрядной сетки.

MySQL и Oracle менее подвержены этому эффекту, т.к. у них «в запасе» есть **DECIMAL** и **NUMBER**, в формате которых и происходит вычисление. Но при достаточном объёме данных там тоже возникает переполнение разрядной сетки.

Особая опасность этой ошибки состоит в том, что на стадии разработки и поверхностного тестирования БД она не проявляется, и лишь со временем, когда у реальных пользователей накопится большой объём данных, в какой-то момент ранее прекрасно работавшие запросы перестают работать.

Исследование 2.1.5.EXP.B. Чтобы больше не возвращаться к особенностям работы агрегирующих функций, рассмотрим их поведение в случае наличия в анализируемом поле **NULL**-значений, а также в случае пустого набора входных значений.

Используем ранее созданную таблицу **table_with_nulls**[28]. В ней по-прежнему находятся следующие данные:

x
1
1
2
NULL

Выполним запросы 2.1.5.EXP.B:

MySQL Исследование 2.1.5.EXP.B

```
SELECT SUM(`x`) AS `sum`,
       MIN(`x`) AS `min`,
       MAX(`x`) AS `max`,
       AVG(`x`) AS `avg`
FROM   `table_with_nulls`
```

MS SQL Исследование 2.1.5.EXP.B

```
SELECT SUM([x]) AS [sum],
       MIN([x]) AS [min],
       MAX([x]) AS [max],
       AVG(CAST([x] AS FLOAT)) AS [avg]
FROM   [table_with_nulls]
```

Oracle Исследование 2.1.5.EXP.B

```
SELECT SUM("x") AS "sum",
       MIN("x") AS "min",
       MAX("x") AS "max",
       AVG("x") AS "avg"
FROM   "table_with_nulls"
```

Все запросы 2.1.5.EXP.B возвращают почти одинаковый результат (разница только в количестве знаков после запятой в значении функции **AVG**: у MySQL там четыре знака, у MS SQL Server и Oracle — 14 и 38 знаков соответственно):

sum	min	max	avg
4	1	2	1.3333

Как легко заметить из полученных данных, ни одна из функций не учитывает **NULL**-значения.

Исследование 2.1.5.EXP.C. Последний эксперимент будет заключаться в применении к выборке такого условия, которому не соответствует ни один ряд: таким образом в выборке окажется пустое множество строк.

MySQL Исследование 2.1.5.EXP.C

```
SELECT SUM(`x`) AS `sum`,
       MIN(`x`) AS `min`,
       MAX(`x`) AS `max`,
       AVG(`x`) AS `avg`
FROM   `table_with_nulls`
WHERE  `x` < 0
```

MS SQL Исследование 2.1.5.EXP.C

```
1  SELECT SUM([x]) AS [sum],
2         MIN([x]) AS [min],
3         MAX([x]) AS [max],
4         AVG(CAST([x] AS FLOAT)) AS [avg]
5  FROM   [table_with_nulls]
6  WHERE  [x] < 0
```

Oracle Исследование 2.1.5.EXP.C

```
1  SELECT SUM("x") AS "sum",
2         MIN("x") AS "min",
3         MAX("x") AS "max",
4         AVG("x") AS "avg"
5  FROM   "table_with_nulls"
6  WHERE  "x" < 0
```

Здесь все три СУБД также работают одинаково, наглядно демонстрируя, что на пустом множестве функции **SUM**, **MIN**, **MAX**, **AVG** возвращают **NULL**:

sum	min	max	avg
NULL	NULL	NULL	NULL

Также обратите внимание, что при вычислении среднего значения не произошло ошибки деления на ноль.

Логику работы на пустом множестве значений функции **COUNT** мы уже рассмотрели ранее (см. исследование 2.1.3.EXP.B{30}).

Задание 2.1.5.TSK.A: показать первую и последнюю даты выдачи книги читателю.

2.1.6. ПРИМЕР 6: УПОРЯДОЧИВАНИЕ ВЫБОРКИ

Задача 2.1.6.a{37}: показать все книги в библиотеке в порядке возрастания их года издания.

Задача 2.1.6.b{38}: показать все книги в библиотеке в порядке убывания их года издания.

Ожидаемый результат 2.1.6.a.

b_name	b_year
Курс теоретической физики	1981
Евгений Онегин	1985
Сказка о рыбаке и рыбке	1990
Искусство программирования	1993
Язык программирования C++	1996
Психология программирования	1998
Основание и империя	2000

Ожидаемый результат 2.1.6.b.

b_name	b_year
Основание и империя	2000
Психология программирования	1998
Язык программирования C++	1996
Искусство программирования	1993
Сказка о рыбаке и рыбке	1990
Евгений Онегин	1985
Курс теоретической физики	1981

Решение 2.1.6.a[36].

Для упорядочивания[1] результатов выборки необходимо применить конструкцию **`ORDER BY`** (строка 4 каждого запроса), в которой мы указываем:

- поле, по которому производится сортировка (**`b_year`**)
- направление сортировки (**`ASC`**).

MySQL Решение 2.1.6.a

```
1  SELECT `b_name`,
2         `b_year`
3  FROM   `books`
4  ORDER  BY `b_year` ASC
```

MS SQL Решение 2.1.6.a

```
1  SELECT [b_name],
2         [b_year]
3  FROM   [books]
4  ORDER  BY [b_year] ASC
```

[1] В повседневной жизни всё равно большинство людей использует тут термин «сортировка» вместо «упорядочивание», но это не совсем идентичные понятия. «Сортировка» больше относится к процессу изменения порядка, а «упорядочивание» к готовому результату.

Oracle Решение 2.1.6.a

```
SELECT "b_name",
       "b_year"
FROM   "books"
ORDER  BY "b_year" ASC
```

Решение 2.1.6.b{36}.

Здесь (в отличие от решения{37} задачи 2.1.6.a{36}) всего лишь необходимо поменять направление сортировки с «по возрастанию» (**ASC**) на «по убыванию» (**DESC**).

MySQL Решение 2.1.6.b

```
SELECT `b_name`,
       `b_year`
FROM   `books`
ORDER  BY `b_year` DESC
```

MS SQL Решение 2.1.6.b

```
SELECT [b_name],
       [b_year]
FROM   [books]
ORDER  BY [b_year] DESC
```

Oracle Решение 2.1.6.b

```
SELECT "b_name",
       "b_year"
FROM   "books"
ORDER  BY "b_year" DESC
```

Альтернативное решение можно получить добавлением знака «минус» перед именем поля, по которому сортировка реализована по возрастанию, т.е. **ORDER BY числовое_поле DESC** эквивалентно **ORDER BY -числовое_поле ASC**.

MySQL Решение 2.1.6.b (альтернативный вариант)

```
SELECT `b_name`,
       `b_year`
FROM   `books`
ORDER  BY -`b_year` ASC
```

MS SQL Решение 2.1.6.b (альтернативный вариант)

```
SELECT [b_name],
       [b_year]
FROM   [books]
ORDER  BY -[b_year] ASC
```

Oracle Решение 2.1.6.b (альтернативный вариант)

```
SELECT "b_name",
       "b_year"
FROM   "books"
ORDER  BY -"b_year" ASC
```

Исследование 2.1.6.EXP.A. Ещё одна неожиданная проблема с упорядочиванием связана с тем, где различные СУБД по умолчанию располагают **NULL**-значения — в начале выборки или в конце. Проверим.

Снова воспользуемся готовой таблицей **table_with_nulls**[28]. В ней по-прежнему находятся следующие данные:

x
1
1
2
NULL

Выполним запросы 2.1.6.EXP.A:

MySQL Исследование 2.1.6.EXP.A

```
SELECT `x`
FROM   `table_with_nulls`
ORDER  BY `x` DESC
```

MS SQL Исследование 2.1.6.EXP.A

```
SELECT [x]
FROM   [table_with_nulls]
ORDER  BY [x] DESC
```

Oracle Исследование 2.1.6.EXP.A

```
SELECT "x"
FROM   "table_with_nulls"
ORDER  BY "x" DESC
```

Результаты будут следующими (обратите внимание на то, где расположено **NULL**-значение):

x
2
1
1
NULL

MySQL

x
2
1
1
NULL

MS SQL Server

x
NULL
2
1
1

Oracle

Получить в MySQL и MS SQL Server поведение, аналогичное поведению Oracle (и наоборот), можно с использованием следующих запросов:

MySQL Исследование 2.1.6.EXP.A

```
SELECT `x`
FROM   `table_with_nulls`
ORDER  BY `x` IS NULL DESC,
          `x` DESC
```

MS SQL Исследование 2.1.6.EXP.A

```
1   SELECT [x]
2   FROM   [table_with_nulls]
3   ORDER  BY ( CASE
4                 WHEN [x] IS NULL THEN 0
5                 ELSE 1
6               END ) ASC,
7              [x] DESC
```

Oracle Исследование 2.1.6.EXP.A

```
1   SELECT "x"
2   FROM   "table_with_nulls"
3   ORDER  BY "x" DESC NULLS LAST
```

В случае с Oracle мы явно указываем, поместить ли **NULL**-значения в начало выборки (**NULLS FIRST**) или в конец выборки (**NULLS LAST**). Поскольку MySQL и MS SQL Server не поддерживают такой синтаксис, выборку приходится упорядочивать по двум уровням: первый уровень (строка 3 для MySQL, строки 3-6 для MS SQL Server) — по признаку «является ли значение поля **NULL**'ом», второй уровень — по самому значению поля.

Задание 2.1.6.TSK.A: показать список авторов в обратном алфавитном порядке (т.е. «Я → А»).

2.1.7. ПРИМЕР 7: ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ УСЛОВИЙ

Задача 2.1.7.a[{41}]: показать книги, изданные в период 1990-2000 годов, представленные в библиотеке в количестве трёх и более экземпляров.

Задача 2.1.7.b[{42}]: показать идентификаторы и даты выдачи книг за лето 2012-го года.

Ожидаемый результат 2.1.7.a.

b_name	b_year	b_quantity
Сказка о рыбаке и рыбке	1990	3
Основание и империя	2000	5
Язык программирования C++	1996	3
Искусство программирования	1993	7

Ожидаемый результат 2.1.7.b.

sb_id	sb_start
42	2012-06-11
57	2012-06-11

Решение 2.1.7.a[40].

Для каждой СУБД приведено два варианта запроса — с ключевым словом **BETWEEN** (часто используемого как раз для указания диапазона дат), и без него — в виде двойного неравенства (что выглядит более привычно для имеющих опыт программирования).

В случае использования **BETWEEN**, гдраницы включаются в диапазон искомых значений.

В представленных ниже решениях с ключевым словом **BETWEEN** отдельные условия осознанно не взяты в скобки. Синтаксически такой вариант верен и отлично работает, но он тем сложнее читается, чем больше составных частей входит в сложное условие. Потому всё же рекомендуется брать каждую отдельную часть в скобки.

MySQL Решение 2.1.7.a

```
1  -- Вариант 1: использование BETWEEN
2  SELECT `b_name`,
3         `b_year`,
4         `b_quantity`
5  FROM   `books`
6  WHERE  `b_year` BETWEEN 1990 AND 2000
7         AND `b_quantity` >= 3

1  -- Вариант 2: использование двойного неравенства
2  SELECT `b_name`,
3         `b_year`,
4         `b_quantity`
5  FROM   `books`
6  WHERE  `b_year` >= 1990
7         AND `b_year` <= 2000
8         AND `b_quantity` >= 3
```

MS SQL Решение 2.1.7.a

```
1  -- Вариант 1: использование BETWEEN
2  SELECT [b_name],
3         [b_year],
4         [b_quantity]
5  FROM   [books]
6  WHERE  [b_year] BETWEEN 1990 AND 2000
7         AND [b_quantity] >= 3

1  -- Вариант 2: использование двойного неравенства
2  SELECT [b_name],
3         [b_year],
4         [b_quantity]
5  FROM   [books]
6  WHERE  [b_year] >= 1990
7         AND [b_year] <= 2000
8         AND [b_quantity] >= 3
```

Oracle Решение 2.1.7.a

```
-- Вариант 1: использование BETWEEN
SELECT "b_name",
       "b_year",
       "b_quantity"
FROM   "books"
WHERE  "b_year" BETWEEN 1990 AND 2000
       AND "b_quantity" >= 3

-- Вариант 2: использование двойного неравенства
SELECT "b_name",
       "b_year",
       "b_quantity"
FROM   "books"
WHERE  "b_year" >= 1990
       AND "b_year" <= 2000
       AND "b_quantity" >= 3
```

Решение 2.1.7.b[40].

Сначала рассмотрим правильный и неправильный вариант решения, а потом поясним, в чём проблема с неправильным.

Правильный вариант:

MySQL Решение 2.1.7.b

```
SELECT `sb_id`,
       `sb_start`
FROM   `subscriptions`
WHERE  `sb_start` >= '2012-06-01'
       AND `sb_start` < '2012-09-01'
```

MS SQL Решение 2.1.7.b

```
SELECT [sb_id],
       [sb_start]
FROM   [subscriptions]
WHERE  [sb_start] >= '2012-06-01'
       AND [sb_start] < '2012-09-01'
```

Oracle Решение 2.1.7.b

```
SELECT "sb_id",
       "sb_start"
FROM   "subscriptions"
WHERE  "sb_start" >= TO_DATE('2012-06-01', 'yyyy-mm-dd')
       AND "sb_start" < TO_DATE('2012-09-01', 'yyyy-mm-dd')
```

Указание правой границы диапазона дат в виде строгого неравенства удобно потому, что не надо запоминать или вычислять последний день месяца (особенно актуально для февраля) или писать конструкции вида 23:59:59.9999 (если надо учесть ещё и время). Какой бы частью даты мы ни оперировали (год, месяц, день, час, минута, секунда, доли секунд), всегда можно сформировать следующее значение, не входящее в искомый диапазон, и использовать строгое неравенство.

Также обратите внимание на строки 4-5 запросов 2.1.7.b: MySQL и MS SQL Server допускают строковое указание даты (и автоматически выполняют необходимые преобразования), в то время как Oracle требует явного преобразования строкового представления даты к соответствующему типу данных.

Пришло время рассмотреть очень распространённый, но **неправильный** вариант решения, который возвращает корректный результат, но является фатальным для производительности. Вместо того, чтобы получить две константы (начала и конца диапазона дат) и напрямую сравнивать с ними значения анализируемого столбца таблицы (используя индекс), СУБД вынуждена **для каждой записи в таблице** выполнять два преобразования, и затем сравнивать результаты с заданными константами (напрямую, без использования индекса, т.к. в таблице нет индексов по результатам извлечения года и месяца из даты).

MySQL Решение 2.1.7.b (неправильный с точки зрения производительности вариант)

```
1  SELECT `sb_id`,
2         `sb_start`
3  FROM   `subscriptions`
4  WHERE  YEAR(`sb_start`) = 2012
5         AND MONTH(`sb_start`) BETWEEN 6 AND 8
```

MS SQL Решение 2.1.7.b (неправильный с точки зрения производительности вариант)

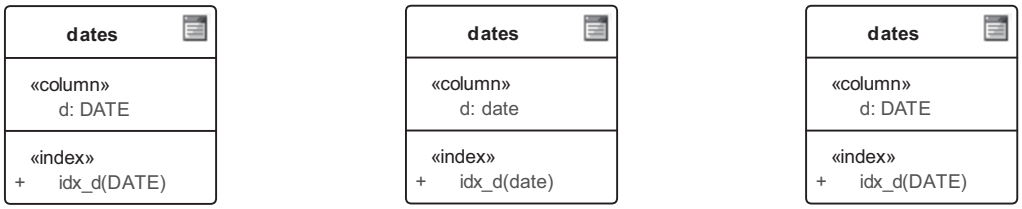
```
1  SELECT [sb_id],
2         [sb_start]
3  FROM   [subscriptions]
4  WHERE  YEAR([sb_start]) = 2012
5         AND MONTH([sb_start]) BETWEEN 6 AND 8
```

Oracle Решение 2.1.7.b (неправильный с точки зрения производительности вариант)

```
1  SELECT "sb_id",
2         "sb_start"
3  FROM   "subscriptions"
4  WHERE  EXTRACT(year FROM "sb_start") = 2012
5         AND EXTRACT(month FROM "sb_start") BETWEEN 6 AND 8
```

Исследование 2.1.7.EXP.A. Продемонстрируем разницу в производительности СУБД при выполнении запросов из правильного и неправильного решения.

Создадим в БД «Исследование» ещё одну таблицу с одним полем для хранения даты, над которым будет построен индекс.



MySQL MS SQL Server Oracle

*Рисунок 2.1.h — **Таблица `dates` во всех трёх СУБД***

Наполним получившуюся таблицу миллионом записей и выполним по сто раз каждый вариант запроса 2.1.7.b — правильный и неправильный.

Медианные значения времени выполнения таковы:

	MySQL	MS SQL Server	Oracle
Правильное решение	0.003	0.059	0.674
Неправильное решение	0.436	0.086	0.966
Разница (раз)	145.3	1.4	1.5

Даже такое тривиальное исследование показывает, что запрос, требующий извлечения части поля, приводит к падению производительности от примерно полутора до примерно ста пятидесяти раз.

К сожалению, иногда у нас нет возможности указать диапазон дат (например, нужно будет показать информацию о книгах, выданных с 3-го по 7-е число каждого месяца каждого года или о книгах, выданных в любое воскресенье, см. задачу 2.3.3.b{198}). Если таких запросов много, стоит либо хранить дату в виде отдельных полей (год, месяц, число, день недели), либо создавать т.н. «вычисляемые поля» для года, месяца, числа, дня недели, и над этими полями также строить индекс.

Задание 2.1.7.TSK.A: показать книги, количество экземпляров которых меньше среднего по библиотеке.

Задание 2.1.7.TSK.B: показать идентификаторы и даты выдачи книг за первый год работы библиотеки (первым годом работы библиотеки считать все даты с первой выдачи книги по 31-е декабря (включительно) того года, когда библиотека начала работать).

2.1.8. ПРИМЕР 8: ПОИСК МНОЖЕСТВА МИНИМАЛЬНЫХ И МАКСИМАЛЬНЫХ ЗНАЧЕНИЙ

Задача 2.1.8.a{нет}: показать книгу, представленную в библиотеке максимальным количеством экземпляров.

В самой формулировке этой задачи скрывается ловушка: в такой формулировке задача 2.1.8.a не имеет правильного решения (потому оно и не будет показано). На самом деле, здесь не одна задача, а три.

Задача 2.1.8.b{45}: показать просто одну любую книгу, количество экземпляров которой максимально (равно максимуму по всем книгам).

Задача 2.1.8.c{47}: показать все книги, количество экземпляров которых максимально (и одинаково для всех этих показанных книг).

Задача 2.1.8.d{49}: показать книгу (если такая есть), количество экземпляров которой больше, чем у любой другой книги.

Ожидаемые результаты по этим трём задачам таковы (причём для первой результат может меняться, т.к. мы не указываем, какую именно книгу из числа соответствующих условию показывать).

Ожидаемый результат 2.1.8.b.

b_name	b_quantity
Курс теоретической физики	12

Ожидаемый результат 2.1.8.c.

b_name	b_quantity
Курс теоретической физики	12

Ожидаемый результат 2.1.8.d.

b_name	b_quantity
Курс теоретической физики	12

Кажется странным, не так ли? Три разных задачи, но три одинаковых результата. Да, при том наборе данных, который сейчас есть в базе данных, все три решения дают одинаковый результат, но стоит, например, привезти в библиотеку ещё десять экземпляров «Евгения Онегина» (чтобы их тоже стало 12, как и у книги «Курс теоретической физики»), как результат становится таким (убедитесь в этом сами, сделав соответствующую правку в базе данных).

Возможный ожидаемый результат 2.1.8.b.

b_name	b_quantity
Евгений Онегин	12

Возможный ожидаемый результат 2.1.8.c.

b_name	b_quantity
Евгений Онегин	12
Курс теоретической физики	12

Возможный ожидаемый результат 2.1.8.d.

b_name	b_quantity
{пустое множество, запрос вернул ноль рядов}	

Решение 2.1.8.b[{44}].

Эта задача — самая простая: нужно упорядочить выборку по убыванию поля **b_quantity** и взять первый ряд выборки.

MySQL Решение 2.1.8.b

```
1  SELECT `b_name`,
2         `b_quantity`
3  FROM   `books`
4  ORDER  BY `b_quantity` DESC
5  LIMIT  1
```

MS SQL Решение 2.1.8.b

```
1  -- Вариант 1: использование TOP
2  SELECT TOP 1 [b_name],
3               [b_quantity]
4  FROM   [books]
5  ORDER  BY [b_quantity] DESC

1  -- Вариант 2: использование FETCH NEXT
2  SELECT   [b_name],
3           [b_quantity]
4  FROM     [books]
5  ORDER BY [b_quantity] DESC
6  OFFSET   0 ROWS
7  FETCH    NEXT 1 ROWS ONLY
```

Oracle Решение 2.1.8.b

```
1  SELECT "b_name",
2         "b_quantity"
3  FROM   (SELECT "b_name",
4                 "b_quantity",
5              ROW_NUMBER() OVER(ORDER BY "b_quantity" DESC) AS "rn"
6          FROM   "books")
7  WHERE  "rn" = 1
```

В MySQL всё просто: достаточно указать, какое количество рядов (**LIMIT 1**) возвращать из упорядоченной выборки.

В MS SQL Server вариант с **TOP 1** вполне аналогичен решению для MySQL: мы говорим СУБД, что нас интересует только один первый («верхний») ряд. Второй запрос 2.1.8.b для MS SQL Server (строки 5-6) говорит СУБД пропустить ноль рядов и вернуть один следующий ряд.

Решение для Oracle самое нетривиальное. Версия Oracle 12c уже поддерживает синтаксис, аналогичный MS SQL Server, но мы работаем с версией Oracle 11gR2, и потому вынуждены реализовывать классический вариант.

В строке 5 запроса 2.1.8.b для Oracle мы используем специальную аналитическую функцию **ROW_NUMBER**, позволяющую присвоить строке номер на основе выражения. В нашем случае выражение имеет упрощённый вариант: не указан принцип разбиения строк на группы и перезапуска нумерации — мы лишь просим СУБД пронумеровать строки в порядке их следования в упорядоченной выборке.

Oracle не позволяет в одном и том же запросе как пронумеровать строки, так и наложить условие на выборку на основе этой нумерации, потому мы вынуждены использовать подзапрос (строки 3-6). Альтернативой подзапросу может быть т.н. CTE (Common Table Expression, общее табличное выражение), но о CTE мы поговорим позже{75}.

Частым вопросом относительно решения для Oracle является применимость здесь не функции **ROW_NUMBER**, а «псевдополя» **ROWNUM**. Его применять нельзя, т.к. нумерация с его использованием происходит **до** срабатывания **ORDER BY**.

Исследование 2.1.8.EXP.A. Продемонстрируем результат неверного использования «псевдополя» **ROWNUM** вместо функции **ROW_NUMBER**.

Oracle Исследование 2.1.8.EXP.A, пример неверного запроса

```
1  SELECT "b_name",
2         "b_quantity"
3  FROM   (SELECT "b_name",
4                 "b_quantity",
5                 ROWNUM AS "rn"
6          FROM   "books"
7          ORDER  BY "b_quantity" DESC)
8  WHERE  "rn" = 1
```

Результатом выполнения этого запроса является:

b_name	b_quantity
Евгений Онегин	2

Такой результат получается потому, что подзапрос (строки 3-7) возвращает следующие данные:

b_name	b_quantity	rn
Курс теоретической физики	12	6
Искусство программирования	7	7
Основание и империя	5	3
Сказка о рыбаке и рыбке	3	2
Язык программирования C++	3	5
Евгений Онегин	**2**	**1**
Психология программирования	1	4

Легко заметить, что нумерация строк произошла до упорядочивания, и первый номер был присвоен книге «Евгений Онегин». При этом вариант с функцией **ROW_NUMBER** работает корректно.

Решение 2.1.8.c[44].

Если нам нужно показать все книги, представленные равным максимальным количеством экземпляров, нужно выяснить это максимальное количество и использовать полученное значение как условие выборки.

MySQL Решение 2.1.8.c

```
1  -- Вариант 1: использование MAX
2  SELECT `b_name`,
3         `b_quantity`
4  FROM   `books`
5  WHERE  `b_quantity` = (SELECT MAX(`b_quantity`)
6                         FROM   `books`)
```

MS SQL Решение 2.1.8.c

```
1  -- Вариант 1: использование MAX
2  SELECT [b_name],
3         [b_quantity]
4  FROM   [books]
5  WHERE  [b_quantity] = (SELECT MAX([b_quantity])
6                         FROM   [books])

1  -- Вариант 2: использование RANK
2  SELECT [b_name],
3         [b_quantity]
4  FROM   (SELECT [b_name],
5                 [b_quantity],
6                 RANK() OVER (ORDER BY [b_quantity] DESC) AS [rn]
7          FROM   [books]) AS [temporary_data]
8  WHERE  [rn] = 1
```

Oracle Решение 2.1.8.c

```
1  -- Вариант 1: использование MAX
2  SELECT "b_name",
3         "b_quantity"
4  FROM   "books"
5  WHERE  "b_quantity" = (SELECT MAX("b_quantity")
6                         FROM   "books")

1  -- Вариант 2: использование RANK
2  SELECT "b_name",
3         "b_quantity"
4  FROM   (SELECT "b_name",
5                 "b_quantity",
6                 RANK() OVER (ORDER BY "b_quantity" DESC) AS "rn"
7          FROM   "books")
8  WHERE  "rn" = 1
```

В случае с MySQL доступен только один вариант решения[2]: подзапросом (строки 5-6) выяснить максимальное количество экземпляров книг и использовать полученное число как условие выборки. Этот же вариант решения прекрасно работает в MS SQL Server и Oracle.

MS SQL Server и Oracle поддерживают т.н. «оконные (ранжирующие) функции», позволяющие реализовать второй вариант решения. Обратите внимание на строку 6 этого варианта: MS SQL Server требует явного именования подзапроса, являющегося источником данных, а Oracle не требует (именование подзапроса допустимо, но в данном случае не является обязательным).

Функция **RANK** позволяет ранжировать строки выборки (т.е. расставить их на 1-е, 2-е, 3-е, 4-е и так далее места) по указанному условию (в нашем случае — по убыванию количества экземпляров книг). Книги с одинаковыми количествами экземпляров будут занимать одинаковые места, а на первом месте будут книги с максимальным количеством экземпляров. Остаётся только показать эти книги, занявшие первое место.

Для наглядности рассмотрим, что возвращает подзапрос, представленный строками 4-7 второго варианта решения для MS SQL Server и Oracle:

---

[2] На самом деле, в MySQL можно эмулировать ранжирующие (оконные) функции. Примеры такой эмуляции представлены в исследовании 2.1.8.EXP.D{48} и решениях 2.2.7.d{119}, 2.2.9.d{139}.

MS SQL Решение 2.1.8.c (фрагмент запроса)

```
SELECT [b_name],
       [b_quantity],
       RANK() OVER (ORDER BY [b_quantity] DESC) AS [rn]
FROM   [books]
```

Oracle Решение 2.1.8.c (фрагмент запроса)

```
SELECT "b_name",
       "b_quantity",
       RANK() OVER (ORDER BY "b_quantity" DESC) AS "rn"
FROM   "books"
```

b_name	b_quantity	rn
Курс теоретической физики	12	1
Искусство программирования	7	2
Основание и империя	5	3
Сказка о рыбаке и рыбке	3	4
Язык программирования C++	3	4
Евгений Онегин	2	6
Психология программирования	1	7

Исследование 2.1.8.EXP.B. Что работает быстрее — **MAX** или **RANK**? Используем ранее созданную и наполненную данными (десять миллионов записей) таблицу **test_counts**{23} и проверим.

Медианные значения времени после ста выполнений запросов таковы:

	MS QL Server	Oracle
MAX	0.009	0.341
RANK	6.940	0.818

Вариант с **MAX** в данном случае работает быстрее. Однако в исследовании 2.2.7.EXP.A{122} будет показана обратная ситуация, когда вариант с ранжированием окажется значительно быстрее варианта с функцией **MAX**. Таким образом, вновь и вновь подтверждается идея о том, что исследование производительности стоит выполнять в конкретной ситуации на конкретном наборе данных.

Решение 2.1.8.d{44}.

Чтобы найти «абсолютного рекордсмена» по количеству экземпляров, мы используем функцию работы с множествами **ALL**, которая позволяет сравнить некоторое значение с каждым элементом множества.

MySQL Решение 2.1.8.d

```
-- Вариант 1: использование ALL и подзапроса
SELECT `b_name`,
       `b_quantity`
FROM   `books` AS `ext`
WHERE  `b_quantity` > ALL (SELECT `b_quantity`
                           FROM   `books` AS `int`
                           WHERE  `ext`.`b_id` != `int`.`b_id`)
```

MS SQL Решение 2.1.8.d

```
-- Вариант 1: использование ALL и подзапроса
SELECT [b_name],
       [b_quantity]
FROM   [books] AS [ext]
WHERE  [b_quantity] > ALL (SELECT [b_quantity]
                           FROM   [books] AS [int]
                           WHERE  [ext].[b_id] != [int].[b_id])
```

```
-- Вариант 2: использование общего табличного выражения и RANK
WITH [ranked]
     AS (SELECT [b_name],
                [b_quantity],
                RANK()
                  OVER (
                    ORDER BY [b_quantity] DESC) AS [rank]
         FROM   [books]),
     [counted]
     AS (SELECT [rank],
                COUNT(*) AS [competitors]
         FROM   [ranked]
         GROUP  BY [rank])
SELECT [b_name],
       [b_quantity]
FROM   [ranked]
       JOIN [counted]
         ON [ranked].[rank] = [counted].[rank]
WHERE  [counted].[rank] = 1
       AND [counted].[competitors] = 1
```

Oracle Решение 2.1.8.d

```
-- Вариант 1: использование ALL и подзапроса
SELECT "b_name",
       "b_quantity"
FROM   "books" "ext"
WHERE  "b_quantity" > ALL (SELECT "b_quantity"
                           FROM   "books" "int"
                           WHERE  "ext"."b_id" != "int"."b_id")
```

Oracle Решение 2.1.8.d (продолжение)

```
1  -- Вариант 2: использование общего табличного выражения и RANK
2  WITH "ranked"
3       AS (SELECT "b_name",
4                  "b_quantity",
5                  RANK()
6                    OVER (
7                      ORDER BY "b_quantity" DESC) AS "rank"
8           FROM   "books"),
9       "counted"
10      AS (SELECT "rank",
11                 COUNT(*) AS "competitors"
12          FROM   "ranked"
13          GROUP  BY "rank")
14 SELECT "b_name",
15        "b_quantity"
16 FROM   "ranked"
17        JOIN "counted"
18          ON "ranked"."rank" = "counted"."rank"
19 WHERE  "counted"."rank" = 1
20        AND "counted"."competitors" = 1
```

Первые варианты решения для всех трёх СУБД идентичны (только в Oracle здесь для именования таблицы между исходным именем и псевдонимом не должно быть ключевого слова **AS**). Теперь поясним, как это работает.

Одна и та же таблица **books** фигурирует в запросе 2.1.8.d дважды — под именем **ext** (для внешней части запроса) и **int** (для внутренней части запроса). Это нужно затем, чтобы СУБД могла применить условие выборки, представленное в строке 7: для каждой строки таблицы **ext** выбрать значение поля **b_quantity** из всех строк таблицы **int** кроме той строки, которая сейчас рассматривается в таблице **ext**.

Это фундаментальный принцип построения т.н. коррелирующих запросов, потому покажем логику работы СУБД графически. Итак, у нас есть семь книг с идентификаторами от 1 до 7:

Строка из таблицы ext	Какие строки анализируются в таблице int
1	2, 3, 4, 5, 6, 7 {т.е. все, кроме 1-й}
2	1, 3, 4, 5, 6, 7 {т.е. все, кроме 2-й}
3	1, 2, 4, 5, 6, 7 {т.е. все, кроме 3-й}
4	1, 2, 3, 5, 6, 7 {т.е. все, кроме 4-й}
5	1, 2, 3, 4, 6, 7 {т.е. все, кроме 5-й}
6	1, 2, 3, 4, 5, 7 {т.е. все, кроме 6-й}
7	1, 2, 3, 4, 5, 6 {т.е. все, кроме 7-й}

Выбрав соответствующие значения поля **b_quantity**, СУБД проверяет, чтобы значение, выбранное из таблицы **ext** было больше каждого из значений, выбранных из таблицы **int**:

Значение ext.b_quantity	Набор значений int.b_quantity
2	3, 5, 1, 3, 12, 7
3	2, 5, 1, 3, 12, 7
5	2, 3, 1, 3, 12, 7
1	2, 3, 5, 3, 12, 7
3	2, 3, 5, 1, 12, 7
12	**2, 3, 5, 1, 3, 7**
7	2, 3, 5, 1, 3, 12

Как мы видим, только для книги с количеством экземпляров, равным 12, заданное условие выполняется. Если бы ещё хотя бы у одной книги было такое же количество экземпляров, ни для одной строки выборки условие бы не выполнилось, и запрос возвратил бы пустой результат (что и показано в начале этого примера, см. «Возможный ожидаемый результат 2.1.8.d»).

Вторые варианты решения доступны только в MS SQL Server и Oracle (т.к. MySQL не поддерживает общие табличные выражения, хотя в нём и возможно написать подобный вариант решения через подзапросы). Идея второго варианта состоит в том, чтобы отказаться от коррелирующих подзапросов. Для этого в строках 1-7 второго варианта решения 2.1.8.d в общем табличном выражении `ranked` производится подготовка данных с ранжированием книг по количеству их экземпляров, в строках 8-12 в общем табличном выражении `counted` определяется количество книг, занявших одно и то же место, а в основной части запроса в строках 13-19 происходит объединение полученных данных с наложением фильтра «должно быть первое место, и на первом месте должна быть только одна книга».

Исследование 2.1.8.EXP.C. Что работает быстрее — вариант с коррелирующим подзапросом или с общим табличным выражением и последующим объединением? Выполним по сто раз соответствующие запросы 2.1.8.d на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений запросов таковы:

	MS QL Server	**Oracle**
Коррелирующий подзапрос	0.198	0.402
Общее табличное выражение с последующим объединением	0.185	0.378

Вариант с общим табличным выражением оказывается пусть и немного, но всё же быстрее.

Исследование 2.1.8.EXP.D. И ещё раз продемонстрируем разницу в скорости работы решений, основанных на коррелирующих запросах, агрегирующих функциях и ранжировании. Представим, что для каждого читателя нам нужно показать **ровно одну** (любую, если их может быть несколько, но — одну) запись из таблицы `subscriptions`, соответствующую первому визиту читателя в библиотеку.

В результате мы ожидаем увидеть:

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
2	1	1	2011-01-12	2011-02-12	N
3	3	3	2012-05-17	2012-07-17	Y
57	4	5	2012-06-11	2012-08-11	N

В каждой из СУБД возможно три варианта получения этого результата (в MS SQL Server и Oracle добавляется ещё вариант с общим табличным выражением, но мы осознанно не будем его рассматривать, ограничившись аналогом с подзапросами):

- на основе коррелирующих запросов (при этом в Oracle придётся очень нетривиальным образом эмулировать в подзапросе ограничение на количество выбранных записей, реализуемое через `LIMIT 1` и `TOP 1` в MySQL и MS SQL Server);
- на основе агрегирующих функций;
- на основе ранжирования (обратите внимание, что в MySQL нет соответствующих готовых решений, потому нам придётся эмулировать поведение доступной в MS SQL Server и Oracle функции `ROW_NUMBER` средствами MySQL).

MySQL Исследование 2.1.8.EXP.D

```
-- Вариант 1: решение на основе коррелирующих запросов
SELECT `sb_id`,
       `sb_subscriber`,
       `sb_book`,
       `sb_start`,
       `sb_finish`,
       `sb_is_active`
FROM   `subscriptions` AS `outer`
WHERE  `sb_id` = (SELECT `sb_id`
                  FROM   `subscriptions` AS `inner`
                  WHERE  `outer`.`sb_subscriber` = `inner`.`sb_subscriber`
                  ORDER  BY `sb_start` ASC
                  LIMIT  1)
```

```
-- Вариант 2: решение на основе агрегирующих функций
SELECT `sb_id`,
       `subscriptions`.`sb_subscriber`,
       `sb_book`,
       `sb_start`,
       `sb_finish`,
       `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` IN (SELECT MIN(`sb_id`)
                   FROM   `subscriptions`
                          JOIN (SELECT `sb_subscriber`,
                                       MIN(`sb_start`) AS `min_date`
                                FROM   `subscriptions`
                                GROUP  BY `sb_subscriber`) AS `prepared`
                            ON `subscriptions`.`sb_subscriber` =
                               `prepared`.`sb_subscriber`
                              AND `subscriptions`.`sb_start` =
                                  `prepared`.`min_date`
                   GROUP  BY `prepared`.`sb_subscriber`,
                             `prepared`.`min_date`)
```

```
-- Вариант 3: решение на основе ранжирования
SELECT `subscriptions`.`sb_id`,
       `sb_subscriber`,
       `sb_book`,
       `sb_start`,
       `sb_finish`,
       `sb_is_active`
FROM   `subscriptions`
       JOIN (SELECT `sb_id`,
                    @row_num := IF(@prev_value = `sb_subscriber`,
                                   @row_num + 1,
                                   1) AS `visit`,
                    @prev_value := `sb_subscriber`
             FROM   `subscriptions`,
                    (SELECT @row_num := 1) AS `x`,
                    (SELECT @prev_value := '') AS `y`
             ORDER  BY `sb_subscriber` ASC,
                       `sb_start` ASC) AS `prepared`
         ON `subscriptions`.`sb_id` = `prepared`.`sb_id`
WHERE  `visit` = 1
```

MS SQL Исследование 2.1.8.EXP.D

```
-- Вариант 1: решение на основе коррелирующих запросов
SELECT [sb_id],
       [sb_subscriber],
       [sb_book],
       [sb_start],
       [sb_finish],
       [sb_is_active]
FROM   [subscriptions] AS [outer]
WHERE  [sb_id] = (SELECT TOP 1 [sb_id]
                  FROM   [subscriptions] AS [inner]
                  WHERE  [outer].[sb_subscriber] = [inner].[sb_subscriber]
                  ORDER  BY [sb_start] ASC)
```

```
-- Вариант 2: решение на основе агрегирующих функций
SELECT [sb_id],
       [subscriptions].[sb_subscriber],
       [sb_book],
       [sb_start],
       [sb_finish],
       [sb_is_active]
FROM   [subscriptions]
WHERE  [sb_id] IN (SELECT MIN([sb_id])
                   FROM   [subscriptions]
                          JOIN (SELECT [sb_subscriber],
                                       MIN([sb_start]) AS [min_date]
                                FROM   [subscriptions]
                                GROUP  BY [sb_subscriber]) AS [prepared]
                            ON [subscriptions].[sb_subscriber] =
                               [prepared].[sb_subscriber]
                               AND [subscriptions].[sb_start] =
                                   [prepared].[min_date]
                   GROUP  BY [prepared].[sb_subscriber],
                             [prepared].[min_date])
```

```
-- Вариант 3: решение на основе ранжирования
SELECT [subscriptions].[sb_id],
       [sb_subscriber],
       [sb_book],
       [sb_start],
       [sb_finish],
       [sb_is_active]
FROM   [subscriptions]
       JOIN (SELECT [sb_id],
                    ROW_NUMBER()
                      OVER (
                        PARTITION BY [sb_subscriber]
                        ORDER BY [sb_start] ASC) AS [visit]
             FROM   [subscriptions]) AS [prepared]
         ON [subscriptions].[sb_id] = [prepared].[sb_id]
WHERE  [visit] = 1
```

Oracle Исследование 2.1.8.EXP.D

```
-- Вариант 1: решение на основе коррелирующих запросов
SELECT "sb_id",
       "sb_subscriber",
       "sb_book",
       "sb_start",
       "sb_finish",
       "sb_is_active"
FROM   "subscriptions" "outer"
WHERE  "sb_id" = (SELECT DISTINCT FIRST_VALUE("inner"."sb_id")
                                    OVER (
                                      ORDER BY "inner"."sb_start" ASC)
                  FROM   "subscriptions" "inner"
                  WHERE  "outer"."sb_subscriber" = "inner"."sb_subscriber")
```

```
-- Вариант 2: решение на основе агрегирующих функций
SELECT "sb_id",
       "subscriptions"."sb_subscriber",
       "sb_book",
       "sb_start",
       "sb_finish",
       "sb_is_active"
FROM   "subscriptions"
WHERE  "sb_id" IN (SELECT MIN("sb_id")
                   FROM   "subscriptions"
                          JOIN (SELECT "sb_subscriber",
                                       MIN("sb_start") AS "min_date"
                                FROM   "subscriptions"
                                GROUP  BY "sb_subscriber") "prepared"
                            ON "subscriptions"."sb_subscriber" =
                               "prepared"."sb_subscriber"
                               AND "subscriptions"."sb_start" =
                                   "prepared"."min_date"
                   GROUP  BY "prepared"."sb_subscriber",
                             "prepared"."min_date")
```

```
-- Вариант 3: решение на основе ранжирования
SELECT "subscriptions"."sb_id",
       "sb_subscriber",
       "sb_book",
       "sb_start",
       "sb_finish",
       "sb_is_active"
FROM   "subscriptions"
       JOIN (SELECT "sb_id",
                    ROW_NUMBER()
                      OVER (
                        partition BY "sb_subscriber"
                        ORDER BY "sb_start" ASC) AS "visit"
             FROM   "subscriptions") "prepared"
         ON "subscriptions"."sb_id" = "prepared"."sb_id"
WHERE  "visit" = 1
```

После выполнения каждого из представленных запросов по одному разу (увидев результаты, вы легко поймёте, почему только под одному разу) на базе данных «Большая библиотека» получились следующие значения времени:

	MySQL	MS SQL Server	Oracle
Решение на основе коррелирующих запросов	43:12:17.674	247:53:21.645	763:32:22.878
Решение на основе агрегирующих функций	44:17:12.736	688:43:58.244	041:19:43.344
Решение на основе ранжирования	00:18:48.828	000:00:41.511	000:02:32.274

Каждая из СУБД оказалась самой быстрой в одном из видов запросов, но во всех трёх СУБД решение на основе ранжирования стало бесспорным лидером (сравните, например, лучший и худший результаты для MS SQL Server: 41.5 секунды вместо почти месяца).

Задание 2.1.8.TSK.A: показать идентификатор одного (любого) читателя, взявшего в библиотеке больше всего книг.

Задание 2.1.8.TSK.B: показать идентификаторы всех «самых читающих читателей», взявших в библиотеке больше всего книг.

Задание 2.1.8.TSK.C: показать идентификатор «читателя-рекордсмена», взявшего в библиотеке больше книг, чем любой другой читатель.

Задание 2.1.8.TSK.D: написать второй вариант решения задачи 2.1.8.d (основанный на общем табличном выражении) для MySQL, проэмулировав общее табличное выражение через подзапросы.

2.1.9. ПРИМЕР 9: ВЫЧИСЛЕНИЕ СРЕДНЕГО ЗНАЧЕНИЯ АГРЕГИРОВАННЫХ ДАННЫХ

Задача 2.1.9.a{57}: показать, сколько в среднем экземпляров книг сейчас на руках у каждого читателя.

Задача 2.1.9.b{58}: показать, сколько в среднем книг сейчас на руках у каждого читателя.

Задача 2.1.9.c{59}: показать, на сколько в среднем дней читатели берут книги (учесть только случаи, когда книги были возвращены).

Задача 2.1.9.d{59}: показать, сколько в среднем дней читатели читают книгу (учесть оба случая — и когда книга была возвращена, и когда книга не была возвращена).

Разница между задачами 2.1.9.a и 2.1.9.b состоит в том, что первая учитывает случаи «у читателя на руках несколько экземпляров одной и той же книги», а вторая любое количество таких дубликатов будет считать одной книгой.

Разница между задачами 2.1.9.c и 2.1.9.d состоит в том, что для решения задачи 2.1.9.c достаточно данных из таблицы, а для решения задачи 2.1.9.d придётся определять текущую дату.

Ожидаемый результат 2.1.9.a.

avg_books
2.5

Ожидаемый результат 2.1.9.b.

avg_books
2.5

Ожидаемые результаты 2.1.9.a и 2.1.9.b совпадают на имеющемся наборе данных, т.к. ни у кого из читателей сейчас нет на руках двух и более экземпляров одной и той же книги, но вы можете изменить данные в базе данных и посмотреть, как изменится результат выполнения запроса.

Ожидаемый результат 2.1.9.c.

avg_days
46

Ожидаемый результат 2.1.9.d.

avg_days
560.6364

Обратите внимание: ожидаемый результат 2.1.9.d зависит от даты, в которую выполнялся запрос. Потому у вас он обязательно будет другим!

Решение 2.1.9.a[{56}].

См. пояснение ниже после решения 2.1.9.b[{58}].

MySQL Решение 2.1.9.a

```
SELECT AVG(`books_per_subscriber`) AS `avg_books`
FROM   (SELECT COUNT(`sb_book`) AS `books_per_subscriber`
        FROM   `subscriptions`
        WHERE  `sb_is_active` = 'Y'
        GROUP  BY `sb_subscriber`) AS `count_subquery`
```

MS SQL Решение 2.1.9.a

```
1  SELECT AVG(CAST([books_per_subscriber] AS FLOAT)) AS [avg_books]
2  FROM   (SELECT COUNT([sb_book]) AS [books_per_subscriber]
3           FROM   [subscriptions]
4           WHERE  [sb_is_active] = 'Y'
5           GROUP  BY [sb_subscriber]) AS [count_subquery]
```

Oracle Решение 2.1.9.a

```
1  SELECT AVG("books_per_subscriber") AS "avg_books"
2  FROM   (SELECT COUNT("sb_book") AS "books_per_subscriber"
3           FROM   "subscriptions"
4           WHERE  "sb_is_active" = 'Y'
5           GROUP  BY "sb_subscriber")
```

Решение 2.1.9.b{56}.

MySQL Решение 2.1.9.b

```
1  SELECT AVG(`books_per_subscriber`) AS `avg_books`
2  FROM   (SELECT COUNT(DISTINCT `sb_book`) AS `books_per_subscriber`
3           FROM   `subscriptions`
4           WHERE  `sb_is_active` = 'Y'
5           GROUP  BY `sb_subscriber`) AS `count_subquery`
```

MS SQL Решение 2.1.9.b

```
1  SELECT AVG(CAST([books_per_subscriber] AS FLOAT)) AS [avg_books]
2  FROM   (SELECT COUNT(DISTINCT [sb_book]) AS [books_per_subscriber]
3           FROM   [subscriptions]
4           WHERE  [sb_is_active] = 'Y'
5           GROUP  BY [sb_subscriber]) AS [count_subquery]
```

Oracle Решение 2.1.9.b

```
1  SELECT AVG("books_per_subscriber") AS "avg_books"
2  FROM   (SELECT COUNT(DISTINCT "sb_book") AS "books_per_subscriber"
3           FROM   "subscriptions"
4           WHERE  "sb_is_active" = 'Y'
5           GROUP  BY "sb_subscriber")
```

Суть решений 2.1.9.a{57} и 2.1.9.b{58} состоит в том, чтобы сначала подготовить агрегированные данные (подзапрос в строках 2-5 всех шести представленных выше запросов 2.1.9.a-2.1.9.b), а затем вычислить среднее значение от этих заранее подготовленных значений.

Разница в решении задач 2.1.9.a{56} и 2.1.9.b{56} состоит в использовании во втором случае ключевого слова **DISTINCT** (строка 2 всех шести запросов 2.1.9.a-2.1.9.b), позволяющего проигнорировать дубликаты книг.

Разница в решениях для трёх разных СУБД состоит в необходимости предварительного приведения аргумента функции **AVG** к дроби в MS SQL Server и отсутствию необходимости именовать подзапрос в Oracle. В остальном решения 2.1.9.a{57} и 2.1.9.b{58} идентичны для всех трёх СУБД.

Для наглядности покажем, какие данные были возвращены подзапросами (строки 2-5 всех шести запросов 2.1.9.a-2.1.9.b):

books_per_subscriber
3
2

Решение 2.1.9.c[56].

MySQL Решение 2.1.9.c

```
1  SELECT AVG(DATEDIFF(`sb_finish`, `sb_start`)) AS `avg_days`
2  FROM   `subscriptions`
3  WHERE  `sb_is_active` = 'N'
```

MS SQL Решение 2.1.9.c

```
1  SELECT AVG(CAST (DATEDIFF(day, [sb_start], [sb_finish]) AS FLOAT))
2         AS [avg_days]
3  FROM   [subscriptions]
4  WHERE  [sb_is_active] = 'N'
```

Oracle Решение 2.1.9.c

```
1  SELECT AVG("sb_finish" - "sb_start") AS "avg_days"
2  FROM   "subscriptions"
3  WHERE  "sb_is_active" = 'N'
```

Для всех трёх СУБД решение задачи 2.1.9.c[56] является одинаковым за исключением синтаксиса вычисления разницы в днях между двумя датами (строка 1 в каждом из трёх запросов 2.1.9.c). Результаты вычисления разницы дат выглядят следующим образом (эти данные поступают на вход функции **AVG**):

data
31
61
61
61
31
31

Решение 2.1.9.d[56].

Здесь самый большой вопрос состоит в том, как определить учитываемый диапазон времени. Его начало хранится в поле **sb_start**, а вот с завершением не всё так просто. Если книга была возвращена, датой завершения периода чтения можно считать значение поля **sb_finish**, если оно находится в прошлом. Если книга не была возвращена, и значение **sb_finish** находится в прошлом, датой завершения чтения можно считать текущую дату.

Но что делать, если значение **sb_finish** находится в будущем? Правильный ответ на подобный вопрос в реальной жизни можно получить только от заказчика разрабатываемого приложения. Мы же в учебных целях решим, что в такой ситуации будем использовать текущую дату, если книга уже возвращена, и значение **sb_finish**, если она ещё не возвращена.

Итого, у нас есть четыре варианта расчёта времени чтения книги:

- **sb_finish** в прошлом, книга возвращена: **sb_finish** - **sb_start**.
- **sb_finish** в прошлом, книга не возвращена: **текущая_дата** - **sb_start**.
- **sb_finish** в будущем, книга возвращена: **текущая_дата** - **sb_start**.
- **sb_finish** в будущем, книга не возвращена: **sb_finish** - **sb_start**.

Легко заметить, что алгоритмов вычисления всего два, но активируется каждый из них двумя независимыми условиями. Самым простым способом решения здесь является объединение результатов двух запросов с помощью оператора **UNION**. Важно помнить, что **UNION** по умолчанию работает в **DISTINCT**-режиме, т.е. нужно явно писать **UNION ALL**.

MySQL Решение 2.1.9.d

```
1   SELECT AVG(`diff`) AS `avg_days`
2   FROM   (
3           SELECT DATEDIFF(`sb_finish`, `sb_start`) AS `diff`
4           FROM   `subscriptions`
5           WHERE    ( `sb_finish` <= CURDATE() AND `sb_is_active` = 'N' )
6                 OR ( `sb_finish` > CURDATE() AND `sb_is_active` = 'Y' )
7           UNION ALL
8           SELECT DATEDIFF(CURDATE(), `sb_start`) AS `diff`
9           FROM   `subscriptions`
10          WHERE    ( `sb_finish` <= CURDATE() AND `sb_is_active` = 'Y' )
11                OR ( `sb_finish` > CURDATE() AND `sb_is_active` = 'N' )
12        ) AS `diffs`
```

Несмотря на громоздкость этого запроса, он прост. В строке 1 решается основная задача — вычисление среднего значения, строки 2-13 лишь подготавливают необходимые данные. В строке 7 используется только что упомянутый оператор **UNION ALL**, с помощью которого объединяются результаты двух отдельных запросов, представленных в строках 3-6 и 8-11 соответственно. Объёмные конструкции **WHERE** в строках 5-6 и 10-11 определяют условия, по которым активируется один из двух алгоритмов вычисления времени чтения книги.

Вот такие наборы данных возвращают запросы в строках 3-6 и 8-11.

Первый запрос (строки 3-6) возвращает:

diff
31
61
61
61
3684
31

Второй запрос (строки 8-11) возвращает.

diff
1266
458
458
28
28

Решения для MS SQL Server и Oracle следуют той же логике и отличаются только синтаксисом получения разницы в днях между датами (и необходимостью приведения аргумента функции **AVG** к дроби для MS SQL Server).

MS SQL Решение 2.1.9.d

```
SELECT AVG(CAST([diff] AS FLOAT)) AS [avg_days]
FROM  (
       SELECT DATEDIFF(day, [sb_start], [sb_finish]) AS [diff]
       FROM   [subscriptions]
       WHERE     ( [sb_finish] <= CONVERT(date, GETDATE())
                   AND [sb_is_active] = 'N' )
             OR ( [sb_finish] > CONVERT(date, GETDATE())
                   AND [sb_is_active] = 'Y' )
       UNION ALL
       SELECT DATEDIFF(day, [sb_start], CONVERT(date, GETDATE())) AS [diff]
       FROM   [subscriptions]
       WHERE     ( [sb_finish] <= CONVERT(date, GETDATE())
                   AND [sb_is_active] = 'Y' )
             OR ( [sb_finish] > CONVERT(date, GETDATE())
                   AND [sb_is_active] = 'N' )
       ) AS [diffs]
```

Oracle Решение 2.1.9.d

```
SELECT AVG("diff") AS "avg_days"
FROM  (
       SELECT ("sb_finish" - "sb_start") AS "diff"
       FROM   "subscriptions"
       WHERE     ( "sb_finish" <= TRUNC(SYSDATE) AND "sb_is_active" = 'N' )
             OR ( "sb_finish" > TRUNC(SYSDATE) AND "sb_is_active" = 'Y' )
       UNION ALL
       SELECT (TRUNC(SYSDATE) - "sb_start") AS "diff"
       FROM   "subscriptions"
       WHERE     ( "sb_finish" <= TRUNC(SYSDATE) AND "sb_is_active" = 'Y' )
             OR ( "sb_finish" > TRUNC(SYSDATE) AND "sb_is_active" = 'N' )
       )
```

В решении для Oracle также стоит отметить, что там не существует такого понятия, как «просто дата без времени», потому мы вынуждены использовать конструкцию **`TRUNC(SYSDATE)`**, чтобы «отрезать» время от даты. Иначе результат вычитания в строке 8 вернёт не целое число дней, а дробное (что отличается от поведения MySQL и MS SQL Server), а также могут неожиданным образом работать все условия, в которых фигурирует дата.

И ещё один очевидный, но достойный упоминания факт: во всех запросах для всех СУБД в условиях, связанных с текущей датой, мы использовали **`<= текущая_дата`** и **`> текущая дата`**, т.е. включали «сегодня» в один из диапазонов. Если использовать два строгих неравенства или два нестрогих, мы рискуем либо «потерять» записи со значением **`sb_finish`**, совпадающим с текущей датой, либо учесть такие случаи дважды.

Если вы внимательно изучили только что рассмотренное решение, у вас обязан был возникнуть вопрос о том, как на корректность вычислений влияет запись из таблицы **`subscriptions`** с идентификатором 91 (в ней дата возврата книги находится в прошлом по отношению к дате выдачи книги):

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
91	4	1	2015-10-07	2015-03-07	Y

Ответ прост и неутешителен: да, из-за этой ошибки результат получается искажённым. Что делать? Ничего. Мы не можем позволить себе роскошь в каждом запросе на выборку учитывать возможность возникновения таких ошибок (например, дата выдачи книги могла оказаться в будущем). Контроль таких ситуаций должен быть возложен на операции вставки и обновления

данных. Соответствующий пример (см. задачу 4.2.1.a{331}) будет рассмотрен в разделе{284}, посвящённом триггерам.

Есть ещё одна проблема с разницами дат, о которой стоит помнить: в повседневной жизни мы привыкли округлять эти значения, в то время как СУБД этой операции не выполняет. Проверьте себя: сколько лет прошло между 2011-01-01 и 2012-01-01? Один год, верно? А между 2011-01-01 и 2012-12-31?

Исследование 2.1.9.EXP.A. Проверка поведения СУБД при вычислении расстояния между двумя датами в годах.

MySQL Исследование 2.1.9.EXP.A

```
1   SELECT YEAR('2012-01-01') - YEAR('2011-01-01') - (
2          DATE_FORMAT('2012-01-01', '%m%d') <
3          DATE_FORMAT('2011-01-01', '%m%d'))

1   SELECT YEAR('2012-12-31') - YEAR('2011-01-01') - (
2          DATE_FORMAT('2012-12-31', '%m%d') <
3          DATE_FORMAT('2011-01-01', '%m%d'))
```

MS SQL Исследование 2.1.9.EXP.A

```
1   SELECT DATEDIFF(year, '2011-01-01', '2012-01-01')

1   SELECT DATEDIFF(year, '2011-01-01', '2012-12-31')
```

Oracle Исследование 2.1.9.EXP.A

```
1   SELECT FLOOR(MONTHS_BETWEEN(DATE '2012-01-01', DATE '2011-01-01')
2   / 12)
    FROM dual;
1
2   SELECT FLOOR(MONTHS_BETWEEN(DATE '2012-12-31', DATE '2011-01-01')
    / 12)
    FROM dual;
```

Все шесть запросов 2.1.6.EXP.A вернут один и тот же результат: 1, т.е. между указанными датами с точки зрения СУБД прошёл один год. И это правда: прошёл «один полный год».

Также обратите внимание, насколько по-разному решается задача вычисления разницы между датами в годах в различных СУБД. Особенно интересны строки 2-3 в запросах для MySQL: они позволяют получить корректный результат, когда значения года различны, но на самом деле год ещё не прошёл (например, 2011-05-01 и 2012-04-01).

Задание 2.1.9.TSK.A: показать, сколько в среднем экземпляров книг есть в библиотеке.

Задание 2.1.9.TSK.B: показать в днях, сколько в среднем времени читатели уже зарегистрированы в библиотеке (временем регистрации считать диапазон от первой даты получения читателем книги до текущей даты).

2.1.10. ПРИМЕР 10: ИСПОЛЬЗОВАНИЕ ГРУППИРОВКИ ДАННЫХ

Задача 2.1.10.a[64]: показать по каждому году, сколько раз в этот год читатели брали книги.

Задача 2.1.10.b[65]: показать по каждому году, сколько читателей в год воспользовалось услугами библиотеки.

Задача 2.1.10.c[66]: показать, сколько книг было возвращено и не возвращено в библиотеку.

Ожидаемый результат 2.1.10.a.

year	books_taken
2011	2
2012	3
2014	3
2015	3

Ожидаемый результат 2.1.10.b.

year	subscribers
2011	1
2012	3
2014	2
2015	2

Ожидаемый результат 2.1.10.c.

status	books
Returned	6
Not returned	5

Как следует из названия главы, все три задачи будут решаться с помощью группировки данных, т.е. использования `GROUP BY`.

Решение 2.1.10.a[63].

MySQL Решение 2.1.10.a

```
1  SELECT YEAR(`sb_start`) AS `year`,
2         COUNT(`sb_id`)   AS `books_taken`
3  FROM   `subscriptions`
4  GROUP  BY `year`
5  ORDER  BY `year`
```

MS SQL Решение 2.1.10.a

```
1  SELECT YEAR([sb_start]) AS [year],
2         COUNT([sb_id])   AS [books_taken]
3  FROM   [subscriptions]
4  GROUP  BY YEAR([sb_start])
5  ORDER  BY [year]
```

Oracle Решение 2.1.10.a

```
1  SELECT EXTRACT(year FROM "sb_start") AS "year",
2         COUNT("sb_id")                AS "books_taken"
3  FROM   "subscriptions"
4  GROUP  BY EXTRACT(year FROM "sb_start")
5  ORDER  BY "year"
```

Обратите внимание на разницу в 4-й строке в запросах 2.1.10.a: MySQL позволяет в **`GROUP BY`** сослаться на имя только что вычисленного выражения, в то время как MS SQL Server и Oracle не позволяют этого сделать.

Ранее мы уже рассматривали логику работы группировок[21], но подчеркнём это ещё раз. После извлечения значения года и «объединения ячеек по признаку равенства значений» полученный СУБД результат условно можно представить так:

year	результат группировки	сколько ячеек объединено
2011	2011	2
2011		
2012	2012	3
2012		
2012		
2014	2014	3
2014		
2014		
2015	2015	3
2015		
2015		

Решение 2.1.10.b{63}.

MySQL Решение 2.1.10.b

```
SELECT YEAR(`sb_start`)                AS `year`,
       COUNT(DISTINCT `sb_subscriber`) AS `subscribers`
FROM   `subscriptions`
GROUP  BY `year`
ORDER  BY `year`
```

MS SQL Решение 2.1.10.b

```
SELECT YEAR([sb_start])                AS [year],
       COUNT(DISTINCT [sb_subscriber]) AS [subscribers]
FROM   [subscriptions]
GROUP  BY YEAR([sb_start])
ORDER  BY [year]
```

Oracle Решение 2.1.10.b

```
SELECT EXTRACT(year FROM "sb_start")   AS "year",
       COUNT(DISTINCT "sb_subscriber") AS "subscribers"
FROM   "subscriptions"
GROUP  BY EXTRACT(year FROM "sb_start")
ORDER  BY "year"
```

Решение 2.1.10.b{65} очень похоже на решение 2.1.10.a{64}: разница лишь в том, что в первом случае нас интересуют все записи за каждый год, а во втором — количество идентификаторов читателей без повторений за каждый год. Покажем это графически:

year	sb_subscriber	группировка по году	число уникальных id читателя
2011	1	2011	1
2011	1		
2012	1	2012	3
2012	3		
2012	4		
2014	1	2014	2
2014	3		
2014	3		
2015	1	2015	2
2015	4		
2015	4		

Решение 2.1.10.c[63].

MySQL Решение 2.1.10.c

```
1  SELECT IF(`sb_is_active` = 'Y', 'Not returned', 'Returned') AS `status`,
2         COUNT(`sb_id`)                                        AS `books`
3  FROM   `subscriptions`
4  GROUP  BY `status`
5  ORDER  BY `status` DESC
```

MS SQL Решение 2.1.10.c

```
1  SELECT (CASE
2            WHEN [sb_is_active] = 'Y'
3            THEN 'Not returned'
4            ELSE 'Returned'
5           END)           AS [status],
6         COUNT([sb_id]) AS [books]
7  FROM   [subscriptions]
8  GROUP  BY (CASE
9               WHEN [sb_is_active] = 'Y'
10              THEN 'Not returned'
11              ELSE 'Returned'
12             END)
13 ORDER  BY [status] DESC
```

Oracle Решение 2.1.10.c

```
1  SELECT (CASE
2            WHEN "sb_is_active" = 'Y'
3            THEN 'Not returned'
4            ELSE 'Returned'
5           END)           AS "status",
6         COUNT("sb_id") AS "books"
7  FROM   "subscriptions"
8  GROUP  BY (CASE
9               WHEN "sb_is_active" = 'Y'
10              THEN 'Not returned'
11              ELSE 'Returned'
12             END)
13 ORDER  BY "status" DESC
```

В решении 2.1.10.c есть одна сложность: нужно на основе значений поля `sb_is_active` `Y` (книга на руках, т.е. не возвращена) и `N` (книга возвращена) получить человекочитаемые осмысленные надписи «Not returned» и «Returned».

В MySQL все необходимые действия помещаются в одну строку (строка 1), а благодаря способности MySQL ссылаться на имя вычисленного выражения в `GROUP BY`, нет необходимости повторять всю эту конструкцию в строке 4.

MS SQL Server и Oracle поддерживают одинаковый, но чуть более громоздкий синтаксис: строки 1-5 запросов для этих СУБД содержат необходимые преобразования, а дублирование этого же кода в строках 8-12 вызвано тем, что эти СУБД не позволяют в `GROUP BY` сослаться на вычисленное выражение по его имени.

И снова покажем графически, с какими данными приходится работать СУБД:

sb_is_active (исходное значение)	status (результат преобразования)	группировка	подсчёт
N	Returned	Returned	6
N	Returned		
N	Returned		
N	Returned		
N	Returned		
N	Returned		
Y	Not returned	Not returned	5
Y	Not returned		
Y	Not returned		
Y	Not returned		
Y	Not returned		

Исследование 2.1.10.EXP.A. Как быстро, в принципе, работает группировка на больших объёмах данных? Используем базу данных «Большая библиотека» и посчитаем, сколько книг находится на руках у каждого читателя.

MySQL Исследование 2.1.10.EXP.A

```
SELECT `sb_subscriber`,
       COUNT(`sb_id`) AS `books_taken`
FROM   `subscriptions`
WHERE  `sb_is_active` = 'Y'
GROUP  BY `sb_subscriber`
```

MS SQL Исследование 2.1.10.EXP.A

```
SELECT [sb_subscriber],
       COUNT([sb_id]) AS [books_taken]
FROM   [subscriptions]
WHERE  [sb_is_active] = 'Y'
GROUP  BY [sb_subscriber]
```

Oracle Исследование 2.1.10.EXP.A

```
SELECT "sb_subscriber",
       COUNT("sb_id") AS "books_taken"
FROM   "subscriptions"
WHERE  "sb_is_active" = 'Y'
GROUP  BY "sb_subscriber"
```

Медианы времени после выполнения по сто раз каждого из запросов 2.1.10.EXP.A:

MySQL	MS SQL Server	Oracle
34.333	8.189	2.306

С одной стороны, результаты не выглядят пугающе, но, если объём данных увеличить в 10, 100, 1000 раз и т.д. — время выполнения уже будет измеряться часами или даже днями.

Задание 2.1.10.TSK.A: переписать решение 2.1.10.c так, чтобы при подсчёте возвращённых и невозвращённых книг СУБД оперировала исходными значениями поля `sb_is_active` (т.е. `Y` и `N`), а преобразование в «Returned» и «Not returned» происходило после подсчёта.

2.2. ВЫБОРКА ИЗ НЕСКОЛЬКИХ ТАБЛИЦ

2.2.1. ПРИМЕР 11: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ КАК СПОСОБ ПОЛУЧЕНИЯ ЧЕЛОВЕКОЧИТАЕМЫХ ДАННЫХ

Следующие две задачи уже были упомянуты ранее{14}, сейчас мы рассмотрим их подробно.

Задача 2.2.1.a{69}: показать всю человекочитаемую информацию обо всех книгах (т.е. название, автора, жанр).

Задача 2.2.1.b{71}: показать всю человекочитаемую информацию обо всех обращениях в библиотеку (т.е. имя читателя, название взятой книги).

Ожидаемый результат 2.2.1.a.

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Классика
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
Курс теоретической физики	Л.Д. Ландау	Классика
Курс теоретической физики	Е.М. Лифшиц	Классика
Искусство программирования	Д. Кнут	Классика
Евгений Онегин	А.С. Пушкин	Поэзия
Сказка о рыбаке и рыбке	А.С. Пушкин	Поэзия
Психология программирования	Д. Карнеги	Программирование
Психология программирования	Б. Страуструп	Программирование
Язык программирования C++	Б. Страуструп	Программирование
Искусство программирования	Д. Кнут	Программирование
Психология программирования	Д. Карнеги	Психология
Психология программирования	Б. Страуструп	Психология
Основание и империя	А. Азимов	Фантастика

Ожидаемый результат 2.2.1.b.

b_name	s_id	s_name	sb_start	sb_finish
Евгений Онегин	1	Иванов И.И.	2011-01-12	2011-02-12
Сказка о рыбаке и рыбке	1	Иванов И.И.	2012-06-11	2012-08-11
Искусство программирования	1	Иванов И.И.	2014-08-03	2014-10-03
Психология программирования	1	Иванов И.И.	2015-10-07	2015-11-07
Основание и империя	1	Иванов И.И.	2011-01-12	2011-02-12
Основание и империя	3	Сидоров С.С.	2012-05-17	2012-07-17
Язык программирования C++	3	Сидоров С.С.	2014-08-03	2014-10-03
Евгений Онегин	3	Сидоров С.С.	2014-08-03	2014-09-03
Язык программирования C++	4	Сидоров С.С.	2012-06-11	2012-08-11
Евгений Онегин	4	Сидоров С.С.	2015-10-07	2015-03-07
Психология программирования	4	Сидоров С.С.	2015-10-08	2025-11-08

Решение 2.2.1.a[{68}].

MySQL Решение 2.2.1.a

```
SELECT `b_name`,
       `a_name`,
       `g_name`
FROM   `books`
       JOIN `m2m_books_authors` USING(`b_id`)
       JOIN `authors` USING(`a_id`)
       JOIN `m2m_books_genres` USING(`b_id`)
       JOIN `genres` USING(`g_id`)
```

MS SQL Решение 2.2.1.a

```
SELECT [b_name],
       [a_name],
       [g_name]
FROM   [books]
       JOIN [m2m_books_authors]
         ON [books].[b_id] = [m2m_books_authors].[b_id]
       JOIN [authors]
         ON [m2m_books_authors].[a_id] = [authors].[a_id]
       JOIN [m2m_books_genres]
         ON [books].[b_id] = [m2m_books_genres].[b_id]
       JOIN [genres]
         ON [m2m_books_genres].[g_id] = [genres].[g_id]
```

Oracle Решение 2.2.1.a

```
SELECT "b_name",
       "a_name",
       "g_name"
FROM   "books"
       JOIN "m2m_books_authors" USING("b_id")
       JOIN "authors" USING("a_id")
       JOIN "m2m_books_genres" USING("b_id")
       JOIN "genres" USING("g_id")
```

Ключевое отличие решений для MySQL и Oracle от решения для MS SQL Server заключается в том, что эти две СУБД поддерживают специальный синтаксис указания полей, по которым необходимо производить объединение: если такие поля имеют одинаковое имя в объединяемых таблицах, вместо конструкции `ON первая_таблица.поле = вторая_таблица.поле` можно использовать `USING(поле)`, что зачастую очень сильно повышает читаемость запроса.

Несмотря на то, что задача 2.2.1.а является, пожалуй, самым простым случаем использования `JOIN`, в ней мы объединяем пять таблиц в одном запросе, потому покажем графически на примере одной строки, как формируется финальная выборка.

В рассматриваемых таблицах есть и другие поля, но здесь показаны лишь те, которые представляют интерес в контексте данной задачи.

Первое действие:

```
[books] JOIN [m2m_books_authors]
ON [books].[b_id] = [m2m_books_authors].[b_id]
```

Таблица `books`:

b_id	b_name
1	Евгений Онегин

Таблица `m2m_books_authors`:

b_id	a_id
1	7

Промежуточный результат:

b_id	b_name	a_id
1	Евгений Онегин	7

Второе действие:

```
{результат первого действия} JOIN [authors]
ON [m2m_books_authors].[a_id] = [authors].[a_id]
```

b_id	b_name	a_id
1	Евгений Онегин	7

Таблица `authors`:

a_id	a_name
7	А.С. Пушкин

Промежуточный результат:

b_id	b_name	a_name
1	Евгений Онегин	А.С. Пушкин

Третье действие:

```
{результат второго действия} JOIN [m2m_books_genres]
ON [books].[b_id] = [m2m_books_genres].[b_id]
```

b_id	b_name	a_name
1	Евгений Онегин	А.С. Пушкин

Таблица `m2m_books_genres`:

b_id	g_id
1	1
1	5

Промежуточный результат:

b_name	a_name	g_id
Евгений Онегин	А.С. Пушкин	1
Евгений Онегин	А.С. Пушкин	5

Четвёртое действие:

```
{результат третьего действия} JOIN [genres]
ON [m2m_books_genres].[g_id] = [genres].[g_id]
```

b_name	a_name	g_id
Евгений Онегин	А.С. Пушкин	1
Евгений Онегин	А.С. Пушкин	5

Таблица **genres**:

g_id	g_name
1	Поэзия
5	Классика

Итоговый результат:

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Поэзия
Евгений Онегин	А.С. Пушкин	Классика

Аналогичная последовательность действий повторяется для каждой строки из таблицы **books**, что и приводит к ожидаемому результату 2.2.1.a.

Решение 2.2.1.b[68].

MySQL Решение 2.2.1.b

```
SELECT `b_name`,
       `s_id`,
       `s_name`,
       `sb_start`,
       `sb_finish`
FROM   `books`
       JOIN `subscriptions`
         ON `b_id` = `sb_book`
       JOIN `subscribers`
         ON `sb_subscriber` = `s_id`
```

MS SQL Решение 2.2.1.b

```
SELECT [b_name],
       [s_id],
       [s_name],
       [sb_start],
       [sb_finish]
FROM   [books]
       JOIN [subscriptions]
         ON [b_id] = [sb_book]
       JOIN [subscribers]
         ON [sb_subscriber] = [s_id]
```

Oracle Решение 2.2.1.b

```
SELECT "b_name",
       "s_id",
       "s_name",
       "sb_start",
       "sb_finish"
FROM   "books"
       JOIN "subscriptions"
         ON "b_id" = "sb_book"
       JOIN "subscribers"
         ON "sb_subscriber" = "s_id"
```

Логика решения 2.2.1.b{71} полностью эквивалентна логике решения 2.2.1.a{68} (здесь приходится объединять даже меньше таблиц — всего три, а не пять). Обратите внимание на тот факт, что если объединение происходит не по одноимённым полям, а по разноимённым, в MySQL и Oracle тоже приходится использовать конструкцию `ON` вместо `USING`.

Задание 2.2.1.TSK.A: показать список книг, у которых более одного автора.

Задание 2.2.1.TSK.B: показать список книг, относящихся ровно к одному жанру.

2.2.2. ПРИМЕР 12: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ И ПРЕОБРАЗОВАНИЕ СТОЛБЦОВ В СТРОКИ

Возможно, вы заметили, что в решении{69} задачи 2.2.1.a{68} есть неудобство: если у книги несколько авторов и/или жанров, информация начинает дублироваться (упорядочим для наглядности выборку по полю `b_name`):

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Классика
Евгений Онегин	А.С. Пушкин	Поэзия
Искусство программирования	Д. Кнут	Классика
Искусство программирования	Д. Кнут	Программирование
Курс теоретической физики	Л.Д. Ландау	Классика
Курс теоретической физики	Е.М. Лифшиц	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Д. Карнеги	Программирование
Психология программирования	Б. Страуструп	Программирование
Психология программирования	Д. Карнеги	Психология
Психология программирования	Б. Страуструп	Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
Сказка о рыбаке и рыбке	А.С. Пушкин	Поэзия
Язык программирования C++	Б. Страуструп	Программирование

Пользователи же куда больше привыкли к следующему представлению данных:

Книга	Автор(ы)	Жанр(ы)
Евгений Онегин	А.С. Пушкин	Классика, Поэзия
Искусство программирования	Д. Кнут	Классика, Программирование
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп, Д. Карнеги	Программирование, Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика, Поэзия
Язык программирования C++	Б. Страуструп	Программирование

Задача 2.2.2.a{74}: показать все книги с их авторами (дублирование названий книг не допускается).

Задача 2.2.2.b{77}: показать все книги с их авторами и жанрами (дублирование названий книг и имён авторов не допускается).

Ожидаемый результат 2.2.2.a.

book	author(s)
Евгений Онегин	А.С. Пушкин
Искусство программирования	Д. Кнут
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау
Основание и империя	А. Азимов
Психология программирования	Б. Страуструп, Д. Карнеги
Сказка о рыбаке и рыбке	А.С. Пушкин
Язык программирования C++	Б. Страуструп

Ожидаемый результат 2.2.2.b.

book	author(s)	genre(s)
Евгений Онегин	А.С. Пушкин	Классика, Поэзия
Искусство программирования	Д. Кнут	Классика, Програмирование
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп, Д. Карнеги	Программирование, Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика, Поэзия
Язык программирования C++	Б. Страуструп	Программирование

Решение 2.2.2.a[{73}].

MySQL Решение 2.2.2.a

```
1  SELECT `b_name`
2         AS `book`,
3         GROUP_CONCAT(`a_name` ORDER BY `a_name` SEPARATOR ', ')
4         AS `author(s)`
5  FROM   `books`
6         JOIN `m2m_books_authors` USING(`b_id`)
7         JOIN `authors` USING(`a_id`)
8  GROUP  BY `b_id`
9  ORDER  BY `b_name`
```

Решение для MySQL получается очень простым и элегантным потому, что эта СУБД поддерживает функцию **GROUP_CONCAT**, которая и выполняет всю основную работу. У этой функции очень развитый синтаксис (даже в нашем случае мы используем сортировку и указание разделителя), с которым обязательно стоит ознакомиться в официальной документации.

Обратите особое внимание на строку 8 запроса: ни в коем случае не стоит выполнять группировку по названию книги! Такая ошибка приводит к тому, что СУБД считает одной и той же книгой несколько разных книг с одинаковым названием (что в реальной жизни может встречаться очень часто). Группировка же по значению первичного ключа таблицы гарантирует, что никакие разные записи не будут смешаны в одну группу.

И ещё одна особенность MySQL заслуживает внимания: в строке 1 мы извлекаем поле **b_name**, которое **не** упомянуто в выражении **GROUP BY** в строке 8 и не является агрегирующей функцией. MS SQL Server и Oracle не позволяют поступать подобным образом и строго требуют, чтобы любое поле из конструкции **SELECT**, не являющееся агрегирующей функций, было явно упомянуто в выражении **GROUP BY**.

Итак, что делает **GROUP_CONCAT**? Фактически, «разворачивает часть столбца в строку» (одновременно упорядочивая авторов по алфавиту и разделяя их набором символов «, »):

b_name	a_name		author(s)
Евгений Онегин	А.С. Пушкин	→	А.С. Пушкин
Искусство программирования	Д. Кнут	→	Д. Кнут
Курс теоретической физики	Л.Д. Ландау	→	Е.М. Лифшиц, Л.Д. Ландау
	Е.М. Лифшиц		
Основание и империя	А. Азимов	→	А. Азимов
Психология программирования	Д. Карнеги	→	Б. Страуструп, Д. Карнеги
	Б. Страуструп		
Сказка о рыбаке и рыбке	А.С. Пушкин	→	А.С. Пушкин
Язык программирования C++	Б. Страуструп	→	Б. Страуструп

MS SQL Server не поддерживает функцию **GROUP_CONCAT**, а потому решение для него весьма нетривиально:

MS SQL Решение 2.2.2.a

```
1  WITH [prepared_data]
2       AS (SELECT [books].[b_id],
3                  [b_name],
4                  [a_name]
5           FROM   [books]
6                  JOIN [m2m_books_authors]
7                    ON [books].[b_id] = [m2m_books_authors].[b_id]
8                  JOIN [authors]
9                   ON [m2m_books_authors].[a_id] = [authors].[a_id]
10         )
11 SELECT [outer].[b_name]
12        AS [book],
13        STUFF ((SELECT ', ' + [inner].[a_name]
14                FROM   [prepared_data] AS [inner]
15                WHERE  [outer].[b_id] = [inner].[b_id]
16                ORDER  BY [inner].[a_name]
17               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
18                1, 2, '')
19        AS [author(s)]
20 FROM   [prepared_data] AS [outer]
21 GROUP  BY [outer].[b_id],
22           [outer].[b_name]
```

Начнём рассмотрение со строк 1-10. В них представлено т.н. CTE (Common Table Expression, общее табличное выражение). Очень упрощённо общее табличное выражение можно считать отдельным поименованным запросом, к результату выполнения которого можно обращаться как к таблице. Это особенно удобно, когда таких обращений в дальнейшем используется несколько (без общего табличного выражения с использованием классического подзапроса, тело такого подзапроса пришлось бы писать везде, где необходимо к нему обратиться).

В нашем случае общее табличное выражение возвращает такие данные:

b_id	b_name	a_name
1	Евгений Онегин	А.С. Пушкин
2	Сказка о рыбаке и рыбке	А.С. Пушкин
3	Основание и империя	А. Азимов
4	Психология программирования	Д. Карнеги
4	Психология программирования	Б. Страуструп
5	Язык программирования C++	Б. Страуструп
6	Курс теоретической физики	Л.Д. Ландау
6	Курс теоретической физики	Е.М. Лифшиц
7	Искусство программирования	Д. Кнут

Это — уже почти готовое решение, останется только «развернуть в строку» части столбца **a_name** с авторами одной и той же книги. Эту задачу выполнят строки 13-19 запроса.

Основной рабочей частью процесса является код коррелирующего подзапроса:

```
SELECT ', ' + [inner].[a_name]
FROM   [prepared_data] AS [inner]
WHERE  [outer].[b_id] = [inner].[b_id]
ORDER  BY [inner].[a_name]
```

Для каждой строки результата, полученного из общего табличного выражения, выполняется подзапрос, возвращающий данные из этого же результата, относящиеся к рассматриваемой на внешнем уровне строке. Более простой вариант коррелирующего подзапроса мы уже рассматривали[51], а теперь покажем, как работает СУБД в данном конкретном случае:

Строка из внешней части запроса (b_id)	Какие строки будут обработаны внутренней частью запроса (b_id)	Какие данные будут собраны
1	1	, А.С. Пушкин
2	2	, А.С. Пушкин
3	3	, А. Азимов
4	4 и 4	, Б. Страуструп, Д. Карнеги
4	4 и 4	, Б. Страуструп, Д. Карнеги
5	5	, Б. Страуструп
6	6 и 6	, Е.М. Лифшиц, Л.Д. Ландау
6	6 и 6	, Е.М. Лифшиц, Л.Д. Ландау
7	7	, Д. Кнут

Символы «, » (запятая и пробел), стоящие в начале каждого значения в столбце «Какие данные будут собраны» — не опечатка. Мы явно говорим СУБД извлекать именно текст в виде `«, » + имя_автора`. Чтобы в конечном результате этих символов не было, мы используем функцию `STUFF`.

Чтобы было проще пояснять, перепишем эту часть запроса в упрощённом виде:

```
STUFF ((SELECT {данные из коррелирующего подзапроса}
FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'), 1, 2, '')
```

Часть `FOR XML PATH(''), TYPE` относится к `SELECT` и говорит СУБД представить результат выборки как часть XML-документа для пути `''` (пустой строки), т.е. — просто в виде обычной строки, хоть пока и выглядящей для СУБД как «некие данные в формате XML».

Промежуточный результат уже выглядит проще:

```
STUFF (({XML-строка}).value('.', 'nvarchar(max)'), 1, 2, '')
```

Метод `value(путь, тип_данных)` применяется в MS SQL Server для извлечения строкового представления из XML. Не вдаваясь в подробности, скажем, что первый параметр `путь`, равный `.` (точка) говорит, что нужно взять текущий элемент XML-документа (у нас «текущим элементом» является наша строка), а параметр `тип_данных` выставлен в `nvarchar(max)` как один из самых удобных в MS SQL Server типов для хранения длинных строк.

Результат уже почти готов:

```
STUFF ({строка с ", " в начале}, 1, 2, '')
```

Остаётся только избавиться от символов «, » в начале каждого списка авторов. Это и делает функция `STUFF`, заменяя в строке «`{строка с ", " в начале}`» символы с первого по второй (см. второй и третий параметры функции, равные `1` и `2` соответственно) на пустую строку (см. четвёртый параметр функции, равный, `''`).

И на этом с MS SQL Server — всё, переходим к Oracle. Здесь реализуется уже третий вариант решения:

Oracle Решение 2.2.2.a

```
1   SELECT "b_name" AS "book",
2          UTL_RAW.CAST_TO_NVARCHAR2
3          (
4             LISTAGG
5             (
6                 UTL_RAW.CAST_TO_RAW("a_name"),
7                 UTL_RAW.CAST_TO_RAW(N', ')
8             )
9             WITHIN GROUP (ORDER BY "a_name")
10         )
11         AS "author(s)"
12  FROM   "books"
13         JOIN "m2m_books_authors" USING ("b_id")
14         JOIN "authors" USING("a_id")
15  GROUP  BY "b_id",
16            "b_name"
```

В Oracle решение получается проще, чем в MS SQL Server, т.к. здесь (начиная с версии 11gR2 поддерживается функция **LISTAGG**, выполняющая практически то же самое, что и **GROUP_CONCAT** в MySQL. Таким образом, строки 4-8 запроса отвечают за «разворачивание в строку части столбца».

Вызовы метода **UTL_RAW.CAST_TO_RAW** в строках 6-7 и метода **UTL_RAW.CAST_TO_NVARCHAR2** в строке 2 нужны для того, чтобы сначала представить текстовые данные в формате, который обрабатывается без интерпретации значений байтов, а затем вернуть обработанные данные из этого формата в текстовый вид. Без этого преобразования информация об авторах превращается в нечитаемый набор спецсимволов.

В остальном поведение Oracle при решении этой задачи вполне эквивалентно поведению MySQL.

Решение 2.2.2.b[73].

MySQL Решение 2.2.2.b

```
1   SELECT `b_name`
2          AS `book`,
3          GROUP_CONCAT(DISTINCT `a_name` ORDER BY `a_name` SEPARATOR ', ')
4          AS `author(s)`,
5          GROUP_CONCAT(DISTINCT `g_name` ORDER BY `g_name` SEPARATOR ', ')
6          AS `genre(s)`
7   FROM   `books`
8          JOIN `m2m_books_authors` USING(`b_id`)
9          JOIN `authors` USING(`a_id`)
10         JOIN `m2m_books_genres` USING(`b_id`)
11         JOIN `genres` USING(`g_id`)
12  GROUP  BY `b_id`
13  ORDER  BY `b_name`
```

Решение 2.2.2.b для MySQL лишь чуть-чуть сложнее решения 2.2.2.a: здесь появился ещё один вызов **GROUP_CONCAT** (строки 5-6), и в обоих вызовах **GROUP_CONCAT** появилось ключевое слово **DISTINCT**, чтобы избежать дублирования информации об авторах и жанрах, которое появляется объективным образом в процессе выполнения объединения:

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Классика
	А.С. Пушкин	Поэзия
Искусство программирования	Д. Кнут	Классика
	Д. Кнут	Программирование
Курс теоретической физики	Е.М. Лифшиц	Классика
	Л.Д. Ландау	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп	Программирование
	Б. Страуструп	Психология
	Д. Карнеги	Программирование
	Д. Карнеги	Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
	А.С. Пушкин	Поэзия
Язык программирования C++	Б. Страуструп	Программирование

MS SQL Решение 2.2.2.b

```
WITH [prepared_data]
     AS (SELECT [books].[b_id],
                [b_name],
                [a_name],
                     [g_name]
           FROM   [books]
                JOIN [m2m_books_authors]
                  ON [books].[b_id] = [m2m_books_authors].[b_id]
                JOIN [authors]
                 ON [m2m_books_authors].[a_id] = [authors].[a_id]
                JOIN [m2m_books_genres]
                  ON [books].[b_id] = [m2m_books_genres].[b_id]
                JOIN [genres]
                  ON [m2m_books_genres].[g_id] = [genres].[g_id]
         )
SELECT [outer].[b_name]
       AS [book],
       STUFF ((SELECT DISTINCT ', ' + [inner].[a_name]
                FROM   [prepared_data] AS [inner]
                WHERE  [outer].[b_id] = [inner].[b_id]
                ORDER  BY ', ' + [inner].[a_name]
              FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
                1, 2, '')
       AS [author(s)],
       STUFF ((SELECT DISTINCT ', ' + [inner].[g_name]
                FROM   [prepared_data] AS [inner]
                WHERE  [outer].[b_id] = [inner].[b_id]
                ORDER  BY ', ' + [inner].[g_name]
              FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
                1, 2, '')
       AS [genre(s)]
FROM   [prepared_data] AS [outer]
GROUP  BY [outer].[b_id],
          [outer].[b_name]
```

Данное решение для MS SQL Server тоже строится на основе решения 2.2.2.a[74] и отличается чуть большим количеством **JOIN** в общем табличном выражении (добавились строки 11-14), а также (по тем же причинам, что и в MySQL) добавлением **DISTINCT** в строках 18 и 25. Блоки строк 18-24 и 25-31 отличаются только именем выбираемого поля (в первом случае — **a_name**, во втором — **g_name**).

Oracle Решение 2.2.2.b

```
1  SELECT "book", "author(s)",
2         UTL_RAW.CAST_TO_NVARCHAR2
3         (
4            LISTAGG
5            (
6                UTL_RAW.CAST_TO_RAW("g_name"),
7                UTL_RAW.CAST_TO_RAW(N', ')
8            )
9            WITHIN GROUP (ORDER BY "g_name")
10        )
11        AS "genre(s)"
12 FROM
13 (
14        SELECT "b_id", "b_name" AS "book",
15               UTL_RAW.CAST_TO_NVARCHAR2
16               (
17                  LISTAGG
18                  (
19                     UTL_RAW.CAST_TO_RAW("a_name"),
20                     UTL_RAW.CAST_TO_RAW(N', ')
21                  )
22                  WITHIN GROUP (ORDER BY "a_name")
23               )
24               AS "author(s)"
25        FROM   "books"
26        JOIN "m2m_books_authors" USING ("b_id")
27        JOIN "authors" USING("a_id")
28        GROUP  BY "b_id",
29               "b_name"
30 ) "first_level"
31 JOIN "m2m_books_genres" USING ("b_id")
32 JOIN "genres" USING("g_id")
33 GROUP  BY "b_id",
34           "book",
35           "author(s)"
```

Здесь подзапрос в строках 13-30 представляет собой решение 2.2.2.a{74}, в котором в **SELECT** добавлено поле **b_id**, чтобы оно было доступно для дальнейших операций **JOIN** и **GROUP BY**. Если переписать запрос с учётом этой информации, получается:

Oracle Решение 2.2.2.b

```
1  SELECT "book", "author(s)",
2         UTL_RAW.CAST_TO_NVARCHAR2
3         (
4            LISTAGG
5            (
6                UTL_RAW.CAST_TO_RAW("g_name"),
7                UTL_RAW.CAST_TO_RAW(N', ')
8            )
9            WITHIN GROUP (ORDER BY "g_name")
10        )
11        AS "genre(s)"
12 FROM {данные_из_решения_2_2_2_a + поле b_id}
13 JOIN "m2m_books_genres" USING ("b_id")
14 JOIN "genres" USING("g_id")
15 GROUP  BY "b_id",
16           "book",
17           "author(s)"
```

Здесь мы применяем такой двухуровневый подход потому, что не существует простого и производительного способа устранить дублирование данных в функции **LISTAGG**. Некуда применить **DISTINCT** и нет никаких специальных встроенных механизмов дедубликации. Альтернативные решения с регулярными выражениями или подготовкой отфильтрованных данных оказываются ещё сложнее и медленнее.

Но ничто не мешает нам разбить эту задачу на два этапа, вынеся первую часть в подзапрос (который теперь можно рассматривать как готовую таблицу), а вторую часть сделав полностью идентичной первой за исключением имени агрегируемого поля (было **a_name**, стало **g_name**).

Почему не работает одноуровневое решение (когда мы пытаемся сразу в одном **SELECT** получить набор авторов и набор жанров)? Итак, у нас есть данные:

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин	Классика
	А.С. Пушкин	Поэзия
Искусство программирования	Д. Кнут	Классика
	Д. Кнут	Программирование
Курс теоретической физики	Е.М. Лифшиц	Классика
	Л.Д. Ландау	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп	Программирование
	Б. Страуструп	Психология
	Д. Карнеги	Программирование
	Д. Карнеги	Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
	А.С. Пушкин	Поэзия
Язык программирования C++	Б. Страуструп	Программирование

Попытка сразу получить в виде строки список авторов и жанров выглядит так:

b_name	a_name	g_name
Евгений Онегин	А.С. Пушкин, А.С. Пушкин	Классика, Поэзия
Искусство программирования	Д. Кнут, Д. Кнут	Классика, Программирование
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау	Классика, Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп, Б. Страуструп, Д. Карнеги, Д. Карнеги	Программирование, Психология, Программирование, Психология
Сказка о рыбаке и рыбке	А.С. Пушкин, А.С. Пушкин	Классика, Поэзия
Язык программирования C++	Б. Страуструп	Программирование

А вот как работает двухходовое решение. Сначала у нас есть такой набор данных (серым фоном отмечены строки, которые при группировке превратятся в одну строку и дадут список из более чем одного автора):

b_name	a_name
Евгений Онегин	А.С. Пушкин
Сказка о рыбаке и рыбке	А.С. Пушкин
Основание и империя	А. Азимов
Психология программирования	Д. Карнеги
Психология программирования	Б. Страуструп
Язык программирования C++	Б. Страуструп
Курс теоретической физики	Л.Д. Ландау
Курс теоретической физики	Е.М. Лифшиц
Искусство программирования	Д. Кнут

Результат выполнения первой группировки и построчного объединения:

book	author(s)
Евгений Онегин	А.С. Пушкин
Сказка о рыбаке и рыбке	А.С. Пушкин
Основание и империя	А. Азимов
Психология программирования	Б. Страуструп, Д. Карнеги
Язык программирования C++	Б. Страуструп
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау
Искусство программирования	Д. Кнут

На втором шаге набор данных о каждой книге и её авторах уже позволяет проводить группировку сразу по двум полям — `book` и `author(s)`, т.к. списки авторов уже подготовлены, и никакая информация о них не будет потеряна:

book	author(s)	g_name
Евгений Онегин	А.С. Пушкин	Классика
Евгений Онегин	А.С. Пушкин	Поэзия
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика
Сказка о рыбаке и рыбке	А.С. Пушкин	Поэзия
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп, Д. Карнеги	Программирование
Психология программирования	Б. Страуструп, Д. Карнеги	Психология
Язык программирования C++	Б. Страуструп	Программирование
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау	Классика
Искусство программирования	Д. Кнут	Классика
Искусство программирования	Д. Кнут	Программирование

И получается итоговый результат:

book	author(s)	genre(s)
Евгений Онегин	А.С. Пушкин	Классика, Поэзия
Искусство программирования	Д. Кнут	Классика, Программирование
Курс теоретической физики	Е.М. Лифшиц, Л.Д. Ландау	Классика
Основание и империя	А. Азимов	Фантастика
Психология программирования	Б. Страуструп, Д. Карнеги	Программирование, Психология
Сказка о рыбаке и рыбке	А.С. Пушкин	Классика, Поэзия
Язык программирования C++	Б. Страуструп	Программирование

Задание 2.2.2.TSK.A: показать все книги с их жанрами (дублирование названий книг не допускается).

Задание 2.2.2.TSK.B: показать всех авторов со всеми написанными ими книгами и всеми жанрами, в которых они работали (дублирование имён авторов, названий книг и жанров не допускается).

2.2.3. ПРИМЕР 13: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ И ПОДЗАПРОСЫ С УСЛОВИЕМ IN

Очень часто запросы на объединение можно преобразовать в запросы с подзапросом и ключевым словом `IN` (обратное преобразование тоже возможно). Рассмотрим несколько типичных примеров:

Задача 2.2.3.a[84]: показать список читателей, когда-либо бравших в библиотеке книги (использовать **`JOIN`**).

Задача 2.2.3.b[85]: показать список читателей, когда-либо бравших в библиотеке книги (не использовать **`JOIN`**).

Задача 2.2.3.c[88]: показать список читателей, никогда не бравших в библиотеке книги (использовать **`JOIN`**).

Задача 2.2.3.d[89]: показать список читателей, никогда не бравших в библиотеке книги (не использовать **`JOIN`**).

Легко заметить, что пары задач 2.2.3.a-2.2.3.b и 2.2.3.c-2.2.3.d как раз являются предпосылкой к использованию преобразования **`JOIN`** в **`IN`**.

Ожидаемый результат 2.2.3.a.

s_id	s_name
1	Иванов И.И.
3	Сидоров С.С.
4	Сидоров С.С.

Ожидаемый результат 2.2.3.b.

s_id	s_name
1	Иванов И.И.
3	Сидоров С.С.
4	Сидоров С.С.

Ожидаемый результат 2.2.3.c.

s_id	s_name
2	Петров П.П.

Ожидаемый результат 2.2.3.d.

s_id	s_name
2	Петров П.П.

Решение 2.2.3.a[83].

MySQL Решение 2.2.3.a

```
SELECT DISTINCT `s_id`,
                `s_name`
FROM   `subscribers`
       JOIN `subscriptions`
         ON `s_id` = `sb_subscriber`
```

MS SQL Решение 2.2.3.a

```
SELECT DISTINCT [s_id],
                [s_name]
FROM   [subscribers]
       JOIN [subscriptions]
         ON [s_id] = [sb_subscriber]
```

Oracle Решение 2.2.3.a

```
SELECT DISTINCT "s_id",
                "s_name"
FROM   "subscribers"
       JOIN "subscriptions"
         ON "s_id" = "sb_subscriber"
```

Для всех трёх СУБД это решение полностью эквивалентно. Важную роль здесь играет ключевое слово **DISTINCT**, т.к. без него результат будет вот таким (в силу того факта, что **JOIN** найдёт все случаи выдачи книг каждому из читателей, а нам по условию задачи важен просто факт того, что человек хотя бы раз брал книгу):

s_id	s_name
1	Иванов И.И.
1	Иванов И.И.
1	Иванов И.И.
1	Иванов И.И.
1	Иванов И.И.
3	Сидоров С.С.
3	Сидоров С.С.
3	Сидоров С.С.
4	Сидоров С.С.
4	Сидоров С.С.
4	Сидоров С.С.

Но у **DISTINCT** есть и опасность. Представьте, что мы решили не извлекать идентификатор читателя, ограничившись его именем (приведём пример только для MySQL, т.к. в MS SQL Server и Oracle ситуация совершенно идентична):

MySQL Решение 2.2.3.a (демонстрация потенциальной проблемы)

```
1  SELECT DISTINCT `s_name`
2  FROM    `subscribers`
3          JOIN `subscriptions`
4            ON `s_id` = `sb_subscriber`
```

Запрос вернул следующие данные:

s_name
Иванов И.И.
Сидоров С.С.

В правильном ожидаемом результате было три записи (в т.ч. двое Сидоровых с разными идентификаторами). Сейчас же эта информация утеряна.

Решение 2.2.3.b[83].

MySQL Решение 2.2.3.b

```
1  SELECT `s_id`,
2         `s_name`
3  FROM    `subscribers`
4  WHERE   `s_id` IN (SELECT DISTINCT `sb_subscriber`
5                     FROM    `subscriptions`)
```

MS SQL Решение 2.2.3.b

```
1  SELECT [s_id],
2         [s_name]
3  FROM    [subscribers]
4  WHERE   [s_id] IN (SELECT DISTINCT [sb_subscriber]
5                     FROM    [subscriptions])
```

Oracle Решение 2.2.3.b

```
1  SELECT "s_id",
2         "s_name"
3  FROM    "subscribers"
4  WHERE   "s_id" IN (SELECT DISTINCT "sb_subscriber"
5                     FROM    "subscriptions")
```

Снова решение для всех трёх СУБД совершенно одинаково. Слово **DISTINCT** в подзапросе (в 4-й строке всех трёх запросов) используется для того, чтобы СУБД приходилось анализировать меньший набор данных. Будет ли разница в производительности существенной, мы рассмотрим сразу после решения следующих двух задач.

А пока покажем графически, как работают эти два запроса. Начнём с варианта с **JOIN**. СУБД перебирает значения **s_id** из таблицы **subscribers** и проверяет, есть ли в таблице **subscriptions** записи, поле **sb_subscriber** которых содержит то же самое значение:

Таблица `subscribers`

s_id	s_name
1	Иванов И.И.
2	Петров П.П.
3	Сидоров С.С.
4	Сидоров С.С.

Таблица `subscriptions`

sb_subscriber
1
1
3
1
4
1
3
3
4
1
4

В результате поиска совпадений получается следующая картина:

s_id	sb_subscriber	s_name
1	1	Иванов И.И.
1	1	Иванов И.И.
3	3	Сидоров С.С.
1	1	Иванов И.И.
4	4	Сидоров С.С.
1	1	Иванов И.И.
3	3	Сидоров С.С.
3	3	Сидоров С.С.
4	4	Сидоров С.С.
1	1	Иванов И.И.
4	4	Сидоров С.С.

Поле `sb_subscriber` мы не указываем в `SELECT`, т.е. остаётся всего два столбца:

s_id	s_name
1	Иванов И.И.
1	Иванов И.И.
3	Сидоров С.С.
1	Иванов И.И.
4	Сидоров С.С.
1	Иванов И.И.
3	Сидоров С.С.
3	Сидоров С.С.
4	Сидоров С.С.
1	Иванов И.И.
4	Сидоров С.С.

И, наконец, благодаря применению `DISTINCT`, СУБД устраняет дубликаты:

s_id	s_name
1	Иванов И.И.
3	Сидоров С.С.
4	Сидоров С.С.

В случае с подзапросом и ключевым словом `IN` ситуация выглядит иначе. Сначала СУБД выбирает все значения `sb_subscriber`:

sb_subscriber
1
1
3
1
4
1
3
3
4
1
4

Потом происходит устранение дубликатов:

sb_subscriber
1
3
4

Теперь СУБД анализирует каждую строку таблицы `subscribers` на предмет того, входит ли её значение `s_id` в этот набор:

s_id	s_name	Набор значений sb_subscriber	Помещать ли запись в выборку
1	Иванов И.И.	***1***, 3, 4	Да
2	Петров П.П.	~~1, 3, 4~~	Нет
3	Сидоров С.С.	1, ***3***, 4	Да
4	Сидоров С.С.	1, 3, ***4***	Да

Итого получается:

s_id	s_name
1	Иванов И.И.
3	Сидоров С.С.
4	Сидоров С.С.

Примечание: здесь, расписывая пошагово логику работы СУБД мы для простоты считаем, что устранение дубликатов происходит в самом конце. На самом деле, это не так. Внутренние алгоритмы обработки данных позволяют СУБД выполнять операции дедубликации как после завершения формирования выборки, так и прямо в процессе её формирования. Решение о применении того или иного варианта будет зависеть от конкретной СУБД, используемых методов доступа (storage engine), плана выполнения запроса и иных факторов.

Решение 2.2.3.c[83].

MySQL Решение 2.2.3.c

```
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
       LEFT JOIN `subscriptions`
              ON `s_id` = `sb_subscriber`
WHERE  `sb_subscriber` IS NULL
```

MS SQL Решение 2.2.3.c

```
SELECT [s_id],
       [s_name]
FROM   [subscribers]
       LEFT JOIN [subscriptions]
              ON [s_id] = [sb_subscriber]
WHERE  [sb_subscriber] IS NULL
```

Oracle Решение 2.2.3.c

```
SELECT "s_id",
       "s_name"
FROM   "subscribers"
       LEFT JOIN "subscriptions"
              ON "s_id" = "sb_subscriber"
WHERE  "sb_subscriber" IS NULL
```

При поиске читателей, никогда не бравших книги, логика работы СУБД выглядит так. Сначала выполняется т.н. «левое (внешнее) объединение», т.е. СУБД извлекает **все** записи из таблицы **subscribers** и пытается найти им «пару» из таблицы **subscriptions**. Если «пару» найти не получилось (её нет), вместо значения поля **sb_subscriber** будет подставлено значение **NULL**:

s_id	s_name	sb_subscriber
1	Иванов И.И.	1
1	Иванов И.И.	1
1	Иванов И.И.	1
1	Иванов И.И.	1
1	Иванов И.И.	1
2	**Петров П.П.**	**NULL**
3	Сидоров С.С.	3
3	Сидоров С.С.	3
3	Сидоров С.С.	3
4	Сидоров С.С.	4
4	Сидоров С.С.	4
4	Сидоров С.С.	4

Благодаря условию **WHERE "sb_subscriber" IS NULL**, только информация о Петрове попадёт в конечную выборку:

s_id	s_name
2	Петров П.П.

Решение 2.2.3.d[{83}].

MySQL Решение 2.2.3.d

```
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` NOT IN (SELECT DISTINCT `sb_subscriber`
                      FROM   `subscriptions`)
```

MS SQL Решение 2.2.3.d

```
SELECT [s_id],
       [s_name]
FROM   [subscribers]
WHERE  [s_id] NOT IN (SELECT DISTINCT [sb_subscriber]
                      FROM   [subscriptions])
```

Oracle Решение 2.2.3.d

```
SELECT "s_id",
       "s_name"
FROM   "subscribers"
WHERE  "s_id" NOT IN (SELECT DISTINCT "sb_subscriber"
                      FROM   "subscriptions")
```

Здесь поведение СУБД аналогично решению 2.2.3.b за исключением того, что при переборе значений `sb_subscriber` нужно **не** обнаружить среди них значение `s_id`:

s_id	s_name	Набор значений sb_subscriber	Помещать ли запись в выборку
1	Иванов И.И.	***1***, 3, 4	Нет
2	Петров П.П.	~~1, 3, 4~~	Да
3	Сидоров С.С.	1, ***3***, 4	Нет
4	Сидоров С.С.	1, 3, ***4***	Нет

Получается:

s_id	s_name
2	Петров П.П.

Исследование 2.2.3.EXP.A: оценка влияния `DISTINCT` в подзапросе на производительность операции.

Настало время поговорить о производительности. Нас будет интересовать два вопроса:

- Влияет ли на производительность наличие `DISTINCT` в подзапросе (для решений с `IN`)?
- Что работает быстрее — `JOIN` или `IN` (в обоих случаях: когда мы ищем как читателей, бравших книги, так и не бравших)?

Проводить исследование будем на базе данных «Большая библиотека». Выполним по сто раз следующие запросы:

MySQL Исследование 2.2.3.EXP.A

```
-- Запрос 1: использование JOIN
SELECT DISTINCT `s_id`,
                `s_name`
FROM   `subscribers`
       JOIN `subscriptions`
         ON `s_id` = `sb_subscriber`

-- Запрос 2: использование IN (... DISTINCT ...)
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` IN (SELECT DISTINCT `sb_subscriber`
                  FROM   `subscriptions`)

-- Запрос 3: использование IN
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` IN (SELECT `sb_subscriber`
                  FROM   `subscriptions`)

-- Запрос 4: использование LEFT JOIN
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
       LEFT JOIN `subscriptions`
              ON `s_id` = `sb_subscriber`
WHERE  `sb_subscriber` IS NULL

-- Запрос 5: использование NOT IN (... DISTINCT ...)
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` NOT IN (SELECT DISTINCT `sb_subscriber`
                      FROM   `subscriptions`)

-- Запрос 6: использование NOT IN
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` NOT IN (SELECT `sb_subscriber`
                      FROM   `subscriptions`)
```

MS SQL Исследование 2.2.3.EXP.A

```
1  -- Запрос 1: использование JOIN
2  SELECT DISTINCT [s_id],
3                  [s_name]
4  FROM   [subscribers]
5         JOIN [subscriptions]
6           ON [s_id] = [sb_subscriber]

1  -- Запрос 2: использование IN (... DISTINCT ...)
2  SELECT [s_id],
3         [s_name]
4  FROM   [subscribers]
5  WHERE  [s_id] IN (SELECT DISTINCT [sb_subscriber]
6                    FROM   [subscriptions])

1  -- Запрос 3: использование IN
2  SELECT [s_id],
3         [s_name]
4  FROM   [subscribers]
5  WHERE  [s_id] IN (SELECT [sb_subscriber]
6                    FROM   [subscriptions])

1  -- Запрос 4: использование LEFT JOIN
2  SELECT [s_id],
3         [s_name]
4  FROM   [subscribers]
5         LEFT JOIN [subscriptions]
6                ON [s_id] = [sb_subscriber]
7  WHERE  [sb_subscriber] IS NULL

1  -- Запрос 5: использование NOT IN (... DISTINCT ...)
2  SELECT [s_id],
3         [s_name]
4  FROM   [subscribers]
5  WHERE  [s_id] NOT IN (SELECT DISTINCT [sb_subscriber]
6                        FROM   [subscriptions])

1  -- Запрос 6: использование NOT IN
2  SELECT [s_id],
3         [s_name]
4  FROM   [subscribers]
5  WHERE  [s_id] NOT IN (SELECT [sb_subscriber]
6                        FROM   [subscriptions])
```

Oracle Исследование 2.2.3.EXP.A

```
1  -- Запрос 1: использование JOIN
2  SELECT DISTINCT "s_id",
3                  "s_name"
4  FROM   "subscribers"
5         JOIN "subscriptions"
6           ON "s_id" = "sb_subscriber"

1  -- Запрос 2: использование IN (... DISTINCT ...)
2  SELECT "s_id",
3         "s_name"
4  FROM   "subscribers"
5  WHERE  "s_id" IN (SELECT DISTINCT "sb_subscriber"
6                    FROM   "subscriptions")

1  -- Запрос 3: использование IN
2  SELECT "s_id",
3         "s_name"
4  FROM   "subscribers"
5  WHERE  "s_id" IN (SELECT "sb_subscriber"
6                    FROM   "subscriptions")

1  -- Запрос 4: использование LEFT JOIN
2  SELECT "s_id",
3         "s_name"
4  FROM   "subscribers"
5         LEFT JOIN "subscriptions"
6                ON "s_id" = "sb_subscriber"
7  WHERE  "sb_subscriber" IS NULL

1  -- Запрос 5: использование NOT IN (... DISTINCT ...)
2  SELECT "s_id",
3         "s_name"
4  FROM   "subscribers"
5  WHERE  "s_id" NOT IN (SELECT DISTINCT "sb_subscriber"
6                        FROM   "subscriptions")

1  -- Запрос 6: использование NOT IN
2  SELECT "s_id",
3         "s_name"
4  FROM   "subscribers"
5  WHERE  "s_id" NOT IN (SELECT "sb_subscriber"
6                        FROM   "subscriptions")
```

Медианы времени, затраченного на выполнение каждого запроса:

	MySQL	MS SQL Server	Oracle
JOIN	91.065	8.574	1.136
IN (… DISTINCT …)	0.068	6.711	0.298
IN	0.039	6.723	0.309
LEFT JOIN	45.788	8.695	0.284
NOT IN (… DISTINCT …)	45.564	7.437	0.329
NOT IN	46.020	7.384	0.311

Перед проведением исследования мы ставили два вопроса, и теперь у нас есть ответы:

- Влияет ли на производительность наличие `DISTINCT` в подзапросе (для решений с `IN`)?
  - В случае с `IN` наличие `DISTINCT` ощутимо замедляет работу MySQL и немного ускоряет работу MS SQL Server и Oracle.
  - В случае с `NOT IN` наличие `DISTINCT` немного ускоряет работу MySQL и немного замедляет работу MS SQL Server и Oracle.
- Что работает быстрее — `JOIN` или `IN` (в обоих случаях: когда мы ищем как читателей, бравших книги, так и не бравших)?
  - `JOIN` работает медленнее `IN`.
  - `LEFT JOIN` работает немного медленнее `NOT IN` в MySQL и MS SQL Server и немного быстрее `NOT IN` в Oracle.

Поскольку большинство результатов крайне близки по значениям, однозначный вывод получается только один: `IN` работает быстрее `JOIN`, в остальных случаях стоит проводить дополнительные исследования.

Задание 2.2.3.TSK.A: показать список книг, которые когда-либо были взяты читателями.

Задание 2.2.3.TSK.B: показать список книг, которые никто из читателей никогда не брал.

2.2.4. ПРИМЕР 14: НЕТРИВИАЛЬНЫЕ СЛУЧАИ ИСПОЛЬЗОВАНИЯ УСЛОВИЯ IN И ЗАПРОСОВ НА ОБЪЕДИНЕНИЕ

Существуют задачи, которые на первый взгляд решаются очень просто. Однако оказывается, что простое и очевидное решение является неверным.

Задача 2.2.4.a{94}: показать список читателей, у которых сейчас на руках нет книг (использовать `JOIN`).

Задача 2.2.4.b{97}: показать список читателей, у которых сейчас на руках нет книг (не использовать `JOIN`).

В задачах 2.2.3.* примера 13{82} всё было просто: если в таблице **subscriptions** есть информация о читателе, значит, он брал книги в библиотеке, а если нет — не брал. Теперь же нас будут интересовать как читатели, никогда не бравшие книги (объективно у них на руках нет книг), так и читатели, бравшие книги (но кто-то вернул всё, что брал, а кто-то ещё что-то читает).

Ожидаемый результат 2.2.4.a.

s_id	s_name
1	Иванов И.И.
2	Петров П.П.

Ожидаемый результат 2.2.4.b.

s_id	s_name
1	Иванов И.И.
2	Петров П.П.

Решение 2.2.4.a{93}.

MySQL Решение 2.2.4.a

```
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
       LEFT OUTER JOIN `subscriptions`
                    ON `s_id` = `sb_subscriber`
GROUP  BY `s_id`
HAVING COUNT(IF(`sb_is_active` = 'Y', `sb_is_active`, NULL)) = 0
```

MS SQL Решение 2.2.4.a

```
SELECT [s_id],
       [s_name]
FROM   [subscribers]
       LEFT OUTER JOIN [subscriptions]
                    ON [s_id] = [sb_subscriber]
GROUP  BY [s_id],
          [s_name]
HAVING COUNT(CASE
                WHEN [sb_is_active] = 'Y' THEN [sb_is_active]
                ELSE NULL
             END) = 0
```

Oracle Решение 2.2.4.a

```
1   SELECT "s_id",
2          "s_name"
3   FROM   "subscribers"
4          LEFT OUTER JOIN "subscriptions"
5                       ON "s_id" = "sb_subscriber"
6   GROUP  BY "s_id",
7             "s_name"
8   HAVING COUNT(CASE
9                  WHEN "sb_is_active" = 'Y' THEN "sb_is_active"
10                 ELSE NULL
11               END) = 0
```

Почему получается такое нетривиальное решение? Строки 1-5 запросов 2.2.4.a возвращают следующие данные (добавим поле `sb_is_active` для наглядности):

s_id	s_name	sb_is_active
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
2	Петров П.П.	NULL
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
4	Сидоров С.С.	N
4	Сидоров С.С.	Y
4	Сидоров С.С.	Y

Признаком того, что читатель вернул все книги, является отсутствие (`COUNT(...) = 0`) у него записей со значением `sb_is_active = 'Y'`. Проблема в том, что мы не можем «заставить» `COUNT` считать только значения `Y` — он будет учитывать любые значения, не равные `NULL`. Отсюда легко следует вывод, что нам осталось превратить в `NULL` любые значения поля `sb_is_active`, не равные `Y`, т.е. получить такой результат:

s_id	s_name	sb_is_active
1	Иванов И.И.	NULL
1	Иванов И.И.	NULL
1	Иванов И.И.	NULL
1	Иванов И.И.	NULL
1	Иванов И.И.	NULL
2	Петров П.П.	NULL
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
4	Сидоров С.С.	NULL
4	Сидоров С.С.	Y
4	Сидоров С.С.	Y

Именно за это действие отвечают выражения **IF** в строке 7 запроса для MySQL и **CASE** в строках 8-11 запросов для MS SQL Server и Oracle — любое значение поля **sb_is_active**, отличное от **Y**, они превращают в **NULL**.

Теперь остаётся проверить результат работы **COUNT**: если он равен нулю, у читателя на руках нет книг.

Типичной ошибкой при решении задачи 2.2.4.a является попытка получить нужный результат следующим запросом:

MySQL Решение 2.2.4.a (**ошибочный** запрос)

```
1  SELECT `s_id`,
2         `s_name`
3  FROM   `subscribers`
4         LEFT OUTER JOIN `subscriptions`
5                       ON `s_id` = `sb_subscriber`
6  WHERE  `sb_is_active` = 'Y'
7          OR `sb_is_active` IS NULL
8  GROUP  BY `s_id`
9  HAVING COUNT(`sb_is_active`) = 0
```

MS SQL Решение 2.2.4.a (**ошибочный** запрос)

```
1   SELECT [s_id],
2          [s_name]
3  FROM   [subscribers]
4         LEFT OUTER JOIN [subscriptions]
5                       ON [s_id] = [sb_subscriber]
6  WHERE  [sb_is_active] = 'Y'
7          OR [sb_is_active] IS NULL
8  GROUP  BY [s_id], [s_name], [sb_is_active]
9  HAVING COUNT([sb_is_active]) = 0
```

Oracle Решение 2.2.4.a (**ошибочный** запрос)

```
1  SELECT "s_id",
2         "s_name"
3  FROM   "subscribers"
4         LEFT OUTER JOIN "subscriptions"
5                       ON "s_id" = "sb_subscriber"
6  WHERE  "sb_is_active" = 'Y'
7          OR "sb_is_active" IS NULL
8  GROUP  BY "s_id", "s_name", "sb_is_active"
9  HAVING COUNT("sb_is_active") = 0
```

Здесь получается такой неверный набор данных:

s_id	s_name
2	Петров П.П.

Это решение учитывает никогда не бравших книги читателей (**sb_is_active IS NULL**), а также тех, у кого есть на руках хотя бы одна книга (**sb_is_active = 'Y'**), но те, кто вернул все книги, под это условие не подходят:

s_id	s_name	sb_is_active
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
1	Иванов И.И.	N
2	Петров П.П.	NULL
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
3	Сидоров С.С.	Y
4	Сидоров С.С.	N
4	Сидоров С.С.	Y
4	Сидоров С.С.	Y

Эти записи не удовлетворяют условию **WHERE** и будут пропущены.

Таким образом, Иванов И.И. оказывается пропущенным. Поэтому нужно использовать решение, представленное в начале рассмотрения задачи 2.2.4.a (т.е. решение с преобразованием значения поля **sb_is_active**).

Решение 2.2.4.b[93].

MySQL Решение 2.2.4.b

```
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` NOT IN (SELECT DISTINCT `sb_subscriber`
                      FROM   `subscriptions`
                      WHERE  `sb_is_active` = 'Y')
```

MS SQL Решение 2.2.4.b

```
SELECT [s_id],
       [s_name]
FROM   [subscribers]
WHERE  [s_id] NOT IN (SELECT DISTINCT [sb_subscriber]
                      FROM   [subscriptions]
                      WHERE  [sb_is_active] = 'Y')
```

Oracle Решение 2.2.4.b

```
SELECT "s_id",
       "s_name"
FROM   "subscribers"
WHERE  "s_id" NOT IN (SELECT DISTINCT "sb_subscriber"
                      FROM   "subscriptions"
                      WHERE  "sb_is_active" = 'Y')
```

Типичной ошибкой при решении задачи 2.2.4.b является попытка получить нужный результат следующим запросом:

MySQL Решение 2.2.4.b (**ошибочный** запрос)

```
SELECT `s_id`,
       `s_name`
FROM   `subscribers`
WHERE  `s_id` IN (SELECT DISTINCT `sb_subscriber`
                  FROM   `subscriptions`
                  WHERE  `sb_is_active` = 'N')
```

MS SQL Решение 2.2.4.b (**ошибочный** запрос)

```
SELECT [s_id],
       [s_name]
FROM   [subscribers]
WHERE  [s_id] IN (SELECT DISTINCT [sb_subscriber]
                  FROM   [subscriptions]
                  WHERE  [sb_is_active] = 'N')
```

Oracle Решение 2.2.4.b (**ошибочный** запрос)

```
SELECT "s_id",
       "s_name"
FROM   "subscribers"
WHERE  "s_id" IN (SELECT DISTINCT "sb_subscriber"
                  FROM   "subscriptions"
                  WHERE  "sb_is_active" = 'N')
```

Получается такой набор данных:

s_id	s_name
1	Иванов И.И.
4	Сидоров С.С.

Но это — решение задачи «показать читателей, которые хотя бы раз вернули книгу». Данная проблема (характерная для многих подобных ситуаций) лежит не столько в области правильности составления запросов, сколько в области понимания смысла (семантики) модели базы данных.

Если по некоторому факту выдачи книги отмечено, что она возвращена (в поле **sb_is_active** стоит значение **N**), это совершенно не означает, что рядом нет другого факта выдачи тому же читателю книги, которую он ещё не вернул. Рассмотрим это на примере читателя с идентификатором 4 («Сидоров С.С.»):

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
57	4	5	2012-06-11	2012-08-11	N
91	4	1	2015-10-07	2015-03-07	Y
99	4	4	2015-10-08	2025-11-08	Y

Он вернул одну книгу, и потому подзапрос вернёт его идентификатор, но ещё две книги он не вернул, и потому по условию исходной задачи он не должен оказаться в списке.

Также очевидно, что читатели, никогда не бравшие книг, не попадут в список людей, у которых на руках нет книг (идентификаторы таких читателей вообще ни разу не встречаются в таблице **subscriptions**).

Задание 2.2.4.TSK.A: показать список книг, ни один экземпляр которых сейчас не находится на руках у читателей.

2.2.5. ПРИМЕР 15: ДВОЙНОЕ ИСПОЛЬЗОВАНИЕ УСЛОВИЯ IN

Задача 2.2.5.a{100}: показать книги из жанров «Программирование» и/или «Классика» (без использования **JOIN**; идентификаторы жанров известны).

Задача 2.2.5.b{101}: показать книги из жанров «Программирование» и/или «Классика» (без использования **JOIN**; идентификаторы жанров неизвестны).

Задача 2.2.5.c{102}: показать книги из жанров «Программирование» и/или «Классика» (с использованием **JOIN**; идентификаторы жанров известны).

Задача 2.2.5.d{103}: показать книги из жанров «Программирование» и/или «Классика» (с использованием **JOIN**; идентификаторы жанров неизвестны).

Вариант такого задания, где вместо «и/или» стоит строгое «и», рассмотрен в примере 18{130}.

Ожидаемый результат 2.2.5.a.

b_id	b_name
1	Евгений Онегин
7	Искусство программирования
6	Курс теоретической физики
4	Психология программирования
2	Сказка о рыбаке и рыбке
5	Язык программирования C++

Ожидаемый результат 2.2.5.b.

b_id	b_name
1	Евгений Онегин
7	Искусство программирования
6	Курс теоретической физики
4	Психология программирования
2	Сказка о рыбаке и рыбке
5	Язык программирования C++

Ожидаемый результат 2.2.5.c.

b_id	b_name
1	Евгений Онегин
7	Искусство программирования
6	Курс теоретической физики
4	Психология программирования
2	Сказка о рыбаке и рыбке
5	Язык программирования C++

Ожидаемый результат 2.2.5.d.

b_id	b_name
1	Евгений Онегин
7	Искусство программирования
6	Курс теоретической физики
4	Психология программирования
2	Сказка о рыбаке и рыбке
5	Язык программирования C++

Решение 2.2.5.a[{99}].

Из таблицы **m2m_books_genres** можно узнать список идентификаторов книг, относящихся к нужным жанрам, а затем на основе этого списка идентификаторов выбрать из таблицы **books** названия книг.

MySQL Решение 2.2.5.a

```
SELECT `b_id`,
       `b_name`
FROM   `books`
WHERE  `b_id` IN (SELECT DISTINCT `b_id`
                  FROM   `m2m_books_genres`
                  WHERE  `g_id` IN ( 2, 5 ))
ORDER  BY `b_name` ASC
```

MS SQL Решение 2.2.5.a

```
SELECT [b_id],
       [b_name]
FROM   [books]
WHERE  [b_id] IN (SELECT DISTINCT [b_id]
                  FROM   [m2m_books_genres]
                  WHERE  [g_id] IN ( 2, 5 ))
ORDER  BY [b_name] ASC
```

Oracle Решение 2.2.5.a

```
1   SELECT "b_id",
2          "b_name"
3   FROM   "books"
4   WHERE  "b_id" IN (SELECT DISTINCT "b_id"
5                     FROM   "m2m_books_genres"
6                     WHERE  "g_id" IN ( 2, 5 ))
7   ORDER  BY "b_name" ASC
```

Подзапросы в строках 4-6 возвращают следующие значения:

b_id
4
5
7
1
2
6

Решение 2.2.5.b{99}.

Результат достигается в три шага, на первом из которых по имени жанра определяется его идентификатор с использованием таблицы **genres**, после чего второй и третий шаги эквивалентны решению 2.2.5.a{100}.

MySQL Решение 2.2.5.b

```
1   SELECT `b_id`,
2          `b_name`
3   FROM   `books`
4   WHERE  `b_id` IN (SELECT DISTINCT `b_id`
5                     FROM   `m2m_books_genres`
6                     WHERE  `g_id` IN (SELECT `g_id`
7                                       FROM   `genres`
8                                       WHERE
9                                          `g_name` IN ( 'Программирование',
10                                                       'Классика' )
11                                      ))
12  ORDER  BY `b_name` ASC
```

MS SQL Решение 2.2.5.b

```
1   SELECT [b_id],
2          [b_name]
3   FROM   [books]
4   WHERE  [b_id] IN (SELECT DISTINCT [b_id]
5                     FROM   [m2m_books_genres]
6                     WHERE  [g_id] IN (SELECT [g_id]
7                                       FROM   [genres]
8                                       WHERE
9                                         [g_name] IN ( N'Программирование',
10                                                      N'Классика' )
11                                      ))
12  ORDER  BY [b_name] ASC
```

Oracle Решение 2.2.5.b

```
1   SELECT "b_id",
2          "b_name"
3   FROM   "books"
4   WHERE  "b_id" IN (SELECT DISTINCT "b_id"
5                     FROM   "m2m_books_genres"
6                     WHERE  "g_id" IN (SELECT "g_id"
7                                       FROM   "genres"
8                                       WHERE
9                                         "g_name" IN ( N'Программирование',
10                                                      N'Классика' )
11                                      ))
12  ORDER  BY "b_name" ASC
```

Подзапросы 2.2.5.b в строках 6-11 возвращают следующие данные:

g_id
5
2

Подзапросы 2.2.5.b в строках 4-10 возвращают те же данные, что и подзапросы в строках 3-5 решения 2.2.5.a.

Решение 2.2.5.c[{99}].

Здесь по-прежнему удобно использовать **IN** для указания условия выборки, но второе использование **IN** по условию задачи необходимо заменить на **JOIN**.

MySQL Решение 2.2.5.c

```
1   SELECT DISTINCT `b_id`,
2                   `b_name`
3   FROM   `books`
4          JOIN `m2m_books_genres` USING ( `b_id` )
5   WHERE  `g_id` IN ( 2, 5 )
6   ORDER  BY `b_name` ASC
```

MS SQL Решение 2.2.5.c

```
1   SELECT DISTINCT [books].[b_id],
2                   [b_name]
3   FROM   [books]
4          JOIN [m2m_books_genres]
5            ON [books].[b_id] = [m2m_books_genres].[b_id]
6   WHERE  [g_id] IN ( 2, 5 )
7   ORDER  BY [b_name] ASC
```

Oracle Решение 2.2.5.c

```
1   SELECT DISTINCT "b_id",
2                   "b_name"
3   FROM   "books"
4          JOIN "m2m_books_genres" USING ( "b_id" )
5   WHERE  "g_id" IN ( 2, 5 )
6   ORDER  BY "b_name" ASC
```

Мы не можем обойтись без ключевого слова **DISTINCT**, т.к. в противном случае получим дублирование результатов для книг, относящихся одновременно к обоим требуемым жанрам:

b_id	b_name
1	Евгений Онегин
7	Искусство программирования
7	Искусство программирования
6	Курс теоретической физики
4	Психология программирования
2	Сказка о рыбаке и рыбке
5	Язык программирования C++

Но благодаря тому, что мы помещаем в выборку идентификатор книги, мы не рискуем «схлопнуть» с помощью **DISTINCT** несколько книг с одинаковыми названиями в одну строку.

Обратите внимание на синтаксические различия этого решения для разных СУБД: MySQL и Oracle не требуют указания на то, из какой таблицы выбирать значение поля **b_id** (строка 1 запросов 2.2.5.c для MySQL и Oracle), а также поддерживают сокращённую форму указания условия объединения (строка 4 запросов 2.2.5.c для MySQL и Oracle). MS SQL Server требует указывать таблицу, из которой будет происходить извлечение значения поля **b_id** (строка 1 запроса 2.2.5.c для MS SQL Server), а также не поддерживает сокращённую форму указания условия объединения (строка 5 запроса 2.2.5.c для MS SQL Server).

Решение 2.2.5.d[{99}]:

Поскольку первое использование **IN** по условию задачи пришлось заменить на **JOIN**, здесь осталось на один уровень **IN** меньше, чем в решении 2.2.5.b[{101}].

MySQL Решение 2.2.5.d

```
1   SELECT DISTINCT `b_id`,
2                   `b_name`
3   FROM    `books`
4           JOIN `m2m_books_genres` USING ( `b_id` )
5   WHERE   `g_id` IN (SELECT `g_id`
6                      FROM   `genres`
7                      WHERE  `g_name` IN ( N'Программирование',
8                                           N'Классика' )
9                     )
10  ORDER  BY `b_name` ASC
```

MS SQL Решение 2.2.5.d

```
1   SELECT DISTINCT [books].[b_id],
2                   [b_name]
3   FROM    [books]
4           JOIN [m2m_books_genres]
5             ON [books].[b_id] = [m2m_books_genres].[b_id]
6   WHERE   [g_id] IN (SELECT [g_id]
7                      FROM   [genres]
8                      WHERE  [g_name] IN ( N'Программирование',
9                                           N'Классика' )
10                    )
11  ORDER  BY [b_name] ASC
```

Oracle Решение 2.2.5.d

```
SELECT DISTINCT "b_id",
                "b_name"
FROM   "books"
       JOIN "m2m_books_genres" USING ( "b_id" )
WHERE  "g_id" IN (SELECT "g_id"
                  FROM   "genres"
                  WHERE  "g_name" IN ( N'Программирование',
                                       N'Классика' )
                 )
ORDER  BY "b_name" ASC
```

Логика решения 2.2.5.d является комбинацией 2.2.5.b (по самым глубоко вложенным **IN**) и 2.2.5.c (по использованию **JOIN**).

Задание 2.2.5.TSK.A: показать книги, написанные Пушкиным и/или Азимовым (индивидуально или в соавторстве — не важно).

Задание 2.2.5.TSK.B: показать книги, написанные Карнеги и Страуструпом **в соавторстве**.

2.2.6. ПРИМЕР 16: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ И ФУНКЦИЯ COUNT

Задача 2.2.6.a{105}: показать книги, у которых более одного автора.

Задача 2.2.6.b{106}: показать, сколько реально экземпляров каждой книги сейчас есть в библиотеке.

Ожидаемый результат 2.2.6.a.

b_id	b_name	authors_count
4	Психология программирования	2
6	Курс теоретической физики	2

Ожидаемый результат 2.2.6.b.

b_id	b_name	real_count
6	Курс теоретической физики	12
7	Искусство программирования	7
3	Основание и империя	4
2	Сказка о рыбаке и рыбке	3
5	Язык программирования C++	2
1	Евгений Онегин	0
4	Психология программирования	0

Решение 2.2.6.a[{104}].

MySQL Решение 2.2.6.a

```
SELECT `b_id`,
       `b_name`,
       COUNT(`a_id`) AS `authors_count`
FROM   `books`
       JOIN `m2m_books_authors` USING (`b_id`)
GROUP  BY `b_id`
HAVING `authors_count` > 1
```

MS SQL Решение 2.2.6.a

```
SELECT [books].[b_id],
       [books].[b_name],
       COUNT([m2m_books_authors].[a_id]) AS [authors_count]
FROM   [books]
       JOIN [m2m_books_authors]
         ON [books].[b_id] = [m2m_books_authors].[b_id]
GROUP  BY [books].[b_id],
          [books].[b_name]
HAVING COUNT([m2m_books_authors].[a_id]) > 1
```

Oracle Решение 2.2.6.a

```
SELECT "b_id",
       "b_name",
       COUNT("a_id") AS "authors_count"
FROM   "books"
       JOIN "m2m_books_authors" USING ("b_id")
GROUP  BY "b_id", "b_name"
HAVING COUNT("a_id") > 1
```

Обратите внимание, насколько решение для MySQL короче и проще решений для MS SQL Server и Oracle за счёт того, что в MySQL нет необходимости указывать имя таблицы для разрешения неоднозначности принадлежности поля **`b_id`**, а также нет необходимости группировать результат по полю **`b_name`**.

Решение 2.2.6.b[{104}].

Мы рассмотрим несколько похожих по сути, но принципиально разных по синтаксису решений, чтобы показать широту возможностей языка SQL и используемых СУБД, а также чтобы сравнить скорость работы различных решений.

В решении для MySQL варианты 2 и 3 представлены только затем, чтобы показать, как с помощью подзапросов можно эмулировать неподдерживаемые MySQL общие табличные выражения.

MySQL Решение 2.2.6.b

```
-- Вариант 1: использование коррелирующего подзапроса
SELECT DISTINCT `b_id`,
                `b_name`,
                ( `b_quantity` - (SELECT COUNT(`int`.`sb_book`)
                                  FROM   `subscriptions` AS `int`
                                  WHERE  `int`.`sb_book` = `ext`.`sb_book`
                                         AND `int`.`sb_is_active` = 'Y')
                                ) AS `real_count`
FROM   `books`
       LEFT OUTER JOIN `subscriptions` AS `ext`
                    ON `books`.`b_id` = `ext`.`sb_book`
ORDER  BY `real_count` DESC

-- Вариант 2: использование подзапроса как эмуляции общего табличного
-- выражения и коррелирующего подзапроса
SELECT `b_id`,
       `b_name`,
       ( `b_quantity` - IFNULL((SELECT `taken`
                                FROM   (SELECT `sb_book`         AS `b_id`,
                                               COUNT(`sb_book`) AS `taken`
                                        FROM   `subscriptions`
                                        WHERE  `sb_is_active` = 'Y'
                                        GROUP  BY `sb_book`
                                       ) AS `books_taken`
                                WHERE  `books`.`b_id` =
                                       `books_taken`.`b_id`), 0
                           ) ) AS
       `real_count`
FROM   `books`
ORDER  BY `real_count` DESC

-- Вариант 3: пошаговое применение нескольких подзапросов
SELECT `b_id`,
       `b_name`,
       ( `b_quantity` - (SELECT `taken`
                         FROM   (SELECT `b_id`,
                                        COUNT(`sb_book`) AS `taken`
                                 FROM   `books`
                                        LEFT OUTER JOIN
                                        (SELECT `sb_book`
                                         FROM   `subscriptions`
                                         WHERE `sb_is_active` = 'Y'
                                        ) AS `books_taken`
                                        ON `b_id` = `sb_book`
                                 GROUP  BY `b_id`) AS `real_taken`
                         WHERE  `books`.`b_id` = `real_taken`.`b_id`)
       ) AS `real_count`
FROM   `books`
ORDER  BY `real_count` DESC
```

MySQL Решение 2.2.6.b (продолжение)

```
1   -- Вариант 4: подзапрос используется как эмуляция общего
2   -- табличного выражения
3   SELECT `b_id`,
4          `b_name`,
5          ( `b_quantity` - IFNULL(`taken`, 0) ) AS `real_count`
6   FROM   `books`
7          LEFT OUTER JOIN (SELECT `sb_book`,
8                                  COUNT(`sb_book`) AS `taken`
9                           FROM   `subscriptions`
10                          WHERE  `sb_is_active` = 'Y'
11                          GROUP  BY `sb_book`) AS `books_taken`
12                    ON `b_id` = `sb_book`
13  ORDER  BY `real_count` DESC
```

Рассмотрим логику работы этих запросов.

Вариант 1 начинается с выполнения объединения. Если временно заменить подзапрос в строках 4-8 на константу, получается следующий код:

MySQL Решение 2.2.6.b (модифицированный запрос)

```
1   -- Вариант 1: использование коррелирующего подзапроса
2   SELECT DISTINCT `b_id`,
3                   `b_name`,
4                   'X' AS `real_count`
5   FROM    `books`
6          LEFT OUTER JOIN `subscriptions` AS `ext`
7                       ON `books`.`b_id` = `ext`.`sb_book`
8   ORDER  BY `real_count` DESC
```

В результате выполнение такого запроса получим:

b_id	b_name	real_count
1	Евгений Онегин	X
2	Сказка о рыбаке и рыбке	X
3	Основание и империя	X
4	Психология программирования	X
5	Язык программирования C++	X
6	Курс теоретической физики	X
7	Искусство программирования	X

Далее из значения поля **`b_count`** для каждой книги вычитается результат выполнения коррелирующего подзапроса (строки 4-7 в исходном запросе):

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 1: использование коррелирующего подзапроса
    -- ...
4   SELECT COUNT(`int`.`sb_book`)
5   FROM    `subscriptions` AS `int`
6   WHERE   `int`.`sb_book` = `ext`.`sb_book`
7          AND `int`.`sb_is_active` = 'Y'
    -- ...
```

Если в таком запросе подставить вместо выражения **`` `ext`.`sb_book` ``** значение идентификатора, то этот фрагмент кода можно будет выполнить как самостоятельный запрос и получить конкретное число (например, для книги с идентификатором 1 это будет число 2, т.е. два экземпляра книги в настоящий момент находится на руках у читателей.

И, наконец, если мы чуть-чуть доработаем исходный запрос так, чтобы видеть все данные по каждой книге (её количество в библиотеке, количество выданных читателям экземпляров, количество оставшихся в библиотеке экземпляров), мы получим следующий неоптимальный, но наглядный запрос (подзапросы в строках 5-8 и 10-13 приходится дублировать, т.к. MySQL не может сразу вычислить количество выданных читателям книг и тут же использовать это значение в выражении, вычисляющем остаток книг в библиотеке):

MySQL Решение 2.2.6.b (переработанный для наглядности запрос)

```
1   -- Вариант 1: использование коррелирующего подзапроса
2   SELECT DISTINCT `b_id`,
3                   `b_name`,
4                   `b_quantity`,
5                   (SELECT COUNT(`int`.`sb_book`)
6                    FROM   `subscriptions` AS `int`
7                    WHERE  `int`.`sb_book` = `ext`.`sb_book`
8                           AND `int`.`sb_is_active` = 'Y')
9                    AS `taken`,
10                  ( `b_quantity` - (SELECT COUNT(`int`.`sb_book`)
11                                    FROM   `subscriptions` AS `int`
12                                    WHERE  `int`.`sb_book` = `ext`.`sb_book`
13                                           AND `int`.`sb_is_active` = 'Y') )
14                   AS `real_count`
15  FROM   `books`
16         LEFT OUTER JOIN `subscriptions` AS `ext`
17                      ON `books`.`b_id` = `ext`.`sb_book`
18  ORDER  BY `real_count` DESC
```

В результате выполнения такого запроса получается:

b_id	b_name	b_quantity	taken	real_count
6	Курс теоретической физики	12	0	12
7	Искусство программирования	7	0	7
3	Основание и империя	5	1	4
2	Сказка о рыбаке и рыбке	3	0	3
5	Язык программирования C++	3	1	2
1	Евгений Онегин	2	2	0
4	Психология программирования	1	1	0

Вариант 2 построен на идее эмуляции общих табличных выражений, которые не поддерживаются MySQL. Как показывают дальнейшие исследования скорости выполнения запросов, это окажется очень плохой идеей, но в качестве эксперимента рассмотрим логику работы такого запроса.

Работа начинается с самого глубоко вложенного подзапроса (строки 6-10 в исходном запросе):

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 2: использование подзапроса как эмуляции общего табличного
    -- выражения и коррелирующего подзапроса
    -- ...
6   SELECT `sb_book`         AS `b_id`,
7          COUNT(`sb_book`) AS `taken`
8   FROM   `subscriptions`
9   WHERE  `sb_is_active` = 'Y'
10  GROUP  BY `sb_book`
    -- ...
```

Результат выполнения данного подзапроса таков (по книгам, ни один экземпляр которых сейчас не находится на руках у читателей, данных здесь нет):

b_id	taken
1	2
3	1
4	1
5	1

С полученными данными работает коррелирующий подзапрос (строки 5-15), в котором функция **IFNULL** позволяет вместо **NULL** вернуть значение **0** для книг, ни один экземпляр которых не был выдан читателям. Для книг, хотя бы один экземпляр которых выдан читателям, возвращается готовое (отличное от нуля) значение. Полученное из подзапроса значение вычитается из значения поля **b_quantity**, и таким образом мы получаем финальный результат.

Вариант 3 построен на идее последовательного выполнения нескольких подзапросов. Первым срабатывает наиболее глубоко вложенный подзапрос, возвращающий идентификаторы книг, находящихся на руках у читателей (строки 9-12 исходного запроса):

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 3: пошаговое применение нескольких подзапросов
    -- ...
9   (SELECT `sb_book`
10   FROM   `subscriptions`
11   WHERE `sb_is_active` = 'Y'
12 ) AS `books_taken`
    -- ...
```

Результат выполнения этого подзапроса таков:

sb_book
3
5
1
1
4

Далее выполняется подзапрос, определяющий находящееся на руках у читателей количество экземпляров каждой книги (строки 5-14 исходного запроса):

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 3: пошаговое применение нескольких подзапросов
    -- ...
5   SELECT          `b_id`,
6                   count(`sb_book`) AS `taken`
7   FROM            `books`
8   LEFT OUTER JOIN
9                   (
10                         SELECT `sb_book`
11                         FROM   `subscriptions`
12                         WHERE  `sb_is_active` = 'Y' ) AS `books_taken`
13  ON              `b_id` = `sb_book`
14  GROUP BY        `b_id`) AS `real_taken`
    -- ...
```

Результат выполнения этого подзапроса таков:

b_id	taken
1	2
2	0
3	1
4	1
5	1
6	0
7	0

Полученные данные используются в коррелирующем подзапросе (строки 4-16 исходного запроса) для определения итогового результата (количества экземпляров книг в библиотеке):

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 3: пошаговое применение нескольких подзапросов
    -- ...
4           ( `b_quantity` - (SELECT `taken`
5                             FROM   (SELECT `b_id`,
6                                            COUNT(`sb_book`) AS `taken`
7                                     FROM   `books`
8                                            LEFT OUTER JOIN
9                                            (SELECT `sb_book`
10                                            FROM   `subscriptions`
11                                            WHERE `sb_is_active` = 'Y'
12                                           ) AS `books_taken`
13                                           ON `b_id` = `sb_book`
14                                    GROUP  BY `b_id`) AS `real_taken`
15                            WHERE  `books`.`b_id` = `real_taken`.`b_id`)
16          ) AS `real_count`
    -- ...
```

Выполнение этого подзапроса позволяет получить конечное требуемое значение, которое появляется в результатах основного запроса.

Вариант 4 похож на вариант 2 в том, что здесь тоже реализуется эмуляция неподдерживаемых MySQL общих табличных выражений через подзапрос, но есть и существенное отличие от варианта 2 — здесь нет коррелирующих подзапросов.

Эмулирующий общее табличное выражение подзапрос (строки 7-11 оригинального запроса) подготавливает все необходимые данные о том, какое количество экземпляров каждой книги находится на руках у читателей:

MySQL Решение 2.2.6.b (фрагмент кода с подзапросом)

```
    -- Вариант 4: подзапрос используется как эмуляция
    -- общего табличного выражения
    -- ...
7   SELECT `sb_book`,
8          COUNT(`sb_book`) AS `taken`
9   FROM   `subscriptions`
10  WHERE  `sb_is_active` = 'Y'
11  GROUP  BY `sb_book`
    -- ...
```

Результат выполнения этого подзапроса таков:

sb_book	taken
1	2
3	1
4	1
5	1

Этот результат используется в операторе объединения **JOIN** (строки 7-12 оригинального запроса), а функция **IFNULL** в 5-й строке оригинального запроса позволяет получить числовое представление количества выданных на руки читателям книг в случае, если ни один экземпляр не выдан. Для наглядной демонстрации перепишем 4-й вариант, добавив в выборку поля **b_quantity** и исходное значение **taken**, полученное в результате объединения с данными подзапроса:

MySQL Решение 2.2.6.b (модифицированный запрос)

```
1  -- Вариант 4: подзапрос используется как эмуляция
2  -- общего табличного выражения
3  SELECT `b_id`,
4         `b_name`,
5         `b_quantity`,
6         `taken`,
7         ( `b_quantity` - IFNULL(`taken`, 0) ) AS `real_count`
8  FROM   `books`
9         LEFT OUTER JOIN (SELECT `sb_book`,
10                                COUNT(`sb_book`) AS `taken`
11                         FROM   `subscriptions`
12                         WHERE  `sb_is_active` = 'Y'
13                         GROUP  BY `sb_book`) AS `books_taken`
14                      ON `b_id` = `sb_book`
15 ORDER  BY `real_count` DESC
```

В результате выполнения такого модифицированного запроса получается:

b_id	b_name	b_quantity	taken	real_count
6	Курс теоретической физики	12	NULL	12
7	Искусство программирования	7	NULL	7
3	Основание и империя	5	1	4
2	Сказка о рыбаке и рыбке	3	NULL	3
5	Язык программирования C++	3	1	2
1	Евгений Онегин	2	2	0
4	Психология программирования	1	1	0

В оригинальном запросе **NULL**-значения поля **taken** преобразуются в 0, затем из значения поля **b_quantity** вычитается значение поля **taken** и таким образом получаются конечные значения поля **real_taken**.

Рассмотрим решение задачи 2.2.6.b для MS SQL Server и Oracle. Вариант 1 этих решений полностью идентичен варианту 1 решения для MySQL, а варианты 2-4 полностью идентичны для MS SQL Server и Oracle, потому мы рассмотрим их только один раз — на примере MS SQL Server.

MS SQL Решение 2.2.6.b

```
-- Вариант 1: использование коррелирующего подзапроса
SELECT DISTINCT [b_id],
                [b_name],
                ( [b_quantity] - (SELECT COUNT([int].[sb_book])
                                  FROM   [subscriptions] AS [int]
                                  WHERE  [int].[sb_book] = [ext].[sb_book]
                                         AND [int].[sb_is_active] = 'Y') )
AS
                [real_count]
FROM   [books]
       LEFT OUTER JOIN [subscriptions] AS [ext]
                    ON [books].[b_id] = [ext].[sb_book]
ORDER  BY [real_count] DESC
```

Описание логики работы **варианта 1** см. выше — этот запрос идентичен во всех трёх СУБД.

MS SQL Решение 2.2.6.b

```
-- Вариант 2: использование общего табличного выражения
-- и коррелирующего подзапроса
WITH [books_taken]
     AS (SELECT [sb_book]        AS [b_id],
                COUNT([sb_book]) AS [taken]
         FROM   [subscriptions]
         WHERE  [sb_is_active] = 'Y'
         GROUP  BY [sb_book])
SELECT [b_id],
       [b_name],
       ( [b_quantity] - ISNULL((SELECT [taken]
                                FROM   [books_taken]
                                WHERE  [books].[b_id] =
                                       [books_taken].[b_id]), 0
                        ) ) AS
       [real_count]
FROM   [books]
ORDER  BY [real_count] DESC

-- Вариант 3: пошаговое применение общего табличного выражения и подзапроса
WITH [books_taken]
     AS (SELECT [sb_book]
         FROM   [subscriptions]
         WHERE  [sb_is_active] = 'Y'),
     [real_taken]
     AS (SELECT [b_id],
                COUNT([sb_book]) AS [taken]
         FROM   [books]
                LEFT OUTER JOIN [books_taken]
                             ON [b_id] = [sb_book]
         GROUP  BY [b_id])
SELECT [b_id],
       [b_name],
       ( [b_quantity] - (SELECT [taken]
                         FROM   [real_taken]
                         WHERE  [books].[b_id] = [real_taken].[b_id]) ) AS
       [real_count]
FROM   [books]
ORDER  BY [real_count] DESC
```

MS SQL Решение 2.2.6.b (продолжение)

```
1   -- Вариант 4: без подзапросов
2   WITH [books_taken]
3        AS (SELECT [sb_book],
4                   COUNT([sb_book]) AS [taken]
5              FROM   [subscriptions]
6              WHERE  [sb_is_active] = 'Y'
7              GROUP  BY [sb_book])
8   SELECT [b_id],
9          [b_name],
10         ( [b_quantity] - ISNULL([taken], 0) ) AS [real_count]
11  FROM   [books]
12         LEFT OUTER JOIN [books_taken]
13                      ON [b_id] = [sb_book]
14  ORDER  BY [real_count] DESC
```

Вариант 2 основан на том, что общее табличное выражение в строках 3-8 подготавливает информацию о количестве экземпляров книг, находящихся на руках у читателей. Выполним отдельно соответствующий фрагмент запроса:

MS SQL Решение 2.2.6.b (модифицированный фрагмент запроса)

```
1   -- Вариант 2: использование общего табличного выражения
2   -- и коррелирующего подзапроса
3   WITH [books_taken]
4        AS (SELECT [sb_book]         AS [b_id],
5                   COUNT([sb_book]) AS [taken]
6              FROM   [subscriptions]
7              WHERE  [sb_is_active] = 'Y'
8              GROUP  BY [sb_book])
9   SELECT * FROM [books_taken]
```

Результат выполнения этого фрагмента запроса таков:

b_id	taken
1	2
3	1
4	1
5	1

Далее в строках 11-16 исходного запроса выполняется коррелирующий подзапрос, возвращающий для каждой книги количество выданных на руки читателям экземпляров или **NULL**, если ни один экземпляр не выдан. Чтобы иметь возможность корректно использовать такой результат в арифметическом выражении, в строке 11 исходного запроса мы используем функцию **ISNULL**, преобразующую **NULL**-значения в 0.

Вариант 3, основанный на пошаговом применении двух общих табличных выражений, подготавливает для коррелирующего подзапроса полностью готовый набор данных.

Первое общее табличное выражение (строки 2-5) возвращает следующие данные:

sb_book
3
5
1
1
4

Второе общее табличное выражение (строки 6-12) возвращает следующие данные:

b_id	taken
1	2
2	0
3	1
4	1
5	1
6	0
7	0

На основе полученных данных коррелирующий подзапрос в строках 15-18 вычисляет реальное количество экземпляров книг в библиотеке. Поскольку из второго общего табличного выражения данные поступают с «готовыми нулями» для книг, ни один экземпляр которых не выдан читателям, здесь нет необходимости использовать функцию **ISNULL**.

Вариант 4 основан на предварительной подготовке в общем табличном выражении информации о том, сколько книг выдано читателям, с последующим вычитанием этого количества из количества зарегистрированных в библиотеке книг. Общее табличное выражение возвращает следующие данные:

sb_book	taken
1	2
3	1
4	1
5	1

Поскольку при группировке для книг, ни один экземпляр которых не выдан читателям, значение **taken** будет равно **NULL**, мы применяем в 10-й строке запроса функцию **ISNULL**, преобразующую значения **NULL** в 0.

Рассмотрим решение задачи 2.2.6.b для Oracle. Единственное заметное отличие этого решения от решения для MS SQL Server заключается в том, что в Oracle используется функция **NVL** для получения поведения, аналогичного функции **ISNULL** в MS SQL Server (подстановка значения 0 вместо **NULL**).

Oracle Решение 2.2.6.b

```
1   -- Вариант 1: использование коррелирующего подзапроса
2   SELECT DISTINCT "b_id",
3                   "b_name",
4                   ( "b_quantity" - (SELECT COUNT("int"."sb_book")
5                                     FROM   "subscriptions" "int"
6                                     WHERE  "int"."sb_book" = "ext"."sb_book"
7                                            AND "int"."sb_is_active" = 'Y') )
8   AS
9                   "real_count"
10  FROM   "books"
11         LEFT OUTER JOIN "subscriptions" "ext"
12                      ON "books"."b_id" = "ext"."sb_book"
13  ORDER  BY "real_count" DESC
```

Oracle Решение 2.2.6.b

```
-- Вариант 2: использование общего табличного выражения
-- и коррелирующего подзапроса
WITH "books_taken"
     AS (SELECT "sb_book"        AS "b_id",
                COUNT("sb_book") AS "taken"
         FROM   "subscriptions"
         WHERE  "sb_is_active" = 'Y'
         GROUP  BY "sb_book")
SELECT "b_id",
       "b_name",
       ( "b_quantity" - NVL((SELECT "taken"
                             FROM   "books_taken"
                             WHERE  "books"."b_id" =
                                    "books_taken"."b_id"), 0
                          ) ) AS
       "real_count"
FROM   "books"
ORDER  BY "real_count" DESC
```

```
-- Вариант 3: пошаговое применение общего табличного выражения и подзапроса
WITH "books_taken"
     AS (SELECT "sb_book"
         FROM   "subscriptions"
         WHERE  "sb_is_active" = 'Y'),
     "real_taken"
     AS (SELECT "b_id",
                COUNT("sb_book") AS "taken"
         FROM   "books"
                LEFT OUTER JOIN "books_taken"
                             ON "b_id" = "sb_book"
         GROUP  BY "b_id")
SELECT "b_id",
       "b_name",
       ( "b_quantity" - (SELECT "taken"
                         FROM   "real_taken"
                         WHERE  "books"."b_id" = "real_taken"."b_id") ) AS
       "real_count"
FROM   "books"
ORDER  BY "real_count" DESC
```

```
-- Вариант 4: без подзапросов
WITH "books_taken"
     AS (SELECT "sb_book",
                COUNT("sb_book") AS "taken"
         FROM   "subscriptions"
         WHERE  "sb_is_active" = 'Y'
         GROUP  BY "sb_book")
SELECT "b_id",
       "b_name",
       ( "b_quantity" - NVL("taken", 0) ) AS "real_count"
FROM   "books"
       LEFT OUTER JOIN "books_taken"
                    ON "b_id" = "sb_book"
ORDER  BY "real_count" DESC
```

Исследование 2.2.6.EXP.A: сравним скорость работы каждого из четырёх вариантов запросов 2.2.6.b для всех трёх СУБД на базе данных «Большая библиотека».

Выполнив по сто раз каждый запрос для каждой СУБД, получаем такие медианы времени:

	MySQL	MS SQL Server	Oracle
Вариант 1	0.545	50.575	1.393
Вариант 2	16240.234	5.814	0.430
Вариант 3	220.918	5.733	0.342
Вариант 4	19061.824	5.309	0.383

Обратите внимание, насколько по-разному ведут себя различные СУБД. Для MS SQL Server и Oracle ожидаемо вариант с коррелирующим подзапросом (вариант 1) оказался самым медленным, но в MySQL он оказался намного быстрее, чем «эмуляция общего табличного выражения».

Задание 2.2.6.TSK.A: показать авторов, написавших более одной книги.

Задание 2.2.6.TSK.B: показать книги, относящиеся к более чем одному жанру.

Задание 2.2.6.TSK.C: показать читателей, у которых сейчас на руках больше одной книги.

Задание 2.2.6.TSK.D: показать, сколько экземпляров каждой книги сейчас выдано читателям.

Задание 2.2.6.TSK.E: показать всех авторов и количество экземпляров книг по каждому автору.

Задание 2.2.6.TSK.F: показать всех авторов и количество книг (не экземпляров книг, а «книг как изданий») по каждому автору.

Задание 2.2.6.TSK.G: показать всех читателей, не вернувших книги, и количество невозвращённых книг по каждому такому читателю.

2.2.7. ПРИМЕР 17: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ, ФУНКЦИЯ COUNT И АГРЕГИРУЮЩИЕ ФУНКЦИИ

Задача 2.2.7.a[118]: показать читаемость авторов, т.е. всех авторов и то количество раз, которое книги этих авторов были взяты читателями.

Задача 2.2.7.b[119]: показать самого читаемого автора, т.е. автора (или авторов, если их несколько), книги которого читатели брали чаще всего.

Задача 2.2.7.c{123}: показать среднюю читаемость авторов, т.е. среднее значение от того, сколько раз читатели брали книги каждого автора.

Задача 2.2.7.d{123}: показать медиану читаемости авторов, т.е. медианное значение от того, сколько раз читатели брали книги каждого автора.

Задача 2.2.7.e{127}: написать запрос, проверяющий, не была ли допущена ошибка в заполнении документов, при которой оказывается, что на руках сейчас большее количество экземпляров некоторой книги, чем их было в библиотеке. Вернуть 1, если ошибка есть и 0, если ошибки нет.

Ожидаемый результат 2.2.7.a.

a_id	a_name	books
7	А.С. Пушкин	4
6	Б. Страуструп	4
3	Д. Карнеги	2
2	А. Азимов	2
1	Д. Кнут	1
5	Е.М. Лифшиц	0
4	Л.Д. Ландау	0

Ожидаемый результат 2.2.7.b.

a_id	a_name	books
6	Б. Страуструп	4
7	А.С. Пушкин	4

Ожидаемый результат 2.2.7.c: количество знаков после запятой по умолчанию различается в MySQL, MS SQL Server и Oracle.

MySQL

avg_reading
1.8571

MS SQL Server

avg_reading
1.85714285714286

Oracle

avg_reading
1.8571428571428571428571428571428571428571

Ожидаемый результат 2.2.7.d: количество знаков после запятой по умолчанию различается в MySQL, MS SQL Server и Oracle.

MySQL

med_reading
2.0000

MS SQL Server

med_reading
2

Oracle

avg_reading
2

Ожидаемый результат 2.2.7.e.

error_exists
0

Решение 2.2.7.a[{116}].

MySQL Решение 2.2.7.a

```
SELECT `a_id`,
       `a_name`,
       COUNT(`sb_book`) AS `books`
FROM   `authors`
       JOIN `m2m_books_authors` USING ( `a_id` )
       LEFT OUTER JOIN `subscriptions`
                    ON `m2m_books_authors`.`b_id` = `sb_book`
GROUP  BY `a_id`
ORDER  BY `books` DESC
```

MS SQL Решение 2.2.7.a

```
SELECT [authors].[a_id],
       [authors].[a_name],
       COUNT([sb_book]) AS [books]
FROM   [authors]
       JOIN [m2m_books_authors]
         ON [authors].[a_id] = [m2m_books_authors].[a_id]
       LEFT OUTER JOIN [subscriptions]
                    ON [m2m_books_authors].[b_id] = [sb_book]
GROUP  BY [authors].[a_id],
          [authors].[a_name]
ORDER  BY COUNT([sb_book]) DESC
```

Oracle Решение 2.2.7.a

```
SELECT "a_id",
       "a_name",
       COUNT("sb_book") AS "books"
FROM   "authors"
       JOIN "m2m_books_authors" USING ( "a_id" )
       LEFT OUTER JOIN "subscriptions"
                    ON "m2m_books_authors"."b_id" = "sb_book"
GROUP  BY "a_id",
          "a_name"
ORDER  BY "books" DESC
```

Решение для всех трёх СУБД отличается только нюансами синтаксиса, а по сути тривиально: нужно собрать воедино информацию об авторах, книгах и фактах выдачи книг (это достигается за счёт двух **JOIN**), после чего подсчитать количество фактов выдачи книг, сгруппировав результаты подсчёта по идентификаторам авторов.

Решение 2.2.7.b{116}.

MySQL Решение 2.2.7.a

```
1   -- Вариант 1: на основе функции MAX
2   SELECT `a_id`,
3          `a_name`,
4          COUNT(`sb_book`) AS `books`
5   FROM   `authors`
6          JOIN `m2m_books_authors` USING (`a_id`)
7          LEFT OUTER JOIN `subscriptions`
8                       ON `m2m_books_authors`.`b_id` = `sb_book`
9   GROUP  BY `a_id`
10  HAVING `books` = (SELECT MAX(`books`)
11                    FROM
12                    (SELECT COUNT(`sb_book`) AS `books`
13                     FROM   `authors`
14                            JOIN `m2m_books_authors` USING ( `a_id` )
15                            LEFT OUTER JOIN `subscriptions`
16                            ON `m2m_books_authors`.`b_id` = `sb_book`
17                            GROUP  BY `a_id`
18                    ) AS `books_per_author`)
```

Поскольку MySQL не поддерживает ни ранжирующие (оконные) функции, ни специфичный для MS SQL Server синтаксис **`TOP ... WITH TIES`**, здесь остаётся единственный вариант: полностью аналогично решению задачи 2.2.7.a{118} подсчитать, сколько раз читатели брали книги каждого из авторов (строки 2-9 запроса), а затем оставить в выборке только тех авторов, для которых это количество совпадает с максимальным значением по всем авторам (этот максимум вычисляется в строках 10-18 запроса).

Если бы MySQL поддерживал общие табличные выражения, можно было бы обойтись без повторного определения количества выдач книг по каждому автору (подзапрос в строках 12-17 отличается от основной части запроса в строках 2-9 только исключением из выборки полей **`a_id`** и **`a_name`** — в остальном это полное дублирование кода).

Модификация этого варианта решения с использованием общих табличных выражений представлена ниже для MS SQL Server и Oracle.

MS SQL Решение 2.2.7.b

```
1   -- Вариант 1: на основе функции MAX
2   WITH [prepared_data]
3        AS (SELECT [authors].[a_id],
4                   [authors].[a_name],
5                   COUNT([sb_book]) AS [books]
6            FROM   [authors]
7                   JOIN [m2m_books_authors]
8                     ON [authors].[a_id] = [m2m_books_authors].[a_id]
9                   LEFT OUTER JOIN [subscriptions]
10                                  ON [m2m_books_authors].[b_id] = [sb_book]
11           GROUP  BY [authors].[a_id],
12                      [authors].[a_name])
13  SELECT [a_id],
14         [a_name],
15         [books]
16  FROM   [prepared_data]
17  WHERE  [books] = (SELECT MAX([books])
18                    FROM   [prepared_data])
```

MS SQL Решение 2.2.7.b (продолжение)

```
1   -- Вариант 2: на основе ранжирования
2   WITH [prepared_data]
3        AS (SELECT [authors].[a_id],
4                   [authors].[a_name],
5                   COUNT([sb_book])                    AS [books],
6                   RANK()
7                     OVER (
8                       ORDER BY COUNT([sb_book]) DESC) AS [rank]
9             FROM    [authors]
10                  JOIN [m2m_books_authors]
11                    ON [authors].[a_id] = [m2m_books_authors].[a_id]
12                  LEFT OUTER JOIN [subscriptions]
13                               ON [m2m_books_authors].[b_id] = [sb_book]
14            GROUP  BY [authors].[a_id],
15                      [authors].[a_name])
16  SELECT [a_id],
17         [a_name],
18         [books]
19  FROM   [prepared_data]
20  WHERE  [rank] = 1

1   -- Вариант 3: на основе конструкции TOP ... WITH TIES
2   WITH [prepared_data]
3        AS (SELECT [authors].[a_id],
4                   [authors].[a_name],
5                   COUNT([sb_book]) AS [books]
6             FROM    [authors]
7                   JOIN [m2m_books_authors]
8                     ON [authors].[a_id] = [m2m_books_authors].[a_id]
9                   LEFT OUTER JOIN [subscriptions]
10                               ON [m2m_books_authors].[b_id] = [sb_book]
11            GROUP  BY [authors].[a_id],
12                      [authors].[a_name])
13  SELECT TOP 1 WITH TIES [a_id],
14                         [a_name],
15                         [books]
16  FROM   [prepared_data]
17  ORDER  BY [books] DESC
```

Вариант 1 решения для MS SQL Server отличается от варианта 1 для MySQL тем, что благодаря возможности использования общих табличных выражений мы можем избежать двукратного определения количества выдач читателям книг каждого автора: эта информация один раз определяется в общем табличном выражении (строки 2-12), и на этих же данных производится поиск максимального значения выдач книг (строки 17-18).

Вариант 2 основан на ранжировании авторов по количеству выдач их книг читателям с последующим отбором авторов, занявших первое место. Общее табличное выражение в строках 2-15 возвращает следующие данные:

a_id	a_name	books	rank
6	Б. Страуструп	4	1
7	А.С. Пушкин	4	1
2	А. Азимов	2	3
3	Д. Карнеги	2	3
1	Д. Кнут	1	5
4	Л.Д. Ландау	0	6
5	Е.М. Лифшиц	0	6

В строке 20 на этот набор налагается ограничение, помещающее в выборку только авторов со значением `rank` равным 1.

Вариант 3 основан на использовании специфичного для MS SQL Server синтаксиса `TOP ... WITH TIES` для выбора ограниченного числа первых записей с присоединением к этому набору ещё нескольких записей, совпадающих с выбранными по значению поля сортировки.

Сначала мы получаем такой набор данных:

a_id	a_name	Books
6	Б. Страуструп	4
7	А.С. Пушкин	4
2	А. Азимов	2
3	Д. Карнеги	2
1	Д. Кнут	1
4	Л.Д. Ландау	0
5	Е.М. Лифшиц	0

Затем, благодаря конструкции `SELECT TOP 1 WITH TIES ... FROM [prepared_data] ORDER BY [books] DESC` MS SQL Server оставляет только первую запись (`TOP 1`) и присоединяет к ней (`WITH TIES`) все другие записи с тем же самым значением в поле `books`, т.к. по нём идёт сортировка (`ORDER BY [books]`). В нашем случае это значение — 4. Так получается финальный результат выборки:

a_id	a_name	books
6	Б. Страуструп	4
7	А.С. Пушкин	4

Представленное ниже решение задачи 2.2.7.b для Oracle полностью эквивалентно решению для MS SQL Server за исключением отсутствия третьего варианта, т.к. Oracle не поддерживает конструкцию `TOP ... WITH TIES`.

Oracle Решение 2.2.7.b

```
1   -- Вариант 1: на основе функции MAX
2   WITH "prepared_data"
3        AS (SELECT "a_id",
4                   "a_name",
5                   COUNT("sb_book") AS "books"
6              FROM   "authors"
7                   JOIN "m2m_books_authors" USING("a_id")
8                   LEFT OUTER JOIN "subscriptions"
9                                ON "m2m_books_authors"."b_id" = "sb_book"
10             GROUP  BY "a_id",
11                       "a_name")
12  SELECT "a_id",
13         "a_name",
14         "books"
15  FROM   "prepared_data"
16  WHERE  "books" = (SELECT MAX("books")
17                    FROM   "prepared_data")

1   -- Вариант 2: на основе ранжирования
2   WITH "prepared_data"
3        AS (SELECT "a_id",
4                   "a_name",
5                   COUNT("sb_book")                    AS "books",
6                   RANK()
7                     OVER (
8                       ORDER BY COUNT("sb_book") DESC) AS "rank"
9              FROM   "authors"
10                  JOIN "m2m_books_authors" USING("a_id")
11                  LEFT OUTER JOIN "subscriptions"
12                               ON "m2m_books_authors"."b_id" = "sb_book"
13             GROUP  BY "a_id",
14                       "a_name")
15  SELECT "a_id",
16         "a_name",
17         "books"
18  FROM   "prepared_data"
19  WHERE  "rank" = 1
```

Исследование 2.2.6.EXP.A: сравним скорость работы каждого из четырёх вариантов запросов 2.2.6.b для всех трёх СУБД на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений каждого запроса:

	MySQL	MS SQL Server	Oracle
Вариант 1 (`MAX()`)	4787.841	16.106	155.918
Вариант 2 (`RANK()`)	-	8.138	1.407
Вариант 3 (`TOP ...`)	-	7.312	-

Здесь мы наблюдаем ситуацию, обратную полученной в исследовании 2.1.8.EXP.B{49}, где вариант с функций `MAX` оказался быстрее варианта с `RANK`. И это совершенно нормально, т.к. эксперименты проводились на разных наборах данных и в контексте разных запросов. Т.е. на скорость выполнения запроса влияет далеко не только лишь использование той или иной функции, но и множество других параметров.

Решение 2.2.7.c[117].

MySQL Решение 2.2.7.c

```
SELECT AVG(`books`) AS `avg_reading`
FROM   (SELECT COUNT(`sb_book`) AS `books`
        FROM   `authors`
               JOIN `m2m_books_authors` USING (`a_id`)
               LEFT OUTER JOIN `subscriptions`
                            ON `m2m_books_authors`.`b_id` = `sb_book`
        GROUP  BY `a_id`) AS `prepared_data`
```

MS SQL Решение 2.2.7.c

```
SELECT AVG(CAST([books] AS FLOAT)) AS [avg_reading]
FROM   (SELECT COUNT([sb_book]) AS [books]
        FROM   [authors]
               JOIN [m2m_books_authors]
                 ON [authors].[a_id] = [m2m_books_authors].[a_id]
               LEFT OUTER JOIN [subscriptions]
                            ON [m2m_books_authors].[b_id] = [sb_book]
        GROUP  BY [authors].[a_id]) AS [prepared_data]
```

Oracle Решение 2.2.7.c

```
SELECT AVG("books") AS "avg_reading"
FROM   (SELECT COUNT("sb_book") AS "books"
        FROM   "authors"
               JOIN "m2m_books_authors" USING ("a_id")
               LEFT OUTER JOIN "subscriptions"
                            ON "m2m_books_authors"."b_id" = "sb_book"
        GROUP  BY "a_id") "prepared_data"
```

Решение этой задачи для MS SQL Server и Oracle может быть представлено в виде общего табличного выражения, но из соображений совместимости оставлено в том же виде, что и решение для MySQL. Подзапрос в секции **FROM** играет роль источника данных и производит подсчёт количества выдач книг по каждому автору. Затем в основной секции запроса (строка 1 для всех трёх СУБД) из подготовленного набора извлекается искомое среднее значение.

Решение 2.2.7.d[117].

Обратите внимание, насколько просто решается эта задача в Oracle (который поддерживает функцию **MEDIAN**), и насколько нетривиальны решения для MySQL и MS SQL Server.

Логика решения для MySQL и MS SQL Server построена на математическом определении медианного значения: набор данных сортируется, после чего для наборов с нечётным количеством элементов медианой является значение центрального элемента, а для наборов с чётным значением элементов медианой является среднее значение двух центральных элементов. Поясним на примере.

Пусть у нас есть следующий набор данных с нечётным количеством элементов:

Номер элемента	Значение элемента
1	40
2	65
3	90

← медиана = 65

Центральным элементом является элемент с номером 2, и его значение 65 является медианой для данного набора.

Если у нас есть набор данных с чётным количеством элементов:

Номер элемента	Значение элемента
1	40
2	65
3	90
4	95

← медиана = (65 + 90) / 2 = 77.5

Центральными элементами являются элементы с номерами 2 и 3, и среднее арифметическое их значений (65 + 90) / 2 является медианой для данного набора.

Таким образом нам нужно:

- Получить отсортированный набор данных.
- Определить количество элементов в этом наборе.
- Определить центральные элементы набора.
- Получить среднее арифметическое этих элементов.

Мы можем не различать случаи, когда центральный элемент один, и когда их два, вычисляя среднее арифметическое в обоих случаях, т.к. среднее арифметическое от любого одного числа — это и есть само число.

Рассмотрим решение для MySQL.

MySQL Решение 2.2.7.d

```
SELECT AVG(`books`) AS `med_reading`
FROM   (SELECT @rownum := @rownum + 1 AS `RowNumber`,
               `books`
        FROM   (SELECT COUNT(`sb_book`) AS `books`
                FROM   `authors`
                       JOIN `m2m_books_authors` USING (`a_id`)
                       LEFT OUTER JOIN `subscriptions`
                       ON `m2m_books_authors`.`b_id` = `sb_book`
                GROUP  BY `a_id`) AS `inner_data`,
               (SELECT @rownum := 0) AS `rownum_initialisation`
        ORDER  BY `books`) AS `popularity`,
       (SELECT COUNT(*) AS `RowCount`
        FROM   (SELECT COUNT(`sb_book`) AS `books`
                FROM   `authors`
                       JOIN `m2m_books_authors` USING (`a_id`)
                       LEFT OUTER JOIN `subscriptions`
                       ON `m2m_books_authors`.`b_id` = `sb_book`
               GROUP  BY `a_id`) AS `inner_data`) AS `total_rows`
WHERE  `RowNumber` IN ( FLOOR(( `RowCount` + 1 ) / 2),
                        FLOOR(( `RowCount` + 2 ) / 2) )
```

Начнём с самых глубоко вложенных подзапросов в строках 4-9 и 13-18: легко заметить, что они полностью дублируются (увы, подготовить эти данные один раз и использовать многократно в MySQL не получится). Оба подзапроса возвращают следующие данные:

books
1
2
2
0
0
4
4

Подзапрос в строках 12-18 определяет количество рядов в этом наборе данных и возвращает одно число: 7.

Подзапрос в строках 2-11 упорядочивает этот набор данных и нумерует его строки. Поскольку в MySQL нет готовых встроенных функций для нумерации строк выборки, приходится получать необходимый эффект в несколько шагов:

- Конструкция `SELECT @rownum := 0` в строке 10 инициализирует переменную `@rownum` значением 0.
- Конструкция `SELECT @rownum := @rownum + 1 AS `RowNumber`` в строке 2 увеличивает на 1 значение переменной `@rownum` для каждого следующего ряда выборки. Колонка, в которой будут располагаться номера рядов, будет называться `RowNumber`.

Результат выполнения подзапроса в строках 2-11 таков:

RowNumber	books
1	0
2	0
3	1
4	2
5	2
6	4
7	4

Поднимемся на уровень выше и посмотрим, что вернёт весь подзапрос в строках 2-18 целиком:

RowNumber	books	RowCount
1	0	7
2	0	7
3	1	7
4	2	7
5	2	7
6	4	7
7	4	7

Повторяющееся значение `RowCount` выглядит несколько раздражающе, но для данного варианта решения такой подход является наиболее простым.

Конструкция **WHERE** в строках 19 и 20 указывает на необходимость взять для финального анализа только те ряды, значения **RowNumber** которых удовлетворяют условиям

- Условие 1: **FLOOR((`RowCount` + 1) / 2)**.
- Условие 2: **FLOOR((`RowCount` + 2) / 2)**.

В нашем конкретном случае RowCount = 7, получается:

- Условие 1: **FLOOR((7 + 1) / 2) = FLOOR (8 / 2) = 4**.
- Условие 2: **FLOOR((7 + 2) / 2) = FLOOR (9 / 2) = 4**.

Оба условия указывают на один и тот же ряд — 4-й. Значение 2 поля books из 4-го ряда передаётся в функцию **AVG** (первая строка запроса), и т.к. **AVG(2) = 2**, мы получаем конечный результат: медиана равна 2.

Если бы количество рядом было чётным (например, 8), условия в строках 19 и 20 приняли бы следующие значения:

- Условие 1: **FLOOR((8 + 1) / 2) = FLOOR (9 / 2) = 4**.
- Условие 2: **FLOOR((8 + 2) / 2) = FLOOR (10 / 2) = 5**.

Переходим к рассмотрению решения для MS SQL Server.

MS SQL Решение 2.2.7.d

```
1   WITH [popularity]
2        AS (SELECT COUNT([sb_book]) AS [books]
3            FROM   [authors]
4                   JOIN [m2m_books_authors]
5                     ON [authors].[a_id] = [m2m_books_authors].[a_id]
6                   LEFT OUTER JOIN [subscriptions]
7                                   ON [m2m_books_authors].[b_id] = [sb_book]
8            GROUP  BY [authors].[a_id]),
9       [meadian_preparation]
10      AS (SELECT CAST([books] AS FLOAT) AS [books],
11                 ROW_NUMBER()
12                   OVER (
13                     ORDER BY [books])  AS [RowNumber],
14                 COUNT(*)
15                   OVER (
16                     PARTITION BY NULL) AS [RowCount]
17          FROM   [popularity])
18  SELECT AVG([books]) AS [med_reading]
19  FROM   [meadian_preparation]
20  WHERE  [RowNumber] IN ( ( [RowCount] + 1 ) / 2, ( [RowCount] + 2 ) / 2 )
```

Благодаря наличию общих табличных выражений и функций нумерации рядов выборки, решение для MS SQL Server получается намного проще.

Первое общее табличное выражение в строках 1-8 возвращает такие данные:

books
1
2
2
0
0
4
4

Второе общее табличное выражение дорабатывает этот набор данных, в результате чего получается:

books	RowNumber	RowCount
0	1	7
0	2	7
1	3	7
2	4	7
2	5	7
4	6	7
4	7	7

Основная часть запроса в строках 18-20 действует совершенно аналогично основной части запроса в решении для MySQL (строки 1 и 19-20): определяются номера центральных рядов и вычисляется среднее арифметическое значений поля books этих рядов, что и является искомым значением медианы.

Переходим к рассмотрению решения для Oracle.

Oracle Решение 2.2.7.d

```
1  WITH "popularity"
2       AS (SELECT COUNT("sb_book") AS "books"
3           FROM   "authors"
4                  JOIN "m2m_books_authors" USING ("a_id")
5                  LEFT OUTER JOIN "subscriptions"
6                               ON "m2m_books_authors"."b_id" = "sb_book"
7           GROUP  BY "a_id")
8  SELECT MEDIAN("books") AS "med_reading" FROM "popularity"
```

Благодаря наличию в Oracle функции **MEDIAN** решение сводится к подготовке множества значений, медиану которого мы ищем, и… вызову функции **MEDIAN**. Общее табличное выражение в строках 1-7 подготавливает уже очень хорошо знакомый нам по решениям для двух других СУБД набор данных:

books
1
2
2
0
0
4
4

Решение 2.2.7.e[117].

Для решения этой задачи необходимо:

- Определить по каждой книге количество её экземпляров, выданных на руки читателям.
- Вычесть полученное значение и количества экземпляров книги, зарегистрированных в библиотеке.
- Проверить, существуют ли книги, для которых результат такого вычитания отказался отрицательным, и вернуть 0, если таких книг нет, и 1, если такие книги есть.

MySQL Решение 2.2.7.e

```
1   SELECT EXISTS (SELECT `b_id`
2                  FROM   `books`
3                         LEFT OUTER JOIN (SELECT `sb_book`,
4                                                 COUNT(`sb_book`) AS `taken`
5                                          FROM   `subscriptions`
6                                          WHERE  `sb_is_active` = 'Y'
7                                          GROUP  BY `sb_book`
8                                          ) AS `books_taken`
9                                       ON `b_id` = `sb_book`
10                 WHERE  ( `b_quantity` - IFNULL(`taken`, 0) ) < 0
11                 LIMIT 1)
12         AS `error_exists`
```

MySQL трактует значения **TRUE** и **FALSE** как 1 и 0 соответственно, потому на верхнем уровне запроса (строка 1) можно просто возвращать значение функции **EXISTS**.

Подзапрос в строках 3-9 возвращает следующие данные (количество экземпляров, выданных на руки читателям, по каждой книге, хотя бы один экземпляр которой выдан):

sb_book	taken
1	2
3	1
4	1
5	1

Подзапрос в строках 1-11 возвращает не более одного ряда выборки, содержащей идентификаторы книг с отрицательным остатком («не более одного», т.к. нас интересует просто факт наличия или отсутствия таких книг). Если этот подзапрос вернёт ноль рядов, функция **EXISTS** на пустой выборке вернёт **FALSE** (0). Если подзапрос вернёт один ряд, множество уже будет непустым, и функция **EXISTS** вернёт **TRUE** (1).

Поскольку в MS SQL Server и Oracle нельзя напрямую получить 1 и 0 из результата работы функции **EXISTS**, решения для этих СУБД будут чуть более сложными.

MS SQL Решение 2.2.7.e

```
1   WITH [books_taken]
2        AS (SELECT [sb_book],
3                   COUNT([sb_book]) AS [taken]
4            FROM   [subscriptions]
5            WHERE  [sb_is_active] = 'Y'
6            GROUP  BY [sb_book])
7   SELECT TOP 1 CASE
8                  WHEN EXISTS (SELECT TOP 1 [b_id]
9                               FROM   [books]
10                                LEFT OUTER JOIN [books_taken]
11                                             ON [b_id] = [sb_book]
12                              WHERE  ([b_quantity] - ISNULL([taken], 0)) < 0)
13                 THEN 1
14                 ELSE 0
15               END AS [error_exists]
16  FROM   [books_taken]
```

Общее табличное выражение в строках 1-6 возвращает информацию о том, сколько экземпляров каждой книги выдано на руки читателям:

sb_book	taken
1	2
3	1
4	1
5	1

Подзапрос в строках 8-12 возвращает не более одного идентификатора книги с отрицательным остатком и передаёт эту информацию в функцию **EXISTS**, которая возвращает **TRUE** или **FALSE** в зависимости от того, нашлись такие книги или не нашлись.

Конструкция **CASE ... WHEN ... THEN ... ELSE ... END** в строках 7-15 позволяет преобразовать логический результат работы функции **EXISTS** в требуемые по условию задачи 1 и 0.

Указание **TOP 1** в строке 7 необходимо для того, чтобы в итоговую выборку попала одна строка, а не такое количество строк, какое возвратит общее табличное выражение.

Oracle Решение 2.2.7.e

```
1   WITH "books_taken"
2        AS (SELECT "sb_book",
3                   COUNT("sb_book") AS "taken"
4            FROM   "subscriptions"
5            WHERE  "sb_is_active" = 'Y'
6            GROUP  BY "sb_book")
7   SELECT CASE
8                  WHEN EXISTS (SELECT "b_id"
9                               FROM   "books"
10                                     LEFT OUTER JOIN "books_taken"
11                                                     ON "b_id" = "sb_book"
12                              WHERE  ("b_quantity" - NVL("taken", 0)) < 0
13                              AND ROWNUM = 1)
14                 THEN 1
15                 ELSE 0
16               END AS "error_exists"
17  FROM   "books_taken"
18  WHERE ROWNUM = 1
```

Решение для Oracle идентично решению для MS SQL Server за исключением использования вместо **TOP 1** конструкции **WHERE ROWNUM = 1**, позволяющей вернуть не более одного ряда выборки.

Задание 2.2.7.TSK.A: показать читаемость жанров, т.е. все жанры и то количество раз, которое книги этих жанров были взяты читателями.

Задание 2.2.7.TSK.B: показать самый читаемый жанр, т.е. жанр (или жанры, если их несколько), относящиеся к которому книги читатели брали чаще всего.

Задание 2.2.7.TSK.C: показать среднюю читаемость жанров, т.е. среднее значение от того, сколько раз читатели брали книги каждого автора.

Задание 2.2.7.TSK.D: показать медиану читаемости жанров, т.е. медианное значение от того, сколько раз читатели брали книги каждого жанра.

2.2.8. ПРИМЕР 18: УЧЁТ ВАРИАНТОВ И КОМБИНАЦИЙ ПРИЗНАКОВ

Задача 2.2.8.a{130}: показать авторов, **одновременно** работавших в двух и более жанрах (т.е. хотя бы одна книга автора должна одновременно относиться к двум и более жанрам).

Задача 2.2.8.b{133}: показать авторов, работавших в двух и более жанрах (даже если каждая отдельная книга автора относится только к одному жанру).

Ожидаемый результат 2.2.8.a.

a_id	a_name	genres_count
1	Д. Кнут	2
3	Д. Карнеги	2
6	Б. Страуструп	2
7	А.С. Пушкин	2

Ожидаемый результат 2.2.8.b.

a_id	a_name	genres_count
1	Д. Кнут	2
3	Д. Карнеги	2
6	Б. Страуструп	2
7	А.С. Пушкин	2

На имеющемся наборе данных ожидаемые результаты 2.2.8.a и 2.2.8.b совпадают, но из этого никоим образом не следует, что они всегда должны быть одинаковыми.

Решение 2.2.8.a{130}.

MySQL Решение 2.2.8.a

```
1  SELECT `a_id`,
2         `a_name`,
3         MAX(`genres_count`) AS `genres_count`
4  FROM   (SELECT `a_id`,
5                 `a_name`,
6                 COUNT(`g_id`) AS `genres_count`
7          FROM   `authors`
8                 JOIN `m2m_books_authors` USING (`a_id`)
9                 JOIN `m2m_books_genres` USING (`b_id`)
10         GROUP  BY `a_id`,
11                   `b_id`
12         HAVING `genres_count` > 1) AS `prepared_data`
13 GROUP  BY `a_id`
```

Для получения результата нужно выполнить два шага.

Первый шаг представлен подзапросом в строках 4-12: здесь происходит объединение трёх таблиц (из таблицы **authors** берётся информация об авторах, из таблицы **m2m_books_authors** — информация о написанных каждым автором книгах, из таблицы **m2m_books_genres** — информация о жанрах, к которым относится каждая книга) и в выборку попадают авторы, у которых есть книги, относящиеся к двум и более жанрам.

Второй шаг (представленный основной частью запроса в строках 1-3 и 13) нужен для корректной обработки ситуации, в которой у некоторого автора оказывается несколько книг, относящихся к двум и более жанрам, но каждая из которых относится к разному количеству жанров. Данные, полученные из подзапроса тогда приняли бы, например, такой вид (обратите внимание, что **DISTINCT** здесь не поможет, т.к. 4-я и 5-я записи отличаются значением поля **genres_count**):

a_id	a_name	genres_count
1	Д. Кнут	2
3	Д. Карнеги	2
6	Б. Страуструп	2
7	А.С. Пушкин	2
7	А.С. Пушкин	3

Но благодаря повторной группировке по идентификаторам авторов с поиском максимального количества жанров эти данные приняли бы такой конечный вид:

a_id	a_name	genres_count
1	Д. Кнут	2
3	Д. Карнеги	2
6	Б. Страуструп	2
7	А.С. Пушкин	3

Поскольку задачи 2.2.8.a и 2.2.8.b во многом схожи, для лучшего понимания их ключевых отличий сейчас мы рассмотрим внутренние данные, которые возвращает подзапрос в строках 4-12. Перепишем эту часть, добавив в выборку информацию о книгах и убрав условие «два и более жанра»:

MySQL Решение 2.2.8.a (модифицированный код подзапроса)

```
SELECT `a_id`,
       `a_name`,
       `b_id`,
       `b_name`,
       COUNT(`g_id`) AS `genres_count`
FROM   `authors`
       JOIN `m2m_books_authors` USING (`a_id`)
       JOIN `m2m_books_genres` USING (`b_id`)
       JOIN `books` USING (`b_id`)
GROUP  BY `a_id`,
          `b_id`
```

Благодаря двойной группировке по идентификаторам автора и книги мы получаем информацию о количестве жанров каждой отдельной книги каждого автора:

a_id	a_name	b_id	b_name	genres_count
1	Д. Кнут	7	Искусство программирования	2
2	А. Азимов	3	Основание и империя	1
3	Д. Карнеги	4	Психология программирования	2
4	Л.Д. Ландау	6	Курс теоретической физики	1
5	Е.М. Лифшиц	6	Курс теоретической физики	1
6	Б. Страуструп	4	Психология программирования	2
6	Б. Страуструп	5	Язык программирования C++	1
7	А.С. Пушкин	1	Евгений Онегин	2
7	А.С. Пушкин	2	Сказка о рыбаке и рыбке	2

Иными словами, здесь мы считаем «количество жанров у каждой книги», а в задаче 2.2.8.b мы будем считать «количество жанров у автора».

Решение этой задачи для MS SQL Server и Oracle отличается от решения для MySQL только синтаксическими нюансами.

MS SQL Решение 2.2.8.a

```
SELECT [a_id],
       [a_name],
       MAX([genres_count]) AS [genres_count]
FROM   (SELECT [authors].[a_id],
               [authors].[a_name],
               COUNT([m2m_books_genres].[g_id]) AS [genres_count]
        FROM   [authors]
               JOIN [m2m_books_authors]
                 ON [authors].[a_id] = [m2m_books_authors].[a_id]
               JOIN [m2m_books_genres]
                 ON [m2m_books_authors].[b_id] = [m2m_books_genres].[b_id]
        GROUP  BY [authors].[a_id],
                  [a_name],
                  [m2m_books_authors].[b_id]
        HAVING COUNT([m2m_books_genres].[g_id]) > 1) AS [prepared_data]
GROUP  BY [a_id],
          [a_name]
```

Oracle Решение 2.2.8.a

```
1   SELECT "a_id",
2          "a_name",
3          MAX("genres_count") AS "genres_count"
4   FROM   (SELECT "a_id",
5                  "a_name",
6                  COUNT("g_id") AS "genres_count"
7           FROM   "authors"
8                  JOIN "m2m_books_authors" USING ("a_id")
9                  JOIN "m2m_books_genres" USING ("b_id")
10          GROUP  BY "a_id",
11                    "a_name",
12                    "b_id"
13          HAVING COUNT("g_id") > 1) "prepared_data"
14  GROUP  BY "a_id",
15            "a_name"
```

Решение 2.2.8.b{130}.

Как только что было подчёркнуто в разборе решения{130} задачи 2.2.8.a{130}, здесь нам придётся считать «количество жанров у автора».

MySQL Решение 2.2.8.b

```
1   SELECT `a_id`,
2          `a_name`,
3          COUNT(`g_id`) AS `genres_count`
4   FROM   (SELECT DISTINCT `a_id`,
5                           `g_id`
6           FROM   `m2m_books_genres`
7                  JOIN `m2m_books_authors` USING (`b_id`)
8          ) AS `prepared_data`
9          JOIN `authors` USING (`a_id`)
10  GROUP  BY `a_id`
11  HAVING `genres_count` > 1
```

Снова доработаем подзапрос (строки 4-8) и посмотрим, какие данные он возвращает:

MySQL Решение 2.2.8.b (модифицированный код подзапроса)

```
1   SELECT DISTINCT `a_id`,
2                   `a_name`,
3                   `g_id`,
4                   `g_name`
5   FROM   `m2m_books_genres`
6          JOIN `m2m_books_authors` USING (`b_id`)
7          JOIN `authors` USING (`a_id`)
8          JOIN `genres` USING (`g_id`)
9   ORDER  BY `a_id`,
10            `g_id`
```

Данные получаются такие (список без повторений всех жанров, в которых работал автор):

a_id	a_name	g_id	g_name
1	Д. Кнут	2	Программирование
1	Д. Кнут	5	Классика
2	А. Азимов	6	Фантастика
3	Д. Карнеги	2	Программирование
3	Д. Карнеги	3	Психология
4	Л.Д. Ландау	5	Классика
5	Е.М. Лифшиц	5	Классика
6	Б. Страуструп	2	Программирование
6	Б. Страуструп	3	Психология
7	А.С. Пушкин	1	Поэзия
7	А.С. Пушкин	5	Классика

В основной части запроса (строки 1-3 и 9-11) остаётся только посчитать количество элементов в списке жанров для каждого автора, а затем оставить в выборке тех авторов, у которых это количество больше единицы. Так мы получаем итоговый результат.

Решение этой задачи для MS SQL Server и Oracle отличается от решения для MySQL только синтаксическими нюансами.

MS SQL Решение 2.2.8.b

```
1   SELECT [prepared_data].[a_id],
2          [a_name],
3          COUNT([g_id]) AS [genres_count]
4   FROM   (SELECT DISTINCT [m2m_books_authors].[a_id],
5                           [m2m_books_genres].[g_id]
6           FROM   [m2m_books_genres]
7                  JOIN [m2m_books_authors]
8                    ON [m2m_books_genres].[b_id] = [m2m_books_authors].[b_id])
9   AS
10         [prepared_data]
11         JOIN [authors]
12           ON [prepared_data].[a_id] = [authors].[a_id]
13  GROUP  BY [prepared_data].[a_id],
14            [a_name]
15  HAVING COUNT([g_id]) > 1
```

Oracle Решение 2.2.8.b

```
1   SELECT "a_id",
2          "a_name",
3          COUNT("g_id") AS "genres_count"
4   FROM   (SELECT DISTINCT "a_id",
5                           "g_id"
6           FROM   "m2m_books_genres"
7                  JOIN "m2m_books_authors" USING ("b_id")
8          ) "prepared_data"
9          JOIN "authors" USING ("a_id")
10  GROUP  BY "a_id",
11            "a_name"
12  HAVING COUNT("g_id") > 1
```

Задание 2.2.8.TSK.A: переписать решения задач 2.2.8.a{130} и 2.2.8.b{133} для MS SQL Server и Oracle с использованием общих табличных выражений.

Задание 2.2.8.TSK.B: показать читателей, бравших самые разножанровые книги (т.е. книги, одновременно относящиеся к максимальному количеству жанров).

Задание 2.2.8.TSK.C: показать читателей наибольшего количества жанров (не важно, брали ли они книги, каждая из которых относится одновременно к многим жанрам, или же просто много книг из разных жанров, каждая из которых относится к небольшому количеству жанров).

2.2.9. ПРИМЕР 19: ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ И ПОИСК МИНИМУМА, МАКСИМУМА, ДИАПАЗОНОВ

Задача 2.2.9.a{136}: показать читателя, первым взявшего в библиотеке книгу.

Задача 2.2.9.b{138}: показать читателя (или читателей, если их окажется несколько), быстрее всего прочитавшего книгу (учитывать только случаи, когда книга возвращена).

Задача 2.2.9.c{141}: показать, какую книгу (или книги, если их несколько) каждый читатель взял в первый день своей работы с библиотекой.

Задача 2.2.9.d{144}: показать первую книгу, которую каждый из читателей взял в библиотеке.

Ожидаемый результат 2.2.9.a.

s_name
Иванов И.И.

Ожидаемый результат 2.2.9.b.

s_id	s_name	days
1	Иванов И.И.	31

Ожидаемый результат 2.2.9.c.

s_id	s_name	books_list
1	Иванов И.И.	Евгений Онегин, Основание и империя
3	Сидоров С.С.	Основание и империя
4	Сидоров С.С.	Язык программирования C++

Это разные Сидоровы!

Ожидаемый результат 2.2.9.d.

s_id	s_name	b_name
1	Иванов И.И.	Евгений Онегин
3	Сидоров С.С.	Основание и империя
4	Сидоров С.С.	Язык программирования C++

Это разные Сидоровы!

Решение 2.2.9.a[{135}].

В данном случае мы сделаем допущение о том, что первичные ключи в таблице **subscriptions** никогда не изменяются, т.е. минимальное значение первичного ключа действительно соответствует первому факту выдачи книги читателю. Тогда решение оказывается очень простым (что будет, если подобного допущения не делать, показано в решении 2.2.9.c[{141}]).

Для всех трёх СУБД рассмотрим два варианта решения, отличающиеся логикой получения минимального значения первичного ключа таблицы **subscriptions**: с использованием функции **MIN** и с использованием сортировки по возрастанию с последующим использованием первого ряда выборки.

MySQL Решение 2.2.9.a

```
1  -- Вариант 1: использование функции MIN
2  SELECT `s_name`
3  FROM   `subscribers`
4  WHERE  `s_id` = (SELECT `sb_subscriber`
5                   FROM   `subscriptions`
6                   WHERE  `sb_id` = (SELECT MIN(`sb_id`)
7                                     FROM   `subscriptions`))

1  -- Вариант 2: использование сортировки
2  SELECT `s_name`
3  FROM   `subscribers`
4  WHERE  `s_id` = (SELECT `sb_subscriber`
5                   FROM   `subscriptions`
6                   ORDER  BY `sb_id` ASC
7                   LIMIT 1)
```

MS SQL Решение 2.2.9.a

```
-- Вариант 1: использование функции MIN
SELECT [s_name]
FROM   [subscribers]
WHERE  [s_id] = (SELECT [sb_subscriber]
                 FROM   [subscriptions]
                 WHERE  [sb_id] = (SELECT MIN([sb_id])
                                   FROM   [subscriptions]))

-- Вариант 2: использование сортировки
SELECT [s_name]
FROM   [subscribers]
WHERE  [s_id] = (SELECT TOP 1 [sb_subscriber]
                 FROM   [subscriptions]
                 ORDER  BY [sb_id] ASC)
```

Oracle Решение 2.2.9.a

```
-- Вариант 1: использование функции MIN
SELECT "s_name"
FROM   "subscribers"
WHERE  "s_id" = (SELECT "sb_subscriber"
                 FROM   "subscriptions"
                 WHERE  "sb_id" = (SELECT MIN("sb_id")
                                   FROM   "subscriptions"))

-- Вариант 2: использование сортировки
SELECT "s_name"
FROM   "subscribers"
WHERE  "s_id" = (SELECT "sb_subscriber"
                 FROM   (SELECT "sb_subscriber",
                                ROW_NUMBER()
                                  OVER(
                                    ORDER BY "sb_id" ASC) AS "rn"
                         FROM   "subscriptions")
                 WHERE  "rn" = 1)
```

Исследование 2.2.9.EXP.A: сравним скорость работы решений этой задачи, выполнив запросы 2.2.9.a на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений каждого запроса:

	MySQL	MS SQL Server	Oracle
Использование функции MIN	0.001	0.020	2.876
Использование сортировки	0.001	0.018	10.330

В MySQL и MS SQL Server оба варианта запросов работают с примерно сопоставимой скоростью, но в Oracle эмуляция конструкций `LIMIT/TOP` через нумерацию рядов приводит к очень ощутимому падению скорости работы запроса.

Решение 2.2.9.b[135].

Т.к. MySQL не поддерживает общие табличные выражения и ранжирующие (оконные) функции, для этой СУБД мы рассмотрим только один вариант решения, а для MS SQL Server и Oracle — три варианта.

MySQL Решение 2.2.9.b

```
1   -- Вариант 1: использование подзапроса и сортировки
2   SELECT DISTINCT `s_id`,
3                   `s_name`,
4                   DATEDIFF(`sb_finish`, `sb_start`) AS `days`
5   FROM    `subscribers`
6           JOIN `subscriptions`
7             ON `s_id` = `sb_subscriber`
8   WHERE   `sb_is_active` = 'N'
9           AND DATEDIFF(`sb_finish`, `sb_start`) =
10             (SELECT    DATEDIFF(`sb_finish`, `sb_start`) AS `days`
11              FROM      `subscriptions`
12              WHERE     `sb_is_active` = 'N'
13              ORDER  BY `days` ASC
14              LIMIT  1)
```

Подзапрос в строках 10-14 возвращает информацию о минимальном количестве дней, за которые была возвращена книга:

days
31

Основная часть запроса в строках 2-8 получает информацию обо всех читателях и количестве дней, которые каждый из читателей держал у себя каждую взятую и возвращённую им книгу:

s_id	s_name	days
1	Иванов И.И.	31
1	Иванов И.И.	61
4	Сидоров С.С.	61

Условие в строке 9 оставляет из этого набора только те записи, в которых количество дней равно количеству, определённому подзапросом в строках 10-14. Так мы получаем конечный набор данных.

Вариант 1 решений для MS SQL Server и Oracle построен аналогичным образом.

MS SQL Решение 2.2.9.b

```
-- Вариант 1: использование подзапроса и сортировки
SELECT DISTINCT [s_id],
                [s_name],
                DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
FROM   [subscribers]
       JOIN [subscriptions]
         ON [s_id] = [sb_subscriber]
WHERE  [sb_is_active] = 'N'
       AND DATEDIFF(day, [sb_start], [sb_finish]) =
       (SELECT TOP 1 DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
        FROM         [subscriptions]
        WHERE        [sb_is_active] = 'N'
        ORDER  BY    [days] ASC)
```

```
-- Вариант 2: использование общего табличного выражения и функции MIN
WITH [prepared_data]
     AS (SELECT DISTINCT [s_id],
                         [s_name],
                         DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
         FROM   [subscribers]
                JOIN [subscriptions]
                  ON [s_id] = [sb_subscriber]
         WHERE  [sb_is_active] = 'N')
SELECT [s_id],
       [s_name],
       [days]
FROM   [prepared_data]
WHERE  [days] = (SELECT MIN([days])
                 FROM   [prepared_data])
```

```
-- Вариант 3: использование общего табличного выражения и ранжирования
WITH [prepared_data]
     AS (SELECT DISTINCT [s_id],
                         [s_name],
                         DATEDIFF(day, [sb_start], [sb_finish]) AS [days],
                         RANK()
                           OVER (
                                 ORDER BY
                                 DATEDIFF(day, [sb_start], [sb_finish]) ASC
                           ) AS [rank]
         FROM   [subscribers]
                JOIN [subscriptions]
                  ON [s_id] = [sb_subscriber]
         WHERE  [sb_is_active] = 'N')
SELECT [s_id],
       [s_name],
       [days]
FROM   [prepared_data]
WHERE  [rank] = 1
```

В **варианте 2** общее табличное выражение (строки 2-9) возвращает тот же набор данных, что и основная часть запроса в варианте 1 (строки 2-8):

s_id	s_name	days
1	Иванов И.И.	31
1	Иванов И.И.	61
4	Сидоров С.С.	61

Подзапрос в строках 14-15 определяет минимальное значение столбца **days**, которое затем используется как условие ограничения выборки в основной части запроса (строки 10-14).

В **варианте 3** общее табличное выражение в строках 2-14 готовит уже знакомый нам набор данных, но ещё и ранжированный по значению столбца **days**:

s_id	s_name	days	rank
1	Иванов И.И.	31	1
1	Иванов И.И.	61	4
4	Сидоров С.С.	61	4

В основной части запроса (строки 15-19) остаётся только извлечь из этих подготовленных результатов строки, «занимающие первое место» (**rank = 1**).

Решение для Oracle аналогично решению для MS SQL Server.

Oracle Решение 2.2.9.b

```
1   -- Вариант 1: использование подзапроса и сортировки
2   SELECT DISTINCT "s_id",
3                   "s_name",
4                   ( "sb_finish" - "sb_start" ) AS "days"
5   FROM   "subscribers"
6          JOIN "subscriptions"
7            ON "s_id" = "sb_subscriber"
8   WHERE  "sb_is_active" = 'N'
9          AND ( "sb_finish" - "sb_start" ) =
10         (SELECT "min_days"
11          FROM   (SELECT ("sb_finish" - "sb_start" )          AS "min_days",
12                         ROW_NUMBER()
13                          OVER(
14                          ORDER BY ( "sb_finish" - "sb_start") ASC) AS "rn"
15                  FROM   "subscriptions"
16                  WHERE  "sb_is_active" = 'N')
17          WHERE  "rn" = 1)

1   -- Вариант 2: использование общего табличного выражения и функции MIN
2   WITH "prepared_data"
3        AS (SELECT DISTINCT "s_id",
4                            "s_name",
5                            ( "sb_finish" - "sb_start" ) AS "days"
6            FROM   "subscribers"
7                   JOIN "subscriptions"
8                     ON "s_id" = "sb_subscriber"
9            WHERE  "sb_is_active" = 'N')
10  SELECT "s_id",
11         "s_name",
12         "days"
13  FROM   "prepared_data"
14  WHERE  "days" = (SELECT MIN("days")
15                   FROM   "prepared_data")
```

Oracle Решение 2.2.9.b (продолжение)

```
-- Вариант 3: использование общего табличного выражения и ранжирования
WITH "prepared_data"
     AS (SELECT DISTINCT "s_id",
                         "s_name",
                         ("sb_finish" - "sb_start")              AS "days",
                         RANK()
                         OVER (
                         ORDER BY ("sb_finish" - "sb_start") ASC ) AS "rank"
           FROM   "subscribers"
                  JOIN "subscriptions"
                    ON "s_id" = "sb_subscriber"
           WHERE  "sb_is_active" = 'N')
SELECT "s_id",
       "s_name",
       "days"
FROM   "prepared_data"
WHERE  "rank" = 1
```

Решение 2.2.9.c{135}.

Здесь мы не будем делать допущений, подобных сделанным в решении задачи 2.2.9.a{136}, и построим запрос исключительно на информации, относящейся к предметной области.

Итак, нам надо по каждому читателю:

- выяснить дату первого визита читателя в библиотеку;
- получить набор взятых им в тот день книг;
- представить набор этих книг в виде строки.

MySQL Решение 2.2.9.c

```
SELECT `s_id`,
       `s_name`,
        GROUP_CONCAT(`b_name` ORDER BY `b_name` SEPARATOR ', ')
        AS `books_list`
FROM   (SELECT `s_id`,
               `s_name`,
               `b_name`
        FROM   `subscribers`
               JOIN (SELECT `subscriptions`.`sb_subscriber`,
                            `subscriptions`.`sb_book`
                     FROM   `subscriptions`
                            JOIN (SELECT `sb_subscriber`,
                                         MIN(`sb_start`) AS `min_date`
                                  FROM   `subscriptions`
                                  GROUP  BY `sb_subscriber`)
                                  AS `first_visit`
                                    ON `subscriptions`.`sb_subscriber` =
                                       `first_visit`.`sb_subscriber`
                                       AND `subscriptions`.`sb_start` =
                                           `first_visit`.`min_date`)
                     AS `books_list`
                       ON `s_id` = `sb_subscriber`
                    JOIN `books`
                      ON `sb_book` = `b_id`) AS `prepared_data`
GROUP  BY `s_id`
```

Наиболее глубоко вложенный подзапрос (строки 12-16) определяет дату первого визита в библиотеку каждого из читателей:

sb_subscriber	min_date
1	2011-01-12
3	2012-05-17
4	2012-06-11

Следующий по уровню вложенности подзапрос (строки 9-20) определяет список книг, взятых каждым из читателей в дату своего первого визита в библиотеку:

sb_subscriber	sb_book
1	1
1	3
3	3
4	5

Следующий по уровню вложенности подзапрос (строки 5-24) подготавливает данные с именами читателей и названиями книг:

s_id	s_name	b_name
1	Иванов И.И.	Евгений Онегин
1	Иванов И.И.	Основание и империя
3	Сидоров С.С.	Основание и империя
4	Сидоров С.С.	Язык программирования C++

И, наконец, основная часть запроса (строки 1-5 и 25) превращает отдельные строки с информацией о книгах читателя (отмечены выше серым фоном) в одну строку с перечнем книг. Так получается итоговый результат.

MS SQL Решение 2.2.9.c

```
1   WITH [step_1]
2        AS (SELECT [sb_subscriber],
3                   MIN([sb_start]) AS [min_date]
4            FROM   [subscriptions]
5            GROUP  BY [sb_subscriber]),
6        [step_2]
7        AS (SELECT [subscriptions].[sb_subscriber],
8                   [subscriptions].[sb_book]
9            FROM   [subscriptions]
10                  JOIN [step_1]
11                    ON [subscriptions].[sb_subscriber] =
12                               [step_1].[sb_subscriber]
13                       AND [subscriptions].[sb_start] = [step_1].[min_date]),
14       [step_3]
15       AS (SELECT [s_id],
16                  [s_name],
17                  [b_name]
18           FROM   [subscribers]
19                  JOIN [step_2]
20                    ON [s_id] = [sb_subscriber]
21                  JOIN [books]
22                    ON [sb_book] = [b_id])
```

MS SQL Решение 2.2.9.c (продолжение)

```
23  SELECT [s_id],
24         [s_name],
25         STUFF
26         ((SELECT    ', ' + [int].[b_name]
27           FROM      [step_3] AS [int]
28           WHERE     [ext].[s_id] = [int].[s_id]
29           ORDER  BY [int].[b_name]
30           FOR xml path(''), type).value('.', 'nvarchar(max)'),
31           1, 2, '') AS [books_list]
32  FROM   [step_3] AS [ext]
33  GROUP BY [s_id], [s_name]
```

В решении для MS SQL Server общие табличные выражения возвращают те же данные, что и подзапросы в решении для MySQL.

Общее табличное выражение **step_1** (строки 1-5) определяет дату первого визита в библиотеку каждого из читателей:

sb_subscriber	min_date
1	2011-01-12
3	2012-05-17
4	2012-06-11

Общее табличное выражение **step_2** (строки 6-13) определяет список книг, взятых каждым из читателей в дату своего первого визита в библиотеку:

sb_subscriber	sb_book
1	1
1	3
3	3
4	5

Общее табличное выражение **step_3** (строки 14-22) подготавливает данные с именами читателей и названиями книг:

s_id	s_name	b_name
1	Иванов И.И.	Евгений Онегин
1	Иванов И.И.	Основание и империя
3	Сидоров С.С.	Основание и империя
4	Сидоров С.С.	Язык программирования C++

И, наконец, основная часть запроса (строки 23-33) превращает отдельные строки с информацией о книгах читателя (отмечены выше серым фоном) в одну строку с перечнем книг. Так получается итоговый результат.

Решение для Oracle аналогично решению для MS SQL Server.

Стоит лишь отметить, что последний шаг (превращение нескольких строк с названиями книг в одну строку) реализуется совершенно по-разному в каждой из СУБД в силу крайне серьёзных различий в синтаксисе и наборе поддерживаемых функций. Подробнее эта операция рассмотрена в задаче 2.2.2.a{73}.

Oracle Решение 2.2.9.c

```
WITH "step_1"
     AS (SELECT "sb_subscriber",
                MIN("sb_start") AS "min_date"
         FROM   "subscriptions"
         GROUP  BY "sb_subscriber"),
     "step_2"
     AS (SELECT "subscriptions"."sb_subscriber",
                "subscriptions"."sb_book"
         FROM   "subscriptions"
                join "step_1"
                  ON "subscriptions"."sb_subscriber" =
                             "step_1"."sb_subscriber"
                     AND "subscriptions"."sb_start" = "step_1"."min_date"),
     "step_3"
     AS (SELECT "s_id",
                "s_name",
                "b_id",
                "b_name"
         FROM   "subscribers"
                JOIN "step_2"
                  ON "s_id" = "sb_subscriber"
                JOIN "books"
                  ON "sb_book" = "b_id")
SELECT "s_id",
       "s_name",
       UTL_RAW.CAST_TO_NVARCHAR2
       (
        LISTAGG
        (
         UTL_RAW.CAST_TO_RAW("b_name"),
         UTL_RAW.CAST_TO_RAW(N', ')
        )
        WITHIN GROUP (ORDER BY "b_name")
       )
       AS "books_list"
FROM   "step_3"
GROUP  BY "s_id",
          "s_name"
```

Исследование 2.2.9.EXP.B: сравним скорость работы трёх СУБД при выполнении запросов 2.2.9.c на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений каждого запроса:

MySQL	MS SQL Server	Oracle
91789.850	203.083	5.167

Результаты говорят сами за себя. В реальной задаче для MySQL и MS SQL Server явно придётся искать иное, пусть технически и более сложное, но более производительное решение.

Решение 2.2.9.d[{135}].

Данная задача является логическим продолжением задачи 2.2.9.c[{135}], только теперь нужно определить, какая книга из тех, что читатель взял в первый свой визит в библиотеку, была выда-

на первой. Поскольку дата выдачи хранится с точностью до дня, у нас не остаётся иного выхода, кроме как предположить, что первой была выдана книга, запись о выдаче которой имеет минимальное значение первичного ключа.

Первый вариант решения основан на последовательном определении первого визита для каждого читателя, первой выдачи книги в рамках этого визита, идентификатора книги в этой выдаче и объединения с таблицами **subscribers** и **books**, из которых мы извлечём имя читателя и название книги.

MySQL Решение 2.2.9.d

```
1   -- Вариант 1: решение в четыре шага без использования ранжирования
2   SELECT `s_id`,
3          `s_name`,
4          `b_name`
5   FROM   (SELECT `subscriptions`.`sb_subscriber`,
6                  `sb_book`
7           FROM   `subscriptions`
8                  JOIN (SELECT `subscriptions`.`sb_subscriber`,
9                               MIN(`sb_id`) AS `min_sb_id`
10                       FROM   `subscriptions`
11                              JOIN (SELECT `sb_subscriber`,
12                                           MIN(`sb_start`) AS `min_sb_start`
13                                    FROM   `subscriptions`
14                                    GROUP  BY `sb_subscriber`)
15                                    AS `step_1`
16                                ON `subscriptions`.`sb_subscriber` =
17                                   `step_1`.`sb_subscriber`
18                                   AND `subscriptions`.`sb_start` =
19                                       `step_1`.`min_sb_start`
20                       GROUP  BY `subscriptions`.`sb_subscriber`,
21                                 `min_sb_start`)
22                       AS `step_2`
23                    ON `subscriptions`.`sb_id` = `step_2`.`min_sb_id`)
24          AS `step_3`
25         JOIN `subscribers`
26           ON `sb_subscriber` = `s_id`
27         JOIN `books`
28           ON `sb_book` = `b_id`
```

Наиболее глубоко вложенный подзапрос (**step_1**) в строках 11-15 возвращает информацию о дате первого визита в библиотеку каждого читателя:

sb_subscriber	min_sb_start
1	2011-01-12
3	2012-05-17
4	2012-06-11

Следующий подзапрос (**step_2**) в строках 8-22 на основе только что полученной информации определяет минимальное значение первичного ключа таблицы **subscribers**, соответствующее каждому читателю и дате его первого визита в библиотеку:

sb_subscriber	min_sb_id
1	2
3	3
4	57

Следующий подзапрос (**step_3**) в строках 5-24 на основе этой информации определяет идентификатор книги, соответствующий каждой выдаче с идентификатором **min_sb_id**:

sb_subscriber	sb_book
1	1
3	3
4	5

Основная часть запроса в строках 2-4 и 25-28 является четвёртым шагом, на котором по имеющимся идентификаторам определяются имя читателя и название книги. Так получается финальный результат.

MySQL Решение 2.2.9.d

```
1   -- Вариант 2: решение в два шага с использованием ранжирования
2   SELECT `s_id`,
3          `s_name`,
4          `b_name`
5   FROM   (SELECT `sb_subscriber`,
6                  `sb_start`,
7                  `sb_id`,
8                  `sb_book`,
9                  ( CASE
10                     WHEN ( @sb_subscriber_value = `sb_subscriber` )
11                      THEN @i := @i + 1
12                     ELSE ( @i := 1 )
13                      AND ( @sb_subscriber_value := `sb_subscriber` )
14                   END ) AS `rank_by_subscriber`,
15                 ( CASE
16                     WHEN ( @sb_subscriber_value = `sb_subscriber` )
17                          AND ( @sb_start_value = `sb_start` )
18                      THEN @j := @j + 1
19                     ELSE ( @j := 1 )
20                          AND ( @sb_subscriber_value := `sb_subscriber` )
21                          AND ( @sb_start_value := `sb_start` )
22                   END ) AS `rank_by_date`
23           FROM   `subscriptions`,
24                  (SELECT @i := 0,
25                          @j := 0,
26                          @sb_subscriber_value := '',
27                          @sb_start_value := ''
28                  ) AS `initialisation`
29           ORDER  BY `sb_subscriber`,
30                     `sb_start`,
31                     `sb_id`) AS `ranked`
32          JOIN `subscribers`
33            ON `sb_subscriber` = `s_id`
34          JOIN `books`
35            ON `sb_book` = `b_id`
36  WHERE  `rank_by_subscriber` = 1
37          AND `rank_by_date` = 1
```

Второй вариант решения построен на идее ранжирования дат визитов каждого читателя и ранжирования выдач книг в рамках каждого визита каждого читателя. Поскольку MySQL не поддерживает никаких ранжирующих (оконных) функций, их поведение придётся эмулировать.

Подзапрос в строках 24-28 отвечает за начальную инициализацию переменных:

- переменная `@i` хранит порядковый номер визита каждого из читателей;
- переменная `@j` хранит порядковый номер выдачи книги в рамках каждого визита каждого читателя;
- переменная `@sb_subscriber_value` используется для определения того факта, что в процессе обработки данных изменился идентификатор читателя;
- переменная `@sb_start_value` используется для определения того факта, что в процессе обработки данных изменилось значение даты визита.

Выражение в строках 9-14 определяет, изменилось ли значение идентификатора читателя, и либо инкрементирует порядковые номер `@i` (если значение идентификатора не изменилось), либо инициализирует его значением 1 (если значение идентификатора изменилось).

Выражение в строках 15-22 определяет, изменились ли значения идентификатора читателя и даты визита, и либо инкрементирует порядковые номер `@j` (если значения идентификатора и даты не изменились), либо инициализирует его значением 1 (если значения идентификатора или даты изменились).

Результатом работы подзапроса в строках 5-21 является следующий набор данных (серым фоном отмечены строки, представляющие для нас интерес в контексте решения задачи):

sb_subscriber	sb_start	sb_id	sb_book	rank_by_subscriber	rank_by_date
1	2011-01-12	2	1	1	1
1	2011-01-12	100	3	2	2
1	2012-06-11	42	2	3	1
1	2014-08-03	61	7	4	1
1	2015-10-07	95	4	5	1
3	2012-05-17	3	3	1	1
3	2014-08-03	62	5	2	1
3	2014-08-03	86	1	3	2
4	2012-06-11	57	5	1	1
4	2015-10-07	91	1	2	1
4	2015-10-08	99	4	3	1

Выбрав из этого набора строки, соответствующие первому визиту читателя и первой выдаче книги в рамках этого визита (строки 36-37 запроса), мы получаем:

sb_subscriber	sb_start	sb_id	sb_book	rank_by_subscriber	rank_by_date
1	2011-01-12	2	1	1	1
3	2012-05-17	3	3	1	1
4	2012-06-11	57	5	1	1

Теперь остаётся по идентификаторам читателе и книг получить их имена и названия (строки 2-4 и 32-35 запроса). Так получается итоговый результат.

Этот запрос можно оптимизировать, выбирая меньше полей и проводя меньше проверок, но тогда он станет сложнее для понимания.

Решение для MS SQL Server и Oracle реализуется проще, и даже допускает ещё один — третий — вариант, т.к. эти СУБД поддерживают ранжирующие (оконные) функции. И всё же первый вариант решения по соображениям совместимости будет реализован без ранжирования.

MS SQL Решение 2.2.9.d

```
1   -- Вариант 1: решение в четыре шага без использования ранжирования
2   WITH [step_1]
3        AS (SELECT [sb_subscriber],
4                   MIN([sb_start]) AS [min_sb_start]
5            FROM   [subscriptions]
6            GROUP  BY [sb_subscriber]),
7        [step_2]
8        AS (SELECT [subscriptions].[sb_subscriber],
9                   MIN([sb_id]) AS [min_sb_id]
10           FROM   [subscriptions]
11                  JOIN [step_1]
12                    ON [subscriptions].[sb_subscriber] =
13                               [step_1].[sb_subscriber]
14                       AND [subscriptions].[sb_start] =
15                               [step_1].[min_sb_start]
16           GROUP  BY [subscriptions].[sb_subscriber],
17                     [min_sb_start]),
18       [step_3]
19       AS (SELECT [subscriptions].[sb_subscriber],
20                  [sb_book]
21           FROM   [subscriptions]
22                  JOIN [step_2]
23                    ON [subscriptions].[sb_id] = [step_2].[min_sb_id])
24  SELECT [s_id],
25         [s_name],
26         [b_name]
27  FROM   [step_3]
28         JOIN [subscribers]
29           ON [sb_subscriber] = [s_id]
30         JOIN [books]
31           ON [sb_book] = [b_id]
```

Здесь общие табличные выражения возвращают те же наборы данных, что и подзапросы с соответствующими именами в MySQL.

Общее табличное выражение **step_1** (строки 2-6 запроса) возвращает информацию о дате первого визита в библиотеку каждого читателя:

sb_subscriber	min_sb_start
1	2011-01-12
3	2012-05-17
4	2012-06-11

Общее табличное выражение **step_2** (строки 7-17 запроса) возвращает минимальное значение первичного ключа таблицы **subscribers**, соответствующее каждому читателю и дате его первого визита в библиотеку:

sb_subscriber	min_sb_id
1	2
3	3
4	57

Общее табличное выражение **step_3** (строки 18-23 запроса) возвращает идентификатор книги, соответствующий каждой выдаче с идентификатором **min_sb_id**:

sb_subscriber	sb_book
1	1
3	3
4	5

Основная часть запроса в строках 24-31 является четвёртым шагом, на котором по имеющимся идентификаторам определяются имя читателя и название книги. Так получается финальный результат.

MS SQL Решение 2.2.9.d

```
1   -- Вариант 2: решение в два шага с использованием ранжирования
2   WITH [step_1]
3        AS (SELECT [sb_subscriber],
4                   [sb_start],
5                   [sb_id],
6                  [sb_book],
7                   ROW_NUMBER()
8                     OVER (
9                       PARTITION BY [sb_subscriber]
10                      ORDER BY [sb_subscriber] ASC)
11                  AS [rank_by_subscriber],
12                  ROW_NUMBER()
13                    OVER (
14                      PARTITION BY [sb_subscriber], [sb_start]
15                      ORDER BY [sb_subscriber], [sb_start] ASC)
16                  AS [rank_by_date]
17            FROM   [subscriptions])
18  SELECT [s_id],
19         [s_name],
20         [b_name]
21  FROM   [step_1]
22         JOIN [subscribers]
23           ON [sb_subscriber] = [s_id]
24         JOIN [books]
25           ON [sb_book] = [b_id]
26  WHERE [rank_by_subscriber] = 1 AND [rank_by_date] = 1
```

Общее табличное выражение в строках 2-17 возвращает те же данные, что и подзапрос в решении для MySQL — информацию о выдаче читателям книг, ранжированную по номеру визита читателя в библиотеку и номеру выдачи книги в рамках каждого визита:

sb_subscriber	sb_start	sb_id	sb_book	rank_by_subscriber	rank_by_date
1	2011-01-12	2	1	1	1
1	2011-01-12	100	3	2	2
1	2012-06-11	42	2	3	1
1	2014-08-03	61	7	4	1
1	2015-10-07	95	4	5	1
3	2012-05-17	3	3	1	1
3	2014-08-03	62	5	2	1
3	2014-08-03	86	1	3	2
4	2012-06-11	57	5	1	1
4	2015-10-07	91	1	2	1
4	2015-10-08	99	4	3	1

Основная часть запроса в строках 18-26 оставляет из этого набора только каждую первую выдачу в каждом первом визите и получает по идентификаторам читателей и книг их имена и названия. Так получается финальный результат.

В MySQL бессмысленно было реализовывать третий вариант решения, т.к. группировка там нарушает логику нумерации рядов. В MS SQL Server и Oracle этого ограничения нет, потому возможно ещё одно решение — третий вариант.

MS SQL Решение 2.2.9.d

```
1   -- Вариант 3: решение в три шага с использованием ранжирования
2   -- и группировки
3   WITH [step_1]
4        AS (SELECT [sb_subscriber],
5                   [sb_start],
6                   MIN([sb_id])                AS [min_sb_id],
7                   RANK()
8                     OVER (
9                       PARTITION BY [sb_subscriber]
10                      ORDER BY [sb_start] ASC) AS [rank]
11           FROM   [subscriptions]
12           GROUP  BY [sb_subscriber],
13                     [sb_start]),
14       [step_2]
15       AS (SELECT [subscriptions].[sb_subscriber],
16                  [subscriptions].[sb_book]
17           FROM   [subscriptions]
18                  JOIN [step_1]
19                    ON [subscriptions].[sb_id] = [step_1].[min_sb_id]
20           WHERE  [rank] = 1)
21  SELECT [s_id],
22         [s_name],
23         [b_name]
24  FROM   [step_2]
25         JOIN [subscribers]
26           ON [sb_subscriber] = [s_id]
27         JOIN [books]
28           ON [sb_book] = [b_id]
```

Первое общее табличное выражение (строки 3-13 запроса) сразу выясняет наименьшее значение первичного ключа таблицы **subscriptions** и ранжирует полученные данные по номеру визита читателя в библиотеку:

sb_subscriber	sb_start	min_sb_id	rank
1	2011-01-12	2	1
1	2012-06-11	42	2
1	2014-08-03	61	3
1	2015-10-07	95	4
3	2012-05-17	3	1
3	2014-08-03	62	2
4	2012-06-11	57	1
4	2015-10-07	91	2
4	2015-10-08	99	3

Второе общее табличное выражение убирает лишние данные и, обладая информацией об идентификаторе записи из таблицы **subscriptions**, извлекает оттуда значение поля **sb_book**, т.е. идентификатор книги. В результате получается:

sb_subscriber	sb_book
1	1
3	3
4	5

Основная часть запроса в строках 21-28 нужна только для замены идентификаторов читателей и книг на их имена и названия. Так получается итоговый результат.

Решение для Oracle аналогично решению для MS SQL Server и отличается лишь незначительными синтаксическими деталями.

Oracle Решение 2.2.9.d

```
1   -- Вариант 1: решение в четыре шага без использования ранжирования
2   WITH "step_1"
3        AS (SELECT "sb_subscriber",
4                   MIN("sb_start") AS "min_sb_start"
5            FROM   "subscriptions"
6            GROUP  BY "sb_subscriber"),
7        "step_2"
8        AS (SELECT "subscriptions"."sb_subscriber",
9                   MIN("sb_id") AS "min_sb_id"
10           FROM   "subscriptions"
11                  JOIN "step_1"
12                    ON "subscriptions"."sb_subscriber" =
13                               "step_1"."sb_subscriber"
14                       AND "subscriptions"."sb_start" =
15                               "step_1"."min_sb_start"
16           GROUP  BY "subscriptions"."sb_subscriber",
17                     "min_sb_start"),
18       "step_3"
19       AS (SELECT "subscriptions"."sb_subscriber",
20                  "sb_book"
21           FROM   "subscriptions"
22                  JOIN "step_2"
23                    ON "subscriptions"."sb_id" = "step_2"."min_sb_id")
24  SELECT "s_id",
25         "s_name",
26         "b_name"
27  FROM   "step_3"
28         JOIN "subscribers"
29           ON "sb_subscriber" = "s_id"
30         JOIN "books"
31           ON "sb_book" = "b_id"
```

Oracle Решение 2.2.9.d (продолжение)

```
-- Вариант 2: решение в два шага с использованием ранжирования
WITH "step_1"
     AS (SELECT "sb_subscriber",
                "sb_start",
                "sb_id",
                       "sb_book",
                ROW_NUMBER()
                  OVER (
                    PARTITION BY "sb_subscriber"
                    ORDER BY "sb_subscriber" ASC)
                AS "rank_by_subscriber",
                ROW_NUMBER()
                  OVER (
                    PARTITION BY "sb_subscriber", "sb_start"
                    ORDER BY "sb_subscriber", "sb_start" ASC)
                AS "rank_by_date"
         FROM   "subscriptions")
SELECT "s_id",
       "s_name",
       "b_name"
FROM   "step_1"
       JOIN "subscribers"
         ON "sb_subscriber" = "s_id"
       JOIN "books"
         ON "sb_book" = "b_id"
WHERE "rank_by_subscriber" = 1 AND "rank_by_date" = 1
-- Вариант 3: решение в три шага с использованием ранжирования
-- и группировки
WITH "step_1"
     AS (SELECT "sb_subscriber",
                "sb_start",
                MIN("sb_id")                  AS "min_sb_id",
                RANK()
                  OVER (
                    PARTITION BY "sb_subscriber"
                    ORDER BY "sb_start" ASC) AS "rank"
         FROM   "subscriptions"
         GROUP  BY "sb_subscriber",
                   "sb_start"),
     "step_2"
     AS (SELECT "subscriptions"."sb_subscriber",
                "subscriptions"."sb_book"
         FROM   "subscriptions"
                JOIN "step_1"
                  ON "subscriptions"."sb_id" = "step_1"."min_sb_id"
         WHERE  "rank" = 1)
SELECT "s_id",
       "s_name",
       "b_name"
FROM   "step_2"
       JOIN "subscribers"
         ON "sb_subscriber" = "s_id"
       JOIN "books"
         ON "sb_book" = "b_id"
```

Исследование 2.2.9.EXP.C: сравним скорость работы решений этой задачи, выполнив запросы 2.2.9.d на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений каждого запроса:

	MySQL	MS SQL Server	Oracle
Вариант 1	87135.510	31.158	2.281
Вариант 2	62.524	35.226	1.040
Вариант 3	-	39.339	1.250

Обратите внимание на разницу в скорости работы первого и второго вариантов для MySQL, а также насколько Oracle превосходит конкурентов в решении задач такого класса.

Задание 2.2.9.TSK.A: показать читателя, последним взявшего в библиотеке книгу.

Задание 2.2.9.TSK.B: показать читателя (или читателей, если их окажется несколько), дольше всего держащего у себя книгу (учитывать только случаи, когда книга не возвращена).

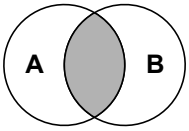
Задание 2.2.9.TSK.C: показать, какую книгу (или книги, если их несколько) каждый читатель взял в свой последний визит в библиотеку.

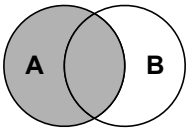
Задание 2.2.9.TSK.D: показать последнюю книгу, которую каждый из читателей взял в библиотеке.

2.2.10. ПРИМЕР 20: ВСЕ РАЗНОВИДНОСТИ ЗАПРОСОВ НА ОБЪЕДИНЕНИЕ В ТРЁХ СУБД

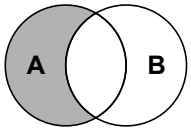
Для начала напомним, какие результаты можно получить, объединяя данные из двух таблиц с помощью классических вариантов `JOIN`[3]:

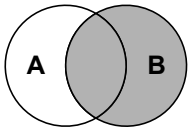
Вид объединения	Графическое представление	Псевдокод запроса

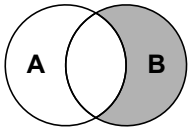
Внутреннее объединение		`SELECT поля`
		`FROM A INNER JOIN B`
		`ON A.поле = B.поле`

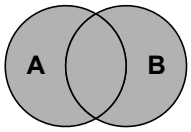
Левое внешнее объединение		`SELECT поля`
		`FROM A LEFT OUTER JOIN B`
		`ON A.поле = B.поле`

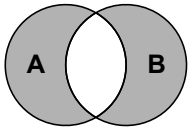
[3] Оригинальный рисунок: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

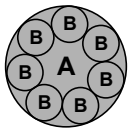
Левое внешнее объединение с исключением		`SELECT поля`
		`FROM A LEFT OUTER JOIN B`
		`ON A.поле = B.поле`
		`WHERE B.поле IS NULL`

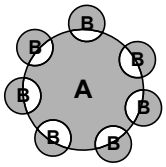
Правое внешнее объединение		`SELECT поля`
		`FROM A RIGHT OUTER JOIN B`
		`ON A.поле = B.поле`

Правое внешнее объединение с исключением		`SELECT поля`
		`FROM A RIGHT OUTER JOIN B`
		`ON A.поле = B.поле`
		`WHERE B.поле IS NULL`

Полное внешнее объединение		`SELECT поля`
		`FROM A FULL OUTER JOIN B`
		`ON A.поле = B.поле`

Полное внешнее объединение с исключением		`SELECT поля`
		`FROM A FULL OUTER JOIN B`
		`ON A.поле = B.поле`
		`WHERE A.поле IS NULL`
		`OR B.поле IS NULL`

Перекрёстное объединение (декартово произведение): попарная комбинация всех записей		`SELECT поля`
		`FROM A CROSS JOIN B`
		или
		`SELECT поля`
		`FROM A, B`

Перекрёстное объединение (декартово произведение) с исключением: попарная комбинация всех записей, за исключением тех, что имеют пары по указанному полю)		`SELECT поля`
		`FROM A CROSS JOIN B`
		`WHERE A.поле != B.поле`
		или
		`SELECT поля`
		`FROM A, B`
		`WHERE A.поле != B.поле`

Для демонстрации конкретных примеров создадим в БД «Исследование» таблицы **rooms** и **computers**, связанные связью «один ко многим»:

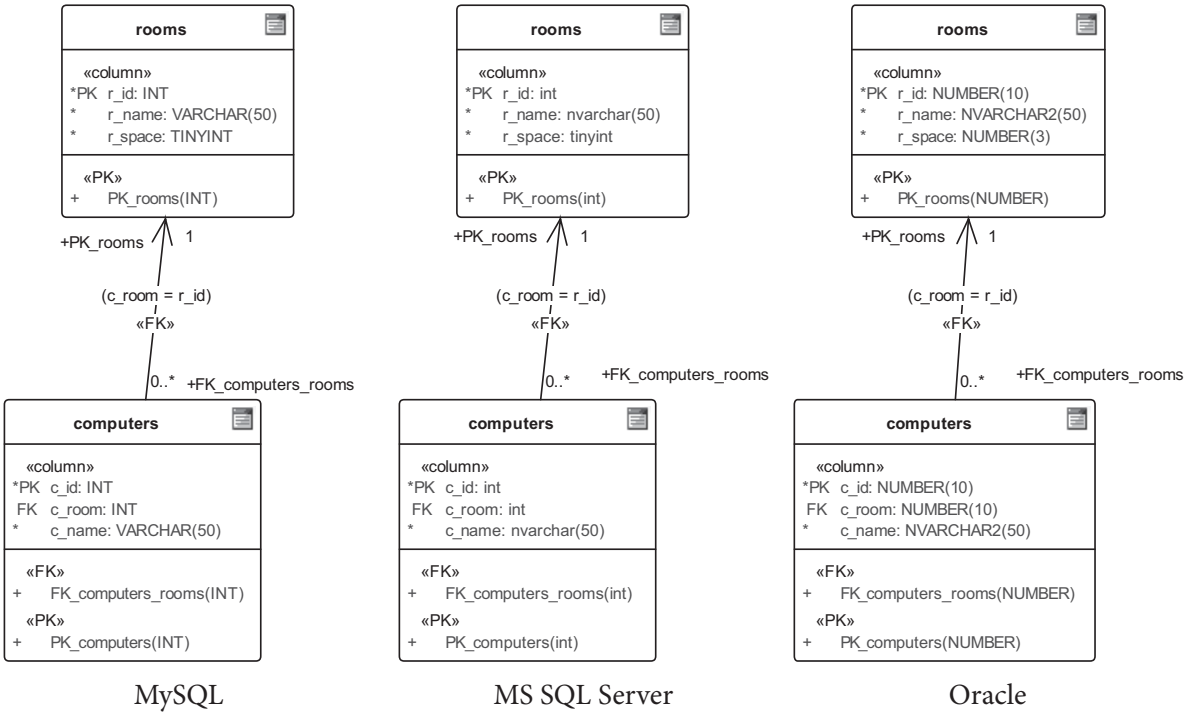


*Рисунок 2.2.а — **Таблицы rooms и computers в трёх СУБД***

Поместим в таблицу **rooms** следующие данные:

r_id	r_name	r_space
1	Комната с двумя компьютерами	5
2	Комната с тремя компьютерами	5
3	Пустая комната 1	2
4	Пустая комната 2	2
5	Пустая комната 3	2

Поместим в таблицу **computers** следующие данные:

c_id	c_room	c_name
1	1	Компьютер A в комнате 1
2	1	Компьютер B в комнате 1
3	2	Компьютер A в комнате 2
4	2	Компьютер B в комнате 2
5	2	Компьютер C в комнате 2
6	NULL	Свободный компьютер A
7	NULL	Свободный компьютер B
8	NULL	Свободный компьютер C

Сразу отметим, что слова **INNER** и **OUTER** в подавляющем большинстве случаев являются т.н. «синтаксическим сахаром» (т.е. добавлены для удобства человека, при этом никак не влияя на выполнения запроса) и потому являются необязательными: «просто **JOIN**» всегда внутренний, а **LEFT JOIN**, **RIGHT JOIN** и **FULL JOIN** всегда внешние.

В этом примере очень много задач, и в них легко запутаться, потому сначала мы перечислим их все, а затем разместим условия, ожидаемые результаты и решения вместе.

Задачи на «классическое объединение»:

- 2.2.10.a{157}: показать информацию о том, как компьютеры распределены по комнатам;
- 2.2.10.b{158}: показать все комнаты с поставленными в них компьютерами;
- 2.2.10.c{159}: показать все пустые комнаты;
- 2.2.10.d{160}: показать все компьютеры с информацией о том, в каких они расположены комнатах;
- 2.2.10.e{161}: показать все свободные компьютеры;
- 2.2.10.f{163}: показать всю информацию о том, как компьютеры размещены по комнатам (включая пустые комнаты и свободные компьютеры);
- 2.2.10.g{165}: показать информацию по всем пустым комнатам и свободным компьютерам;
- 2.2.10.h{168}: показать возможные варианты расстановки компьютеров по комнатам (не учитывать вместимость комнат);
- 2.2.10.i{170}: показать возможные варианты перестановки компьютеров по комнатам (компьютер не должен оказаться в той комнате, в которой он сейчас стоит, не учитывать вместимость комнат).

Задачи на «неклассическое объединение»:

- 2.2.10.j{172}: показать возможные варианты расстановки компьютеров по комнатам (учитывать вместимость комнат);
- 2.2.10.k{174}: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (не учитывать вместимость комнат);
- 2.2.10.l{176}: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (учитывать вместимость комнат);
- 2.2.10.m{179}: показать возможные варианты расстановки свободных компьютеров по комнатам (учитывать остаточную вместимость комнат);
- 2.2.10.n{181}: показать расстановку компьютеров по непустым комнатам так, чтобы в выборку не попало больше компьютеров, чем может поместиться в комнату;
- 2.2.10.o{185}: показать расстановку компьютеров по всем комнатам так, чтобы в выборку не попало больше компьютеров, чем может поместиться в комнату.

Задачи на «классическое объединение» предполагают решение на основе прямого использования предоставляемого СУБД синтаксиса (без подзапросов и прочих ухищрений).

Задачи на «неклассическое объединение» предполагают решение на основе либо специфичного для той или иной СУБД синтаксиса, либо дополнительных действий (как правило — подзапросов).

Задача 2.2.10.a: показать информацию о том, как компьютеры распределены по комнатам.

Ожидаемый результат 2.2.10.a.

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2

Решение 2.2.10.a: используем внутреннее объединение.

MySQL Решение 2.2.10.a

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       JOIN `computers`
         ON `r_id` = `c_room`
```

MS SQL Решение 2.2.10.a

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       JOIN [computers]
         ON [r_id] = [c_room]
```

Oracle Решение 2.2.10.a

```
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       JOIN "computers"
         ON "r_id" = "c_room"
```

Логика внутреннего объединения состоит в том, чтобы подобрать из двух таблиц пары записей, у которых совпадает значение поля, по которому происходит объединение. Для наглядности ещё раз рассмотрим ожидаемый результат и поместим рядом поля, по которым происходит объединение:

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2

Задача 2.2.10.b: показать все комнаты с поставленными в них компьютерами.

Ожидаемый результат 2.2.10.b.

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL

Решение 2.2.10.b: используем левое внешнее объединение. В этом решении нужно показать все записи из таблицы **rooms** — как те, для которых есть соответствие в таблице **computers**, так и те, для которых такого соответствия нет.

MySQL Решение 2.2.10.b

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       LEFT JOIN `computers`
              ON `r_id` = `c_room`
```

MS SQL Решение 2.2.10.b

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       LEFT JOIN [computers]
              ON [r_id] = [c_room]
```

Oracle Решение 2.2.10.b

```
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       LEFT JOIN "computers"
              ON "r_id" = "c_room"
```

В случае левого внешнего объединения СУБД извлекает **все** записи из левой таблицы и пытается найти им пару из правой таблицы. Если пары не находится, соответствующая часть записи в итоговой таблице заполняется **NULL**-значениями.

Модифицируем ожидаемый результат так, чтобы эта идея была более наглядной:

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	NULL	NULL
Пустая комната 2	4	NULL	NULL	NULL
Пустая комната 3	5	NULL	NULL	NULL

Задача 2.2.10.c: показать все пустые комнаты.

Ожидаемый результат 2.2.10.c.

r_id	r_name	c_id	c_room	c_name
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL

Решение 2.2.10.c: используем левое внешнее объединение с исключением, т.е. выберем только те записи из таблицы **`rooms`**, для которых нет соответствия в таблице **`computers`**.

MySQL Решение 2.2.10.c

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       LEFT JOIN `computers`
              ON `r_id` = `c_room`
WHERE  `c_room` IS NULL
```

MS SQL Решение 2.2.10.c

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       LEFT JOIN [computers]
              ON [r_id] = [c_room]
WHERE  [c_room] IS NULL
```

Oracle Решение 2.2.10.c

```
1  SELECT "r_id",
2         "r_name",
3         "c_id",
4         "c_room",
5         "c_name"
6  FROM   "rooms"
7         LEFT JOIN "computers"
8                ON "r_id" = "c_room"
9  WHERE  "c_room" IS NULL
```

Благодаря условию в 9-й строке каждого запроса из набора данных, эквивалентного получаемому в предыдущей задаче (2.2.10.b) в конечную выборку проходят только строки со значением **NULL** в поле **c_room**:

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	NULL	NULL
Пустая комната 2	4	NULL	NULL	NULL
Пустая комната 3	5	NULL	NULL	NULL

Задача 2.2.10.d: показать все компьютеры с информацией о том, в каких они расположены комнатах.

Ожидаемый результат 2.2.10.d.

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Решение 2.2.10.d: используем правое внешнее объединение. Эта задача обратна задаче 2.2.10.b[157]: здесь нужно показать все записи из таблицы **computers** вне зависимости от того, есть ли им соответствие из таблицы **rooms**.

MySQL Решение 2.2.10.d

```
1  SELECT `r_id`,
2         `r_name`,
3         `c_id`,
4         `c_room`,
5         `c_name`
6  FROM   `rooms`
7         RIGHT JOIN `computers`
8                ON `r_id` = `c_room`
```

MS SQL Решение 2.2.10.d

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       RIGHT JOIN [computers]
               ON [r_id] = [c_room]
```

Oracle Решение 2.2.10.d

```
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       RIGHT JOIN "computers"
               ON "r_id" = "c_room"
```

В случае правого внешнего объединения СУБД извлекает **все** записи из правой таблицы и пытается найти им пару из левой таблицы. Если пары не находится, соответствующая часть записи в итоговой таблице заполняется `NULL`-значениями.

Модифицируем ожидаемый результат так, чтобы эта идея была более наглядной:

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
NULL	NULL	NULL	6	Свободный компьютер A
NULL	NULL	NULL	7	Свободный компьютер B
NULL	NULL	NULL	8	Свободный компьютер C

Задача 2.2.10.e: показать все свободные компьютеры.

Ожидаемый результат 2.2.10.e.

r_id	r_name	c_id	c_room	c_name
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Решение 2.2.10.e: используем правое внешнее объединение с исключением. Эта задача обратна задаче 2.2.10.c: здесь мы выберем только те записи из таблицы `computers`, для которых нет соответствия в таблице `rooms`.

MySQL Решение 2.2.10.e

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       RIGHT JOIN `computers`
               ON `r_id` = `c_room`
WHERE  `r_id` IS NULL
```

MS SQL Решение 2.2.10.e

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       RIGHT JOIN [computers]
               ON [r_id] = [c_room]
WHERE  [r_id] IS NULL
```

Oracle Решение 2.2.10.e

```
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       RIGHT JOIN "computers"
               ON "r_id" = "c_room"
WHERE  "r_id" IS NULL
```

Аналогичный же результат (как правило, в таких задачах нас не интересуют поля из родительской таблицы, т.к. там по определению будет **NULL**) можно получить и без **JOIN**. Такой способ срабатывает, когда источником информации является дочерняя таблица, но задачу 2.2.10.c{159} таким тривиальным способом решить не получится (там понадобилось выполнять подзапрос с конструкцией **NOT IN**):

c_id	c_room	c_name
6	NULL	Свободный компьютер A
7	NULL	Свободный компьютер B
8	NULL	Свободный компьютер C

MySQL Решение 2.2.10.e (упрощённый вариант)

```
SELECT `c_id`,
       `c_room`,
       `c_name`
FROM   `computers`
WHERE  `c_room` IS NULL
```

MS SQL Решение 2.2.10.e (упрощённый вариант)

```
SELECT [c_id],
       [c_room],
       [c_name]
FROM   [computers]
WHERE  [c_room] IS NULL
```

Oracle Решение 2.2.10.e (упрощённый вариант)

```
SELECT "c_id",
       "c_room",
       "c_name"
FROM   "computers"
WHERE  "c_room" IS NULL
```

Задача 2.2.10.f: показать всю информацию о том, как компьютеры размещены по комнатам (включая пустые комнаты и свободные компьютеры).

Ожидаемый результат 2.2.10.f.

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Решение 2.2.10.f: используем полное внешнее объединение. Эта задача является комбинацией задач 2.2.10.b{157} и 2.2.10.d{160}: нужно показать все записи из таблицы **rooms** вне зависимости от наличия соответствия в таблице **computers**, а также все записи из таблицы **computers** вне зависимости от наличия соответствия в таблице **rooms**.

Важно! MySQL не поддерживает полное внешнее объединение, потому использование там **FULL JOIN** даёт неверный результат.

MySQL Решение 2.2.10.f (ошибочный запрос)

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       FULL JOIN `computers`
              ON `r_id` = `c_room`
```

В результате выполнения такого запроса получается тот же набор данных, что и в задаче 2.2.10.a{156}:

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2

Самым простым[4] решением этой задачи для MySQL является объединение решений задач 2.2.10.b{157} и 2.2.10.d{160} с помощью конструкции **UNION**.

MySQL Решение 2.2.10.f

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       LEFT JOIN `computers`
              ON `r_id` = `c_room`
UNION
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       RIGHT JOIN `computers`
               ON `r_id` = `c_room`
```

MS SQL Server и Oracle поддерживают полное внешнее объединение, и там эта задача решается намного проще:

MS SQL Решение 2.2.10.f

```
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       FULL JOIN [computers]
              ON [r_id] = [c_room]
```

[4] Несколько альтернативных решений рассмотрено в этой статье: http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/

Oracle Решение 2.2.10.f

```
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       FULL JOIN "computers"
              ON "r_id" = "c_room"
```

При выполнении полного внешнего объединения СУБД извлекает **все** записи из **обеих** таблиц и ищет их пары. Там, где пары находятся, в итоговой выборке получается строка с данными из обеих таблиц. Там, где пары нет, недостающие данные заполняются `NULL`-значениями. Покажем это графически:

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	NULL	NULL
Пустая комната 2	4	NULL	NULL	NULL
Пустая комната 3	5	NULL	NULL	NULL
NULL	NULL	NULL	6	Свободный компьютер A
NULL	NULL	NULL	7	Свободный компьютер B
NULL	NULL	NULL	8	Свободный компьютер C

Задача 2.2.10.g: показать информацию по всем пустым комнатам и свободным компьютерам.

Ожидаемый результат 2.2.10.g.

r_id	r_name	c_id	c_room	c_name
3	Пустая комната 1	NULL	NULL	NULL
4	Пустая комната 2	NULL	NULL	NULL
5	Пустая комната 3	NULL	NULL	NULL
NULL	NULL	6	NULL	Свободный компьютер A
NULL	NULL	7	NULL	Свободный компьютер B
NULL	NULL	8	NULL	Свободный компьютер C

Решение 2.2.10.g: используем полное внешнее объединение с исключением. Эта задача является комбинацией задач 2.2.10.c{159} и 2.2.10.e{161}: нужно показать все записи из таблицы `rooms`, для которых нет соответствия в таблице `computers`, а также все записи из таблицы `computers`, для которых нет соответствия в таблице `rooms`.

С MySQL здесь та же проблема, что и в предыдущей задаче — отсутствие поддержки полного внешнего объединения, что вынуждает нас опять использовать два отдельных запроса, результаты которых объединяются с помощью `UNION`:

MySQL Решение 2.2.10.g

```
1  SELECT `r_id`,
2         `r_name`,
3         `c_id`,
4         `c_room`,
5         `c_name`
6  FROM   `rooms`
7         LEFT JOIN `computers`
8                ON `r_id` = `c_room`
9  WHERE  `c_id` IS NULL
10 UNION
11 SELECT `r_id`,
12        `r_name`,
13        `c_id`,
14        `c_room`,
15        `c_name`
16 FROM   `rooms`
17        RIGHT JOIN `computers`
18                ON `r_id` = `c_room`
19 WHERE  `r_id` IS NULL
```

В MS SQL Server и Oracle всё проще: достаточно указать, в каких полях мы ожидаем наличие **NULL**.

MS SQL Решение 2.2.10.g

```
1  SELECT [r_id],
2         [r_name],
3         [c_id],
4         [c_room],
5         [c_name]
6  FROM   [rooms]
7         FULL JOIN [computers]
8                ON [r_id] = [c_room]
9  WHERE      [r_id] IS NULL
10          OR [c_id] IS NULL
```

Oracle Решение 2.2.10.g

```
1  SELECT "r_id",
2         "r_name",
3         "c_id",
4         "c_room",
5         "c_name"
6  FROM   "rooms"
7         FULL JOIN "computers"
8                ON "r_id" = "c_room"
9  WHERE      "r_id" IS NULL
10          OR "c_id" IS NULL
```

Условия в строках 9-10 запросов для MS SQL Server и Oracle не допускают попадания в конечную выборку строк, отличных от имеющих **NULL**-значение в полях **r_id** или **c_id**. Эти поля выбраны не случайно: они являются первичными ключами своих таблиц, и потому появление в них **NULL**-значения, изначально записанного в таблицу крайне маловероятно в отличие от «обычных полей», где **NULL** вполне может храниться как признак отсутствия значения.

Графически набор попадающих в конечную выборку записей выглядит так (отмечено серым фоном):

r_name	r_id	c_room	c_id	c_name
Комната с двумя компьютерами	1	1	1	Компьютер A в комнате 1
Комната с двумя компьютерами	1	1	2	Компьютер B в комнате 1
Комната с тремя компьютерами	2	2	3	Компьютер A в комнате 2
Комната с тремя компьютерами	2	2	4	Компьютер B в комнате 2
Комната с тремя компьютерами	2	2	5	Компьютер C в комнате 2
Пустая комната 1	3	NULL	**NULL**	NULL
Пустая комната 2	4	NULL	**NULL**	NULL
Пустая комната 3	5	NULL	**NULL**	NULL
NULL	**NULL**	NULL	6	Свободный компьютер A
NULL	**NULL**	NULL	7	Свободный компьютер B
NULL	**NULL**	NULL	8	Свободный компьютер C

Задача 2.2.10.h: показать возможные варианты расстановки компьютеров по комнатам (не учитывать вместимость комнат).

Ожидаемый результат 2.2.10.h.

r_id	r_name	c_id	c_room	c_name
1	Комната с двумя компьютерами	1	1	Компьютер A в комнате 1
2	Комната с тремя компьютерами	1	1	Компьютер A в комнате 1
3	Пустая комната 1	1	1	Компьютер A в комнате 1
4	Пустая комната 2	1	1	Компьютер A в комнате 1
5	Пустая комната 3	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	2	1	Компьютер B в комнате 1
3	Пустая комната 1	2	1	Компьютер B в комнате 1
4	Пустая комната 2	2	1	Компьютер B в комнате 1
5	Пустая комната 3	2	1	Компьютер B в комнате 1
1	Комната с двумя компьютерами	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	3	2	Компьютер A в комнате 2
3	Пустая комната 1	3	2	Компьютер A в комнате 2
4	Пустая комната 2	3	2	Компьютер A в комнате 2
5	Пустая комната 3	3	2	Компьютер A в комнате 2
1	Комната с двумя компьютерами	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	4	2	Компьютер B в комнате 2
3	Пустая комната 1	4	2	Компьютер B в комнате 2
4	Пустая комната 2	4	2	Компьютер B в комнате 2
5	Пустая комната 3	4	2	Компьютер B в комнате 2
1	Комната с двумя компьютерами	5	2	Компьютер C в комнате 2
2	Комната с тремя компьютерами	5	2	Компьютер C в комнате 2
3	Пустая комната 1	5	2	Компьютер C в комнате 2
4	Пустая комната 2	5	2	Компьютер C в комнате 2
5	Пустая комната 3	5	2	Компьютер C в комнате 2
1	Комната с двумя компьютерами	6	NULL	Свободный компьютер A

(продолжение)

2	Комната с тремя компьютерами	6	NULL	Свободный компьютер A
3	Пустая комната 1	6	NULL	Свободный компьютер A
4	Пустая комната 2	6	NULL	Свободный компьютер A
5	Пустая комната 3	6	NULL	Свободный компьютер A
1	Комната с двумя компьютерами	7	NULL	Свободный компьютер B
2	Комната с тремя компьютерами	7	NULL	Свободный компьютер B
3	Пустая комната 1	7	NULL	Свободный компьютер B
4	Пустая комната 2	7	NULL	Свободный компьютер B
5	Пустая комната 3	7	NULL	Свободный компьютер B
1	Комната с двумя компьютерами	8	NULL	Свободный компьютер C
2	Комната с тремя компьютерами	8	NULL	Свободный компьютер C
3	Пустая комната 1	8	NULL	Свободный компьютер C
4	Пустая комната 2	8	NULL	Свободный компьютер C
5	Пустая комната 3	8	NULL	Свободный компьютер C

Решение 2.2.10.h: используем перекрёстное объединение (декартово произведение).

MySQL Решение 2.2.10.h

```
1  -- Вариант 1: без ключевого слова JOIN
2  SELECT `r_id`,
3         `r_name`,
4         `c_id`,
5         `c_room`,
6         `c_name`
7  FROM   `rooms`,
8         `computers`

1  -- Вариант 2: с ключевым словом JOIN
2  SELECT `r_id`,
3         `r_name`,
4         `c_id`,
5         `c_room`,
6         `c_name`
7  FROM   `rooms`
8         CROSS JOIN `computers`
```

MS SQL Решение 2.2.10.h

```
1  -- Вариант 1: без ключевого слова JOIN
2  SELECT [r_id],
3         [r_name],
4         [c_id],
5         [c_room],
6         [c_name]
7  FROM   [rooms],
8         [computers]
```

MS SQL Решение 2.2.10.h (продолжение)

```
-- Вариант 2: с ключевым словом JOIN
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       CROSS JOIN [computers]
```

Oracle Решение 2.2.10.h

```
-- Вариант 1: без ключевого слова JOIN
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms",
       "computers"

-- Вариант 2: с ключевым словом JOIN
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       CROSS JOIN "computers"
```

При выполнении перекрёстного объединения (декартового произведения) СУБД каждой записи из левой таблицы ставит в соответствие все записи из правой таблицы. Иными словами, СУБД находит все возможные попарные комбинации записей из обеих таблиц.

Задача 2.2.10.i: показать возможные варианты перестановки компьютеров по комнатам (компьютер не должен оказаться в той комнате, в которой он сейчас стоит, не учитывать вместимость комнат).

Ожидаемый результат 2.2.10.i.

r_id	r_name	c_id	c_room	c_name
2	Комната с тремя компьютерами	1	1	Компьютер A в комнате 1
3	Пустая комната 1	1	1	Компьютер A в комнате 1
4	Пустая комната 2	1	1	Компьютер A в комнате 1
5	Пустая комната 3	1	1	Компьютер A в комнате 1
2	Комната с тремя компьютерами	2	1	Компьютер B в комнате 1
3	Пустая комната 1	2	1	Компьютер B в комнате 1
4	Пустая комната 2	2	1	Компьютер B в комнате 1
5	Пустая комната 3	2	1	Компьютер B в комнате 1
1	Комната с двумя компьютерами	3	2	Компьютер A в комнате 2

(продолжение)

3	Пустая комната 1	3	2	Компьютер A в комнате 2
4	Пустая комната 2	3	2	Компьютер A в комнате 2
5	Пустая комната 3	3	2	Компьютер A в комнате 2
1	Комната с двумя компьютерами	4	2	Компьютер B в комнате 2
3	Пустая комната 1	4	2	Компьютер B в комнате 2
4	Пустая комната 2	4	2	Компьютер B в комнате 2
5	Пустая комната 3	4	2	Компьютер B в комнате 2
1	Комната с двумя компьютерами	5	2	Компьютер C в комнате 2
3	Пустая комната 1	5	2	Компьютер C в комнате 2
4	Пустая комната 2	5	2	Компьютер C в комнате 2
5	Пустая комната 3	5	2	Компьютер C в комнате 2

Решение 2.2.10.i: используем перекрёстное объединение (декартово произведение) с исключением.

MySQL Решение 2.2.10.i

```
-- Вариант 1: без ключевого слова JOIN
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`,
       `computers`
WHERE  `r_id` != `c_room`

-- Вариант 2: с ключевым словом JOIN
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   `rooms`
       CROSS JOIN `computers`
WHERE  `r_id` != `c_room`
```

MS SQL Решение 2.2.10.i

```
-- Вариант 1: без ключевого слова JOIN
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms],
       [computers]
WHERE  [r_id] != [c_room]
```

MS SQL Решение 2.2.10.i (продолжение)

```
-- Вариант 2: с ключевым словом JOIN
SELECT [r_id],
       [r_name],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       CROSS JOIN [computers]
WHERE  [r_id] != [c_room]
```

Oracle Решение 2.2.10.i

```
-- Вариант 1: без ключевого слова JOIN
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms",
       "computers"
WHERE  "r_id" != "c_room"

-- Вариант 2: с ключевым словом JOIN
SELECT "r_id",
       "r_name",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       CROSS JOIN "computers"
WHERE  "r_id" != "c_room"
```

При выполнении декартового произведения с исключением СУБД не допускает в результирующую выборку реально существующие пары записей из обеих таблиц, т.е. получает все возможные попарные комбинации кроме тех, которые реально существуют.

На этом с классическими вариантами объединений — всё.

Задачи на «неклассическое объединение» предполагают решение на основе либо специфичного для той или иной СУБД синтаксиса, либо дополнительных действий (как правило — подзапросов).

Многие задачи в этом подразделе обязаны своим возникновением существованию в MS SQL Server и Oracle (начиная с версии 12c) операторов **`CROSS APPLY`** и **`OUTER APPLY`**. Потому здесь и далее решение для MS SQL Server будет первичным, а решения для MySQL и Oracle будут построены через эмуляцию соответствующего поведения.

Задача 2.2.10.j: показать возможные варианты расстановки компьютеров по комнатам (учитывать вместимость комнат).

Ожидаемый результат 2.2.10.j.

r_id	r_name	r_space	c_id	c_room	c_name
1	Комната с двумя компьютерами	5	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	5	2	1	Компьютер B в комнате 1
1	Комната с двумя компьютерами	5	3	2	Компьютер A в комнате 2
1	Комната с двумя компьютерами	5	4	2	Компьютер B в комнате 2
1	Комната с двумя компьютерами	5	5	2	Компьютер C в комнате 2
2	Комната с тремя компьютерами	5	1	1	Компьютер A в комнате 1
2	Комната с тремя компьютерами	5	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	5	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	5	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	5	2	Компьютер C в комнате 2
3	Пустая комната 1	2	1	1	Компьютер A в комнате 1
3	Пустая комната 1	2	3	2	Компьютер A в комнате 2
4	Пустая комната 2	2	1	1	Компьютер A в комнате 1
4	Пустая комната 2	2	3	2	Компьютер A в комнате 2
5	Пустая комната 3	2	1	1	Компьютер A в комнате 1
5	Пустая комната 3	2	3	2	Компьютер A в комнате 2

Обратите внимание, что ни к одной комнате не было приписано компьютеров больше, чем значение в поле **`r_space`**.

Решение 2.2.10.j: используем **`CROSS APPLY`** в MS SQL Server и эмуляцию аналогичного поведения в MySQL и Oracle.

MySQL Решение 2.2.10.j

```
1  SELECT `r_id`,
2         `r_name`,
3         `r_space`,
4         `c_id`,
5         `c_room`,
6         `c_name`
7  FROM   `rooms`
8         CROSS JOIN (SELECT `c_id`,
9                            `c_room`,
10                           `c_name`,
11                           @row_num := @row_num + 1 AS `position`
12                    FROM   `computers`,
13                           (SELECT @row_num := 0) AS `x`
14                    ORDER  BY `c_name` ASC) AS `cross_apply_data`
15 WHERE  `position` <= `r_space`
16 ORDER  BY `r_id`,
17           `c_id`
```

Подзапрос в строках 8-14 возвращает пронумерованный список компьютеров:

c_id	c_room	c_name	position
1	1	Компьютер A в комнате 1	1
3	2	Компьютер A в комнате 2	2
2	1	Компьютер B в комнате 1	3
4	2	Компьютер B в комнате 2	4
5	2	Компьютер C в комнате 2	5
6	NULL	Свободный компьютер A	6
7	NULL	Свободный компьютер B	7
8	NULL	Свободный компьютер C	8

Условие в строке 15 позволяет исключить из итоговой выборки компьютеры с номерами, превышающими вместимость комнаты. Таким образом получается итоговый результат.

MS SQL Решение 2.2.10.j

```
1  SELECT [r_id],
2         [r_name],
3         [r_space],
4         [c_id],
5         [c_room],
6         [c_name]
7  FROM   [rooms]
8         CROSS APPLY (SELECT TOP ([r_space])
9                             [c_id],
10                            [c_room],
11                            [c_name]
12                     FROM   [computers]
13                     ORDER  BY [c_name] ASC) AS [cross_apply_data]
14 ORDER  BY [r_id],
15           [c_id]
```

В MS SQL Server оператор **CROSS APPLY** позволяет без никаких дополнительных действий обращаться из правой части запроса к данным из соответствующих строк левой части запроса. Благодаря этому конструкция **SELECT TOP ([r_space]) ...** приводит к выборке и подстановке из таблицы **computers** количества записей, не большего, чем значение **r_space** в соответствующей анализируемой строке из таблицы **rooms**. Поясним это графически:

r_id	r_name	r_space	Что подставляется в TOP x
1	Комната с двумя компьютерами	5	**SELECT TOP 5 ...**
2	Комната с тремя компьютерами	5	**SELECT TOP 5 ...**
3	Пустая комната 1	2	**SELECT TOP 2 ...**
4	Пустая комната 2	2	**SELECT TOP 2 ...**
5	Пустая комната 3	2	**SELECT TOP 2 ...**

Благодаря такому поведению каждой строке из таблицы **rooms** подставляется определённое количество записей из таблицы **computers**, и так получается итоговый результат.

Oracle Решение 2.2.10.j

```
1  SELECT "r_id",
2         "r_name",
3         "r_space",
4         "c_id",
5         "c_room",
6         "c_name"
7  FROM   "rooms"
8         CROSS JOIN (SELECT "c_id",
9                            "c_room",
10                           "c_name",
11                        ROW_NUMBER() OVER (ORDER BY "c_name" ASC)
12                          AS "position"
13                   FROM   "computers"
14                   ORDER  BY "c_name" ASC) "cross_apply_data"
15 WHERE  "position" <= "r_space"
16 ORDER  BY "r_id",
17           "c_id"
```

Решение для Oracle эквивалентно решению для MySQL и отличается только способом нумерации компьютеров: здесь мы можем использовать готовую функцию **ROW_NUMBER**.

Задача 2.2.10.k: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (не учитывать вместимость комнат).

Ожидаемый результат 2.2.10.k.

r_id	r_name	c_id	c_room	c_name
3	Пустая комната 1	6	NULL	Свободный компьютер A
3	Пустая комната 1	7	NULL	Свободный компьютер B
3	Пустая комната 1	8	NULL	Свободный компьютер C
4	Пустая комната 2	6	NULL	Свободный компьютер A
4	Пустая комната 2	7	NULL	Свободный компьютер B
4	Пустая комната 2	8	NULL	Свободный компьютер C
5	Пустая комната 3	6	NULL	Свободный компьютер A
5	Пустая комната 3	7	NULL	Свободный компьютер B
5	Пустая комната 3	8	NULL	Свободный компьютер C

Решение 2.2.10.k: используем перекрёстное объединение с некоторой предварительной подготовкой.

Единственная сложность этой задачи — в получении списка пустых комнат (т.к. свободные компьютеры мы элементарно определяем по значению **NULL** в поле **c_room**). Также эта задача отлично подходит для демонстрации одной типичной ошибки.

MySQL Решение 2.2.10.k

```
SELECT `r_id`,
       `r_name`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   (SELECT `r_id`,
               `r_name`
        FROM   `rooms`
        WHERE  `r_id` NOT IN (SELECT DISTINCT `c_room`
                              FROM   `computers`
                              WHERE  `c_room` IS NOT NULL))
       AS `empty_rooms`
       CROSS JOIN `computers`
WHERE  `c_room` IS NULL
```

MS SQL Решение 2.2.10.k

```
1  SELECT [r_id],
2         [r_name],
3         [c_id],
4         [c_room],
5         [c_name]
6  FROM   (SELECT [r_id],
7                 [r_name]
8          FROM   [rooms]
9          WHERE  [r_id] NOT IN (SELECT DISTINCT [c_room]
10                               FROM   [computers]
11                               WHERE  [c_room] IS NOT NULL))
12        AS [empty_rooms]
13        CROSS JOIN [computers]
14 WHERE  [c_room] IS NULL
```

Oracle Решение 2.2.10.k

```
1  SELECT "r_id",
2         "r_name",
3         "c_id",
4         "c_room",
5         "c_name"
6  FROM   (SELECT "r_id",
7                 "r_name"
8          FROM   "rooms"
9          WHERE  "r_id" NOT IN (SELECT DISTINCT "c_room"
10                               FROM   "computers"
11                               WHERE  "c_room" IS NOT NULL))
12        "empty_rooms"
13        CROSS JOIN "computers"
14 WHERE  "c_room" IS NULL
```

В этом решении 14-я строка во всех трёх запросах отвечает за учёт только свободных компьютеров. Свободные комнаты определяются подзапросом в строках 6-12 (он возвращает список комнат, идентификаторы которых не встречаются в таблице `computers`):

r_id	r_name
3	Пустая комната 1
4	Пустая комната 2
5	Пустая комната 3

Очень частая типичная ошибка заключается в **отсутствии** условия `WHERE c_room IS NOT NULL` во внутреннем подзапросе в строках 9-11. Из-за этого в его результаты попадает `NULL`-значение, при обработке которого конструкция `NOT IN` возвращает `FALSE` для любого значения `r_id`, и в итоге подзапрос в строках 6-12 возвращает пустой результат. Объединение с пустым результатом тоже даёт пустой результат. Так из-за одного неочевидного условия весь запрос перестаёт возвращать какие бы то ни было данные.

Таким образом ведёт себя именно `NOT IN`. Просто `IN` работает ожидаемым образом, т.е. возвращает `TRUE` для входящих в анализируемое множество значений и `FALSE` для не входящих.

Задача 2.2.10.l: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (учитывать вместимость комнат).

Ожидаемый результат 2.2.10.l.

r_id	r_name	r_space	c_id	c_room	c_name
3	Пустая комната 1	2	6	NULL	Свободный компьютер A
3	Пустая комната 1	2	7	NULL	Свободный компьютер B
4	Пустая комната 2	2	6	NULL	Свободный компьютер A
4	Пустая комната 2	2	7	NULL	Свободный компьютер B
5	Пустая комната 3	2	6	NULL	Свободный компьютер A
5	Пустая комната 3	2	7	NULL	Свободный компьютер B

Решение 2.2.10.l: используем CROSS APPLY в MS SQL Server и эмуляцию аналогичного поведения в MySQL и Oracle.

Решение этой задачи сводится к комбинации решений задач 2.2.10.j{171} и 2.2.10.k{174}: из первой мы возьмём логику **`CROSS APPLY`**, из второй — логику получения списка пустых комнат.

MySQL Решение 2.2.10.l

```
SELECT `r_id`,
       `r_name`,
       `r_space`,
       `c_id`,
       `c_room`,
       `c_name`
FROM   (SELECT `r_id`,
               `r_name`,
               `r_space`
        FROM   `rooms`
        WHERE  `r_id` NOT IN (SELECT `c_room`
                              FROM   `computers`
                              WHERE  `c_room` IS NOT NULL))
       AS `empty_rooms`
       CROSS JOIN (SELECT `c_id`,
                          `c_room`,
                          `c_name`,
                          @row_num := @row_num + 1 AS `position`
                   FROM   `computers`,
                          (SELECT @row_num := 0) AS `x`
                   WHERE  `c_room` IS NULL
                   ORDER  BY `c_name` ASC)
                   AS `cross_apply_data`
WHERE  `position` <= `r_space`
ORDER  BY `r_id`,
          `c_id`
```

Подзапрос в строках 8-14 возвращает список пустых комнат:

r_id	r_name	r_space
3	Пустая комната 1	2
4	Пустая комната 2	2
5	Пустая комната 3	2

Подзапрос в строках 15-23 возвращает пронумерованный список свободных компьютеров:

c_id	c_room	c_name	position
6	NULL	Свободный компьютер A	1
7	NULL	Свободный компьютер B	2
8	NULL	Свободный компьютер C	3

`CROSS JOIN` этих двух результатов даёт следующее декартово произведение (поле `position` добавлено для наглядности):

r_id	r_name	r_space	c_id	c_room	c_name	position
3	Пустая комната 1	2	6	NULL	Свободный компьютер A	1
3	Пустая комната 1	2	7	NULL	Свободный компьютер B	2
3	Пустая комната 1	2	8	NULL	Свободный компьютер C	3
4	Пустая комната 2	2	6	NULL	Свободный компьютер A	1
4	Пустая комната 2	2	7	NULL	Свободный компьютер B	2
4	Пустая комната 2	2	8	NULL	Свободный компьютер C	3
5	Пустая комната 3	2	6	NULL	Свободный компьютер A	1
5	Пустая комната 3	2	7	NULL	Свободный компьютер B	2
5	Пустая комната 3	2	8	NULL	Свободный компьютер C	3

Условие в строке 24 не допускает в выборку записи (отмечены серым фоном), в которых значение поля `position` больше значения поля `r_space`. Таким образом получается финальный результат.

В MS SQL Server всё снова намного проще.

MS SQL Решение 2.2.10.l

```
1  SELECT [r_id],
2         [r_name],
3         [r_space],
4         [c_id],
5         [c_room],
6         [c_name]
7  FROM   (SELECT [r_id],
8                 [r_name],
9                 [r_space]
10         FROM   [rooms]
11         WHERE  [r_id] NOT IN (SELECT [c_room]
12                               FROM   [computers]
13                               WHERE  [c_room] IS NOT NULL))
14        AS [empty_rooms]
15        CROSS APPLY (SELECT TOP ([r_space]) [c_id],
16                                            [c_room],
17                                            [c_name]
18                     FROM   [computers]
19                     WHERE  [c_room] IS NULL
20                     ORDER  BY [c_name] ASC)
21                   AS [cross_apply_data]
22 ORDER  BY [r_id],
23           [c_id]
```

Подзапрос в строках 7-14 возвращает список пустых комнат:

r_id	r_name	r_space
3	Пустая комната 1	2
4	Пустая комната 2	2
5	Пустая комната 3	2

Затем благодаря `CROSS APPLY` в качестве аргумента `TOP` используется значение поля `r_space`:

r_id	r_name	r_space	Что подставляется в TOP x
3	Пустая комната 1	2	`SELECT TOP 2 ...`
4	Пустая комната 2	2	`SELECT TOP 2 ...`
5	Пустая комната 3	2	`SELECT TOP 2 ...`

Таким образом получается финальный результат.

Oracle Решение 2.2.10.l

```
1   SELECT "r_id",
2          "r_name",
3          "r_space",
4          "c_id",
5          "c_room",
6          "c_name"
7   FROM    (SELECT "r_id",
8                   "r_name",
9                   "r_space"
10           FROM   "rooms"
11           WHERE  "r_id" NOT IN (SELECT "c_room"
12                                 FROM   "computers"
13                                 WHERE  "c_room" IS NOT NULL))
14         "empty_rooms"
15         CROSS JOIN (SELECT "c_id",
16                            "c_room",
17                            "c_name",
18                            ROW_NUMBER()
19                              OVER (
20                                 ORDER BY "c_name" ASC) AS "position"
21                     FROM   "computers"
22                     WHERE  "c_room" IS NULL
23                     ORDER  BY "c_name" ASC)
24         "cross_apply_data"
25  WHERE  "position" <= "r_space"
26  ORDER  BY "r_id",
27            "c_id"
```

Решение для Oracle эквивалентно решению для MySQL и отличается только способом нумерации компьютеров: здесь мы можем использовать готовую функцию `ROW_NUMBER`.

Задача 2.2.10.m: показать возможные варианты расстановки свободных компьютеров по комнатам (учитывать остаточную вместимость комнат).

Ожидаемый результат 2.2.10.m.

r_id	r_name	r_space	r_space_left	c_id	c_name
1	Комната с двумя компьютерами	5	3	6	Свободный компьютер A
1	Комната с двумя компьютерами	5	3	7	Свободный компьютер B
1	Комната с двумя компьютерами	5	3	8	Свободный компьютер C
2	Комната с тремя компьютерами	5	2	6	Свободный компьютер A
2	Комната с тремя компьютерами	5	2	7	Свободный компьютер B
3	Пустая комната 1	2	2	6	Свободный компьютер A
3	Пустая комната 1	2	2	7	Свободный компьютер B
4	Пустая комната 2	2	2	6	Свободный компьютер A
4	Пустая комната 2	2	2	7	Свободный компьютер B
5	Пустая комната 3	2	2	6	Свободный компьютер A
5	Пустая комната 3	2	2	7	Свободный компьютер B

Решение 2.2.10.m: используем **CROSS APPLY** в MS SQL Server и эмуляцию аналогичного поведения в MySQL и Oracle.

Данная задача похожа на задачу 2.2.10.j[171] за тем исключением, что здесь мы учитываем не общую вместимость комнаты, а остаточную — т.е. разницу между вместимостью комнаты и количеством уже расположенных в ней компьютеров.

MySQL Решение 2.2.10.m

```
1  SELECT `r_id`,
2         `r_name`,
3         `r_space`,
4         ( `r_space` - IFNULL(`r_used`, 0) ) AS `r_space_left`,
5         `c_id`,
6         `c_name`
7  FROM   `rooms`
8         LEFT JOIN (SELECT `c_room`         AS `c_room_inner`,
9                           COUNT(`c_room`) AS `r_used`
10                  FROM   `computers`
11                  GROUP  BY `c_room`) AS `computers_in_room`
12                ON `r_id` = `c_room_inner`
13        CROSS JOIN (SELECT `c_id`,
14                           `c_room`,
15                           `c_name`,
16                           @row_num := @row_num + 1 AS `position`
17                    FROM   `computers`,
18                           (SELECT @row_num := 0) AS `x`
19                    WHERE  `c_room` IS NULL
20                    ORDER  BY `c_name` ASC) AS `cross_apply_data`
21 WHERE `position` <= (`r_space` - IFNULL(`r_used`, 0))
22 ORDER  BY `r_id`,
23           `c_id`
```

Подзапрос в строках 13-20 возвращает пронумерованный список свободных компьютеров:

c_id	c_room	c_name	position
6	NULL	Свободный компьютер A	1
7	NULL	Свободный компьютер B	2
8	NULL	Свободный компьютер C	3

Подзапрос в строках 8-11 возвращает информацию о количестве компьютеров в каждой комнате:

c_room_inner	r_used
NULL	0
1	2
2	3

После выполнения **LEFT JOIN** в строке 8 информация о количестве компьютеров в комнате объединяется со списком комнат:

r_id	r_name	r_space	r_used
1	Комната с двумя компьютерами	5	2
2	Комната с тремя компьютерами	5	3
3	Пустая комната 1	2	NULL
4	Пустая комната 2	2	NULL
5	Пустая комната 3	2	NULL

В строках 4 и 21 информация о количестве компьютеров в комнате используется для вычисления оставшегося количества свободных мест (так получается значение поля **r_space_left**). Обратите внимание на необходимость использования функции **IFNULL** для преобразования к 0 значения **NULL** поля **r_used** у пустых комнат.

Условие в строке 21 не допускает попадание в выборку свободных компьютеров с порядковым номером большим, чем количество оставшихся в комнате свободных мест. Так получается финальный результат.

В MS SQL Server решение с использованием **CROSS APPLY** оказывается более простым и компактным.

MS SQL Решение 2.2.10.m

```
1   SELECT [r_id],
2          [r_name],
3          [r_space],
4          [r_space_left],
5          [c_id],
6          [c_name]
7   FROM   [rooms]
8          CROSS APPLY (SELECT TOP ([r_space] - (SELECT COUNT([c_room]) FROM
9                                    [computers] WHERE [c_room] =  [r_id]))
10                                 [c_id],
11                                 ( [r_space] - (SELECT COUNT([c_room])
12                                                FROM   [computers]
13                                                WHERE  [c_room] = [r_id]) )
14                                 AS [r_space_left],
15                                 [c_room],
16                                 [c_name]
17                          FROM   [computers]
18                          WHERE  [c_room] IS NULL
19                          ORDER  BY [c_name] ASC) AS [cross_apply_data]
20  ORDER BY [r_id],
21           [c_id]
```

Благодаря доступу к полям анализируемой записи из левой таблицы подзапросы в строках 8-9 и 11-14 могут сразу получить всю необходимую информацию, и в результате подзапрос в строках 8-19 возвращает для каждой записи из результата левой части **CROSS APPLY** не большее количество свободных компьютеров, чем в соответствующей комнате осталось свободных мест:

r_id	r_name	r_space	TOP x …	r_space_left
1	Комната с двумя компьютерами	5	TOP 2	2
2	Комната с тремя компьютерами	5	TOP 3	3
3	Пустая комната 1	2	TOP 2	2
4	Пустая комната 2	2	TOP 2	2
5	Пустая комната 3	2	TOP 2	2

Решение для Oracle эквивалентно решению для MySQL за исключением использования функции `NVL` вместо функции `IFNULL` и способа нумерации свободных компьютеров с помощью функции `ROW_NUMBER` вместо использования инкрементируемой переменной.

Oracle Решение 2.2.10.m

```
SELECT "r_id",
       "r_name",
       "r_space",
       ( "r_space" - NVL("r_used", 0) ) AS "r_space_left",
       "c_id",
       "c_name"
FROM   "rooms"
       LEFT JOIN (SELECT "c_room"         AS "c_room_inner",
                         COUNT("c_room") AS "r_used"
                  FROM   "computers"
                  GROUP  BY "c_room") "computers_in_room"
              ON "r_id" = "c_room_inner"
       CROSS JOIN (SELECT "c_id",
                          "c_room",
                          "c_name",
                          ROW_NUMBER() OVER (ORDER BY "c_name" ASC)
                          AS "position"
                   FROM   "computers" WHERE "c_room" IS NULL
                   ORDER  BY "c_name" ASC) "cross_apply_data"
WHERE "position" <= ("r_space" - NVL("r_used", 0))
ORDER  BY "r_id",
          "c_id"
```

Задача 2.2.10.n: показать расстановку компьютеров по непустым комнатам так, чтобы в выборку не попало больше компьютеров, чем может поместиться в комнату.

Ожидаемый результат 2.2.10.n.

r_id	r_name	r_space	c_id	c_room	c_name
1	Комната с двумя компьютерами	5	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	5	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	5	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	5	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	5	2	Компьютер C в комнате 2

Решение 2.2.10.n: используем `CROSS APPLY` в MS SQL Server и эмуляцию аналогичного поведения в MySQL и Oracle.

Эта и следующая (2.2.10.o{185}) задачи являются самыми классическими случаями использования **CROSS APPLY** и **OUTER APPLY**: производится объединение таблиц по некоторому условию, а также учитывается дополнительное условие, данные для которого берутся из левой таблицы.

В нашем случае условием объединения является совпадение значений полей **r_id** и **c_room**, а дополнительным условием, ограничивающим выборку из правой таблицы, является вместимость комнаты, представленная в поле **r_space**.

Обратите внимание, что в эмуляции **CROSS APPLY** для MySQL и Oracle в данном случае используется внутреннее объединение (**JOIN**), а не декартово произведение (**CROSS JOIN**).

Решение для MySQL получается несколько громоздким по синтаксису, но очень простым по сути.

MySQL Решение 2.2.10.n

```
1   SELECT `r_id`,
2          `r_name`,
3          `r_space`,
4          `c_id`,
5          `c_room`,
6          `c_name`
7   FROM   `rooms`
8          JOIN (SELECT `c_id`,
9                       `c_room`,
10                      `c_name`,
11                      @row_num := IF(@prev_value = `c_room`, @row_num + 1, 1)
12                      AS `position`,
13                      @prev_value := `c_room`
14                FROM   `computers`,
15                       (SELECT @row_num := 1) AS `x`,
16                       (SELECT @prev_value := '') AS `y`
17                ORDER  BY `c_room`,
18                          `c_name` ASC) AS `cross_apply_data`
19            ON `r_id` = `c_room`
20  WHERE  `position` <= `r_space`
21  ORDER  BY `r_id`,
22            `c_id`
```

Подзапрос в строках 8-18 возвращает список всех компьютеров с их нумерацией в контексте комнаты.

c_id	c_room	c_name	position
6	NULL	Свободный компьютер A	1
7	NULL	Свободный компьютер B	1
8	NULL	Свободный компьютер C	1
1	1	Компьютер A в комнате 1	1
2	1	Компьютер B в комнате 1	2
3	2	Компьютер A в комнате 2	1
4	2	Компьютер B в комнате 2	2
5	2	Компьютер C в комнате 2	3

В задачах 2.2.10.j{171}, 2.2.10.l{176} и 2.210.m{178} мы выполняли сквозную нумерацию компьютеров, не учитывая их расстановку по комнатам. Для декартового произведения (**CROSS JOIN**) нужен как раз такой сквозной номер, т.к. **CROSS JOIN** обрабатывает **все** записи из правой таблицы. Но для внутреннего объединения (**JOIN**) нужно как раз обратное — номер компьютера в контексте комнаты, в которой он расположен, т.к. **JOIN** будет искать соответствие между комнатами и расположенными в них компьютерами.

Результат выполнения **JOIN** (строка 8) выглядит так:

r_id	r_name	r_space	c_id	c_room	c_name	position
1	Комната с двумя компьютерами	5	1	1	Компьютер A в комнате 1	1
1	Комната с двумя компьютерами	5	2	1	Компьютер B в комнате 1	2
2	Комната с тремя компьютерами	5	3	2	Компьютер A в комнате 2	1
2	Комната с тремя компьютерами	5	4	2	Компьютер B в комнате 2	2
2	Комната с тремя компьютерами	5	5	2	Компьютер C в комнате 2	3

Условие **WHERE `position` <= `r_space`** в строке 20 гарантирует, что в выборку не попадёт ни один компьютер, порядковый номер которого (в контексте комнаты, в которой он расположен) больше вместимости комнаты. Так получается итоговый результат.

MS SQL Решение 2.2.10.n

```
1  SELECT [r_id],
2         [r_name],
3         [r_space],
4         [c_id],
5         [c_room],
6         [c_name]
7  FROM   [rooms]
8         CROSS APPLY (SELECT TOP ([r_space])
9                             [c_id],
10                            [c_room],
11                            [c_name]
12                     FROM   [computers]
13                     WHERE [c_room] = [r_id]
14                     ORDER  BY [c_name] ASC) AS [cross_apply_data]
15 ORDER BY [r_id],
16          [c_id]
```

В MS SQL Server условие объединения указывается в строке 13 (**WHERE [c_room] = [r_id]**), а дополнительное условие (выбрать не больше компьютеров, чем вмещает комната) указывается в строке 8 (**SELECT TOP ([r_space])**).

Таким образом, правая часть **CROSS APPLY** в один приём получает полностью готовый набор данных: для каждой комнаты получается список её компьютеров, в котором позиций не больше, чем вместимость комнат:

c_id	c_room	c_name	
1	1	Компьютер A в комнате 1	Набор данных для комнаты 1
2	1	Компьютер B в комнате 1	
3	2	Компьютер A в комнате 2	Набор данных для комнаты 2
4	2	Компьютер B в комнате 2	
5	2	Компьютер C в комнате 2	

Остаётся только добавить к каждой строке этого набора информацию о соответствующей комнате (что и делает **CROSS APPLY**), и таким образом получается итоговый результат.

Решение для Oracle аналогично решению для MySQL, за исключением одной особенности нумерации компьютеров.

Oracle Решение 2.2.10.n

```
SELECT "r_id",
       "r_name",
       "r_space",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       JOIN (SELECT "c_id",
                    "c_room",
                    "c_name",
                    ( CASE
                        WHEN "c_room" IS NULL THEN 1
                        ELSE ROW_NUMBER()
                               OVER (
                                 PARTITION BY "c_room"
                                 ORDER BY "c_name" ASC)
                      END ) AS "position"
             FROM   "computers") "cross_apply_data"
         ON "r_id" = "c_room"
WHERE  "position" <= "r_space"
ORDER  BY "r_id",
          "c_id"
```

Особенность нумерации состоит в том, как MySQL и Oracle нумеруют свободные компьютеры. MySQL для каждого свободного компьютера считает его «номер в комнате» равным 1 (что вполне логично). Этот эффект получается в силу логики условия **`@row_num := IF(@prev_value = `c_room`, @row_num + 1, 1)`**: если предыдущее значение было **`NULL`**, и следующее — тоже **`NULL`**, то они не равны (**`NULL`** не равен сам себе). Итого MySQL получает:

c_id	c_room	c_name	position
6	NULL	Свободный компьютер A	1
7	NULL	Свободный компьютер B	1
8	NULL	Свободный компьютер C	1
1	1	Компьютер A в комнате 1	1
2	1	Компьютер B в комнате 1	2
3	2	Компьютер A в комнате 2	1
4	2	Компьютер B в комнате 2	2
5	2	Компьютер C в комнате 2	3

Oracle же учитывает в поведении функции **`ROW_NUMBER`** такую ситуацию и обрабатывает все **`NULL`**-значения как равные друг другу:

c_id	c_room	c_name	position
1	1	Компьютер A в комнате 1	1
2	1	Компьютер B в комнате 1	2
3	2	Компьютер A в комнате 2	1
4	2	Компьютер B в комнате 2	2
5	2	Компьютер C в комнате 2	3
6	NULL	Свободный компьютер A	1
7	NULL	Свободный компьютер B	2
8	NULL	Свободный компьютер C	3

Чтобы добиться от Oracle поведения, аналогичного поведению MySQL, мы используем выражение **CASE** (строки 11-17).

Для решения данной конкретной задачи эта особенность не важна (свободные компьютеры никак не учитываются, а потому их порядковый номер не важен), но знать и помнить о таком различии в поведении этих двух СУБД полезно.

Задача 2.2.10.o: показать расстановку компьютеров по всем комнатам так, чтобы в выборку не попало больше компьютеров, чем может поместиться в комнату.

Ожидаемый результат 2.2.10.o.

r_id	r_name	r_space	c_id	c_room	c_name
1	Комната с двумя компьютерами	5	1	1	Компьютер A в комнате 1
1	Комната с двумя компьютерами	5	2	1	Компьютер B в комнате 1
2	Комната с тремя компьютерами	5	3	2	Компьютер A в комнате 2
2	Комната с тремя компьютерами	5	4	2	Компьютер B в комнате 2
2	Комната с тремя компьютерами	5	5	2	Компьютер C в комнате 2
3	Пустая комната 1	2	NULL	NULL	NULL
4	Пустая комната 2	2	NULL	NULL	NULL
5	Пустая комната 3	2	NULL	NULL	NULL

Решение 2.2.10.o: используем **OUTER APPLY** в MS SQL Server и эмуляцию аналогичного поведения в MySQL и Oracle.

Единственное отличие этой задачи от задачи 2.2.10.n{181} состоит в том, что нужно добавить в выборку пустые комнаты. Для MySQL и Oracle это делается заменой **JOIN** на **LEFT JOIN**, а для MS SQL Server — заменой **CROSS APPLY** на **OUTER APPLY**. Также в MySQL и Oracle нужно немного изменить условие, отвечающее за сравнение номера компьютера и вместимости комнаты.

MySQL Решение 2.2.10.n

```
1   SELECT `r_id`,
2          `r_name`,
3          `r_space`,
4          `c_id`,
5          `c_room`,
6          `c_name`
7   FROM   `rooms`
8          LEFT JOIN (SELECT `c_id`,
9                            `c_room`,
10                           `c_name`,
11                           @row_num := IF(@prev_value = `c_room`, @row_num + 1, 1)
12                           AS `position`,
13                           @prev_value := `c_room`
14                 FROM      `computers`,
15                           (SELECT @row_num := 1) AS `x`,
16                           (SELECT @prev_value := '') AS `y`
17                 ORDER  BY `c_room`,
18                           `c_name` ASC) AS `cross_apply_data`
19           ON `r_id` = `c_room`
20  WHERE  `position` <= `r_space` OR `position` IS NULL
21  ORDER  BY `r_id`,
22            `c_id`
```

MS SQL Решение 2.2.10.n

```
SELECT [r_id],
       [r_name],
       [r_space],
       [c_id],
       [c_room],
       [c_name]
FROM   [rooms]
       OUTER APPLY (SELECT TOP ([r_space])
                           [c_id],
                           [c_room],
                           [c_name]
                    FROM   [computers]
                    WHERE [c_room] = [r_id]
                    ORDER  BY [c_name] ASC) AS [cross_apply_data]
ORDER BY [r_id],
         [c_id]
```

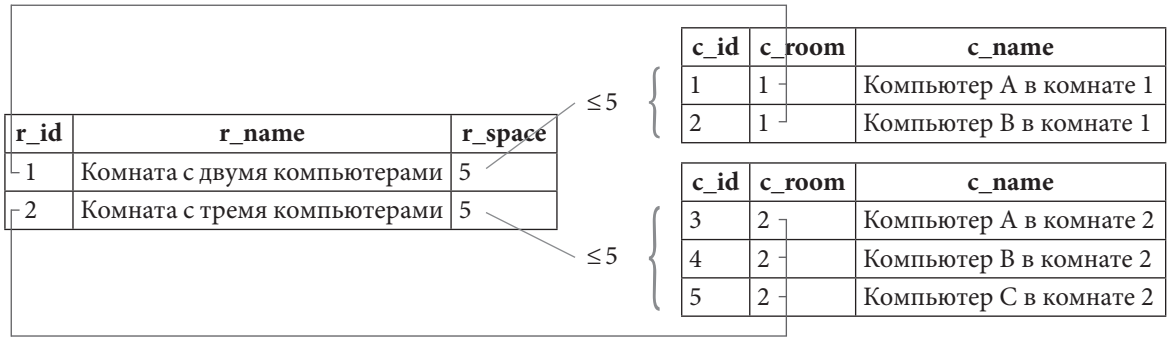
Oracle Решение 2.2.10.n

```
SELECT "r_id",
       "r_name",
       "r_space",
       "c_id",
       "c_room",
       "c_name"
FROM   "rooms"
       LEFT JOIN (SELECT "c_id",
                         "c_room",
                         "c_name",
                         ( CASE
                             WHEN "c_room" IS NULL THEN 1
                             ELSE ROW_NUMBER()
                                    OVER (
                                      PARTITION BY "c_room"
                                      ORDER BY "c_name" ASC)
                           END ) AS "position"
                  FROM   "computers") "cross_apply_data"
         ON "r_id" = "c_room"
WHERE  "position" <= "r_space" OR "position" IS NULL
ORDER  BY "r_id",
          "c_id"
```

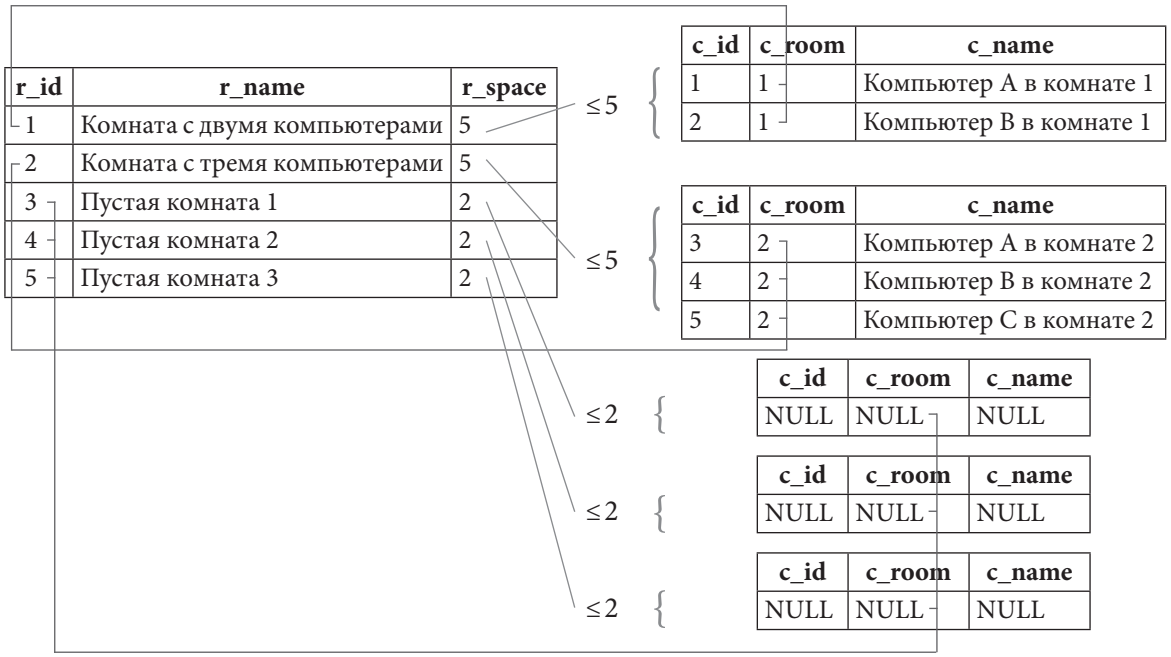
Если бы мы не добавили в строки 20 запросов для MySQL и Oracle вторую часть условия (**`OR position IS NULL`**), пустые комнаты не попали бы в итоговую выборку, т.к. им нет соответствия компьютеров, т.е. «номер» любого компьютера для них равен **`NULL`**, а сравнение **`NULL`** со значением поля **`r_space`** даёт **`FALSE`**.

Поясним ещё раз на графическом примере логику работы и разницу **`CROSS APPLY`** и **`OUTER APPLY`**.

В задаче 2.2.10.n{181} (**CROSS APPLY**):



В задаче 2.2.10.m{185} (**OUTER APPLY**):



Задание 2.2.10.TSK.A: показать информацию о том, кто из читателей и когда брал в библиотеке книги.

Задание 2.2.10.TSK.B: показать информацию обо всех читателях и датах выдачи им в библиотеке книг.

Задание 2.2.10.TSK.C: показать информацию о читателях, никогда не бравших в библиотеке книги.

Задание 2.2.10.TSK.D: показать книги, которые ни разу не были взяты никем из читателей.

Задание 2.2.10.TSK.E: показать информацию о том, какие книги в принципе может взять в библиотеке каждый из читателей.

Задание 2.2.10.TSK.F: показать информацию о том, какие книги (при условии, что он их ещё не брал) каждый из читателей может взять в библиотеке.

Задание 2.2.10.TSK.G: показать информацию о том, какие изданные до 2010-го года книги в принципе может взять в библиотеке каждый из читателей.

Задание 2.2.10.TSK.H: показать информацию о том, какие изданные до 2010-го года книги (при условии, что он их ещё не брал) может взять в библиотеке каждый из читателей.

2.3. МОДИФИКАЦИЯ ДАННЫХ

2.3.1. ПРИМЕР 21: ВСТАВКА ДАННЫХ

Очень полезным источником хороших примеров выполнения вставки данных для любой СУБД является изучение дампов баз данных (там же вы увидите множество готовых примеров SQL-конструкций по созданию таблиц, связей, триггеров и т.д.)

Все дальнейшие рассуждения относительно способа передачи строковых данных построены на предположении, что базы данных созданы в следующих кодировках:

- MySQL: utf8 / utf8_general_ci;
- MS SQL Server: UNICODE / Cyrillic_General_CI_AS;
- Oracle: AL32UTF8 / AL16UTF16.

Тема кодировок и работы с ними огромна, очень специфична для каждой СУБД и почти не будет рассмотрена в этой книге (кроме задач 7.3.2.a{547} и 7.3.2.b{547}) — обратитесь к документации по соответствующей СУБД.

Задача 2.3.1.a{189}: добавить в базу данных информацию о том, что читатель с идентификатором 4 взял 15-го января 2016-го года в библиотеке книгу с идентификатором 3 и обещал вернуть её 30-го января 2016-го года.

Задача 2.3.1.b{191}: добавить в базу данных информацию о том, что читатель с идентификатором 2 взял 25-го января 2016-го года в библиотеке книги с идентификаторами 1, 3, 5 и обещал вернуть их 30-го апреля 2016-го года.

Ожидаемый результат 2.3.1.a: в таблице `subscriptions` должна появиться такая новая запись.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
101	4	3	2016-01-15	2016-01-30	N

Ожидаемый результат 2.3.1.b в таблице `subscriptions` должны появиться три таких новых записи.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
102	2	1	2016-01-25	2016-04-30	N
103	2	3	2016-01-25	2016-04-30	N
104	2	5	2016-01-25	2016-04-30	N

Решение 2.3.1.a[188].

Во всех трёх СУБД в таблице **subscriptions** у нас созданы автоинкрементируемые первичные ключи, потому их значение указывать не надо. Остаётся только передать известные нам данные.

MySQL Решение 2.3.1.a

```
1  INSERT INTO `subscriptions`
2              (`sb_id`,
3               `sb_subscriber`,
4               `sb_book`,
5               `sb_start`,
6               `sb_finish`,
7               `sb_is_active`)
8  VALUES      (NULL,
9               4,
10              3,
11              '2016-01-15',
12              '2016-01-30',
13              'N')
```

Если вставка выполняется во все поля таблицы, часть запроса в строках 2-7 можно не писать, в противном случае эта часть является обязательной.

Если мы не хотим явно указывать значение автоинкрементируемого первичного ключа, в MySQL можно вместо его значения передать **NULL** или исключить это поле из списка передаваемых полей и не передавать никаких данных (т.е. убрать имя поля в строке 2 и значение **NULL** в строке 8).

MS SQL Решение 2.3.1.a

```
1  INSERT INTO [subscriptions]
2              ([sb_subscriber],
3               [sb_book],
4               [sb_start],
5               [sb_finish],
6               [sb_is_active])
7  VALUES      (4,
8               3,
9               CAST(N'2016-01-15' AS DATE),
10              CAST(N'2016-01-30' AS DATE),
11              N'N')
```

В MS SQL Server автоинкрементация первичного ключа осуществляется за счёт того, что соответствующее поле помечается как **IDENTITY**. Вставка значений **NULL** и **DEFAULT** в такое поле запрещена, потому мы обязаны исключить его из списка полей и не передавать в него никаких значений.

Даже если бы мы точно знали значение первичного ключа этой вставляемой записи, всё равно для его вставки нам пришлось бы выполнить две дополнительных операции:

- перед вставкой данных выполнить команду
  `SET IDENTITY_INSERT [subscriptions] ON`;
- после вставки данных выполнить команду
  `SET IDENTITY_INSERT [subscriptions] OFF`.

Эти команды соответственно разрешают и снова запрещают вставку в `IDENTITY`-поле явно переданных значений.

В строках 9-10 запроса производится явное преобразование строки, содержащей дату, к типу данных `DATE`. MS SQL Server позволяет этого не делать (конвертация происходит автоматически), но из соображений надёжности рекомендуется использовать явное преобразование.

Теми же соображениями надёжности вызвана необходимость ставить букву `N` перед строковыми константами (строки 9-11 запроса), чтобы явно указать СУБД на то, что строки представлены в т.н. «национальной кодировке» (и юникоде как форме представления символов). Для символов английского алфавита и символов, из которых состоит дата, этим правилом можно пренебречь, но как только в строке появится хотя бы один символ, по-разному представленный в разных кодировках, вы рискуете повредить данные.

Интересен тот факт, что даже в нашем конкретном случае (формат поля `sb_is_active — CHAR(1)`, а не `NCHAR(1)`) в создаваемом средствами MS SQL Server Management Studio дампе базы данных буква `N` присутствует перед значениями поля `sb_is_active`. Краткий вывод: если есть сомнения, использовать букву `N` перед строковыми константами, или нет, — лучше использовать.

Oracle Решение 2.3.1.a

```
1   INSERT INTO "subscriptions"
2               ("sb_id",
3                "sb_subscriber",
4                "sb_book",
5                "sb_start",
6                "sb_finish",
7                "sb_is_active")
8   VALUES      (NULL,
9                4,
10               3,
11               TO_DATE('2016-01-15', 'YYYY-MM-DD'),
12               TO_DATE('2016-01-30', 'YYYY-MM-DD'),
13               'N')
```

В Oracle обязательно нужно преобразовывать строковое представление дат к соответствующему типу.

В отличие от MS SQL Server здесь нет необходимости указывать букву `N` перед строковыми константами со значениями дат и поля `sb_is_active` (можно и указать — это не приведёт к ошибке). Но для большинства остальных данных (например, текста на русском языке) букву `N` лучше указывать.

Особый интерес представляет передача значения автоинкрементируемого первичного ключа. В Oracle такой ключ реализуется нетривиальным способом — созданием `SEQUENCE` как «источника чисел» и триггера, который получает очередное число из `SEQUENCE` и использует его в качестве значения первичного ключа вставляемой записи.

Из этого следует, что в качестве значения автоинкрементируемого ключа вы можете передавать… что угодно: `NULL`, `DEFAULT`, число — триггер всё равно заменит переданное вами значение на очередное полученное из `SEQUENCE` число.

Если же вам необходимо явно указать значение поля **sb_id**, нужно выполнить две дополнительных операции:

- перед вставкой данных выполнить команду
  ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE;
- после вставки данных выполнить команду
  ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE.

Эти команды соответственно отключают и снова включают триггер, устанавливающий значение автоинкрементируемого первичного ключа.

Решение 2.3.1.b{188}.

Формально мы можем свести решение этой задачи к выполнению трёх отдельных запросов, аналогичных представленным в решении задачи 2.3.1.a{189}, но существует более эффективный способ выполнения вставки набора данных, который включает три действия:

- временную приостановку работы индексов;
- вставку данных одним большим блоком;
- возобновление работы индексов.

Наличие индексов может сильно снизить скорость модификации данных, т.к. СУБД будет вынуждена тратить много ресурсов на постоянное обновление индексов для поддержания их в актуальном состоянии. Также выполнение множества отдельных запросов вместо одного (пусть и большого) приводит к дополнительным накладным расходам на управление транзакциями и иными внутренними механизмами работы СУБД.

Отключение и повторное включение индексов имеет смысл только при модификации действительно большого набора данных (десятки тысяч записей и более), но соответствующие команды мы всё равно рассмотрим.

Обязательно изучите соответствующие разделы документации к вашей СУБД. Рекомендации по использованию тех или иных команд (и даже сама применимость команд) могут очень сильно отличаться в зависимости от множества факторов.

В MySQL есть специальная команда по отключению и повторному включению индексов (**ALTER TABLE ... DISABLE KEYS** и **ALTER TABLE ... ENABLE KEYS**), но она действует только на таблицы, работающей с методом доступа MyISAM. На повсеместно распространённый ныне метод доступа InnoDB она не действует.

Что можно сделать в методе доступа InnoDB?

- Отключить и затем включить контроль уникальности (фактически, выключить и включить уникальные индексы).
- Отключить и затем включить контроль внешних ключей.
- Отключить и затем снова включить автоматическое подтверждение транзакций (если мы собираемся выполнить несколько отдельных запросов).
- Можно «поиграть» с автоинкрементируемыми первичными ключами[5], но здесь нет универсальных рекомендаций.

Отключение уникальных индексов и контроля внешних ключей — крайне опасная операция, которая в перспективе может привести к катастрофическим повреждениям данных. Выполняйте её только как последнее средство, если ничто другое не помогает повысить производительность, и вы на 101% уверены в том, что передаваемые вами данные полностью удовлетворяют требованиям, соблюдение которых призваны контролировать уникальные индексы и внешние ключи.

---

[5] https://dev.mysql.com/doc/refman/5.6/en/innodb-auto-increment-handling.html#innodb-auto-increment-lock-mode-usage-implications

Если теперь отбросить все нюансы и неочевидные рекомендации, остаётся один основной вывод: выполняйте вставку одним запросом (вставляйте сразу несколько строк) — это более быстрый вариант в сравнении с несколькими отдельными запросами, каждый из которых вставляет по одной строке.

MySQL Решение 2.3.1.b

```
1   -- Актуально для MyISAM, не актуально для InnoDB:
2   ALTER TABLE `subscriptions` DISABLE KEYS; -- Отключение индексов.
3
4   -- Следующие две команды -- ОЧЕНЬ опасное решение!
5   SET foreign_key_checks = 0; -- Отключение проверки внешних ключей.
6   SET unique_checks = 0; -- Отключение уникальных индексов.
7
8   SET autocommit = 0; -- Отключение автоматической фиксации транзакций.
9
10  -- Сам запрос на вставку:
11  INSERT INTO `subscriptions`
12              (`sb_subscriber`,
13               `sb_book`,
14               `sb_start`,
15               `sb_finish`,
16               `sb_is_active`)
17  VALUES      ( 2,
18                1,
19                '2016-01-25',
20                '2016-04-30',
21                'N' ),
22              ( 2,
23                3,
24                '2016-01-25',
25                '2016-04-30',
26                'N' ),
27              ( 2,
28                5,
29                '2016-01-25',
30                '2016-04-30',
31                'N' );
32
33  COMMIT; -- Фиксация транзакции.
34
35  SET autocommit = 1; -- Включение автоматической фиксации транзакций.
36
37  SET unique_checks = 1; -- Включение уникальных индексов.
38  SET foreign_key_checks = 1; -- Включение проверки внешних ключей.
39
40  -- Актуально для MyISAM, не актуально для InnoDB:
41  ALTER TABLE `subscriptions` ENABLE KEYS; -- Включение индексов.
```

Ещё раз отметим, что в общем случае можно и нужно ограничиться запросом, показанным в строках 10-16. Остальные идеи приведены как справочная информация.

В MS SQL Server вы тоже можете временно выключить и затем снова включить индексы командами `ALTER INDEX ALL ON ... DISABLE` и `ALTER INDEX ALL ON ... REBUILD`.

Обязательно ознакомьтесь хотя бы с двумя[6, 7] соответствующими разделами документации — у этих операций могут быть крайне неожиданные и опасные побочные эффекты. Их выполнение может поставить под серьёзную угрозу способность СУБД контролировать консистентность данных.

MS SQL Решение 2.3.1.b

```
-- ОЧЕНЬ опасное решение!
ALTER INDEX ALL ON [subscriptions] DISABLE;

-- Обязательно включите кластерный индекс
-- (в нашем случае -- первичный ключ) перед дальнейшими
-- операциями, иначе запросы будут завершаться ошибкой.
ALTER INDEX [PK_subscriptions] ON [subscriptions] REBUILD;

-- Сам запрос на вставку:
INSERT INTO [subscriptions]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (2,
             1,
             CAST(N'2016-01-25' AS DATE),
             CAST(N'2016-04-30' AS DATE),
             N'N'),
            (2,
             3,
             CAST(N'2016-01-25' AS DATE),
             CAST(N'2016-04-30' AS DATE),
             N'N'),
            (2,
             5,
             CAST(N'2016-01-25' AS DATE),
             CAST(N'2016-04-30' AS DATE),
             N'N');

-- Включение индексов после их отключения:
ALTER INDEX ALL ON [subscriptions] REBUILD;
```

В отличие от MySQL и MS SQL Server в Oracle нет готового решения для выключения и включения всех индексов. Существует много алгоритмических решений[8] этой задачи, но подавляющее большинство авторов совершенно справедливо предупреждают о множестве потенциальных проблем, рекомендуют этого не делать и даже высказывают предположения о том, что удаление и повторное создание индекса может оказаться более выгодным, чем его выключение и включение.

Единственный индекс на таблице `subscriptions` — это её первичный ключ, потому в приведённом ниже примере именно его мы будем выключать и включать.

---

6 https://msdn.microsoft.com/en-us/library/ms177456.aspx

7 https://msdn.microsoft.com/en-us/library/ms190645.aspx

8 http://johnlevandowski.com/oracle-disable-constraints-and-make-indexes-unusable/

Снова и снова напоминаем: это очень опасная операция, вы рискуете фатально повредить вашу базу данных, если что-то пойдёт не так, как вы запланировали или предполагали. Ни в коем случае не пытайтесь проводить подобные эксперименты на реальных работающих базах данных, тренируйтесь только на их копиях, которые вам совершенно не жалко безвозвратно потерять.

Ещё один важный нюанс решения для Oracle состоит в том, что эта СУБД не поддерживает классический синтаксис вставки одним запросом нескольких записей, потому приходится использовать альтернативную запись.

Oracle Решение 2.3.1.b

```
-- ОЧЕНЬ опасное решение!
ALTER TABLE "subscriptions" MODIFY CONSTRAINT "PK_subscriptions" DISABLE;

-- Сам запрос на вставку:
INSERT ALL
 INTO   "subscriptions"
        ("sb_subscriber",
         "sb_book",
         "sb_start",
         "sb_finish",
         "sb_is_active")
 VALUES (2,
         1,
         TO_DATE('2016-01-25', 'YYYY-MM-DD'),
         TO_DATE('2016-04-30', 'YYYY-MM-DD'),
         'N')
 INTO   "subscriptions"
        ("sb_subscriber",
         "sb_book",
         "sb_start",
         "sb_finish",
         "sb_is_active")
 VALUES (2,
         3,
         TO_DATE('2016-01-25', 'YYYY-MM-DD'),
         TO_DATE('2016-04-30', 'YYYY-MM-DD'),
         'N')
 INTO   "subscriptions"
        ("sb_subscriber",
         "sb_book",
         "sb_start",
         "sb_finish",
         "sb_is_active")
 VALUES (2,
         5,
         TO_DATE('2016-01-25', 'YYYY-MM-DD'),
         TO_DATE('2016-04-30', 'YYYY-MM-DD'),
         'N')
SELECT 1 FROM "DUAL";

-- Включение индекса после его отключения:
ALTER TABLE "subscriptions" MODIFY CONSTRAINT "PK_subscriptions" ENABLE;
```

Исследование 2.3.1.EXP.A: сравним скорость вставки данных, основанной на циклическом повторении (1000 итераций) запроса, вставляющего одну строку, и выполнении одного запроса, вставляющего за один раз 1000 строк. Выполним по сто раз каждый вариантов поочерёдно.

Значения медиан времени, затраченного на вставку тысячи записей каждым из вариантов, таковы:

	MySQL	MS SQL Server	Oracle
Цикл	3.000	2.000	6.000
Один запрос	0.112	0.937	5.807

Числа для первого (циклического) варианта имеют такие аккуратные значения потому, что время выполнения каждого отдельного запроса считалось с точностью до тысячной доли секунды, затем полученное значение медианы умножалось на тысячу.

В каждой СУБД соотношение времени работы двух представленных решения разное, но везде по скорости выигрывает вариант со вставкой большого количества строк одним запросом.

В Oracle такая небольшая разница во времени работы двух представленных вариантов обусловлена тем, что, фактически, это один и тот же вариант, только записанный разными способами. В этой СУБД тоже можно добиться очень высокой производительности вставки данных, но это достигается более сложными способами[9].

Для повышения надёжности операций вставки рекомендуется после их выполнения получать на уровне приложения информацию о том, какое количество рядов было затронуто операцией. Если полученное число не совпадает с числом переданных на вставку строк, где-то произошла ошибка.

Задание 2.3.1.TSK.A: добавить в базу данных информацию о троих новых читателях: «Орлов О.О.», «Соколов С.С.», «Беркутов Б.Б.»

Задание 2.2.1.TSK.B: отразить в базе данных информацию о том, что каждый из добавленных в задании 2.3.1.TSK.A читателей 20-го января 2016-го года на месяц взял в библиотеке книгу «Курс теоретической физики».

Задание 2.2.1.TSK.C: добавить в базу данных пять любых авторов и десять книг этих авторов (по две на каждого); если понадобится, добавить в базу данных соответствующие жанры. Отразить авторство добавленных книг и их принадлежность к соответствующим жанрам.

2.3.2. ПРИМЕР 22: ОБНОВЛЕНИЕ ДАННЫХ

Задача 2.3.2.a{196}: у выдачи с идентификатором 99 изменить дату возврата на текущую и отметить, что книга возвращена.

Задача 2.3.2.b{197}: изменить ожидаемую дату возврата для всех книг, которые читатель с идентификатором 2 взял в библиотеке 25-го января 2016-го года, на «плюс два месяца» (т.е. читатель будет читать их на два месяца дольше, чем планировал).

9 http://www.oracle.com/technetwork/issue-archive/2012/12-sep/o52plsql-1709862.html

Ожидаемый результат 2.3.2.a: строка с первичным ключом, равным 99, примет примерно такой вид (у вас дата в поле **sb_finish** будет другой).

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
99	4	4	2015-10-08	2016-01-06	N

Ожидаемый результат 2.3.2.b: следующие строки примут такой вид.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
102	2	1	2016-01-25	2016-06-30	N
103	2	3	2016-01-25	2016-06-30	N
104	2	5	2016-01-25	2016-06-30	N

Решение 2.3.2.a[{195}]:

Значение текущей даты можно получить на стороне приложения и передать в СУБД, тогда решение будет предельно простым (см. вариант 1), но текущую дату можно получить и на стороне СУБД (см. вариант 2), что не сильно усложняет запрос.

MySQL Решение 2.3.2.a

```
-- Вариант 1: прямая подстановка текущей даты в запрос.
UPDATE `subscriptions`
SET    `sb_finish` = '2016-01-06',
       `sb_is_active` = 'N'
WHERE  `sb_id` = 99

-- Вариант 2: получение текущей даты на стороне СУБД.
UPDATE `subscriptions`
SET    `sb_finish` = CURDATE(),
       `sb_is_active` = 'N'
WHERE  `sb_id` = 99
```

MS SQL Решение 2.3.2.a

```
-- Вариант 1: прямая подстановка текущей даты в запрос.
UPDATE [subscriptions]
SET    [sb_finish] = CAST(N'2016-01-06' AS DATE),
       [sb_is_active] = N'N'
WHERE  [sb_id] = 99

-- Вариант 2: получение текущей даты на стороне СУБД.
UPDATE [subscriptions]
SET    [sb_finish] = CONVERT(date, GETDATE()),
       [sb_is_active] = N'N'
WHERE  [sb_id] = 99
```

Oracle Решение 2.3.2.a

```
1  -- Вариант 1: прямая подстановка текущей даты в запрос.
2  UPDATE "subscriptions"
3  SET    "sb_finish" = TO_DATE('2016-01-06', 'YYYY-MM-DD'),
4         "sb_is_active" = 'N'
5  WHERE  "sb_id" = 99
6
7  -- Вариант 2: получение текущей даты на стороне СУБД.
8  UPDATE "subscriptions"
9  SET    "sb_finish" = TRUNC(SYSDATE),
10        "sb_is_active" = 'N'
11 WHERE  "sb_id" = 99
```

Применение функции **TRUNC** в 9-й строке запроса нужно для получения из полного формата представления даты-времени только даты.

Внимательно следите за тем, чтобы ваши запросы на обновление данных содержали условие (в этой задаче — строки 5 и 11 всех трёх запросов). **UPDATE** без условия обновит **все** записи в таблице, т.е. вы «покалечите» данные.

Существует ещё две очень распространённых ошибки при решении любой задачи, связанной с понятием «текущая дата»:

1) Если пользователь и сервер с СУБД находятся в разных часовых поясах, каждые сутки возникает отрезок времени, когда «текущая дата» у пользователя и на сервере отличается. Это следует учитывать, внося соответствующие поправки либо на уровне отображения данных пользователю, либо на уровне хранения данных.

2) Если дата хранит в себе не только информацию о годе, месяце и дне, но и о часах, минутах, секундах (дробных долях секунд), операции сравнения могут дать неожиданный результат (например, что с 01.01.2011 по 01.01.2012 не прошёл год, если в базе данных первая дата представлена как «2011-01-01 11:23:17» и «сегодня» представлено как «2012-01-01 15:28:27» — до прохождения года остаётся чуть больше четырёх часов).

Универсального решения этих ситуаций не существует. Требуется комплексный подход, начиная с выбора оптимального формата хранения, заканчивая реализацией и глубоким тестированием алгоритмов обработки данных.

Решение 2.3.2.b[195].

В отличие от предыдущей задачи, здесь крайне нерационален вариант с получением новой даты на стороне приложения и подстановкой готового значения в запрос (мы ведь не знаем, сколько записей нам придётся обработать, и у разных записей вполне могут быть разные исходные значения обновляемой даты). Таким образом, придётся добавлять два месяца к дате средствами СУБД.

Типичная ошибка: разобрать дату на строковые фрагменты, получить числовое представление месяца, добавить к нему нужное «смещение», преобразовать назад в строку, и «собрать» новую дату. Представьте, что к 15-му декабря 2011-го года надо добавить шесть месяцев: 2011-12-15 → 2011-**18**-15. Получился 18-й месяц, что несколько нелогично. С отрицательными смещениями получается ещё более «забавная» картина.

Вывод: операции с добавлением или вычитанием временных интервалов нужно производить с помощью специальных средств, умеющих учитывать годы, месяцы, дни, часы, минуты, секунды и т.д.

MySQL Решение 2.3.2.b

```
UPDATE `subscriptions`
SET    `sb_finish` = DATE_ADD(`sb_finish`, INTERVAL 2 MONTH)
WHERE  `sb_subscriber` = 2
       AND `sb_start` = '2016-01-25';
```

MS SQL Решение 2.3.2.b

```
UPDATE [subscriptions]
SET    [sb_finish] = DATEADD(month, 2, [sb_finish])
WHERE  [sb_subscriber] = 2
       AND [sb_start] = CONVERT(date, '2016-01-25');
```

Oracle Решение 2.3.2.b

```
UPDATE "subscriptions"
SET    "sb_finish" = ADD_MONTHS("sb_finish", 2)
WHERE  "sb_subscriber" = 2
       AND "sb_start" = TO_DATE('2016-01-25', 'YYYY-MM-DD');
```

Во всех трёх СУБД решения достаточно просты и реализуются идентичным образом, за исключением специфики функций приращения даты.

Для повышения надёжности операций обновления рекомендуется после их выполнения получать на уровне приложения информацию о том, какое количество рядов было затронуто операцией. Если полученное число не совпадает с ожидаемым, где-то произошла ошибка.

Да, мы не всегда можем заранее знать, сколько записей затронет обновление, но всё же существуют случаи, когда это известно (например, библиотекарь отметил чек-боксами некие три выдачи и нажал «Книги возвращены», значит, обновление должно затронуть три записи).

Задание 2.3.2.TSK.A: отметить все выдачи с идентификаторами ≤50 как возвращённые.

Задание 2.2.2.TSK.B: для всех выдач, произведённых до 1-го января 2012-го года, уменьшить значение дня выдачи на 3.

Задание 2.2.2.TSK.C: отметить как невозвращённые все выдачи, полученные читателем с идентификатором 2.

2.3.3. ПРИМЕР 23: УДАЛЕНИЕ ДАННЫХ

Задача 2.3.3.a{199}: удалить информацию о том, что читатель с идентификатором 4 взял 15-го января 2016-го года в библиотеке книгу с идентификатором 3.

Задача 2.3.3.b{199}: удалить информацию обо всех посещениях библиотеки читателем с идентификатором 3 по воскресеньям.

Ожидаемый результат 2.3.3.a: следующая запись должна быть удалена.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
101	4	3	2016-01-15	2016-01-30	N

Ожидаемый результат 2.3.3.b: следующие записи должны быть удалены.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
62	3	5	2014-08-03	2014-10-03	Y
86	3	1	2014-08-03	2014-09-03	Y

Решение 2.3.3.a{198}.

В данном случае решение тривиально и одинаково для всех трёх СУБД за исключением синтаксиса формирования значения поля **sb_start**.

MySQL Решение 2.3.3.a

```
1  DELETE FROM `subscriptions`
2  WHERE  `sb_subscriber` = 4
3         AND `sb_start` = '2016-01-15'
4         AND `sb_book` = 3
```

MS SQL Решение 2.3.3.a

```
1  DELETE FROM [subscriptions]
2  WHERE  [sb_subscriber] = 4
3         AND [sb_start] = CONVERT(date, '2016-01-15')
4         AND [sb_book] = 3
```

Oracle Решение 2.3.3.a

```
1  DELETE FROM "subscriptions"
2  WHERE  "sb_subscriber" = 4
3         AND "sb_start" = TO_DATE('2016-01-15', 'YYYY-MM-DD')
4         AND "sb_book" = 3
```

Внимательно следите за тем, чтобы ваши запросы на удаление данных содержали условие (в этой задаче — строки 2-4 всех трёх запросов). **DELETE** без условия удалит **все** записи в таблице, т.е. вы потеряете все данные.

Решение 2.3.3.b{198}.

В решении{42} задачи 2.1.7.b{40} мы подробно рассматривали, почему в условиях, затрагивающих дату, выгоднее использовать диапазоны, а не извлекать фрагменты даты, и оговаривали, что могут быть исключения, при которых диапазонами обойтись не получится. Данная задача как раз и является таким исключением: нам придётся получать информацию о номере дня недели на основе исходного значения даты.

MySQL Решение 2.3.3.b

```
1  DELETE FROM `subscriptions`
2  WHERE  `sb_subscriber` = 3
3         AND DAYOFWEEK(`sb_start`) = 1
```

Обратите внимание: функция **DAYOFWEEK** в MySQL возвращает номер дня, начиная с воскресенья (1) и заканчивая субботой (7).

MS SQL Решение 2.3.3.b

```
1  DELETE FROM [subscriptions]
2  WHERE  [sb_subscriber] = 3
3         AND DATEPART(weekday, [sb_start]) = 1
```

В MS SQL Server функция **DATEPART** также нумерует дни недели, начиная с воскресенья.

Oracle Решение 2.3.3.b

```
1  DELETE FROM "subscriptions"
2  WHERE  "sb_subscriber" = 3
3         AND TO_CHAR("sb_start", 'D') = 1
```

Oracle ведёт себя аналогичным образом: получая номер дня недели, функция **TO_CHAR** начинает нумерацию с воскресенья.

Логика поведения функций, определяющих номер дня недели, может различаться в разных СУБД и зависеть от настроек СУБД, операционной системы и иных факторов. Не полагайтесь на то, что такие функции всегда и везде будут работать так, как вы привыкли.

Для повышения надёжности операций удаления рекомендуется после их выполнения получать на уровне приложения информацию о том, какое количество рядов было затронуто операцией. Если полученное число не совпадает с ожидаемым, где-то произошла ошибка.

Да, мы не всегда можем заранее знать, сколько записей затронет удаление, но всё же существуют случаи, когда это известно (например, библиотекарь отметил чек-боксами некие три выдачи и нажал «Удалить», значит, удаление должно затронуть три записи).

Задание 2.3.3.TSK.A: удалить информацию обо всех выдачах читателям книги с идентификатором 1.

Задание 2.2.3.TSK.B: удалить все книги, относящиеся к жанру «Классика».

Задание 2.2.3.TSK.C: удалить информацию обо всех выдачах книг, произведённых после 20-го числа любого месяца любого года.

2.3.4. ПРИМЕР 24: СЛИЯНИЕ ДАННЫХ

Задача 2.3.4.a[201]: добавить в базу данных жанры «Философия», «Детектив», «Классика».

Задача 2.3.4.b[203]: скопировать (без повторений) в базу данных «Библиотека» содержимое таблицы `genres` из базы данных «Большая библиотека»; в случае совпадения первичных ключей добавить к существующему имени жанра слово « [OLD]».

Ожидаемый результат 2.3.4.a: содержимое таблицы `genres` должно принять следующий вид (обратите внимание: жанр «Классика» уже был в этой таблице, и он **не** должен дублироваться).

g_id	g_name
1	Поэзия
2	Программирование
3	Психология
4	Наука
5	Классика
6	Фантастика
7	Философия
8	Детектив

Ожидаемый результат 2.3.4.b: этот результат получен на «эталонных данных»; если вы сначала решите задачу 2.3.4.a, в вашем ожидаемом результате будут находиться все восемь жанров из «Библиотеки» и 92 жанра со случайными именами из «Большой библиотеки».

g_id	g_name
1	Поэзия [OLD]
2	Программирование [OLD]
3	Психология [OLD]
4	Наука [OLD]
5	Классика [OLD]
6	Фантастика [OLD]
И ещё 94 строки со случайными именами жанров из «Большой библиотеки»	

Решение 2.3.4.a[201].

Во всех трёх СУБД на поле `g_name` таблицы `genres` построен уникальный индекс, чтобы исключить возможность появления одноимённых жанров.

Эту задачу можно решить, последовательно выполнив три `INSERT`-запроса, отслеживая их результаты (для вставки жанра «Классика» запрос завершится ошибкой). Но можно реализовать и более элегантное решение.

MySQL поддерживает оператор **REPLACE**, который работает аналогично **INSERT**, но учитывает значения первичного ключа и уникальных индексов, добавляя новую строку, если совпадений нет, и обновляя имеющуюся, если совпадение есть.

MySQL Решение 2.3.4.a

```
1  REPLACE INTO `genres`
2              (`g_id`,
3               `g_name`)
4  VALUES      (NULL,
5               'Философия'),
6              (NULL,
7               'Детектив'),
8              (NULL,
9               'Классика')
```

При таком подходе за один запрос (который выполняется без ошибок) мы можем передать много новых данных, не опасаясь проблем, связанных с дублированием первичных ключей и уникальных индексов.

К сожалению, MS SQL Server и Oracle не поддерживают оператор **REPLACE**, но аналогичного поведения можно добиться с помощью оператора **MERGE**.

MS SQL Решение 2.3.4.a

```
1  MERGE INTO [genres]
2  USING ( VALUES (N'Философия'),
3                 (N'Детектив'),
4                 (N'Классика') ) AS [new_genres]([g_name])
5  ON [genres].[g_name] = [new_genres].[g_name]
6  WHEN NOT MATCHED BY TARGET THEN
7    INSERT ([g_name])
8    VALUES ([new_genres].[g_name]);
```

В этом запросе строки 2-4 представляют собой нетривиальный способ динамического создания таблицы, т.к. оператор **MERGE** не может получать «на вход» ничего, кроме таблиц и условия их слияния.

Строка 5 описывает проверяемое условие (совпадение имени нового жанра с именем уже существующего), а строки 6-8 предписывают выполнить вставку данных в целевую таблицу только в том случае, когда условие не выполнилось (т.е. дублирования нет).

Обратите внимание на то, что в конце строки 8 присутствует символ **;**. MS SQL Server требует его наличия в конце оператора **MERGE**.

Oracle Решение 2.3.4.a

```
1  MERGE INTO "genres"
2  USING (SELECT ( N'Философия' ) AS "g_name"
3         FROM   dual
4         UNION
5         SELECT ( N'Детектив' ) AS "g_name"
6         FROM   dual
7         UNION
8         SELECT ( N'Классика' ) AS "g_name"
9         FROM   dual) "new_genres"
10 ON ( "genres"."g_name" = "new_genres"."g_name" )
11 WHEN NOT MATCHED THEN
12   INSERT ("g_name")
13   VALUES ("new_genres"."g_name")
```

Решение для Oracle построено по той же логике, что и решение для MS SQL Server. Отличие состоит только в способе динамического формирования таблицы (строки 2-9) из новых данных.

Решение 2.3.4.b[201].

Для решения этой задачи в MySQL вам нужно установить соединение с СУБД от имени пользователя, имеющего права на работу с обеими базами данных (предположим, что «Библиотека» у вас называется `` `library` ``, а «Большая библиотека» — `` `huge_library` ``).

Далее остаётся воспользоваться вполне классическим **`INSERT ... SELECT`** (позволяет вставить в таблицу данные, полученные в результате выполнения запроса; строки 1-8), а условие задачи о добавлении слова « [OLD]» реализовать через специальный синтаксис **`ON DUPLICATE KEY UPDATE`** (строки 9-11), предписывающий MySQL выполнять обновление записи в целевой таблице, если возникает ситуация дублирования по первичному ключу или уникальному индексу между уже существующими и добавляемыми данными.

MySQL Решение 2.3.4.b

```
1  INSERT INTO `library`.`genres`
2              (
3               `g_id`,
4               `g_name`
5              )
6  SELECT `g_id`,
7         `g_name`
8  FROM   `huge_library`.`genres`
9  ON DUPLICATE KEY
10 UPDATE `library`.`genres`.`g_name` =
11         CONCAT(`library`.`genres`.`g_name`, ' [OLD]')
```

Для решения этой задачи в MS SQL Server (как и в случае с MySQL) вам нужно установить соединение с СУБД от имени пользователя, имеющего права на работу с обеими базами данных (предположим, что «Библиотека» у вас называется **`[library]`**, а «Большая библиотека» — **`[huge_library]`**).

MS SQL Server не поддерживает **`ON DUPLICATE KEY UPDATE`**, но аналогичного поведения можно добиться с использованием **`MERGE`**.

MS SQL Решение 2.3.4.b

```
1  -- Разрешение вставки явно переданных значений в IDENTITY-поле:
2  SET IDENTITY_INSERT [genres] ON;
3
4  -- Слияние данных:
5  MERGE [library].[dbo].[genres]      AS [destination]
6  USING [huge_library].[dbo].[genres] AS [source]
7  ON [destination].[g_id] = [source].[g_id]
8  WHEN MATCHED THEN
9    UPDATE SET [destination].[g_name] =
10              CONCAT([destination].[g_name], N' [OLD]')
11 WHEN NOT MATCHED THEN
12   INSERT ([g_id],
13           [g_name])
14   VALUES ([source].[g_id],
15           [source].[g_name]);
16
17 -- Запрет вставки явно переданных значений в IDENTITY-поле:
18 SET IDENTITY_INSERT [genres] OFF;
```

Поскольку поле `g_id` является `IDENTITY`-полем, нужно явно разрешить вставку туда данных (строка 2) перед выполнением вставки, а затем запретить (строка 18).

Строки 5-7 запроса описывают таблицу-источник, таблицу-приёмник и проверяемое условие совпадения значений.

Строки 8-10 и 11-15 описывают, соответственно, реакцию на совпадение (выполнить обновление) и несовпадение (выполнить вставку) первичных ключей таблицы-источника и таблицы-приёмника.

Для решения этой задачи в Oracle, (как и в случае с MySQL и MS SQL Server) вам нужно установить соединение с СУБД от имени пользователя, имеющего права на работу с обеими базами данных (предположим, что «Библиотека» у вас называется `"library"`, а «Большая библиотека» — `"huge_library"`).

Oracle как и MS SQL Server не поддерживает `ON DUPLICATE KEY UPDATE`, но аналогичного поведения можно добиться с использованием `MERGE`.

Решение для Oracle аналогично решению для MS SQL Server, за исключением необходимости отключать (строка 2) перед вставкой данных триггер, отвечающий за автоинкремент первичного ключа, и снова включать его (строка 18) после вставки.

Oracle Решение 2.3.4.b

```
1   -- Отключение триггера, обеспечивающего автоинкремент первичного ключа:
2   ALTER TRIGGER "library"."TRG_genres_g_id" DISABLE;
3
4   -- Слияние данных:
5   MERGE INTO "library"."genres" "destination"
6   USING "hude_library"."genres" "source"
7   ON ("destination"."g_id" = "source"."g_id")
8   WHEN MATCHED THEN
9     UPDATE SET "destination"."g_name" =
10          CONCAT("destination"."g_name", N' [OLD]')
11  WHEN NOT MATCHED THEN
12    INSERT ("g_id",
13            "g_name")
14    VALUES ("source"."g_id",
15            "source"."g_name");
16
17  -- Включение триггера, обеспечивающего автоинкремент первичного ключа:
18  ALTER TRIGGER "library"."TRG_genres_g_id" ENABLE;
```

Если по неким причинам вы не можете соединиться с СУБД от имени пользователя, имеющего доступ к обеим интересующим вас схемам, можно пойти по следующему пути (такие варианты не были рассмотрены для MySQL и MS SQL Server, т.к. в этих СУБД нет поддержки некоторых возможностей, а сложность обходных решений выходит за рамки данной книги — на порядки проще создать пользователя с правами доступа к обеим базам данных (схемам)).

Строки 6-23 представленного ниже большого набора запросов аналогичны только что рассмотренному решению, а весь остальной код (строки 1-4, 25-37) нужен для того, чтобы обеспечить возможность одновременного доступа к данным в двух разных схемах.

Строки 2-4 отвечают за создание и открытие соединения к схеме-источнику.

В строке 26 команда `COMMIT` нужна потому, что без неё при закрытии соединения мы рискуем потерять часть данных.

В строках 29 и 32-34 представлены два варианта закрытия соединения. Как правило, срабатывает первые вариант, но если не сработал — есть второй.

В строке 37 ранее созданное соединение уничтожается.

Oracle Решение 2.3.4.b (работа от имени двух пользователей)

```
-- Создание соединения со схемой-источником:
CREATE DATABASE LINK "huge"
CONNECT TO "логин" IDENTIFIED BY "пароль"
USING 'localhost:1521/xe';

-- Отключение триггера, обеспечивающего автоинкремент первичного ключа:
ALTER TRIGGER "TRG_genres_g_id" DISABLE;

-- Слияние данных:
MERGE INTO "genres" "destination"
USING "genres"@"huge" "source"
ON ("destination"."g_id" = "source"."g_id")
WHEN MATCHED THEN
  UPDATE SET "destination"."g_name" =
             CONCAT("destination"."g_name", N' [OLD]')
WHEN NOT MATCHED THEN
  INSERT ("g_id",
          "g_name")
  VALUES ("source"."g_id",
          "source"."g_name");

-- Включение триггера, обеспечивающего автоинкремент первичного ключа:
ALTER TRIGGER "TRG_genres_g_id" ENABLE;

-- Явное подтверждение сохранения всех изменений:
COMMIT;

-- Закрытие соединения со схемой источником:
ALTER SESSION CLOSE DATABASE LINK "huge";

-- Если не помогло ALTER SESSION CLOSE ... :
BEGIN
DBMS_SESSION.CLOSE_DATABASE_LINK('huge');
END;

-- Удаление соединения со схемой-источником:
DROP DATABASE LINK "huge";
```

Задание 2.3.4.TSK.A: добавить в базу данных жанры «Политика», «Психология», «История».

Задание 2.3.4.TSK.B: скопировать (без повторений) в базу данных «Библиотека» содержимое таблицы **`subscribers`** из базы данных «Большая библиотека»; в случае совпадения первичных ключей добавить к существующему имени читателя слово « [OLD]».

2.3.5. ПРИМЕР 25: ИСПОЛЬЗОВАНИЕ УСЛОВИЙ ПРИ МОДИФИКАЦИИ ДАННЫХ

Задача 2.3.5.a{207}: добавить в базу данных информацию о том, что читатель с идентификатором 4 взял в библиотеке книги с идентификаторами 2 и 3 1 февраля 2015-го года, и планировал вернуть их не позднее 20-го июля 2015-го года; если текущая дата меньше 20-го июля 2015-го года, отметить выдачи как невозвращённые, если больше — как возвращённые.

Задача 2.3.5.b{210}: изменить даты возврата всех книг на «два месяца от текущего дня», если книга не возвращена и у соответствующего читателя сейчас на руках больше двух книг, и на «месяц от текущего дня» в противном случае (книга возвращена или у соответствующего читателя на руках сейчас не более двух книг).

Задача 2.3.5.c{212}: обновить все имена читателей, добавив в конец в квадратных скобках количество невозвращённых книг (например, « [3]») и слова « [RED]», « [YELLOW]», « [GREEN]», соответственно, если у читателя сейчас на руках более пяти книг, от трёх до пяти, менее трёх.

Ожидаемые результаты представлены, исходя из предположения, что вы используете «чистую» копию базы данных «Библиотека», не изменённую предыдущими примерами модификации данных, но каждая задача данного примера работает с данными, изменёнными предыдущей задачей.

Ожидаемый результат 2.3.5.a: в таблице `subscriptions` должны появиться две следующие записи.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
101	4	2	2015-02-01	2015-07-20	Y
102	4	3	2015-02-01	2015-07-20	Y

Ожидаемый результат 2.3.5.b.

				Было	Стало	
sb_id	**sb_subscriber**	**sb_book**	**sb_start**	**sb_finish**	**sb_finish**	**sb_is_active**
2	1	1	2011-01-12	2011-02-12	2011-03-12	N
3	3	3	2012-05-17	2012-07-17	2012-09-17	Y
42	1	2	2012-06-11	2012-08-11	2012-09-11	N
57	4	5	2012-06-11	2012-08-11	2012-09-11	N
61	1	7	2014-08-03	2014-10-03	2014-11-03	N
62	3	5	2014-08-03	2014-10-03	2014-12-03	Y
86	3	1	2014-08-03	2014-09-03	2014-11-03	Y
91	4	1	2015-10-07	2015-03-07	2015-05-07	Y
95	1	4	2015-10-07	2015-11-07	2015-12-07	N
99	4	4	2015-10-08	2025-11-08	2026-01-08	Y
100	1	3	2011-01-12	2011-02-12	2011-03-12	N
101	4	2	2015-02-01	2015-07-20	2015-09-20	Y
102	4	3	2015-02-01	2015-07-20	2015-09-20	Y

Ожидаемый результат 2.3.5.c.

s_id	s_name
1	Иванов И.И. [0] [GREEN]
2	Петров П.П. [0] [GREEN] [0]
3	Сидоров С.С. [3] [YELLOW]
4	Сидоров С.С. [4] [YELLOW]

Решения всех задач данного примера построены на использовании выражения CASE, позволяющего в рамках одного запроса учитывать несколько вариантов данных или несколько вариантов поведения СУБД.

Решение 2.3.5.a[{206}].

MySQL Решение 2.3.5.a

```
INSERT INTO `subscriptions`
            (
             `sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`
            )
            VALUES
            (
             NULL,
             4,
             2,
             '2015-02-01',
             '2015-07-20',
             CASE
              WHEN CURDATE() < '2015-07-20'
              THEN (SELECT 'N')
              ELSE (SELECT 'Y')
             END
            ),
            (
             NULL,
             4,
             3,
             '2015-02-01',
             '2015-07-20',
             CASE
              WHEN CURDATE() < '2015-07-20'
              THEN (SELECT 'N')
              ELSE (SELECT 'Y')
             END
            )
```

В строках 17-21 и 28-33 выражение **CASE** позволяет на уровне СУБД определить текущую дату, сравнить её с заданной и прийти к выводу о том, какое значение (Y или N) должно принять поле **sb_is_active**.

В MySQL срабатывает и упрощённый синтаксис вида **THEN 'N'** вместо **THEN (SELECT 'N')**. Это не ошибка, но такой код намного сложнее читается и воспринимается, т.к. в контексте SQL привычным способом получения некоего значения является именно использование **SELECT**.

Решение для MS SQL Server реализуется аналогичным образом:

MS SQL Решение 2.3.5.a

```
1   INSERT INTO [subscriptions]
2               (
3                [sb_subscriber],
4                [sb_book],
5                [sb_start],
6                [sb_finish],
7                [sb_is_active]
8               )
9               VALUES
10              (
11               4,
12               2,
13               CONVERT(date, '2015-02-01'),
14               CONVERT(date, '2015-07-20'),
15               CASE
16                WHEN CONVERT(date, GETDATE()) < CONVERT(date, '2015-07-20')
17                THEN (SELECT N'N')
18                ELSE (SELECT N'Y')
19               END
20              ),
21              (
22               4,
23               3,
24               CONVERT(date, '2015-02-01'),
25               CONVERT(date, '2015-07-20'),
26               CASE
27                WHEN CONVERT(date, GETDATE()) < CONVERT(date, '2015-07-20')
28                THEN (SELECT N'N')
29                ELSE (SELECT N'Y')
30               END
31              )
```

Как и в MySQL, в MS SQL Server можно использовать синтаксис вида **THEN 'N'** вместо **THEN (SELECT 'N')**.

Решение для Oracle реализуется аналогичным образом.

Как и в MySQL, и в MS SQL Server, в Oracle можно использовать синтаксис вида **THEN 'N'** вместо **THEN (SELECT 'N' FROM "DUAL")**.

В данной конкретной задаче нет принципиальной разницы в использовании синтаксиса вида `THEN 'N'` или вида `THEN (SELECT 'N')` — оптимизатор запросов в СУБД всё равно «поймёт», что выбирается константа, и заменит её значением всё выражение `SELECT`.

Но гораздо чаще потребуется не выбирать значение константы, а выполнять полноценный запрос — именно потому для сохранения единого стиля рекомендуется и в этом простом случае писать `THEN (SELECT 'N')`.

Oracle Решение 2.3.5.a

```
INSERT ALL
       INTO "subscriptions"
            (
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active"
            )
            VALUES
            (
             4,
             2,
             TO_DATE('2015-02-01', 'YYYY-MM-DD'),
             TO_DATE('2015-07-20', 'YYYY-MM-DD'),
             CASE
              WHEN TRUNC(SYSDATE) < TO_DATE('2015-07-20', 'YYYY-MM-DD')
              THEN (SELECT 'N' FROM "DUAL")
              ELSE (SELECT 'Y' FROM "DUAL")
             END
            )
       INTO "subscriptions"
            (
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active"
            )
            VALUES
            (
             4,
             3,
             TO_DATE('2015-02-01', 'YYYY-MM-DD'),
             TO_DATE('2015-07-20', 'YYYY-MM-DD'),
             CASE
              WHEN TRUNC(SYSDATE) < TO_DATE('2015-07-20', 'YYYY-MM-DD')
              THEN (SELECT 'N' FROM "DUAL")
              ELSE (SELECT 'Y' FROM "DUAL")
             END
            )
SELECT 1 FROM "DUAL";
```

Решение 2.3.5.b[{206}].

В отличие от предыдущей задачи, где условие было крайне тривиальным и наглядным, здесь придётся решать две проблемы:

- Создать достаточно сложное составное условие, опирающееся на коррелирующий подзапрос.
- «Убедить» MySQL разрешить использование в одном запросе одной и той же таблицы (**`subscriptions`**) как для обновления, так и для чтения — MySQL запрещает такие ситуации.

За решение второй проблемы отвечают строки 5-8 запроса: мы «оборачиваем» операцию чтения из обновляемой таблицы в подзапрос, что позволяет обойти налагаемое MySQL ограничение. Это довольно распространённое, но в то же время опасное решение, т.к. оно не только снижает производительность, но и потенциально может привести к некорректной работе некоторых запросов — причём универсального ответа на вопрос «сработает или нет» не существует: нужно проверять в конкретной ситуации.

MySQL Решение 2.3.5.b

```
1   UPDATE `subscriptions` AS `ext`
2   SET    `sb_finish` = CASE
3                          WHEN `sb_is_active` = 'Y'
4                          AND EXISTS (SELECT `int`.`sb_subscriber`
5                                      FROM   (SELECT `sb_subscriber`,
6                                                     `sb_book`,
7                                                     `sb_is_active`
8                                              FROM   `subscriptions`) AS `int`
9                                      WHERE  `int`.`sb_is_active` = 'Y'
10                                            AND `int`.`sb_subscriber` =
11                                                `ext`.`sb_subscriber`
12                                     GROUP  BY `int`.`sb_subscriber`
13                                     HAVING COUNT(`int`.`sb_book`) > 2)
14                         THEN (SELECT DATE_ADD(`sb_finish`, INTERVAL 2 MONTH))
15                         ELSE (SELECT DATE_ADD(`sb_finish`, INTERVAL 1 MONTH))
16                        END
```

Составное условие, являющееся ядром решения поставленной задачи, представлено в строках 3-13.

Его первая часть (строка 3) проста: мы определяем значение поля.

Его вторая часть (строки 4-13) представляет собой коррелирующий подзапрос, результаты которого передаются в функцию **`EXISTS`**, т.е. нас интересует сам факт того, вернул ли подзапрос хотя бы одну строку, или нет.

Необходимость ещё одного вложенного запроса в строках 5-8 только что была рассмотрена — так мы обходим ограничение MySQL на чтение из обновляемой таблицы.

В остальном — это классический коррелирующий подзапрос. Если выполнить его отдельно, вручную подставляя значения идентификатора читателя, получится следующая картина:

MySQL Решение 2.3.5.b (модифицированный фрагмент)

```
1   SELECT `int`.`sb_subscriber`
2   FROM   (SELECT `sb_subscriber`,
3                  `sb_book`,
4                  `sb_is_active`
5           FROM   `subscriptions`) AS `int`
6   WHERE  `int`.`sb_is_active` = 'Y'
7          AND `int`.`sb_subscriber` = {id читателя из основной части запроса}
8   GROUP  BY `int`.`sb_subscriber`
9   HAVING COUNT(`int`.`sb_book`) > 2
```

Для читателей с идентификаторами 1 и 2 этот подзапрос вернёт ноль строк, а для читателей с идентификаторами 3 и 4 результат будет непустым.

Строки 14 и 15 описывают желаемое поведение СУБД в случае, когда составное условие соответственно выполнилось и не выполнилось.

MS SQL Решение 2.3.5.b

```
1   UPDATE [subscriptions]
2   SET    [sb_finish] = CASE
3                           WHEN [sb_is_active] = 'Y'
4                           AND EXISTS (SELECT [int].[sb_subscriber]
5                                       FROM   [subscriptions] AS [int]
6                                       WHERE [int].[sb_is_active] = 'Y'
7                                         AND [int].[sb_subscriber] =
8                                             [subscriptions].[sb_subscriber]
9                                       GROUP  BY [int].[sb_subscriber]
10                                      HAVING COUNT([int].[sb_book]) > 2)
11                          THEN (SELECT DATEADD(month, 2, [sb_finish]))
12                          ELSE (SELECT DATEADD(month, 1, [sb_finish]))
13                        END
```

Решение для MS SQL сервер построено на той же логике. Оно даже чуть проще в реализации, т.к. MS SQL Server не запрещает в одном запросе обновлять данные в таблице и читать данные из этой же таблицы — поэтому нет необходимости «оборачивать» выборку в строке 5 в подзапрос.

Решение для Oracle отличается от решения для MS SQL Server только синтаксисом увеличения даты на нужное количество месяцев (строки 11-12). Как и MS SQL Server, Oracle не запрещает в одном запросе обновлять данные в таблице и читать данные из этой же таблицы.

Oracle Решение 2.3.5.b

```
1   UPDATE "subscriptions"
2   SET    "sb_finish" = CASE
3                           WHEN "sb_is_active" = 'Y'
4                           AND EXISTS (SELECT "int"."sb_subscriber"
5                                       FROM   "subscriptions" "int"
6                                       WHERE "int"."sb_is_active" = 'Y'
7                                         AND "int"."sb_subscriber" =
8                                             "subscriptions"."sb_subscriber"
9                                       GROUP  BY "int"."sb_subscriber"
10                                      HAVING COUNT("int"."sb_book") > 2)
11                          THEN (SELECT ADD_MONTHS("sb_finish", 2) FROM "DUAL")
12                          ELSE (SELECT ADD_MONTHS("sb_finish", 1) FROM "DUAL")
13                        END
```

Решение 2.3.5.c[{206}].

В отличие от предыдущих задач данного примера, здесь для каждой СУБД решения будет принципиально разными. Самый универсальный вариант реализован для Oracle (его можно с минимальными синтаксическими доработками реализовать и в MySQL, и в MS SQL Server).

MySQL Решение 2.3.5.c

```
1   UPDATE `subscribers`
2   SET    `s_name` = CONCAT(`s_name`,
3          (
4           SELECT `postfix`
5           FROM   (SELECT `s_id`,
6                          @x := IFNULL
7                                (
8                                 (SELECT COUNT(`sb_book`)
9                                  FROM  `subscriptions` AS `int`
10                                 WHERE `int`.`sb_is_active` = 'Y'
11                                       AND `int`.`sb_subscriber` =
12                                           `ext`.`sb_subscriber`
13                                 GROUP  BY `int`.`sb_subscriber`), 0
14                               ),
15                         CASE
16                          WHEN @x > 5 THEN
17                           (SELECT CONCAT(' [', @x, '] [RED]'))
18                          WHEN @x >= 3 AND @x <= 5 THEN
19                           (SELECT CONCAT(' [', @x, '] [YELLOW]'))
20                          ELSE (SELECT CONCAT(' [', @x, '] [GREEN]'))
21                         END AS `postfix`
22                  FROM   `subscribers`
23                  LEFT JOIN `subscriptions` AS `ext`
24                       ON `s_id` = `sb_subscriber`
25                  GROUP  BY `sb_subscriber`) AS `data`
26          WHERE  `data`.`s_id` = `subscribers`.`s_id`)
27         )
```

Начнём рассмотрение с наиболее глубоко вложенного подзапроса (строки 6-14). Это — коррелирующий подзапрос, выполняющий подсчёт книг, находящихся на руках у того читателя, который в данный момент анализируется подзапросом в строках 5-25. Результат выполнения подзапроса помещается в переменную **@x**:

s_id	@x
2	0
1	0
3	3
4	4

Выражение **CASE** в строках 15-21 на основе значения переменной **@x** определяет суффикс, который необходимо добавить к имени читателя:

s_id	@x	postfix
2	0	[0] [GREEN]
1	0	[0] [GREEN]
3	3	[3] [YELLOW]
4	8	[4] [YELLOW]

Коррелирующий подзапрос в строках 3-27 передаёт в функцию **CONCAT** суффикс, соответствующий идентификатору читателя, имя которого обновляется (в строке 26 производится соответствующее сравнение). Имя читателя заменяется на результат работы функции **CONCAT**, и таким образом получается итоговый результат.

Для лучшего понимания данного решения приведём модифицированный запрос, который ничего не обновляет, но показывает всю необходимую информацию:

MySQL Решение 2.3.5.c (модифицированный запрос)

```
1  SELECT `s_id`,
2         `s_name`,
3         @x := IFNULL
4               (
5                (SELECT COUNT(`sb_book`)
6                 FROM  `subscriptions` AS `int`
7                 WHERE `int`.`sb_is_active` = 'Y'
8                   AND `int`.`sb_subscriber` =
9                       `ext`.`sb_subscriber`
10                GROUP  BY `int`.`sb_subscriber`), 0
11              ),
12        CASE
13         WHEN @x > 5 THEN
14          (SELECT CONCAT(' [', @x, '] [RED]'))
15         WHEN @x >= 3 AND @x <= 5 THEN
16          (SELECT CONCAT(' [', @x, '] [YELLOW]'))
17         ELSE (SELECT CONCAT(' [', @x, '] [GREEN]'))
18        END AS `postfix`
19 FROM   `subscribers`
20 LEFT JOIN `subscriptions` AS `ext`
21        ON `s_id` = `sb_subscriber`
22 GROUP  BY `sb_subscriber`
```

Результат выполнения этого модифицированного запроса:

s_id	s_name	@x	postfix
2	Петров П.П.	0	[0] [GREEN]
1	Иванов И.И.	0	[0] [GREEN]
3	Сидоров С.С.	3	[3] [YELLOW]
4	Сидоров С.С.	4	[4] [YELLOW]

MS SQL Server не позволяет использовать переменные так же гибко, как MySQL — существуют ограничения на одновременную выборку данных и изменение значения переменной, а также требования по предварительному объявлению переменных.

Однако MS SQL Server поддерживает общие табличные выражения, куда мы и перенесём логику определения того, сколько книг в настоящий момент находится на руках у каждого читателя (строки 1-10). В отличие от решения для MySQL здесь вместо коррелирующего подзапроса используется подзапрос как источник данных (строки 4-8), возвращающий информацию только о книгах, находящихся на руках у читателей.

MS SQL Решение 2.3.5.c

```
1   WITH [prepared_data]
2        AS (SELECT [s_id], COUNT([sb_subscriber]) AS [x]
3            FROM   [subscribers]
4                   LEFT JOIN (
5                              SELECT [sb_subscriber]
6                              FROM [subscriptions]
7                              WHERE [sb_is_active] = 'Y'
8                              ) AS [active_only]
9                           ON [s_id] = [sb_subscriber]
10           GROUP  BY [s_id])
11  UPDATE [subscribers]
12  SET    [s_name] =
13                 (SELECT
14                   CASE
15                    WHEN [x] > 5
16                     THEN (SELECT CONCAT([s_name], ' [', [x], '] [RED]'))
17                    WHEN [x] >= 3 AND [x] <= 5
18                     THEN (SELECT CONCAT([s_name], ' [', [x], '] [YELLOW]'))
19                    ELSE (SELECT CONCAT([s_name], ' [', [x], '] [GREEN]'))
20                   END
21                  FROM   [prepared_data]
22                  WHERE  [subscribers].[s_id] = [prepared_data].[s_id])
```

Результат работы общего табличного выражения таков:

s_id	x
1	0
2	0
3	3
4	4

Коррелирующий подзапрос в строках 13-22 на основе значения колонки **x** формирует значение суффикса имени соответствующего читателя. Так получается итоговый результат.

Если есть потребность избавиться от коррелирующего подзапроса в строках 13-22, можно переписать основную часть решения (общее табличное выражение остаётся таким же) с использованием **JOIN**.

MS SQL Решение 2.3.5.c (вариант решения с JOIN вместо подзапроса)

```
1   WITH [prepared_data]
2        AS (SELECT [s_id], COUNT([sb_subscriber]) AS [x]
3            FROM   [subscribers]
4                   LEFT JOIN (
5                             SELECT [sb_subscriber]
6                             FROM [subscriptions]
7                             WHERE [sb_is_active] = 'Y'
8                             ) AS [active_only]
9                          ON [s_id] = [sb_subscriber]
10           GROUP  BY [s_id])
11  UPDATE [subscribers]
12  SET    [s_name] =
13                 (CASE
14                    WHEN [x] > 5
15                     THEN (SELECT CONCAT([s_name], ' [', [x], '] [RED]'))
16                    WHEN [x] >= 3 AND [x] <= 5
17                     THEN (SELECT CONCAT([s_name], ' [', [x], '] [YELLOW]'))
18                    ELSE (SELECT CONCAT([s_name], ' [', [x], '] [GREEN]'))
19                   END)
20  FROM [subscribers]
21  JOIN [prepared_data]
22    ON [subscribers].[s_id] = [prepared_data].[s_id]
```

Такой вариант решения (с использованием **JOIN** в контексте **UPDATE**) является самым оптимальным, но в то же время и наименее привычным (особенно для начинающих).

Oracle не поддерживает только что рассмотренное использование **JOIN** в контексте **UPDATE**, потому от коррелирующего подзапроса в строках 2-20 избавиться не удастся.

Также Oracle не поддерживает использование общих табличных выражений в контексте **UPDATE**, что приводит к необходимости переноса подготовки данных в подзапрос (строки 11-19).

Oracle Решение 2.3.5.c

```
1   UPDATE "subscribers"
2   SET    "s_name" =
3    (SELECT
4         CASE
5          WHEN "x" > 5
6           THEN (SELECT "s_name"  || ' [' || "x" || '] [RED]' FROM "DUAL")
7          WHEN "x" >= 3 AND "x" <= 5
8           THEN (SELECT "s_name" || ' [' || "x" || '] [YELLOW]' FROM "DUAL")
9          ELSE (SELECT "s_name" || ' [' || "x" || '] [GREEN]' FROM "DUAL")
10        END
11    FROM   (SELECT "s_id",
12                   COUNT("sb_subscriber") AS "x"
13            FROM   "subscribers"
14            LEFT   JOIN
15                      (SELECT "sb_subscriber"
16                       FROM "subscriptions"
17                       WHERE "sb_is_active" = 'Y')
18                   ON "s_id" = "sb_subscriber"
19            GROUP  BY "s_id") "prepared_data"
20   WHERE  "subscribers"."s_id" = "prepared_data"."s_id")
```

Ещё одно ограничение Oracle проявляется в том, что здесь функция **CONCAT** может принимать только два параметра (а нам нужно передать четыре), но мы можем обойти это ограничение с использованием оператора конкатенации строк **||**.

Ранее было отмечено, что решение для Oracle является самым универсальным и может быть легко перенесено в другие СУБД. Продемонстрируем это.

MySQL Решение 2.3.5.c (получено на основе решения для Oracle)

```
UPDATE `subscribers`
SET    `s_name` =
 (SELECT
   CASE
    WHEN `x` > 5 THEN
     (SELECT CONCAT(`s_name`, ' [', `x`, '] [RED]'))
    WHEN `x` >= 3 AND `x` <= 5 THEN
     (SELECT CONCAT(`s_name`, ' [', `x`, '] [YELLOW]'))
   ELSE (SELECT CONCAT(`s_name`, ' [', `x`, '] [GREEN]'))
  END
  FROM   (SELECT `s_id`,
                 COUNT(`sb_subscriber`) AS `x`
          FROM   `subscribers`
          LEFT   JOIN
                    (SELECT `sb_subscriber`
                     FROM `subscriptions`
                     WHERE `sb_is_active` = 'Y') AS `active_only`
                 ON `s_id` = `sb_subscriber`
          GROUP  BY `s_id`) AS `prepared_data`
 WHERE  `subscribers`.`s_id` = `prepared_data`.`s_id`)
```

MS SQL Решение 2.3.5.c (получено на основе решения для Oracle)

```
UPDATE [subscribers]
SET    [s_name] =
 (SELECT
   CASE
    WHEN [x] > 5
     THEN (SELECT CONCAT([s_name], ' [', [x], '] [RED]'))
    WHEN [x] >= 3 AND [x] <= 5
   THEN (SELECT CONCAT([s_name], ' [', [x], '] [YELLOW]'))
    ELSE (SELECT CONCAT([s_name], ' [', [x], '] [GREEN]'))
  END
  FROM   (SELECT [s_id],
                 COUNT([sb_subscriber]) AS [x]
          FROM   [subscribers]
          LEFT   JOIN
                    (SELECT [sb_subscriber]
                     FROM [subscriptions]
                     WHERE [sb_is_active] = 'Y') AS [active_only]
                 ON [s_id] = [sb_subscriber]
          GROUP  BY [s_id]) AS [prepared_data]
 WHERE  [subscribers].[s_id] = [prepared_data].[s_id])
```

Задание 2.3.5.TSK.A: добавить в базу данных читателей с именами «Сидоров С.С.», «Иванов И.И.», «Орлов О.О.»; если читатель с таким именем уже существует, добавить в конец имени нового читателя порядковый номер в квадратных скобках (например, если при добавлении читателя «Сидоров С.С.» выяснится, что в базе данных уже есть четыре таких читателя, имя добавляемого должно превратиться в «Сидоров С.С. [5]»).

Задание 2.3.5.TSK.B: обновить все имена авторов, добавив в конец имени « [+]», если в библиотеке есть более трёх книг этого автора, или добавив в конец имени « [-]» в противном случае.

ИСПОЛЬЗОВАНИЕ ПРЕДСТАВЛЕНИЙ

ВЫБОРКА ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ПРЕДСТАВЛЕНИЙ

3.1.1. ПРИМЕР 26: ВЫБОРКА ДАННЫХ С ИСПОЛЬЗОВАНИЕМ НЕКЭШИРУЮЩИХ ПРЕДСТАВЛЕНИЙ

Классические представления не содержат в себе данных, они лишь являются способом обращения к реальным таблицам базы данных. Альтернативой являются т.н. кэширующие (материализованные, индексированные) представления, которые будут рассмотрены в следующем разделе{222}.

Задача 3.1.1.a{218}: упростить использование решения задачи 2.2.9.d{135} так, чтобы для получения нужных данных не приходилось использовать представленные в решении{144} объёмные запросы.

Задача 3.1.1.b{221}: создать представление, позволяющее получать список авторов и количество имеющихся в библиотеке книг по каждому автору, но отображающее только таких авторов, по которым имеется более одной книги.

Ожидаемый результат 3.1.1.a.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить ожидаемый результат задачи 2.2.9.d{135}, т.е.:

s_id	s_name	b_name
1	Иванов И.И.	Евгений Онегин
3	Сидоров С.С.	Основание и империя
4	Сидоров С.С.	Язык программирования C++

Ожидаемый результат 3.1.1.b.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить результат следующего вида (ни при каких условиях здесь не должны отображаться авторы, по которым в библиотеке зарегистрировано менее двух книг):

a_id	a_name	books_in_library
6	Б. Страуструп	2
7	А.С. Пушкин	2

Решение 3.1.1.a{217}.

Построим решение этой задачи для MySQL на основе следующего уже написанного ранее (см. решение 2.2.9.d{144}) запроса:

MySQL Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
SELECT `s_id`,
       `s_name`,
       `b_name`
FROM   (SELECT `subscriptions`.`sb_subscriber`,
               `sb_book`
        FROM   `subscriptions`
               JOIN (SELECT `subscriptions`.`sb_subscriber`,
                            MIN(`sb_id`) AS `min_sb_id`
                     FROM   `subscriptions`
                            JOIN (SELECT `sb_subscriber`,
                                         MIN(`sb_start`) AS `min_sb_start`
                                  FROM   `subscriptions`
                                  GROUP  BY `sb_subscriber`)
                                  AS `step_1`
                              ON `subscriptions`.`sb_subscriber` =
                                 `step_1`.`sb_subscriber`
                                 AND `subscriptions`.`sb_start` =
                                     `step_1`.`min_sb_start`
                     GROUP  BY `subscriptions`.`sb_subscriber`,
                               `min_sb_start`)
                     AS `step_2`
                 ON `subscriptions`.`sb_id` = `step_2`.`min_sb_id`)
        AS `step_3`
       JOIN `subscribers`
         ON `sb_subscriber` = `s_id`
       JOIN `books`
         ON `sb_book` = `b_id`
```

В идеале нам бы хотелось просто построить представление на этом запросе. Но MySQL младше версии 5.7.7 не позволяет создавать представления, опирающиеся на запросы, в секции `FROM` которых есть подзапросы. К сожалению, у нас таких подзапроса аж три — `` `step_1` ``, `` `step_2` ``, `` `step_3` ``.

Для обхода существующего ограничения есть не очень элегантное, но очень простое решение — для каждого из таких подзапросов надо построить своё отдельное представление. Тогда в секции `FROM` будет не обращение к подзапросу (что запрещено), а обращение к представлению (что разрешено).

MySQL Решение 3.1.1.a

```
-- Замена первого подзапроса представлением:
CREATE OR REPLACE VIEW `first_book_step_1`
AS
  SELECT `sb_subscriber`,
         MIN(`sb_start`) AS `min_sb_start`
  FROM   `subscriptions`
  GROUP  BY `sb_subscriber`

-- Замена второго подзапроса представлением:
CREATE OR REPLACE VIEW `first_book_step_2`
AS
  SELECT `subscriptions`.`sb_subscriber`,
         MIN(`sb_id`) AS `min_sb_id`
  FROM   `subscriptions`
         JOIN `first_book_step_1`
           ON `subscriptions`.`sb_subscriber` =
              `first_book_step_1`.`sb_subscriber`
             AND `subscriptions`.`sb_start` =
                 `first_book_step_1`.`min_sb_start`
  GROUP  BY `subscriptions`.`sb_subscriber`,
            `min_sb_start`
-- Замена третьего подзапроса представлением:
CREATE OR REPLACE VIEW `first_book_step_3`
AS
  SELECT `subscriptions`.`sb_subscriber`,
         `sb_book`
  FROM   `subscriptions`
         JOIN `first_book_step_2`
           ON `subscriptions`.`sb_id` = `first_book_step_2`.`min_sb_id`

-- Создание основного представления:
CREATE OR REPLACE VIEW `first_book`
AS
  SELECT `s_id`,
         `s_name`,
         `b_name`
  FROM   `subscribers`
         JOIN `first_book_step_3`
           ON `sb_subscriber` = `s_id`
         JOIN `books`
           ON `sb_book` = `b_id`
```

Обратите внимание, что каждое следующее представление в этом наборе опирается на предыдущее.

Теперь для получения данных достаточно выполнить запрос **`SELECT * FROM `first_book``**, что и требовалось по условию задачи.

Решение для MS SQL Server также построим на основе ранее написанного запроса (см. решение 2.2.9.d{144}):

MS SQL Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
WITH [step_1]
     AS (SELECT [sb_subscriber],
                MIN([sb_start]) AS [min_sb_start]
         FROM   [subscriptions]
         GROUP  BY [sb_subscriber]),
```

MS SQL Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении) (продолжение)

```
WITH [step_1]
     AS (SELECT [sb_subscriber],
                MIN([sb_start]) AS [min_sb_start]
         FROM   [subscriptions]
         GROUP  BY [sb_subscriber]),
     [step_2]
     AS (SELECT [subscriptions].[sb_subscriber],
                MIN([sb_id]) AS [min_sb_id]
         FROM   [subscriptions]
                JOIN [step_1]
                  ON [subscriptions].[sb_subscriber] =
                             [step_1].[sb_subscriber]
                     AND [subscriptions].[sb_start] =
                             [step_1].[min_sb_start]
         GROUP  BY [subscriptions].[sb_subscriber],
                   [min_sb_start]),
     [step_3]
     AS (SELECT [subscriptions].[sb_subscriber],
                [sb_book]
         FROM   [subscriptions]
                JOIN [step_2]
                  ON [subscriptions].[sb_id] = [step_2].[min_sb_id])
SELECT [s_id],
       [s_name],
       [b_name]
FROM   [step_3]
       JOIN [subscribers]
         ON [sb_subscriber] = [s_id]
       JOIN [books]
         ON [sb_book] = [b_id]
```

Поскольку в MS SQL Server нет характерных для MySQL ограничений на запросы, на которых строится представление, конечное решение задачи получается добавлением в начало запроса одной строки:

MS SQL Решение 3.1.1.a

```
CREATE VIEW [first_book] AS
{текст исходного запроса, который мы «прячем» в представлении}
```

Теперь для получения данных достаточно выполнить запрос **`SELECT * FROM [first_book]`**, что и требовалось по условию задачи.

Решение для Oracle также построим на основе ранее написанного запроса (см. решение 2.2.9.d{144}):

Oracle Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
WITH "step_1"
     AS (SELECT "sb_subscriber",
                MIN("sb_start") AS "min_sb_start"
         FROM   "subscriptions"
         GROUP  BY "sb_subscriber"),
```

Oracle Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении) (продолжение)

```
      "step_2"
      AS (SELECT "subscriptions"."sb_subscriber",
                 MIN("sb_id") AS "min_sb_id"
          FROM   "subscriptions"
                 JOIN "step_1"
                   ON "subscriptions"."sb_subscriber" =
                              "step_1"."sb_subscriber"
                      AND "subscriptions"."sb_start" =
                              "step_1"."min_sb_start"
          GROUP  BY "subscriptions"."sb_subscriber",
                    "min_sb_start"),
      "step_3"
      AS (SELECT "subscriptions"."sb_subscriber",
                 "sb_book"
          FROM   "subscriptions"
                 JOIN "step_2"
                   ON "subscriptions"."sb_id" = "step_2"."min_sb_id")
SELECT "s_id",
       "s_name",
       "b_name"
FROM   "step_3"
       JOIN "subscribers"
         ON "sb_subscriber" = "s_id"
       JOIN "books"
         ON "sb_book" = "b_id"
```

Поскольку в Oracle, как и в MS SQL Server нет характерных для MySQL ограничений на запросы, на которых строится представление, конечное решение задачи получается добавлением в начало запроса одной строки:

Oracle Решение 3.1.1.a

```
CREATE OR REPLACE VIEW "first_book" AS
{текст исходного запроса, который мы «прячем» в представлении}
```

Теперь для получения данных достаточно выполнить запрос `SELECT * FROM "first_book"`, что и требовалось по условию задачи.

Решение 3.1.1.b[217].

Решение этой задачи идентично во всех трёх СУБД и сводится к написанию запроса, отображающего данные об авторах и количестве их книг в библиотеке с учётом указанного в задаче условия: таких книг должно быть больше одной. Затем на полученном запросе строится представление.

Имя представления в Oracle не может превышать 30 символов, потому там слово `with` в имени представления пришлось сократить до одной буквы `w`.

MySQL Решение 3.1.1.b

```
CREATE OR REPLACE VIEW `authors_with_more_than_one_book`
AS
  SELECT `a_id`,
         `a_name`,
         COUNT(`b_id`) AS `books_in_library`
  FROM   `authors`
         JOIN `m2m_books_authors` USING (`a_id`)
  GROUP  BY `a_id`
  HAVING `books_in_library` > 1
```

MS SQL Решение 3.1.1.b

```
1   CREATE VIEW [authors_with_more_than_one_book]
2   AS
3     SELECT [authors].[a_id],
4            [a_name],
5            COUNT([b_id]) AS [books_in_library]
6     FROM   [authors]
7            JOIN [m2m_books_authors]
8              ON [authors].[a_id] = [m2m_books_authors].[a_id]
9     GROUP  BY [authors].[a_id],
10              [a_name]
11    HAVING COUNT([b_id]) > 1
```

Oracle Решение 3.1.1.b

```
1   CREATE OR REPLACE VIEW "authors_w_more_than_one_book"
2   AS
3     SELECT "a_id",
4            "a_name",
5            COUNT("b_id") AS "books_in_library"
6     FROM   "authors"
7            JOIN "m2m_books_authors" USING ("a_id")
8     GROUP  BY "a_id", "a_name"
9     HAVING COUNT("b_id") > 1
```

Задание 3.1.1.TSK.A: упростить использование решения задачи 2.2.8.b{130} так, чтобы для получения нужных данных не приходилось использовать представленные в решении{133} объёмные запросы.

Задание 3.1.1.TSK.B: создать представление, позволяющее получать список читателей с количеством находящихся у каждого читателя на руках книг, но отображающее только таких читателей, по которым имеются задолженности, т.е. на руках у читателя есть хотя бы одна книга, которую он должен был вернуть до наступления текущей даты.

3.1.2. ПРИМЕР 27: ВЫБОРКА ДАННЫХ С ИСПОЛЬЗОВАНИЕМ КЭШИРУЮЩИХ ПРЕДСТАВЛЕНИЙ И ТАБЛИЦ

Кэширующие (материализованные, индексированные) представления в отличие от своих классических аналогов{217} формируют и сохраняют отдельный подготовленный набор данных. Поскольку такие представления поддерживаются не всеми СУБД и/или на них налагается ряд серьёзных ограничений, их аналог может быть реализован с помощью кэширующих или агрегирующих таблиц и триггеров{284}.

Использование решений, подобных представленным в данном примере, в реальной жизни может совершенно непредсказуемо повлиять на производительность — как резко увеличить её, так и очень сильно снизить. Каждый случай требует своего отдельного исследования. Потому представленные ниже задачи и их решения стоит воспринимать лишь как демонстрацию возможностей СУБД, а не как прямое руководство к действию.

Задача 3.1.2.a{223}: создать представление, ускоряющее получение информации о количестве экземпляров книг: имеющихся в библиотеке, выданных на руки, оставшихся в библиотеке.

Задача 3.1.2.b{240}: создать представление, ускоряющее получение всей информации из таблицы `subscriptions` в человекочитаемом виде (где идентификаторы читателей и книг заменены на имена и названия).

Ожидаемый результат 3.1.2.a.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить результат следующего вида:

total	given	rest
33	5	28

Ожидаемый результат 3.1.2.b.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить результат следующего вида:

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
2	Иванов И.И.	Евгений Онегин	2011-01-12	2011-02-12	N
42	Иванов И.И.	Сказка о рыбаке и рыбке	2012-06-11	2012-08-11	N
61	Иванов И.И.	Искусство программирования	2014-08-03	2014-10-03	N
95	Иванов И.И.	Психология программирования	2015-10-07	2015-11-07	N
100	Иванов И.И.	Основание и империя	2011-01-12	2011-02-12	N
3	Сидоров С.С.	Основание и империя	2012-05-17	2012-07-17	Y
62	Сидоров С.С.	Язык программирования C++	2014-08-03	2014-10-03	Y
86	Сидоров С.С.	Евгений Онегин	2014-08-03	2014-09-03	Y
57	Сидоров С.С.	Язык программирования C++	2012-06-11	2012-08-11	N
91	Сидоров С.С.	Евгений Онегин	2015-10-07	2015-03-07	Y
99	Сидоров С.С.	Психология программирования	2015-10-08	2025-11-08	Y

Решение 3.1.2.a{223}.

Традиционно мы начнём рассматривать первым решение для MySQL, и тут есть проблема: MySQL не поддерживает т.н. «кэширующие представления». Единственный способ добиться в данной СУБД необходимого результата — создать настоящую таблицу, в которой будут храниться нужные нам данные.

Эти данные придётся обновлять, для чего могут применяться различные подходы:

- однократное наполнение (для случая, когда исходные данные не меняются);
- периодическое обновление, например, с помощью хранимой процедуры (подходит для случая, когда мы можем позволить себе периодически получать не самую актуальную информацию);
- автоматическое обновление с помощью триггеров (позволяет в каждый момент времени получать актуальную информацию).

Мы будем реализовывать именно последний вариант — работу через триггеры (подробнее о триггерах см. в соответствующем разделе{284}). Этот подход также распадается на два возможных варианта решения: триггеры могут каждый раз обновлять все данные или реагировать только на поступившие изменения (что работает намного быстрее, но требует изначальной инициализации данных в агрегирующей / кэширующей таблице).

Итак, мы реализуем самый производительный (пусть и самый сложный) вариант — создадим агрегирующую таблицу, напишем запрос для инициализации её данных и создадим триггеры, реагирующие на изменения агрегируемых данных.

Создадим агрегирующую таблицу:

MySQL Решение 3.1.2.a (создание агрегирующей таблицы)

```
CREATE TABLE `books_statistics`
(
      `total` INTEGER UNSIGNED NOT NULL,
      `given` INTEGER UNSIGNED NOT NULL,
      `rest`  INTEGER UNSIGNED NOT NULL
)
```

Легко заметить, что в этой таблице нет первичного ключа. Он и не нужен, т.к. в ней предполагается хранить ровно одну строку.

В реальных приложениях обязательно должен быть механизм реакции на случаи, когда в подобных таблицах оказывается либо ноль строк, либо более одной строки. Обе такие ситуации потенциально могут привести к краху приложения или его некорректной работе.

Проинициализируем данные в созданной таблице:

MySQL Решение 3.1.2.a (очистка таблицы и инициализация данных)

```
-- Очистка таблицы:
TRUNCATE TABLE `books_statistics`;

-- Инициализация данных:
INSERT INTO `books_statistics`
            (`total`,
             `given`,
             `rest`)
SELECT IFNULL(`total`, 0),
       IFNULL(`given`, 0),
       IFNULL(`total` - `given`, 0) AS `rest`
FROM   (SELECT (SELECT SUM(`b_quantity`)
                FROM   `books`)              AS `total`,
               (SELECT COUNT(`sb_book`)
                FROM   `subscriptions`
                WHERE  `sb_is_active` = 'Y') AS `given`)
       AS `prepared_data`;
```

Напишем триггеры, модифицирующие данные в агрегирующей таблице. Агрегация происходит на основе информации, представленной в таблицах **books** и **subscriptions**, потому придётся создавать триггеры для обеих этих таблиц.

Изменения в таблице **books** влияют на поля **total** и **rest**, а изменения в таблице **subscriptions** — на поля **given** и **rest**. Данные могут измениться в результате всех трёх операций модификации данных — вставки, удаления, обновления — потому придётся создавать триггеры на всех трёх операциях.

Важно! В MySQL триггеры не активируются каскадными операциями, потому изменения в таблице **subscriptions**, вызванные удалением читателей, останутся «незаметными» для триггеров на этой таблице. В задании 3.1.2.TSK.D{249} вам предлагается доработать данное решение, устранив эту проблему.

MySQL Решение 3.1.2.a (триггеры для таблицы books)

```
-- Удаление старых версий триггеров
-- (удобно в процессе разработки и отладки):
DROP TRIGGER `upd_bks_sts_on_books_ins`;
DROP TRIGGER `upd_bks_sts_on_books_del`;
DROP TRIGGER `upd_bks_sts_on_books_upd`;

-- Переключение разделителя завершения запроса,
-- т.к. сейчас запросом будет создание триггера,
-- внутри которого есть свои, классические запросы:
DELIMITER $$

-- Создание триггера, реагирующего на добавление книг:
CREATE TRIGGER `upd_bks_sts_on_books_ins`
BEFORE INSERT
ON `books`
 FOR EACH ROW
  BEGIN
   UPDATE `books_statistics` SET
          `total` = `total` + NEW.`b_quantity`,
          `rest` = `total` - `given`;
  END;
$$
-- Создание триггера, реагирующего на удаление книг:
CREATE TRIGGER `upd_bks_sts_on_books_del`
BEFORE DELETE
ON `books`
 FOR EACH ROW
  BEGIN
   UPDATE `books_statistics` SET
          `total` = `total` - OLD.`b_quantity`,
          `given` = `given` - (SELECT COUNT(`sb_book`)
                               FROM   `subscriptions`
                               WHERE  `sb_book`=OLD.`b_id`
                                  AND `sb_is_active` = 'Y'),
          `rest` = `total` - `given`;
  END;
$$

```

MySQL Решение 3.1.2.a (триггеры для таблицы books) (продолжение)

```
39 -- Создание триггера, реагирующего на
40 -- изменение количества книг:
42 CREATE TRIGGER `upd_bks_sts_on_books_upd`
42 BEFORE UPDATE
43 ON `books`
44  FOR EACH ROW
45   BEGIN
46    UPDATE `books_statistics` SET
47           `total` = `total` - OLD.`b_quantity` + NEW.`b_quantity`,
48           `rest` = `total` - `given`;
49   END;
50 $$
51
52 -- Восстановление разделителя завершения запросов:
53 DELIMITER ;
```

Переключение разделителя (признака) завершения запроса (строки 7-10) необходимо для того, чтобы MySQL не воспринимал символы `;`, встречающиеся в конце запросов внутри триггера как конец самого запроса по созданию триггера. После всех операций по созданию триггеров этот разделитель возвращается в исходное состояние (строка 53).

Выражение **`FOR EACH ROW`** (строки 16, 27, 44) означают, что тело триггера будет выполнено для каждой записи (по-другому триггеры в MySQL и не работают), которую затрагивает операция с таблицей (добавляться, изменяться и удаляться может несколько записей за один раз).

Ключевые слова **`NEW`** и **`OLD`** позволяют обращаться:

- при операциях вставки через **`NEW`** к новым (добавляемым) данным;
- при операциях обновления через **`OLD`** к старым значениям данных и через **`NEW`** к новым значениям данных;
- при операциях удаления через **`OLD`** к значениям удаляемых данных.

MySQL (в отличие от MS SQL Server) «на лету» вычисляет новые значения полей таблицы, что позволяет нам во всех трёх триггерах производить все необходимые действия одним запросом и использовать выражение **`` `rest` = `total` - `given` ``**, т.к. значения **`` `total` ``** и/или **`` `given` ``** уже обновлены ранее встретившимися в запросах командами. В MS SQL Server же придётся выполнять отдельный запрос, чтобы вычислить значение **`` `rest` ``**, т.к. значения **`` `total` ``** и/или **`` `given` ``** не меняются до завершения выполнения первого запроса.

В триггерах на таблице **`books`** сами запросы (строки 18-20, 29-35, 46-48) совершенно тривиальны, и единственная их непривычность заключается в использовании ключевых слов **`OLD`** и **`NEW`**, которые мы только что рассмотрели.

MySQL Решение 3.1.2.a (триггеры для таблицы subscriptions)

```
1  -- Удаление старых версий триггеров
2  -- (удобно в процессе разработки и отладки):
3  DROP TRIGGER `upd_bks_sts_on_subscriptions_ins`;
4  DROP TRIGGER `upd_bks_sts_on_subscriptions_del`;
5  DROP TRIGGER `upd_bks_sts_on_subscriptions_upd`;
6
7
8  -- Переключение разделителя завершения запроса,
9  -- т.к. сейчас запросом будет создание триггера,
10 -- внутри которого есть свои, классические запросы:
11 DELIMITER $$
12
```

MySQL Решение 3.1.2.a (триггеры для таблицы books) (продолжение)

```
-- Создание триггера, реагирующего на добавление выдачи книг:
CREATE TRIGGER `upd_bks_sts_on_subscriptions_ins`
BEFORE INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN

   SET @delta = 0;

   IF (NEW.`sb_is_active` = 'Y') THEN
    SET @delta = 1;
   END IF;

   UPDATE `books_statistics` SET
          `rest` = `rest` - @delta,
          `given` = `given` + @delta;
  END;
$$

-- Создание триггера, реагирующего на удаление выдачи книг:
CREATE TRIGGER `upd_bks_sts_on_subscriptions_del`
BEFORE DELETE
ON `subscriptions`
 FOR EACH ROW
  BEGIN

   SET @delta = 0;

   IF (OLD.`sb_is_active` = 'Y') THEN
    SET @delta = 1;
   END IF;

   UPDATE `books_statistics` SET
          `rest` = `rest` + @delta,
          `given` = `given` - @delta;
  END;
$$
-- Создание триггера, реагирующего на обновление выдачи книг:
CREATE TRIGGER `upd_bks_sts_on_subscriptions_upd`
BEFORE UPDATE
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   SET @delta = 0;

   IF ((NEW.`sb_is_active` = 'Y') AND (OLD.`sb_is_active` = 'N')) THEN
    SET @delta = -1;
   END IF;

   IF ((NEW.`sb_is_active` = 'N') AND (OLD.`sb_is_active` = 'Y')) THEN
    SET @delta = 1;
   END IF;

   UPDATE `books_statistics` SET
          `rest` = `rest` + @delta,
          `given` = `given` - @delta;
   END;
$$

-- Восстановление разделителя завершения запросов:
DELIMITER ;
```

Триггеры на таблице `subscriptions` оказываются чуть более сложными, тем триггеры на таблице `books`: здесь приходится анализировать происходящее и предпринимать действия в зависимости от ситуации.

В триггере, реагирующем на добавление выдачи книг (строки 13-30) мы должны изменить значения `` `rest` `` и `` `given` `` только в том случае, если книга в добавляемой выдаче отмечена как находящаяся на руках у читателя. Изначально мы предполагаем, что это не так, и инициализируем в строке 20 переменную `@delta` значением 0. Если далее оказывается, что книга всё же выдана, мы изменяем значение этой переменной на 1 (строки 22-24). Таким образом, в запросе в строках 26-28 значения полей агрегирующей таблицы будут меняться на 0 (т.е. оставаться неизменными) или на 1 в зависимости от того, выдана ли книга читателю.

Абсолютно аналогичной логикой мы руководствуемся в триггере, реагирующем на удаление выдачи книги (строки 32-49).

В триггере, реагирующем на обновление выдачи книги, нам нужно рассмотреть четыре случая (из которых нас на самом деле интересуют только два последних):

- книга была на руках у читателя и там же осталась (значение `` `sb_is_active` `` было равно `Y` и таким же осталось);
- книга не была на руках у читателя и там же осталась (значение `` `sb_is_active` `` было равно `N` и таким же осталось);
- книга была на руках у читателя, и он её вернул (значение `` `sb_is_active` `` было равно `Y`, но поменялось на `N` — строки 58-60 запроса);
- книга не была на руках у читателя, но он её забрал (значение `` `sb_is_active` `` было равно `N`, но поменялось на `Y` — строки 62-64 запроса).

Очевидно, что количество выданных и оставшихся в библиотеке книг изменяется только в двух последних случаях, которые и учтены в условиях, представленных в строках 58-64. Запрос в строках 66-68 использует значение переменной `@delta`, изменённое этими условиями, для модификации агрегированных данных.

Проверим, как работает то, что мы создали. Будем модифицировать данные в таблицах `books` и `subscriptions` и выбирать данные из таблицы `books_statistics`.

Добавим две книги с количеством экземпляров 5 и 10:

MySQL Решение 3.1.2.a (проверка реакции на добавление книг)

```
1   INSERT INTO `books`
2               (`b_id`,
3                `b_name`,
4                `b_quantity`,
5                `b_year`)
6   VALUES      (NULL,
7                'Новая книга 1',
8                5,
9                2001),
10              (NULL,
11               'Новая книга 2',
12               10,
13               2002)
```

	total	given	rest
Было	33	5	28
Стало	48	5	43

Увеличим на пять единиц количество экземпляров книги, которой сейчас в библиотеке зарегистрировано 10 экземпляров (такая книга у нас одна):

MySQL Решение 3.1.2.a (проверка реакции на изменение количества книги)

```
1  UPDATE `books`
2  SET    `b_quantity` = `b_quantity` + 5
3  WHERE  `b_quantity` = 10
```

	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находится на руках у читателей (книга с идентификатором 1).

MySQL Решение 3.1.2.a (проверка реакции на удаление книги)

```
1  DELETE FROM `books`
2  WHERE  `b_id` = 1
```

	total	given	rest
Было	53	5	48
Стало	51	3	48

Отметим, что по выдаче с идентификатором 3 книга возвращена:

MySQL Решение 3.1.2.a (проверка реакции на возврат книги)

```
1  UPDATE `subscriptions`
2  SET    `sb_is_active` = 'N'
3  WHERE  `sb_id` = 3
```

	total	given	rest
Было	51	3	48
Стало	51	2	49

Отменим эту операцию (снова отметим книгу как невозвращённую):

MySQL Решение 3.1.2.a (проверка реакции на отмену возврата книги)

```
1  UPDATE `subscriptions`
2  SET    `sb_is_active` = 'Y'
3  WHERE  `sb_id` = 3
```

	total	given	rest
Было	51	2	49
Стало	51	3	48

Добавим в базу данных информацию о том, что читатель с идентификатором 2 взял в библиотеке книги с идентификаторами 5 и 6:

MySQL Решение 3.1.2.a (проверка реакции на выдачу книг)

```
INSERT INTO `subscriptions`
            (`sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`)
VALUES      (NULL,
             2,
             5,
             '2016-01-10',
             '2016-02-10',
             'Y'),
            (NULL,
             2,
             6,
             '2016-01-10',
             '2016-02-10',
             'Y')
```

	total	given	rest
Было	51	3	48
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 42 (книга по этой выдаче уже возвращена):

MySQL Решение 3.1.2.a (проверка реакции на удаление выдачи с возвращённой книгой)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` = 42
```

	total	given	rest
Было	51	5	46
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 62 (книга по этой выдаче ещё не возвращена):

MySQL Решение 3.1.2.a (проверка реакции на удаление выдачи с не возвращённой книгой)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` = 62
```

	total	given	rest
Было	51	5	46
Стало	51	4	47

Наконец, удалим все книги (что также приведёт к каскадному удалению всех выдач):

MySQL Решение 3.1.2.a (проверка реакции на удаление всех книг)

```
DELETE FROM `books`
```

	total	given	rest
Было	51	4	47
Стало	0	0	0

Итак, все операции модификации данных в таблицах **books** и **subscriptions** вызывают соответствующие изменения в агрегирующей таблице **books_statistics**, которая в MySQL выступает в роли кэширующего представления.

Переходим к MS SQL Server. Теоретически, здесь всё должно быть хорошо, т.к. эта СУБД поддерживает т.н. индексированные представления, но если мы внимательно изучим перечень ограничений[10], то придём к неутешительному выводу: придётся идти по пути MySQL и создавать агрегирующую таблицу и триггеры.

Создадим агрегирующую таблицу:

MS SQL Решение 3.1.2.a (создание агрегирующей таблицы)

```
CREATE TABLE [books_statistics]
(
      [total] INTEGER NOT NULL,
      [given] INTEGER NOT NULL,
      [rest]  INTEGER NOT NULL
)
```

Проинициализируем данные в созданной таблице:

MS SQL Решение 3.1.2.a (очистка таблицы и инициализация данных)

```
-- Очистка таблицы:
TRUNCATE TABLE [books_statistics];

-- Инициализация данных:
INSERT INTO [books_statistics]
            ([total],
             [given],
             [rest])
SELECT ISNULL([total], 0) AS [total],
       ISNULL([given], 0) AS [given],
       ISNULL([total] - [given], 0) AS [rest]
FROM   (SELECT (SELECT SUM([b_quantity])
                FROM   [books])              AS [total],
               (SELECT COUNT([sb_book])
                FROM   [subscriptions]
                WHERE  [sb_is_active] = 'Y') AS [given])
       AS [prepared_data];
```

До сих пор всё было совершенно идентично MySQL, но внутренняя логика работы триггеров у MS SQL Server совершенно иная, хотя нам по-прежнему придётся создать триггеры на всех трёх операциях (вставки, обновления удаления) для обеих таблиц (**books** и **subscriptions**).

MS SQL Решение 3.1.2.a (триггеры для таблицы books)

```
-- Удаление старых версий триггеров
-- (удобно в процессе разработки и отладки):
DROP TRIGGER [upd_bks_sts_on_books_ins];
DROP TRIGGER [upd_bks_sts_on_books_del];
DROP TRIGGER [upd_bks_sts_on_books_upd];
GO

```

[10] https://msdn.microsoft.com/en-us/library/ms191432%28v=sql.110%29.aspx

MS SQL Решение 3.1.2.a (триггеры для таблицы books) (продолжение)

```
8   -- Создание триггера, реагирующего на добавление книг:
9   CREATE TRIGGER [upd_bks_sts_on_books_ins]
10  ON [books]
11  AFTER INSERT
12  AS
13    UPDATE [books_statistics] SET
14           [total] = [total]  + (SELECT SUM([b_quantity])
15                                 FROM   [inserted]);
16    UPDATE [books_statistics] SET
17           [rest] = [total] - [given];
18  GO
19
20  -- Создание триггера, реагирующего на удаление книг:
21  CREATE TRIGGER [upd_bks_sts_on_books_del]
22  ON [books]
23  AFTER DELETE
24  AS
25    UPDATE [books_statistics] SET
26           [total] = [total] - (SELECT SUM([b_quantity])
27                                FROM   [deleted]),
28           [given] = [given] - (SELECT COUNT([sb_book])
29                                FROM   [subscriptions]
30                                WHERE  [sb_book] IN (SELECT [b_id]
31                                                     FROM   [deleted])
32                                   AND [sb_is_active] = 'Y');
33    UPDATE [books_statistics] SET
34           [rest] = [total] - [given];
35  GO
36
37  -- Создание триггера, реагирующего на
38  -- изменение количества книг:
39  CREATE TRIGGER [upd_bks_sts_on_books_upd]
40  ON [books]
42  AFTER UPDATE
42  AS
43    UPDATE [books_statistics] SET
44           [total] = [total] - (SELECT SUM([b_quantity])
45                                FROM   [deleted]) + (SELECT SUM([b_quantity])
46                                                     FROM   [inserted]);
47    UPDATE [books_statistics] SET
48           [rest] = [total] - [given];
49  GO
```

Основных отличия от решения для MySQL здесь два:

- тело триггера выполняется не для каждого ряда модифицируемых данных (как это происходит в MySQL), а один раз для всего набора данных — отсюда следует не обращение к отдельному полю через ключевые слова **`OLD`** и **`NEW`**, а работа с «псевдотаблицами» **`[deleted]`** (содержит информацию об удаляемых строках и старые данные обновляемых строк) и **`[inserted]`** (содержит информацию о добавляемых строках и новые данные обновляемых строк);
- MS SQL Server не позволяет в одном запросе модифицировать значения полей и сразу же использовать их новые значения — потому во всех трёх триггерах для вычисления значения поля **`[rest]`** используется отдельный запрос.

В остальном поведение триггеров в MS SQL Server на таблице **`books`** идентично поведению соответствующих триггеров в MySQL. А в триггерах на таблице **`subscriptions`** есть существенные отличия.

MS SQL Решение 3.1.2.a (триггеры для таблицы subscriptions)

```
-- Удаление старых версий триггеров
-- (удобно в процессе разработки и отладки):
DROP TRIGGER [upd_bks_sts_on_subscriptions_ins];
DROP TRIGGER [upd_bks_sts_on_subscriptions_del];
DROP TRIGGER [upd_bks_sts_on_subscriptions_upd];
GO

-- Создание триггера, реагирующего на добавление выдачи книг:
CREATE TRIGGER [upd_bks_sts_on_subscriptions_ins]
ON [subscriptions]
AFTER INSERT
AS
 DECLARE @delta INT = (SELECT COUNT(*)
                       FROM   [inserted]
                       WHERE  [sb_is_active] = 'Y');
 UPDATE [books_statistics] SET
        [rest] = [rest] - @delta,
        [given] = [given] + @delta;
GO

-- Создание триггера, реагирующего на удаление выдачи книг:
CREATE TRIGGER [upd_bks_sts_on_subscriptions_del]
ON [subscriptions]
AFTER DELETE
AS
 DECLARE @delta INT = (SELECT COUNT(*)
                       FROM   [deleted]
                       WHERE  [sb_is_active] = 'Y');
 UPDATE [books_statistics] SET
        [rest] = [rest] + @delta,
        [given] = [given] - @delta;
GO

-- Создание триггера, реагирующего на обновление выдачи книг:
CREATE TRIGGER [upd_bks_sts_on_subscriptions_upd]
ON [subscriptions]
AFTER UPDATE
AS
 DECLARE @taken INT = (
  SELECT COUNT(*)
  FROM   [inserted]
         JOIN [deleted]
           ON [inserted].[sb_id] = [deleted].[sb_id]
  WHERE  [inserted].[sb_is_active] = 'Y'
     AND [deleted].[sb_is_active] = 'N');

 DECLARE @returned INT = (
  SELECT COUNT(*)
  FROM   [inserted]
         JOIN [deleted]
           ON [inserted].[sb_id] = [deleted].[sb_id]
  WHERE  [inserted].[sb_is_active] = 'N'
     AND [deleted].[sb_is_active] = 'Y');

 DECLARE @delta INT = @taken - @returned;

 UPDATE [books_statistics] SET
        [rest] = [rest] - @delta,
        [given] = [given] + @delta;
GO
```

В MySQL мы вычисляли значение переменной `@delta`, которое могло становиться равным 0, 1, -1 в зависимости от того, как изменение в анализируемой строке должно повлиять на данные в агрегирующей таблице.

В MS SQL Server мы должны реализовывать реакцию на изменение не одной отдельной строки, а всего набора модифицируемых строк целиком. Именно поэтому в строках 13-15 и 26-28 значение переменной `@delta` определяется как количество записей, удовлетворяющих условию работы триггера.

В строках 39-55 этот подход ещё больше усложняется: мы должны определить количество выданных (строки 39-45) и возвращённых (строки 47-53) книг, а затем в строке 55 мы можем определить разность полученных чисел и использовать её значение (строки 57-59) для изменения данных в агрегирующей таблице.

Снова (как и в случае с MySQL) проверим, как работает то, что мы создали. Будем модифицировать данные в таблицах `books` и `subscriptions` и выбирать данные из таблицы `books_statistics`.

Добавим две книги с количеством экземпляров 5 и 10:

MS SQL Решение 3.1.2.a (проверка реакции на добавление книг)

```
1   INSERT INTO [books]
2               ([b_name],
3                [b_quantity],
4                [b_year])
5   VALUES      (N'Новая книга 1',
6                5,
7                2001),
8               (N'Новая книга 2',
9                10,
10               2002)
```

	total	given	rest
Было	33	5	28
Стало	48	5	43

Увеличим на пять единиц количество экземпляров книги, которой сейчас в библиотеке зарегистрировано 10 экземпляров (такая книга у нас одна):

MS SQL Решение 3.1.2.a (проверка реакции на изменение количества книги)

```
1   UPDATE [books]
2   SET    [b_quantity] = [b_quantity] + 5
3   WHERE  [b_quantity] = 10
```

	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находится на руках у читателей (книга с идентификатором 1).

MS SQL Решение 3.1.2.a (проверка реакции на удаление книги)

```
1   DELETE FROM [books]
2   WHERE  [b_id] = 1
```

	total	given	rest
Было	53	5	48
Стало	51	3	48

Отметим, что по выдаче с идентификатором 3 книга возвращена:

MS SQL Решение 3.1.2.a (проверка реакции на возврат книги)

```
UPDATE [subscriptions]
SET    [sb_is_active] = 'N'
WHERE  [sb_id] = 3
```

	total	given	rest
Было	51	3	48
Стало	51	2	49

Отменим эту операцию (снова отметим книгу как невозвращённую):

MS SQL Решение 3.1.2.a (проверка реакции на отмену возврата книги)

```
UPDATE [subscriptions]
SET    [sb_is_active] = 'Y'
WHERE  [sb_id] = 3
```

	total	given	rest
Было	51	2	49
Стало	51	3	48

Добавим в базу данных информацию о том, что читатель с идентификатором 2 взял в библиотеке книги с идентификаторами 5 и 6:

MS SQL Решение 3.1.2.a (проверка реакции на выдачу книг)

```
INSERT INTO [subscriptions]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (2,
             5,
             CAST(N'2016-01-10' AS DATE),
             CAST(N'2016-02-10' AS DATE),
             'Y'),
            (2,
             6,
             CAST(N'2016-01-10' AS DATE),
             CAST(N'2016-02-10' AS DATE),
             'Y')
```

	total	given	rest
Было	51	3	48
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 42 (книга по этой выдаче уже возвращена):

MS SQL Решение 3.1.2.a (проверка реакции на удаление выдачи с возвращённой книгой)

```
DELETE FROM [subscriptions]
WHERE  [sb_id] = 42
```

	total	given	rest
Было	51	5	46
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 62 (книга по этой выдаче ещё не возвращена):

MS SQL Решение 3.1.2.a (проверка реакции на удаление выдачи с не возвращённой книгой)

```
DELETE FROM [subscriptions]
WHERE  [sb_id] = 62
```

	total	given	rest
Было	51	5	46
Стало	51	4	47

Наконец, удалим все книги (что также приведёт к каскадному удалению всех выдач):

MS SQL Решение 3.1.2.a (проверка реакции на удаление всех книг)

```
DELETE FROM [books]
```

	total	given	rest
Было	51	4	47
Стало	0	0	0

Итак, все операции модификации данных в таблицах **books** и **subscriptions** вызывают соответствующие изменения в агрегирующей таблице **books_statistics**, которая в MS SQL Server выступает в роли кэширующего представления.

Переходим к решению для Oracle.

Oracle — единственная СУБД, в которой данная задача полноценно решается с использованием материализованных представлений. Если бы мы использовали более новую или коммерческую версию Oracle, мы могли бы реализовать самый элегантный вариант — **REFRESH FAST ON COMMIT**, указывающий СУБД оптимальным образом обновлять данные в материализованном представлении каждый раз, когда завершается очередная транзакция, затрагивающая таблицы, из которых собираются данные. Но в Oracle 11gR2 Express Edition эта опция нам недоступна, и вместо неё мы используем **REFRESH FORCE START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)**, т.е. будем принудительно обновлять данные в представлении раз в минуту.

Oracle Решение 3.1.2.a

```
-- Удаление старой версии материализованного представления
-- (удобно при разработке и отладке):
DROP MATERIALIZED VIEW "books_statistics";

-- Создание материализованного представления:
CREATE MATERIALIZED VIEW "books_statistics"
BUILD IMMEDIATE
REFRESH FORCE
START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)
AS
   SELECT "total",
          "given",
          "total" - "given" AS "rest"
   FROM   (SELECT SUM("b_quantity") AS "total"
           FROM   "books")
          JOIN (SELECT COUNT("sb_book") AS "given"
                FROM   "subscriptions"
                WHERE  "sb_is_active" = 'Y')
            ON 1 = 1
```

Выражение **BUILD IMMEDIATE** в строке 7 предписывает СУБД немедленно инициализировать материализованное представление данными.

Выражения в строках 8-9 описывают способ обновления данных в материализованном представлении (**REFRESH FORCE** — СУБД сама выбирает оптимальный способ из доступных — либо быстрое обновление, либо полное) и периодичность этой операции (**START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)** — начать немедленно и повторять каждую 1/1440-ю часть суток, т.е. каждую минуту).

Запрос в строках 11-19 по здравому смыслу должен быть идентичен запросам, которые мы использовали в MySQL и MS SQL Server для инициализации данных в агрегирующей таблице. Но Oracle не позволяет в материализованных представлениях использовать запросы с подзапросами в секции **FROM**, а вот объединять данные из двух подзапросов позволяет.

Поэтому мы сформировали два подзапроса (вычисляющий общее количество книг — строки 14-15 и вычисляющий количество выданных читателям книг — строки 16-18), а затем объединили их результаты (каждый подзапрос возвращает просто по одному числу) с применением гарантированно выполняющегося условия **1 = 1**.

Результат работы SQL-кода в строках 14-15 таков:

total	given
33	5

Остаётся только выбрать из него значения полей **"total"** и **"given"** и вычислить на их основе значение поля **"rest"**, что и происходит в строках 11-13.

Снова (как и в случае с MySQL и MS SQL Server) проверим, как работает то, что мы создали. Будем модифицировать данные в таблицах **books** и **subscriptions** и выбирать данные из материализованного представления **books_statistics**. Обратите внимание, что после каждого запроса явно выполняется подтверждение транзакции (**COMMIT**), чтобы исключить ситуацию, в которой Oracle будет ждать этого события и не обновит материализованное представление.

Добавим две книги с количеством экземпляров 5 и 10:

Oracle Решение 3.1.2.a (проверка реакции на добавление книг)

```
1  INSERT ALL
2    INTO "books" ("b_name",
3                  "b_quantity",
4                  "b_year")
5          VALUES (N'Новая книга 1',
6                  5,
7                  2001)
8    INTO "books" ("b_name",
9                  "b_quantity",
10                 "b_year")
11         VALUES (N'Новая книга 2',
12                 10,
13                 2002)
14 SELECT 1 FROM "DUAL";
15 COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	33	5	28
Стало	48	5	43

Увеличим на пять единиц количество экземпляров книги, которой сейчас в библиотеке зарегистрировано 10 экземпляров (такая книга у нас одна):

Oracle Решение 3.1.2.a (проверка реакции на изменение количества книги)

```
1  UPDATE "books"
2  SET    "b_quantity" = "b_quantity" + 5
3  WHERE  "b_quantity" = 10;
4  COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находится на руках у читателей (книга с идентификатором 1).

Oracle Решение 3.1.2.a (проверка реакции на удаление книги)

```
1  DELETE FROM "books"
2  WHERE  "b_id" = 1;
3  COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	53	5	48
Стало	51	3	48

Отметим, что по выдаче с идентификатором 3 книга возвращена:

Oracle Решение 3.1.2.a (проверка реакции на возврат книги)

```
1  UPDATE "subscriptions"
2  SET    "sb_is_active" = 'N'
3  WHERE  "sb_id" = 3;
4  COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	3	48
Стало	51	2	49

Отменим эту операцию (снова отметим книгу как невозвращённую):

Oracle Решение 3.1.2.a (проверка реакции на отмену возврата книги)

```
1  UPDATE "subscriptions"
2  SET    "sb_is_active" = 'Y'
3  WHERE  "sb_id" = 3;
4  COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	2	49
Стало	51	3	48

Добавим в базу данных информацию о том, что читатель с идентификатором 2 взял в библиотеке книги с идентификаторами 5 и 6:

Oracle Решение 3.1.2.a (проверка реакции на выдачу книг)

```
INSERT ALL
 INTO "subscriptions" ("sb_subscriber",
                       "sb_book",
                       "sb_start",
                       "sb_finish",
                       "sb_is_active")
              VALUES (2,
                      5,
                      TO_DATE('2016-01-10', 'YYYY-MM-DD'),
                      TO_DATE('2016-02-10', 'YYYY-MM-DD'),
                      'Y')
 INTO "subscriptions" ("sb_subscriber",
                       "sb_book",
                       "sb_start",
                       "sb_finish",
                       "sb_is_active")
              VALUES (2,
                      6,
                      TO_DATE('2016-01-10', 'YYYY-MM-DD'),
                      TO_DATE('2016-02-10', 'YYYY-MM-DD'),
                      'Y')
SELECT 1 FROM "DUAL";
COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	3	48
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 42 (книга по этой выдаче уже возвращена):

Oracle Решение 3.1.2.a (проверка реакции на удаление выдачи с возвращённой книгой)

```
DELETE FROM "subscriptions"
WHERE "sb_id" = 42;
COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	5	46
Стало	51	5	46

Удалим информацию о выдаче с идентификатором 62 (книга по этой выдаче ещё не возвращена):

Oracle Решение 3.1.2.a (проверка реакции на удаление выдачи с не возвращённой книгой)

```
DELETE FROM "subscriptions"
WHERE "sb_id" = 62;
COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	5	46
Стало	51	4	47

Наконец, удалим все книги (что также приведёт к каскадному удалению всех выдач):

Oracle Решение 3.1.2.a (проверка реакции на удаление всех книг)

```
1  DELETE FROM "books";
2  COMMIT; -- И подождать минуту.
```

	total	given	rest
Было	51	4	47
Стало	0	0	0

Итак, все операции модификации данных в таблицах **books** и **subscriptions** вызывают соответствующие изменения в материализованном представлении **books_statistics**.

Решение 3.1.2.b{223}.

Если вы пропустили решение{223} задачи 3.1.2.a{223}, настоятельно рекомендуется ознакомиться с ним перед тем, как продолжить чтение, т.к. многие неочевидные моменты, которые встретят в данном решении, были рассмотрены ранее.

По сравнению с предыдущей задачей{223} здесь всё будет намного проще, т.к. запрос, формирующий необходимый набор данных, не содержит выражений, подпадающих под ограничения индексированных представлений MS SQL Server и материализованных представлений Oracle.

Проблема будет только с MySQL, т.к. в нём подобных представлений нет как явления, и нам снова придётся создавать кэширующую таблицу и триггеры.

Создадим кэширующую таблицу (именно кэширующую, а не агрегирующую, как в задаче 3.1.2.a{223}, т.к. здесь мы ничего не агрегируем, а лишь сохраняем готовый результат). Код её создания можно почти полностью взять из кода создания таблицы **subscritions**, а для полей **sb_subscriber** и **sb_book** взять их определения из таблицы **subscribers** и **books**.

MySQL Решение 3.1.2.b (создание кэширующей таблицы)

```
1  CREATE TABLE `subscriptions_ready`
2  (
3        `sb_id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
4        `sb_subscriber` VARCHAR(150) NOT NULL,
5        `sb_book` VARCHAR(150) NOT NULL,
6        `sb_start` DATE NOT NULL,
7        `sb_finish` DATE NOT NULL,
8        `sb_is_active` ENUM ('Y', 'N') NOT NULL,
9       CONSTRAINT `PK_subscriptions` PRIMARY KEY (`sb_id`)
10 )
```

Проинициализируем данные в созданной таблице:

MySQL Решение 3.1.2.b (очистка таблицы и инициализация данных)

```
-- Очистка таблицы:
TRUNCATE TABLE `subscriptions_ready`;

-- Инициализация данных:
INSERT INTO `subscriptions_ready`
            (`sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`)
SELECT `sb_id`,
       `s_name` AS `sb_subscriber`,
       `b_name` AS `sb_book`,
       `sb_start`,
       `sb_finish`,
       `sb_is_active`
FROM   `books`
       JOIN `subscriptions`
         ON `b_id` = `sb_book`
       JOIN `subscribers`
         ON `sb_subscriber` = `s_id`;
```

Напишем триггеры, модифицирующие данные в кэширующей таблице. Источником данных являются таблицы **books**, **subscribers** и **subscriptions**, потому придётся создавать триггеры для всех трёх таблиц.

Поскольку MySQL версии 5.6 не позволяет создавать несколько однотипных триггеров на одной и той же таблице, перед выполнением следующего кода придётся удалить созданные в задаче 3.2.1.a{223} триггеры с таблиц **books** и **subscriptions**.

Регистрация в библиотеке новых книг и читателей никак не влияет на содержимое таблицы **subscriptions**, потому нет необходимости создавать **INSERT**-триггеры на таблицах **books** и **subscribers** — достаточно **DELETE**- и **UPDATE**-триггеров.

Обратите внимание на несколько важных моментов в представленном ниже коде:

- внутри триггера в MySQL мы не можем явно или неявно подтверждать транзакцию, а потому не можем использовать для очистки таблицы оператор **TRUNCATE** (он является «нетранзакционным», и потому приводит к неявному подтверждению транзакции);
- мы не храним в нашей кэширующей таблице идентификаторы книг, и потому не можем найти и удалить только отдельные записи (искать и удалять по названию книги тоже нельзя — несколько разных книг могут иметь одинаковое название), потому мы вынуждены каждый раз очищать всю таблицу и заново наполнять её данными;
- в задаче 3.2.1.a{223} мы использовали **BEFORE**-триггеры, хотя могли использовать и **AFTER** — там это не имело значения, здесь же мы обязаны использовать только **AFTER**-триггеры, т.к. в противном случае в силу особенностей логики транзакций поведение MySQL отличается от ожидаемого, и информация в кэширующей таблице может не обновиться.

За исключением только что рассмотренных нюансов код тела обоих триггеров совершенно тривиален и представляет собой два запроса — на удаление всех данных из таблицы и на наполнение таблицы данными на основе полученной выборки. Оба эти запроса можно выполнить совершенно отдельно от триггеров как самостоятельные SQL-конструкции.

В триггере, реагирующем на обновление информации о книге, есть проверка (строка 45) того, поменялось ли название книги. Если не поменялось, то нет необходимости обновлять закэшированные данные.

MySQL Решение 3.1.2.b (триггеры для таблицы books)

```
1  -- Удаление старых версий триггеров
2  -- (удобно в процессе разработки и отладки):
3  DROP TRIGGER `upd_sbs_rdy_on_books_del`;
4  DROP TRIGGER `upd_sbs_rdy_on_books_upd`;
5
6  -- Переключение разделителя завершения запроса,
7  -- т.к. сейчас запросом будет создание триггера,
8  -- внутри которого есть свои, классические запросы:
9  DELIMITER $$
10
11 -- Создание триггера, реагирующего на удаление книг:
12 CREATE TRIGGER `upd_sbs_rdy_on_books_del`
13 AFTER DELETE
14 ON `books`
15  FOR EACH ROW
16   BEGIN
17    DELETE FROM `subscriptions_ready`;
18    INSERT INTO `subscriptions_ready`
19                (`sb_id`,
20                 `sb_subscriber`,
21                 `sb_book`,
22                 `sb_start`,
23                 `sb_finish`,
24                 `sb_is_active`)
25          SELECT `sb_id`,
26                 `s_name`,
27                 `b_name`,
28                 `sb_start`,
29                 `sb_finish`,
30                 `sb_is_active`
31          FROM   `books`
32                 JOIN `subscriptions`
33                   ON `b_id` = `sb_book`
34                 JOIN `subscribers`
35                   ON `sb_subscriber` = `s_id`;
36   END;
37 $$
```

MySQL Решение 3.1.2.b (триггеры для таблицы books) (продолжение)

```
-- Создание триггера, реагирующего на
-- изменение данных о книгах:
CREATE TRIGGER `upd_sbs_rdy_on_books_upd`
AFTER UPDATE
ON `books`
 FOR EACH ROW
  BEGIN
   IF (OLD.`b_name` != NEW.`b_name`)
    THEN
     DELETE FROM `subscriptions_ready`;
     INSERT INTO `subscriptions_ready`
                 (`sb_id`,
                  `sb_subscriber`,
                  `sb_book`,
                  `sb_start`,
                  `sb_finish`,
                  `sb_is_active`)
            SELECT `sb_id`,
                   `s_name`,
                   `b_name`,
                   `sb_start`,
                   `sb_finish`,
                   `sb_is_active`
            FROM   `books`
                   JOIN `subscriptions`
                     ON `b_id` = `sb_book`
                   JOIN `subscribers`
                     ON `sb_subscriber` = `s_id`;
   END IF;
  END;
$$

-- Восстановление разделителя завершения запросов:
DELIMITER ;
```

Вся логика проверки работы таких триггеров подробно показана в решении{223} задачи 3.1.2.a{223}. Вы можете самостоятельно провести эксперимент, изменяя произвольным образом данные в таблице **books** и отслеживая соответствующие изменения в таблице **subscriptions_ready**.

Триггеры на таблице **subscribers** отличаются от триггеров на таблице **books** только своими именами, названиями своих таблиц и именем поля, изменение значения которого проверяется для определения необходимости обновления закэшированных данных (в коде выше проверяется поле **b_name**, в коде ниже — **s_name**).

MySQL Решение 3.1.2.b (триггеры для таблицы subscribers)

```
1  -- Удаление старых версий триггеров
2  -- (удобно в процессе разработки и отладки):
3  DROP TRIGGER `upd_sbs_rdy_on_subscribers_del`;
4  DROP TRIGGER `upd_sbs_rdy_on_subscribers_upd`;
5
6  -- Переключение разделителя завершения запроса,
7  -- т.к. сейчас запросом будет создание триггера,
8  -- внутри которого есть свои, классические запросы:
9  DELIMITER $$
10 -- Создание триггера, реагирующего на удаление книг:
11 CREATE TRIGGER `upd_sbs_rdy_on_subscribers_del`
12 AFTER DELETE
13 ON `subscribers`
14  FOR EACH ROW
15   BEGIN
16    DELETE FROM `subscriptions_ready`;
17    INSERT INTO `subscriptions_ready`
18                (`sb_id`,
19                 `sb_subscriber`,
20                 `sb_book`,
21                 `sb_start`,
22                 `sb_finish`,
23                 `sb_is_active`)
24          SELECT `sb_id`,
25                 `s_name`,
26                 `b_name`,
27                 `sb_start`,
28                 `sb_finish`,
29                 `sb_is_active`
30          FROM   `books`
31                 JOIN `subscriptions`
32                   ON `b_id` = `sb_book`
33                 JOIN `subscribers`
34                   ON `sb_subscriber` = `s_id`;
35   END;
36 $$
37
```

MySQL Решение 3.1.2.b (триггеры для таблицы subscribers) (продолжение)

```
38 -- Создание триггера, реагирующего на
39 -- изменение данных о книгах:
40 CREATE TRIGGER `upd_sbs_rdy_on_subscribers_upd`
41 AFTER UPDATE
42 ON `subscribers`
43  FOR EACH ROW
44   BEGIN
45    IF (OLD.`s_name` != NEW.`s_name`)
46     THEN
47      DELETE FROM `subscriptions_ready`;
48      INSERT INTO `subscriptions_ready`
49                  (`sb_id`,
50                   `sb_subscriber`,
51                   `sb_book`,
52                   `sb_start`,
53                   `sb_finish`,
54                   `sb_is_active`)
55            SELECT `sb_id`,
56                   `s_name`,
57                   `b_name`,
58                   `sb_start`,
59                   `sb_finish`,
60                   `sb_is_active`
61            FROM   `books`
62                   JOIN `subscriptions`
63                     ON `b_id` = `sb_book`
64                   JOIN `subscribers`
65                     ON `sb_subscriber` = `s_id`;
66    END IF;
67   END;
68 $$
69
70 -- Восстановление разделителя завершения запросов:
71 DELIMITER ;
```

На таблице **subscriptions** придётся создавать все три триггера (**INSERT**, **UPDATE** и **DELETE**), т.к. каждая из этих операций может повлиять на содержимое кэширующей таблицы **subscriptions_ready**. И код всех этих трёх триггеров будет сильно различаться.

MySQL Решение 3.1.2.b (триггеры для таблицы subscriptions)

```
-- Удаление старых версий триггеров
-- (удобно в процессе разработки и отладки):
DROP TRIGGER `upd_sbs_rdy_on_subscriptions_ins`;
DROP TRIGGER `upd_sbs_rdy_on_subscriptions_del`;
DROP TRIGGER `upd_sbs_rdy_on_subscriptions_upd`;

-- Переключение разделителя завершения запроса,
-- т.к. сейчас запросом будет создание триггера,
-- внутри которого есть свои, классические запросы:
DELIMITER $$

-- Создание триггера, реагирующего на добавление выдачи книг:
CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_ins`
AFTER INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   INSERT INTO `subscriptions_ready`
               (`sb_id`,
                `sb_subscriber`,
                `sb_book`,
                `sb_start`,
                `sb_finish`,
                `sb_is_active`)
          SELECT `sb_id`,
                 `s_name`,
                 `b_name`,
                 `sb_start`,
                 `sb_finish`,
                 `sb_is_active`
          FROM   `books`
                 JOIN `subscriptions`
                   ON `b_id` = `sb_book`
                 JOIN `subscribers`
                   ON `sb_subscriber` = `s_id`
          WHERE `s_id` = NEW.`sb_subscriber`
            AND `b_id` = NEW.`sb_book`;
  END;
$$

-- Создание триггера, реагирующего на удаление выдачи книг:
CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_del`
AFTER DELETE
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   DELETE FROM `subscriptions_ready`
   WHERE `subscriptions_ready`.`sb_id` = OLD.`sb_id`;
  END;
$$
```

MySQL Решение 3.1.2.b (триггеры для таблицы subscriptions) (продолжение)

```
51  -- Создание триггера, реагирующего на обновление выдачи книг:
52  CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_upd`
53  AFTER UPDATE
54  ON `subscriptions`
55   FOR EACH ROW
56    BEGIN
57     UPDATE `subscriptions_ready`
58            JOIN (SELECT `sb_id`,
59                         `s_name`,
60                         `b_name`
61                  FROM   `books`
62                         JOIN `subscriptions`
63                           ON `b_id` = `sb_book`
64                         JOIN `subscribers`
65                           ON `sb_subscriber` = `s_id`
66                  WHERE  `s_id` = NEW.`sb_subscriber`
67                     AND `b_id` = NEW.`sb_book`
68                     AND `sb_id` = NEW.`sb_id`) AS `new_data`
69     SET    `subscriptions_ready`.`sb_id` = NEW.`sb_id`,
70            `subscriptions_ready`.`sb_subscriber` = `new_data`.`s_name`,
71            `subscriptions_ready`.`sb_book` = `new_data`.`b_name`,
72            `subscriptions_ready`.`sb_start` = NEW.`sb_start`,
73            `subscriptions_ready`.`sb_finish` = NEW.`sb_finish`,
74            `subscriptions_ready`.`sb_is_active` = NEW.`sb_is_active`
75     WHERE  `subscriptions_ready`.`sb_id` = OLD.`sb_id`;
76    END;
77  $$
78
79  -- Восстановление разделителя завершения запросов:
80  DELIMITER ;
```

Код **INSERT**-триггера (строки 12-39) очень похож на код аналогичного триггера на таблице **books**. Отличие состоит в том, что в данном случае мы знаем идентификаторы читателя и книги, и это избавляет нас от необходимости формировать полную выборку.

Код **DELETE**-триггера (строки 41-50) получаемся самым компактным: нам нужно просто удалить из таблицы **subscriptions_ready** записи, идентификаторы которых совпадают с идентификаторами записей, удаляемых из таблицы **subscriptions**.

Код **UPDATE**-триггера (строки 51-77) — самый нетривиальный. Сам по себе синтаксис обновления на основе выборки выглядит непривычно (мы вынуждены объединять результаты выборки из обновляемой таблицы и выборки-источника — строки 57-68).

Условия в строках 66-67 позволяют сократить количество выбираемых рядов, а условие в строке 68 гарантирует, что мы получим данные о новой записи таблицы **subscriptions**, даже если у неё изменился первичный ключ. Следуя этой же логике, мы не указываем условие объединения **`subscriptions_ready` JOIN ... `new_data`**, т.к. единственным здравым условием объединения здесь может быть совпадение значений **sb_id**, но если первичный ключ записи в таблице **subscriptions** поменялся, то такого совпадения не будет, т.к. в таблице **subscriptions_ready** всё ещё хранится старое значение первичного ключа обновляемой записи.

В строках 69-74 новые данные для обновления полей мы берём из двух источников: из явно переданных данных через ключевое слово **NEW** (все значения, которые мы можем получить напрямую) и из результатов выборки **new_data** (имя читателя и название книги, т.к. их нет и не может быть в явно переданных данных, доступных через ключевое слово **NEW**).

Условие в строке 75 гарантирует, что мы обновим нужную запись: здесь необходимо сравнение идентификатора обновляемой записи именно с `OLD.`sb_id``, а не с `NEW.`sb_id``, т.к. у обновляемой записи в таблице **subscriptions** мог поменяться первичный ключ, и нам нужно найти его старое значение в таблице **subscriptions_ready** (благодаря выражению в строке 69 это значение тоже обновится, если оно изменилось).

Вся логика проверки работы таких триггеров подробно показана в решении{223} задачи 3.1.2.a{223}. Вы можете самостоятельно провести эксперимент, изменяя произвольным образом данные в таблице **books**, **subscribers** и **subscriptions** и отслеживая соответствующие изменения в таблице **subscriptions_ready**.

По сравнению с решением для MySQL, решения для MS SQL Server и Oracle предельно просты. В их основе лежит обычный запрос на выборку, который можно выполнить и сам по себе.

MS SQL Решение 3.1.2.b

```
1   -- Удаление старой версии индексированного представления
2   -- (удобно при разработке и отладке):
3   DROP VIEW [subscriptions_ready];
4
5   -- Создание представления:
6   CREATE VIEW [subscriptions_ready]
7   WITH SCHEMABINDING
8   AS
9   SELECT [sb_id],
10         [s_name] AS [sb_subscriber],
11         [b_name] AS [sb_book],
12         [sb_start],
13         [sb_finish],
14         [sb_is_active]
15  FROM   [dbo].[books]
16         JOIN [dbo].[subscriptions]
17           ON [b_id] = [sb_book]
18         JOIN [dbo].[subscribers]
19           ON [sb_subscriber] = [s_id];
20
21  -- Создание уникального кластерного индекса на представлении.
22  -- Именно эта операция "включает" автоматическое обновление
23  -- представления при изменении данных в таблицах,
24  -- на которых оно построено:
25  CREATE UNIQUE CLUSTERED INDEX [idx_subscriptions_ready]
26    ON [subscriptions_ready] ([sb_id]);
```

Чтобы данные в представлении **subscriptions_ready** обновлялись автоматически, его нужно сделать индексированным, т.е. создать на нём уникальный кластерный индекс (строки 25-26).

Необходимым условием создания такого индекса на представлении является привязка представления к схеме базы данных (строка 7), указывающая СУБД на необходимость установить и отслеживать соответствие между использованием в коде представления объектов базы данных и реальным состоянием таких объектов (их существованием, доступностью и т.д.) не только в момент создания представления, но и в момент любой модификации объектов базы данных, на которые ссылается представление.

Решение данной задачи для Oracle ещё проще: берётся запрос на выборку, позволяющий получить желаемые данные, и используется как тело материализованного представления. СУБД будет автоматически обновлять данные в этом представлении раз в минуту (логика такого поведения и остальные особенности создания материализованных представлений в Oracle была рассмотрена в решении[223] задачи 3.1.2.a[223].).

Oracle Решение 3.1.2.b

```
-- Удаление старой версии материализованного представления
-- (удобно при разработке и отладке):
DROP MATERIALIZED VIEW "subscriptions_ready";

-- Создание материализованного представления:
CREATE MATERIALIZED VIEW "subscriptions_ready"
BUILD IMMEDIATE
REFRESH FORCE
START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)
AS
  SELECT "sb_id",
         "s_name" AS "sb_subscriber",
         "b_name" AS "sb_book",
         "sb_start",
         "sb_finish",
         "sb_is_active"
  FROM   "books"
         JOIN "subscriptions"
           ON "b_id" = "sb_book"
         JOIN "subscribers"
           ON "sb_subscriber" = "s_id";
```

Задание 3.1.2.TSK.A: проверить корректность обновления кэширующей таблицы и представлений из решения задачи 3.1.2.b[223].

Задание 3.1.2.TSK.B: создать кэширующее представление, позволяющее получать список всех книг и их жанров (две колонки: первая — название книги, вторая — жанры книги, перечисленные через запятую).

Задание 3.1.2.TSK.C: создать кэширующее представление, позволяющее получать список всех авторов и их книг (две колонки: первая — имя автора, вторая — написанные автором книги, перечисленные через запятую).

Задание 3.1.2.TSK.D: доработать решение[223] задачи 3.1.2.a[223] для MySQL таким образом, чтобы оно учитывало изменения в таблице `subscriptions`, вызванные операцией каскадного удаления (при удалении читателей). Убедиться, что решения для MS SQL Server и Oracle не требуют такой доработки.

3.1.3. ПРИМЕР 28: ИСПОЛЬЗОВАНИЕ ПРЕДСТАВЛЕНИЙ ДЛЯ СОКРЫТИЯ ЗНАЧЕНИЙ И СТРУКТУР ДАННЫХ

Задача 3.1.3.a[{251}]: создать представление, через которое невозможно получить информацию о том, какой конкретно читатель взял ту или иную книгу.

Задача 3.1.3.b[{252}]: создать представление, возвращающее всю информацию из таблицы `subscriptions`, преобразуя даты из полей `sb_start` и `sb_finish` в формат UNIXTIME.

Ожидаемый результат 3.1.3.a.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить результат следующего вида:

sb_id	sb_book	sb_start	sb_finish	sb_is_active
2	1	2011-01-12	2011-02-12	N
3	3	2012-05-17	2012-07-17	Y
42	2	2012-06-11	2012-08-11	N
57	5	2012-06-11	2012-08-11	N
61	7	2014-08-03	2014-10-03	N
62	5	2014-08-03	2014-10-03	Y
86	1	2014-08-03	2014-09-03	Y
91	1	2015-10-07	2015-03-07	Y
95	4	2015-10-07	2015-11-07	N
99	4	2015-10-08	2025-11-08	Y
100	3	2011-01-12	2011-02-12	N

Ожидаемый результат 3.1.3.b.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить результат следующего вида:

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
2	1	1	1294779600	1297458000	N
3	3	3	1337202000	1342472400	Y
42	1	2	1339362000	1344632400	N
57	4	5	1339362000	1344632400	N
61	1	7	1407013200	1412283600	N
62	3	5	1407013200	1412283600	Y
86	3	1	1407013200	1409691600	Y
91	4	1	1444165200	1425675600	Y
95	1	4	1444165200	1446843600	N
99	4	4	1444251600	1762549200	Y
100	1	3	1294779600	1297458000	N

Решение 3.1.3.a[250].

Единственное, что нужно сделать для решения этой задачи, — построить представление, выбирающее все поля таблицы **subscriptions**, кроме поля **sb_subscriber**. Внутренняя логика работы представлений будет совершенно идентична для всех трёх СУБД, и решения будут отличаться только особенностями синтаксиса создания представлений.

MySQL Решение 3.1.3.a

```
1  CREATE VIEW `subscriptions_anonymous`
2  AS
3    SELECT `sb_id`,
4           `sb_book`,
5           `sb_start`,
6           `sb_finish`,
7           `sb_is_active`
8    FROM   `subscriptions`
```

MS SQL Решение 3.1.3.a

```
1  CREATE VIEW [subscriptions_anonymous]
2  WITH SCHEMABINDING
3  AS
4    SELECT [sb_id],
5           [sb_book],
6           [sb_start],
7           [sb_finish],
8           [sb_is_active]
9    FROM   [dbo].[subscriptions]
```

В MS SQL Server мы можем позволить себе повысить надёжность работы представления, указав (строка 2 запроса) СУБД на необходимость установить и отслеживать соответствие между использованием в коде представления объектов базы данных и реальным состоянием таких объектов (их существованием, доступностью и т.д.) не только в момент создания представления, но и в момент любой модификации объектов базы данных, на которые ссылается представление. Для включения этой опции мы также должны указать имя таблицы вместе с именем схемы (**[dbo]**), которой она принадлежит (строка 9 запроса).

Oracle Решение 3.1.3.a

```
1  CREATE VIEW "subscriptions_anonymous"
2  AS
3    SELECT "sb_id",
4           "sb_book",
5           "sb_start",
6           "sb_finish",
7           "sb_is_active"
8    FROM   "subscriptions"
```

Oracle (как и MySQL) не поддерживает опцию **WITH SCHEMABINDING**, потому здесь мы создаём обычное представление с использованием тривиального запроса на выборку.

Решение 3.1.3.b{250}.

Решение данной задачи сводится к построению представления на выборке всех полей из таблицы **subscriptions**, где к полям **sb_start** и **sb_finish** применены функции преобразования из внутреннего формата СУБД представления даты-времени в формат UNIXTIME.

MySQL Решение 3.1.3.b

```
1  CREATE VIEW `subscriptions_unixtime`
2  AS
3    SELECT `sb_id`,
4           `sb_subscriber`,
5           `sb_book`,
6           UNIX_TIMESTAMP(`sb_start`)  AS `sb_start`,
7           UNIX_TIMESTAMP(`sb_finish`) AS `sb_finish`,
8           `sb_is_active`
9    FROM   `subscriptions`
```

В MySQL есть готовая функция для представления даты-времени в формате UNIXTIME, её мы и использовали в строках 6-7.

MS SQL Решение 3.1.3.b

```
1  CREATE VIEW [subscriptions_unixtime]
2  WITH SCHEMABINDING
3  AS
4    SELECT [sb_id],
5           [sb_subscriber],
6           [sb_book],
7          DATEDIFF(SECOND, CAST(N'1970-01-01' AS DATE), [sb_start])
8           AS [sb_start],
9          DATEDIFF(SECOND, CAST(N'1970-01-01' AS DATE), [sb_finish])
10          AS [sb_finish],
11          [sb_is_active]
12   FROM   [subscriptions]
```

В MS SQL Server нет готовой функции для представления даты-времени в формате UNIXTIME, потому в строках 7-10 мы вычисляем UNIXTIME-значение по его определению — т.е. находим количество секунд, прошедших с 1 января 1970 года до указанной даты.

Пояснения относительно опции **WITH SCHEMABINDING** см. в решении{251} задачи 3.1.3.a{250}.

Oracle Решение 3.1.3.b

```
1   CREATE VIEW "subscriptions_unixtime"
2   AS
3     SELECT "sb_id",
4            "sb_subscriber",
5            "sb_book",
6            (("sb_start" - TO_DATE('01-01-1970','DD-MM-YYYY')) * 86400)
7            AS "sb_start",
8            (("sb_finish" - TO_DATE('01-01-1970','DD-MM-YYYY')) * 86400)
9            AS "sb_finish",
10           "sb_is_active"
11    FROM   "subscriptions"
```

В Oracle (как и в MS SQL Server) нет готовой функции для представления даты-времени в формате UNIXTIME, потому в строках 6-9 мы вычисляем UNIXTIME-значение по его определению — т.е. находим количество секунд, прошедших с 1 января 1970 года до указанной даты.

Несмотря на кажущуюся простоту, в самом условии этой задачи кроется ловушка, которая находится не столько в области написания SQL-запросов, сколько в области работы с разными представлениями даты-времени.

Если вы создадите описанные выше представления и выберете с их помощью данные, вы увидите, что в разных СУБД они немного различаются. Так, например, дата «12 января 2011 года» преобразуется следующим образом:

MySQL	MS SQL Server	Oracle
1294779600	1294790400	1294790400

Результаты MS SQL Server и Oracle совпадают и отличаются от результата MySQL на 10800 секунд (т.е. 180 минут, т.е. три часа). Причём оба результата — верные. Просто в решении для MySQL не учтена временная зона (в нашем случае UTC+3), а в двух других решениях учтена.

Чтобы получить в MySQL такой же результат, как в MS SQL Server и Oracle, можно воспользоваться функций **`CONVERT_TZ`** для преобразования временной зоны:

MySQL Решение 3.1.3.b (вариант с преобразованием временной зоны)

```
1   CREATE VIEW `subscriptions_unixtime_tz`
2   AS
3     SELECT `sb_id`,
4            `sb_subscriber`,
5            `sb_book`,
6            UNIX_TIMESTAMP(CONVERT_TZ(`sb_start`, '+00:00', '+03:00'))
7            AS `sb_start`,
8            UNIX_TIMESTAMP(CONVERT_TZ(`sb_finish`, '+00:00', '+03:00'))
9            AS `sb_finish`,
10           `sb_is_active`
11    FROM   `subscriptions`
```

Задание 3.1.3.TSK.A: создать представление, через которое невозможно получить информацию о том, какая конкретно книга была выдана читателю в любой из выдач.

Задание 3.1.3.TSK.B: создать представление, возвращающее всю информацию из таблицы **`subscriptions`**, преобразуя даты из полей **`sb_start`** и **`sb_finish`** в формат «ГГГГ-ММ-ДД НН», где «НН» — день недели в виде своего полного названия (т.е. «Понедельник», «Вторник» и т.д.)

3.2. МОДИФИКАЦИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ПРЕДСТАВЛЕНИЙ

3.2.1. ПРИМЕР 29: МОДИФИКАЦИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ «ПРОЗРАЧНЫХ» ПРЕДСТАВЛЕНИЙ

Задача 3.2.1.a{255}: создать представление, извлекающее информацию о читателях, переводя весь текст в верхний регистр и при этом допускающее модификацию списка читателей.

Задача 3.2.1.b{260}: создать представление, извлекающее информацию о датах выдачи и возврата книг в виде единой строки и при этом допускающее обновление информации в таблице `subscriptions`.

Ожидаемый результат 3.2.1.a.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить представленный ниже результат, и при этом выполненные с представлением операции `INSERT`, `UPDATE`, `DELETE` модифицируют соответствующие данные в исходной таблице.

s_id	s_name
1	ИВАНОВ И.И.
2	ПЕТРОВ П.П.
3	СИДОРОВ С.С.
4	СИДОРОВ С.С.

Ожидаемый результат 3.2.1.b.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить представленный ниже результат, и при этом выполненные с представлением операции `INSERT`, `UPDATE`, `DELETE` модифицируют соответствующие данные в исходной таблице.

sb_id	sb_subscriber	sb_book	sb_dates	sb_is_active
2	1	1	2011-02-12 - 2011-02-12	N
3	3	3	2012-05-17 - 2012-07-17	Y
42	1	2	2012-06-11 - 2012-08-11	N
57	4	5	2012-06-11 - 2012-08-11	N
61	1	7	2014-08-03 - 2014-10-03	N
62	3	5	2014-08-03 - 2014-10-03	Y
86	3	1	2014-08-03 - 2014-09-03	Y
91	4	1	2015-10-07 - 2015-03-07	Y
95	1	4	2015-10-07 - 2015-11-07	N
99	4	4	2015-10-08 - 2025-11-08	Y
100	1	3	2011-01-12 - 2011-02-12	N

Решение 3.2.1.a{254}.

Создать представление, позволяющее извлечь из базы данных информацию в требуемой форме, легко. Достаточно использовать функцию **UPPER** для приведения имени читателя к верхнему регистру (строка 4).

MySQL Решение 3.2.1.a (представление, допускающее только удаление данных)

```
1  CREATE VIEW `subscribers_upper_case`
2  AS
3    SELECT `s_id`,
4           UPPER(`s_name`) AS `s_name`
5    FROM   `subscribers`
```

Проблема заключается в том, что MySQL налагает широкий спектр ограничений[11] на обновляемые представления, среди которых есть и использование функций для преобразования значений полей. Мы используем функцию **UPPER**, что приводит к невозможности выполнить операции вставки и обновления с использованием полученного представления (удаление будет работать).

Обойти ограничение на обновление можно следующим образом: в представлении нужно выбирать как «нетронутое» исходное поле, так и его обработанную копию. Этим мы частично нарушим условие задачи, по которому представление должно возвращать только два поля, одноимённые полям исходной таблицы, но зато получим возможность выполнять обновление:

MySQL Решение 3.2.1.a (представление, допускающее удаление и обновление данных)

```
1  CREATE VIEW `subscribers_upper_case_trick`
2  AS
3    SELECT `s_id`,
4           `s_name`,
5           UPPER(`s_name`) AS `s_name_upper`
6    FROM   `subscribers`
```

Теперь у нас уже работают и удаление, и обновление. Вы можете выполнить следующие запросы, чтобы проверить данное утверждение.

11 http://dev.mysql.com/doc/refman/5.6/en/view-updatability.html

MySQL Решение 3.2.1.a (проверка работы обновления и удаления)

```
UPDATE `subscribers_upper_case_trick`
SET    `s_name` = 'Сидоров А.А.'
WHERE  `s_id` = 4;

UPDATE `subscribers_upper_case_trick`
SET    `s_id` = 10
WHERE  `s_id` = 4;

DELETE FROM `subscribers_upper_case`
WHERE  `s_id` = 10;
```

К сожалению, реализовать вставку через такое представление не получится: чтобы вставка работала, представление не должно несколько раз ссылаться на одно и то же поле исходной таблицы.

Таким образом, для MySQL поставленная задача решается лишь частично.

В MS SQL Server тоже есть серия ограничений[12], налагаемых на представления, с помощью которых планируется модифицировать данные. Однако (в отличие от MySQL) MS SQL Server допускает создание на представлениях триггеров, с помощью которых мы можем решить поставленную задачу.

MS SQL Решение 3.2.1.a (создание представления)

```
CREATE VIEW [subscribers_upper_case]
WITH SCHEMABINDING
AS
  SELECT [s_id],
         UPPER([s_name]) AS [s_name]
  FROM   [dbo].[subscribers]
```

Такое представление уже позволяет извлекать данные в указанном в условии задачи формате, а также выполнять удаление данных. Для того, чтобы через это представление можно было выполнять вставку и обновление данных, нужно создать на нём два триггера — на операциях вставки и обновления.

MS SQL Решение 3.2.1.a (создание триггера для реализации операции вставки)

```
CREATE TRIGGER [subscribers_upper_case_ins]
ON [subscribers_upper_case]
INSTEAD OF INSERT
AS
    SET IDENTITY_INSERT [subscribers] ON;
    INSERT INTO [subscribers]
                ([s_id],
                 [s_name])
    SELECT ( CASE
               WHEN [s_id] IS NULL
                    OR [s_id] = 0 THEN IDENT_CURRENT('subscribers')
                                       + IDENT_INCR('subscribers')
                                       + ROW_NUMBER() OVER (ORDER BY
                                                            (SELECT 1))
                                       - 1
               ELSE [s_id]
             END ) AS [s_id],
           [s_name]
    FROM   [inserted];
    SET IDENTITY_INSERT [subscribers] OFF;
GO
```

[12] https://msdn.microsoft.com/en-us/library/ms187956.aspx

Такой триггер выполняется **вместо** (**INSTEAD OF**) операции вставки данных в представление и внутри себя выполняет вставку данных в таблицу, на которой построено представление.

Некоторая сложность обусловлена тем, что мы хотим разрешить вставку как только одного поля (имени читателя), так и обоих полей (идентификатора читателя и имени читателя), и мы не знаем заранее, будет ли при операции вставки передано значение идентификатора **s_id**.

В строках 5 и 20 мы соответственно разрешаем и снова запрещаем явную вставку данных в поле **s_id** (оно является **IDENTITY**-полем для таблицы **subscribers**).

В строках 9-17 мы проверяем, получили ли мы явно указанное значение **s_id**. Если явно указанное значение не было передано, поле **s_id** псевдотаблицы **inserted** может принять значение **NULL** или 0 — в таком случае мы вычисляем новое значение поля **s_id** для таблицы **subscribers** на основе функций, возвращающих текущее значение **IDENTITY**-поля (**IDENT_CURRENT**) и шаг его инкремента (**IDENT_INCR**), а также номера строки из таблицы **inserted**. Иными словами, формула вычисления нового значения **IDENTITY**-поля такова: **текущее_значение + шаг_инкремента + номер_вставляемой_строки - 1**.

Теперь одинаково корректно будет выполняться каждый из следующих запросов на вставку данных:

MS SQL Решение 3.2.1.a (проверка работоспособности вставки)

```
1  INSERT INTO [subscribers_upper_case]
2              ([s_name])
3  VALUES      (N'Орлов О.О.');
4
5  INSERT INTO [subscribers_upper_case]
6              ([s_name])
7  VALUES      (N'Соколов С.С.'),
8              (N'Беркутов Б.Б.');
9
10 INSERT INTO [subscribers_upper_case]
11             ([s_id],
12              [s_name])
13 VALUES      (30,
14              N'Ястребов Я.Я.');
15
16 INSERT INTO [subscribers_upper_case]
17             ([s_id],
18              [s_name])
19 VALUES      (31,
20              N'Синицын С.С.'),
21             (32,
22              N'Воронов В.В.');
```

Переходим к реализации обновления данных.

MS SQL Решение 3.2.1.a (создание триггера для реализации операции обновления)

```
1  CREATE TRIGGER [subscribers_upper_case_upd]
2  ON [subscribers_upper_case]
3  INSTEAD OF UPDATE
4  AS
5      IF UPDATE([s_id])
6        BEGIN
7            RAISERROR ('UPDATE of Primary Key through
8                        [subscribers_upper_case_upd]
9                       view is prohibited.', 16, 1);
10           ROLLBACK;
11       END
12       ELSE
13        UPDATE [subscribers]
14        SET    [subscribers].[s_name] = [inserted].[s_name]
15        FROM   [subscribers]
16          JOIN [inserted]
17            ON [subscribers].[s_id] = [inserted].[s_id];
18 GO
```

При выполнении обновления данных «старые» строки копируются в псевдотаблицу **deleted**, а «новые» в псевдотаблицу **inserted**, но существует непреодолимая проблема: не существует никакого способа **гарантированно** определить взаимное соответствие строк в этих двух псевдотаблицах, т.е. в случае изменения значения первичного ключа, мы не сможем определить его новое значение.

Потому в строках 5-11 кода триггера мы проверяем, была ли попытка обновить значение первичного ключа, с помощью функции **UPDATE** (да, здесь это **не** оператор вставки, а функция). Если такая попытка была, мы запрещаем выполнение операции и откатываем транзакцию. Если же первичный ключ не был затронут обновлением, мы можем легко определить новое значение имени читателя и использовать его для настоящего обновления данных в таблице **subscribers** (строки 12-17).

Теперь следующие запросы выполняются корректно (к слову, удаление и так не требовало никаких доработок, но проверить всё же стоит):

MS SQL Решение 3.2.1.a (проверка работоспособности обновления и удаления)

```
1  UPDATE [subscribers_upper_case]
2  SET    [s_name] = N'Новое имя'
3  WHERE  [s_id] = 30;
4
5  UPDATE [subscribers_upper_case]
6  SET    [s_name] = N'И ещё одно имя'
7  WHERE  [s_id] >= 31;
8
9  DELETE FROM [subscribers_upper_case]
10 WHERE  [s_id] = 30;
11
12 DELETE FROM [subscribers_upper_case]
13 WHERE  [s_id] >= 31;
```

Попытка обновить значение первичного ключа закономерно приведёт к блокировке операции:

MS SQL Решение 3.2.1.a (проверка невозможности обновления значения первичного ключа)

```
1  UPDATE [subscribers_upper_case]
2  SET    [s_id] = 50
3  WHERE  [s_id] = 1;
```

В результате выполнения такого запроса будет получено следующее сообщение об ошибке:

```
Msg 50000, Level 16, State 1, Procedure subscribers_upper_case_upd, Line 7
UPDATE of Primary Key through [subscribers_upper_case_upd] view is prohibited.
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

На этом решение данной задачи для MS SQL Server завершено, и мы переходим к рассмотрению решения для Oracle.

Создадим представление.

Oracle Решение 3.2.1.a (создание представления)

```
1  CREATE VIEW "subscribers_upper_case"
2  AS
3    SELECT "s_id",
4           UPPER("s_name") AS "s_name"
5    FROM   "subscribers"
```

Через это представление уже можно извлекать данные в требуемом формате и удалять данные.

Важно: если поиск записей для удаления происходит по полю **s_name**, необходимо передавать искомые данные **в верхнем регистре**. Эта особенность касается только Oracle, т.к. MySQL и MS SQL Server не налагают подобного ограничения.

Для того, чтобы через это представление можно было выполнять вставку и обновление данных, нужно создать на нём два триггера — на операциях вставки и обновления.

Поскольку (в отличие от MS SQL Server) Oracle поддерживает триггеры уровня отдельных записей, их код получается очень простым, а реализация позволяет обойти имеющееся в MS SQL Server ограничение на обновление первичного ключа.

Oracle Решение 3.2.1.a (создание триггера для реализации операции вставки)

```
1  CREATE OR REPLACE TRIGGER "subscribers_upper_case_ins"
2  INSTEAD OF INSERT ON "subscribers_upper_case"
3  FOR EACH ROW
4   BEGIN
5      INSERT INTO "subscribers"
6                  ("s_id",
7                   "s_name")
8      VALUES     (:new."s_id",
9                  :new."s_name");
10  END;
```

Oracle Решение 3.2.1.a (создание триггера для реализации операции обновления)

```
1  CREATE OR REPLACE TRIGGER "subscribers_upper_case_ins"
2  INSTEAD OF UPDATE ON "subscribers_upper_case"
3  FOR EACH ROW
4   BEGIN
5      UPDATE "subscribers"
6      SET    "s_id" = :new."s_id",
7             "s_name" = :new."s_name"
8      WHERE  "s_id" = :old."s_id";
9   END;
```

Код обоих триггеров сводится к выполнению запроса на вставку или обновление к таблице, на которой построено представление. Как и в MySQL мы можем использовать в Oracle ключевые слова **old** и **new** для обращения к «старым» (удаляемым или обновляемым) и новым (добавляемым или обновлённым) данным.

Теперь следующие запросы выполняются корректно (к слову, удаление и так не требовало никаких доработок, но проверить всё же стоит):

Oracle Решение 3.2.1.a (проверка работоспособности модификации данных)

```
INSERT ALL
 INTO "subscribers" ("s_id", "s_name") VALUES (1, N'Соколов С.С.')
 INTO "subscribers" ("s_id", "s_name") VALUES (2, N'Беркутов Б.Б.')
 INTO "subscribers" ("s_id", "s_name") VALUES (3, N'Филинов Ф.Ф.')
SELECT 1 FROM "DUAL";

UPDATE "subscribers_upper_case"
SET    "s_name" = N'Синицын З.З.'
WHERE  "s_id" = 6;

-- Такой запрос НЕ НАЙДЁТ искомое, т.к. имя должно быть в верхнем регистре:
UPDATE "subscribers_upper_case"
SET    "s_name" = N'Синцын С.С.'
WHERE  "s_name" = N'Синицын З.З.';

-- А такой запрос найдёт искомое:
UPDATE "subscribers_upper_case"
SET    "s_name" = N'Синцын С.С.'
WHERE  "s_name" = N'СИНИЦЫН З.З.';

DELETE FROM "subscribers_upper_case"
WHERE  "s_id" = 6;

-- Такой запрос НЕ НАЙДЁТ искомое, т.к. имя должно быть в верхнем регистре:
DELETE FROM "subscribers_upper_case"
WHERE  "s_name" = N'Филинов Ф.Ф.';

-- А такой запрос найдёт искомое:
DELETE FROM "subscribers_upper_case"
WHERE  "s_name" = N'ФИЛИНОВ Ф.Ф.';

DELETE FROM "subscribers_upper_case"
WHERE  "s_id" > 4;
```

Решение 3.2.1.b{254}.

Решение этой задачи для MySQL подпадает под все ограничения, характерные для решения{255} задачи 3.2.1.a{254}: мы также можем создать представление или удовлетворяющее формату выборки и допускающее лишь удаление данных, или не удовлетворяющее формату выборки (с дополнительными полями) и допускающее обновление и удаление данных. Но вставка данных всё равно работать не будет.

MySQL Решение 3.2.1.b (представление, допускающее только удаление данных)

```
CREATE VIEW `subscriptions_wcd`
AS
  SELECT `sb_id`,
         `sb_subscriber`,
         `sb_book`,
         CONCAT(`sb_start`, ' - ', `sb_finish`) AS `sb_dates`,
         `sb_is_active`
  FROM   `subscriptions`
```

MySQL Решение 3.2.1.b (представление, допускающее удаление и обновление данных)

```
CREATE VIEW `subscriptions_wcd_trick`
AS
  SELECT `sb_id`,
         `sb_subscriber`,
         `sb_book`,
         CONCAT(`sb_start`, ' - ', `sb_finish`) AS `sb_dates`,
         `sb_start`,
         `sb_finish`,
         `sb_is_active`
  FROM   `subscriptions`
```

За исключением уже неоднократно упомянутой проблемы со вставкой, не позволяющей полностью решить поставленную задачу в MySQL, код представлений совершенно тривиален и построен на элементарных запросах на выборку.

Упомянутые в решении{255} задачи 3.2.1.a{254} ограничения MS SQL Server относительно обновляемых представлений актуальны и в данной задаче: мы снова будем вынуждены создавать **INSTEAD OF** триггеры для реализации обновления и вставки данных.

MS SQL Решение 3.2.1.b (создание представления)

```
CREATE VIEW [subscriptions_wcd]
WITH SCHEMABINDING
AS
  SELECT [sb_id],
         [sb_subscriber],
         [sb_book],
         CONCAT([sb_start], ' - ', [sb_finish]) AS [sb_dates],
         [sb_is_active]
  FROM   [dbo].[subscriptions]
```

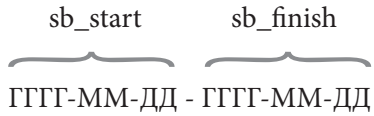
Такое представление уже позволяет извлекать данные в указанном в условии задачи формате, а также выполнять удаление данных. Для того, чтобы через это представление можно было выполнять вставку и обновление данных, нужно создать на нём два триггера — на операциях вставки и обновления.

MS SQL Решение 3.2.1.b (создание триггера для реализации операции вставки)

```
1   CREATE TRIGGER [subscriptions_wcd_ins]
2   ON [subscriptions_wcd]
3   INSTEAD OF INSERT
4   AS
5       SET IDENTITY_INSERT [subscriptions] ON;
6       INSERT INTO [subscriptions]
7                   ([sb_id],
8                    [sb_subscriber],
9                    [sb_book],
10                   [sb_start],
11                   [sb_finish],
12                   [sb_is_active])
13      SELECT ( CASE
14                  WHEN [sb_id] IS NULL
15                       OR [sb_id] = 0 THEN IDENT_CURRENT('subscriptions')
16                                           + IDENT_INCR('subscriptions')
17                                           + ROW_NUMBER() OVER (ORDER BY
18                                                                (SELECT 1))
19                                           - 1
20                  ELSE [sb_id]
21                END ) AS [sb_id],
22              [sb_subscriber],
23              [sb_book],
24              SUBSTRING([sb_dates], 1, (CHARINDEX(' ', [sb_dates]) - 1))
25              AS [sb_start],
26              SUBSTRING([sb_dates], (CHARINDEX(' ', [sb_dates]) + 3),
27                        DATALENGTH([sb_dates]) -
28                        (CHARINDEX(' ', [sb_dates]) + 2))
29              AS [sb_finish],
30              [sb_is_active]
31      FROM    [inserted];
32      SET IDENTITY_INSERT [subscriptions] OFF;
33  GO
```

Нетривиальная логика получения значения первичного ключа (строки 13-21) подробно объяснена в решении{255} задачи 3.2.1.a{254}.

Что касается получения значений полей **sb_start** и **sb_finish** (строки 24-25 и 26-29), то здесь мы наблюдаем последствия ещё одного ограничения MS SQL Server: в псевдотаблице **inserted** нет полей, которых нет в представлении, на котором построен триггер. Т.е. единственный способ[13] получить значения этих полей — извлечь их из значения поля **sb_dates**. Это выглядит следующим образом (части строки, содержащей две даты, извлекаются с помощью строковых функций):



Такой подход является медленным и ненадёжным, но альтернатив ему нет. Если мы хотим повысить надёжность работы триггера, можно добавить дополнительную проверку на корректность формата «комбинированной даты» в поле **sb_dates**, но каждая такая дополнительная операция негативно отразится на производительности.

Проверим, как будет работать вставка данных с использованием созданного триггера:

13 https://technet.microsoft.com/en-us/library/ms190188%28v=sql.105%29.aspx

MS SQL Решение 3.2.1.b (проверка работоспособности вставки)

```
INSERT INTO [subscriptions_wcd]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_dates],
             [sb_is_active])
VALUES      (1000,
             1,
             3,
             '2017-01-12 - 2017-03-15',
             'N'),
            (2000,
             1,
             1,
             '2017-01-12 - 2017-03-15',
             'N');

INSERT INTO [subscriptions_wcd]
            ([sb_subscriber],
             [sb_book],
             [sb_dates],
             [sb_is_active])
VALUES      (1,
             3,
             '2019-01-12 - 2019-03-15',
             'N'),
            (1,
             1,
             '2019-01-12 - 2019-03-15',
             'N');

INSERT INTO [subscriptions_wcd]
            ([sb_subscriber],
             [sb_book],
             [sb_dates],
             [sb_is_active])
VALUES      (1,
             3,
             'Это -- не даты, а ерунда.',
             'N');
```

Первые два запроса работают корректно, а третий ожидаемо приводит к возникновению ошибочной ситуации:

```
Msg 241, Level 16, State 1, Procedure subscriptions_wcd_ins, Line 6
Conversion failed when converting date and/or time from character string.
```

Переходим к реализации обновления данных.

MS SQL Решение 3.2.1.b (создание триггера для реализации операции обновления)

```
1   CREATE TRIGGER [subscriptions_wcd_upd]
2   ON [subscriptions_wcd]
3   INSTEAD OF UPDATE
4   AS
5       IF UPDATE([sb_id])
6         BEGIN
7             RAISERROR ('UPDATE of Primary Key through
8                         [subscriptions_wcd_upd]
9                         view is prohibited.', 16, 1);
10            ROLLBACK;
11        END
12        ELSE
13         UPDATE [subscriptions]
14         SET    [subscriptions].[sb_subscriber] = [inserted].[sb_subscriber],
15                [subscriptions].[sb_book] = [inserted].[sb_book],
16                [subscriptions].[sb_start] =
17                                    SUBSTRING([sb_dates], 1,
18                                    (CHARINDEX(' ', [sb_dates]) - 1)),
19                [subscriptions].[sb_finish] =
20                                    SUBSTRING([sb_dates],
21                                    (CHARINDEX(' ', [sb_dates]) + 3),
22                                    DATALENGTH([sb_dates]) -
23                                    (CHARINDEX(' ', [sb_dates]) + 2)),
24                [subscriptions].[sb_is_active] = [inserted].[sb_is_active]
25         FROM   [subscriptions]
26           JOIN [inserted]
27             ON [subscriptions].[sb_id] = [inserted].[sb_id];
28  GO
```

Логика запрета обновления первичного ключа (строки 5-11) подробно объяснена в решении задачи 3.2.1.a.

Необходимость получать значения полей **sb_start** и **sb_finish** (строки 16-23) только что была рассмотрена в реализации **INSERT**-триггера.

В остальном запрос в строках 14-27 представляет собой классическую реализацию обновления на основе выборки.

Остаётся убедиться, что обновление и удаление работает корректно:

MS SQL Решение 3.2.1.b (проверка работоспособности обновления и удаления)

```
1   UPDATE [subscriptions_wcd]
2   SET    [sb_dates] = '2021-01-12 - 2021-03-15'
3   WHERE  [sb_id] = 1000;
4
5   DELETE FROM [subscriptions_wcd]
6   WHERE  [sb_id] = 2000;
7
8   DELETE FROM [subscriptions_wcd]
9   WHERE  [sb_dates] = '2021-01-12 - 2021-03-15';
```

Попытка обновить значение первичного ключа закономерно приведёт к блокировке операции:

MS SQL Решение 3.2.1.b (проверка невозможности обновления значения первичного ключа)

```
1   UPDATE [subscriptions_wcd]
2   SET    [sb_id] = 999
3   WHERE  [sb_id] = 1000;
```

В результате выполнения такого запроса будет получено следующее сообщение об ошибке:

```
Msg 50000, Level 16, State 1, Procedure subscriptions_wcd_upd, Line 7
UPDATE of Primary Key through [subscriptions_wcd_upd] view is prohibited.
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

На этом решение данной задачи для MS SQL Server завершено, и мы переходим к рассмотрению решения для Oracle.

Oracle Решение 3.2.1.b (создание представления)

```
1  CREATE VIEW "subscriptions_wcd"
2  AS
3    SELECT "sb_id",
4           "sb_subscriber",
5           "sb_book",
6           TO_CHAR("sb_start", 'YYYY-MM-DD') || ' - ' ||
7           TO_CHAR("sb_finish", 'YYYY-MM-DD') AS "sb_dates",
8           "sb_is_active"
9    FROM   "subscriptions"
```

Использование функции **TO_CHAR** в строках 6-7 позволяет получить строку с датами в определённом условием задачи формате.

Oracle Решение 3.2.1.b (создание триггера для реализации операции вставки)

```
1  CREATE OR REPLACE TRIGGER "subscriptions_wcd_ins"
2    INSTEAD OF INSERT ON "subscriptions_wcd"
3    FOR EACH ROW
4     BEGIN
5      INSERT INTO "subscriptions"
6                  ("sb_id",
7                   "sb_subscriber",
8                   "sb_book",
9                   "sb_start",
10                  "sb_finish",
11                  "sb_is_active")
12     VALUES       (:new."sb_id",
13                   :new."sb_subscriber",
14                   :new."sb_book",
15                   TO_DATE(SUBSTR(:new."sb_dates", 1,
16                          (INSTR(:new."sb_dates", ' ') - 1 )), 'YYYY-MM-DD'),
17                   TO_DATE(SUBSTR(:new."sb_dates",
18                          (INSTR(:new."sb_dates", ' ') + 3 )), 'YYYY-MM-DD'),
19                   :new."sb_is_active");
20    END;
```

Oracle Решение 3.2.1.b (создание триггера для реализации операции обновления)

```
CREATE OR REPLACE TRIGGER "subscriptions_wcd_ins"
INSTEAD OF UPDATE ON "subscriptions_wcd"
FOR EACH ROW
  BEGIN
    UPDATE "subscriptions"
    SET    "sb_id" = :new."sb_id",
           "sb_subscriber" = :new."sb_subscriber",
           "sb_book" = :new."sb_book",
           "sb_start" = TO_DATE(SUBSTR(:new."sb_dates", 1,
                                (INSTR(:new."sb_dates", ' ') - 1)),
                                'YYYY-MM-DD'),
           "sb_finish" = TO_DATE(SUBSTR(:new."sb_dates",
                                (INSTR(:new."sb_dates", ' ') + 3 )),
                                'YYYY-MM-DD'),
           "sb_is_active" = :new."sb_is_active"
    WHERE  "sb_id" = :old."sb_id";
  END;
```

Благодаря поддержке триггеров уровня отдельных записей, решение этой задачи в Oracle оказывается достаточно простым: код обоих триггеров сводится к выполнению запроса на вставку или обновление к таблице, на которой построено представление. Как и в MySQL мы можем использовать в Oracle ключевые слова **old** и **new** для обращения к «старым» (удаляемым или обновляемым) и новым (добавляемым или обновлённым) данным.

Значения полей **sb_start** и **sb_finish** (как и в решении для MS SQL Server) в обоих триггерах приходится вычислять с помощью строковых функций, но в Oracle их синтаксис немного проще, и решение получается короче.

Теперь все следующие запросы работают корректно:

Oracle Решение 3.2.1.b (проверка работоспособности модификации данных)

```
INSERT INTO "subscriptions_wcd"
            ("sb_subscriber",
             "sb_book",
             "sb_dates",
             "sb_is_active")
VALUES      (1,
             3,
             '2019-01-12 - 2019-02-12',
             'N');

UPDATE "subscriptions_wcd"
SET    "sb_dates" = '2019-01-12 - 2019-02-12'
WHERE  "sb_id" = 100;

DELETE FROM "subscriptions_wcd"
WHERE  "sb_id" = 100;

DELETE FROM "subscriptions_wcd"
WHERE  "sb_dates" = '2012-05-17 - 2012-07-17';
```

Важно! Oracle допускает вставку данных через представление только с использованием синтаксиса вида

```
INSERT INTO ... (...) VALUES (...)
```

но не вида

```
INSERT ALL
 INTO ... (...) VALUES (…)
 INTO ... (...) VALUES (…)
SELECT 1 FROM "DUAL"
```

При попытке использовать второй вариант вы получите сообщение об ошибке «ORA-01702: a view is not appropriate here».

Задание 3.2.1.TSK.A: создать представление, извлекающее информацию о книгах, переводя весь текст в верхний регистр и при этом допускающее модификацию списка книг.

Задание 3.2.1.TSK.B: создать представление, извлекающее информацию о датах выдачи и возврата книг и состоянии выдачи книги в виде единой строки в формате «ГГГГ-ММ-ДД - ГГГГ-ММ-ДД - Возвращена» и при этом допускающее обновление информации в таблице `subscriptions`.

3.2.2. ПРИМЕР 30: МОДИФИКАЦИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТРИГГЕРОВ НА ПРЕДСТАВЛЕНИЯХ

Поскольку MySQL не позволяет создавать триггеры на представлениях, все задачи этого примера имеют полноценные решения только для MS SQL Server и Oracle.

Задача 3.2.2.a{268}: создать представление, извлекающее из таблицы `subscriptions` человекочитаемую (с именами читателей и названиями книг вместо идентификаторов) информацию, и при этом позволяющее модифицировать данные в таблице `subscriptions`.

Задача 3.2.2.b{281}: создать представление, показывающее список книг с относящимися к этим книгам жанрами, и при этом позволяющее добавлять новые жанры.

Ожидаемый результат 3.2.2.a.

Выполнение запроса вида `SELECT * FROM {представление}` позволяет получить представленный ниже результат, и при этом выполненные с представлением операции `INSERT`, `UPDATE`, `DELETE` модифицируют соответствующие данные в исходной таблице.

sb_id	sb_subscriber	sb_book	sb_start	sb_finish	sb_is_active
2	Иванов И.И.	Евгений Онегин	2011-01-12	2011-02-12	N
3	Сидоров С.С.	Основание и империя	2012-05-17	2012-07-17	Y
42	Иванов И.И.	Сказка о рыбаке и рыбке	2012-06-11	2012-08-11	N
57	Сидоров С.С.	Язык программирования C++	2012-06-11	2012-08-11	N
61	Иванов И.И.	Искусство программирования	2014-08-03	2014-10-03	N
62	Сидоров С.С.	Язык программирования C++	2014-08-03	2014-10-03	Y
86	Сидоров С.С.	Евгений Онегин	2014-08-03	2014-09-03	Y
91	Сидоров С.С.	Евгений Онегин	2015-10-07	2015-03-07	Y
95	Иванов И.И.	Психология программирования	2015-10-07	2015-11-07	N
99	Сидоров С.С.	Психология программирования	2015-10-08	2025-11-08	Y
100	Иванов И.И.	Основание и империя	2011-01-12	2011-02-12	N

Ожидаемый результат 3.2.2.b.

Выполнение запроса вида **`SELECT * FROM {представление}`** позволяет получить представленный ниже результат, и при этом выполненная с представлением операция **`INSERT`** приводит к добавлению нового жанра в таблицу **`genres`**.

b_id	b_name	genres
1	Евгений Онегин	Классика,Поэзия
2	Сказка о рыбаке и рыбке	Классика,Поэзия
3	Основание и империя	Фантастика
4	Психология программирования	Программирование,Психология
5	Язык программирования C++	Программирование
6	Курс теоретической физики	Классика
7	Искусство программирования	Программирование,Классика

Решение 3.2.2.a[{267}].

К сожалению, для MySQL эта задача не имеет решения, т.к. MySQL не позволяет создавать триггеры на представлениях. Максимум, что мы можем сделать, это создать само представление, но данные через него модифицировать не получится:

MySQL Решение 3.2.2.a (создание представления)

```
1  CREATE VIEW `subscriptions_with_text`
2  AS
3  SELECT `sb_id`,
4         `s_name` AS `sb_subscriber`,
5         `b_name` AS `sb_book`,
6         `sb_start`,
7         `sb_finish`,
8         `sb_is_active`
9  FROM   `subscriptions`
10        JOIN `subscribers` ON `sb_subscriber` = `s_id`
11        JOIN `books` ON `sb_book` = `b_id`
```

В MS SQL Server код самого представления идентичен:

MS SQL Решение 3.2.2.a (создание представления)

```
1   CREATE VIEW [subscriptions_with_text]
2   WITH SCHEMABINDING
3   AS
4   SELECT [sb_id],
5          [s_name] AS [sb_subscriber],
6          [b_name] AS [sb_book],
7          [sb_start],
8          [sb_finish],
9          [sb_is_active]
10  FROM   [dbo].[subscriptions]
11         JOIN [dbo].[subscribers] ON [sb_subscriber] = [s_id]
12         JOIN [dbo].[books] ON [sb_book] = [b_id]
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

Поскольку исходная таблица **subscriptions** содержит в полях **sb_subscriber** и **sb_book** числовые идентификаторы читателя и книги, мы обязаны получать вставляемые значения этих полей в числовом виде.

Однако попытка «на лету» получить идентификаторы читателя или книги на основе их имени или названия не может быть реализована потому, что как имена читателей, так и названия книг могут дублироваться, и мы не можем гарантировать получение корректного значения.

Чтобы минимизировать вероятность неверного использования полученного триггера, в его строках 5-14 происходит проверка того, не переданы ли во время вставки нечисловые значения в поля **sb_subscriber** или **sb_book**. Если такая ситуация возникла, выводится сообщение об ошибке и транзакция отменяется.

В остальном весь код триггера предельно похож на рассмотренный в решении{255} задачи 3.2.1.a{254} (там же подробно описана логика вычисления нового значения первичного ключа, если таковое не передано явно в запросе на вставку данных — строки 25-33 представленного ниже триггера).

MS SQL Решение 3.2.2.a (создание триггера для реализации операции вставки)

```
1   CREATE TRIGGER [subscriptions_with_text_ins]
2   ON [subscriptions_with_text]
3   INSTEAD OF INSERT
4   AS
5      IF EXISTS(SELECT 1
6                FROM   [inserted]
7                WHERE  PATINDEX('%[^0-9]%', [sb_subscriber]) > 0
8                    OR PATINDEX('%[^0-9]%', [sb_book]) > 0)
9         BEGIN
10         RAISERROR ('Use digital identifiers for [sb_subscriber]
11                     and [sb_book]. Do not use subscribers'' names
12                     or books'' titles', 16, 1);
13         ROLLBACK;
14        END
15        ELSE
16         BEGIN
17         SET IDENTITY_INSERT [subscriptions] ON;
18         INSERT INTO [subscriptions]
19                     ([sb_id],
20                      [sb_subscriber],
21                      [sb_book],
22                      [sb_start],
23                      [sb_finish],
24                      [sb_is_active])
```

MS SQL Решение 3.2.2.a (создание триггера для реализации операции вставки) (продолжение)

```
25          SELECT ( CASE
26                     WHEN [sb_id] IS NULL
27                          OR [sb_id] = 0 THEN IDENT_CURRENT('subscriptions')
28                                              + IDENT_INCR('subscriptions')
29                                              + ROW_NUMBER() OVER (ORDER BY
30                                                                   (SELECT 1))
31                                              - 1
32                     ELSE [sb_id]
33                   END ) AS [sb_id],
34                 [sb_subscriber],
35                 [sb_book],
36                 [sb_start],
37                 [sb_finish],
38                 [sb_is_active]
39          FROM   [inserted];
40          SET IDENTITY_INSERT [subscriptions] OFF;
41        END
42  GO
```

Проверим, как выполняются следующие запросы на вставку. Вполне ожидаемо запросы 1-2 выполняются корректно, а запросы 3-5 приводят к срабатыванию кода в строках 5-14 триггера и отмене транзакции, т.к. передача нечисловых значений для полей **sb_subscriber** и/или **sb_book** запрещена.

MS SQL Решение 3.2.2.a (проверка работоспособности операции вставки)

```
1   -- Запрос 1 (вставка выполняется):
2   INSERT INTO [subscriptions_with_text]
3               ([sb_id],
4                [sb_subscriber],
5                [sb_book],
6                [sb_start],
7                [sb_finish],
8                [sb_is_active])
9   VALUES      (5000,
10               1,
11               3,
12               '2015-01-12',
13               '2015-02-12',
14               'N'),
15              (5005,
16               1,
17               1,
18               '2015-01-12',
19               '2015-02-12',
20               'N');
21
```

MS SQL Решение 3.2.2.а (проверка работоспособности операции вставки) (продолжение)

```
-- Запрос 2 (вставка выполняется):
INSERT INTO [subscriptions_with_text]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (1,
             3,
             '2015-01-12',
             '2015-02-12',
             'N'),
            (1,
             1,
             '2015-01-12',
             '2015-02-12',
             'N');

-- Запрос 3 (вставка НЕ выполняется):
INSERT INTO [subscriptions_with_text]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (N'Иванов И.И.',
             3,
             '2015-01-12',
             '2015-02-12',
             'N');

-- Запрос 4 (вставка НЕ выполняется):
INSERT INTO [subscriptions_with_text]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (1,
             N'Какая-то книга',
             '2015-01-12',
             '2015-02-12',
             'N');
-- Запрос 5 (вставка НЕ выполняется):
INSERT INTO [subscriptions_with_text]
            ([sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (N'Какой-то читатель',
             N'Какая-то книга',
             '2015-01-12',
             '2015-02-12',
             'N');
```

Создадим триггер, позволяющий реализовать операцию обновления данных. Его логика будет несколько сложнее, чем в только что рассмотренном **INSTEAD OF INSERT** триггере.

Мы должны реагировать на нечисловые значения в полях **sb_subscriber** и **sb_book** только в том случае, если эти значения были явно переданы в запросе, — этим вызвана необходимость создания более сложного условия в строках 7-10.

Если поля **sb_subscriber** и **sb_book** не были переданы в **UPDATE**-запросе, в них естественным образом появятся человекочитаемые имена читателя и название книги (т.к. псевдотаблицы **inserted** и **deleted** наполняются данными из представления).

Эту ситуацию мы рассматриваем и исправляем в строках 29-42: если в полях оказываются нечисловые данные (и выполнение триггера дошло до этой части), значит в **UPDATE**-запросе эти поля не фигурируют, и их значения нужно взять из исходной таблицы (**subscriptions**).

И, наконец, как и было подчёркнуто в решении{255} задачи 3.2.1.a{254}, мы не можем корректно определить взаимоотношение записей в псевдотаблицах **inserted** и **deleted**, если было изменено значение первичного ключа, поэтому мы запрещаем эту операцию (строки 19-25).

MS SQL Решение 3.2.2.a (создание триггера для реализации операции обновления)

```
1   CREATE TRIGGER [subscriptions_with_text_upd]
2   ON [subscriptions_with_text]
3   INSTEAD OF UPDATE
4   AS
5      IF EXISTS(SELECT 1
6                FROM [inserted]
7                WHERE (    UPDATE([sb_subscriber])
8                       AND PATINDEX('%[^0-9]%', [sb_subscriber]) > 0)
9                   OR (    UPDATE([sb_book])
10                      AND PATINDEX('%[^0-9]%', [sb_book]) > 0))
11        BEGIN
12         RAISERROR ('Use digital identifiers for [sb_subscriber]
13                     and [sb_book]. Do not use subscribers'' names
14                     or books'' titles', 16, 1);
15         ROLLBACK;
16        END
17        ELSE
18         BEGIN
19          IF UPDATE([sb_id])
20           BEGIN
21            RAISERROR ('UPDATE of Primary Key through
22                        [subscriptions_with_text]
23                        view is prohibited.', 16, 1);
24            ROLLBACK;
25           END
26           ELSE
27            BEGIN
28             UPDATE [subscriptions]
29             SET    [subscriptions].[sb_subscriber] =
30                    CASE
31                     WHEN (PATINDEX('%[^0-9]%',
32                           [inserted].[sb_subscriber]) = 0)
33                     THEN [inserted].[sb_subscriber]
34                     ELSE [subscriptions].[sb_subscriber]
35                    END,
36                    [subscriptions].[sb_book] =
37                    CASE
```

MS SQL Решение 3.2.2.a (создание триггера для реализации операции обновления) (продолжение)

```
                    WHEN (PATINDEX('%[^0-9]%',
                         [inserted].[sb_book]) = 0)
                    THEN [inserted].[sb_book]
                    ELSE [subscriptions].[sb_book]
                  END,
                [subscriptions].[sb_start] = [inserted].[sb_start],
                [subscriptions].[sb_finish] = [inserted].[sb_finish],
                  [subscriptions].[sb_is_active] =
                                    [inserted].[sb_is_active]
            FROM   [subscriptions]
              JOIN [inserted]
                ON [subscriptions].[sb_id] = [inserted].[sb_id];
          END
        END
GO
```

Проверим, как работают следующие запросы на обновление данных. Запросы 1-3 выполнятся успешно, а запросы 4-6 — нет, т.к. в запросах 4-5 происходит попытка передать нечисловые значения имени читателя и названия книги, а в запросе 6 происходит попытка обновить значение первичного ключа.

MS SQL Решение 3.2.2.a (проверка работоспособности операции обновления)

```
-- Запрос 1 (обновление выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_start] = '2021-01-12'
WHERE  [sb_id] = 5000;

-- Запрос 2 (обновление выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_subscriber] = 3
WHERE  [sb_id] = 5000;

-- Запрос 3 (обновление выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_book] = 4
WHERE  [sb_id] = 5000;

-- Запрос 4 (обновление НЕ выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_subscriber] = N'Читатель'
WHERE  [sb_id] = 5000;

-- Запрос 5 (обновление НЕ выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_book] = N'Книга'
WHERE  [sb_id] = 5000;

-- Запрос 6 (обновление НЕ выполняется):
UPDATE [subscriptions_with_text]
SET    [sb_id] = 5001
WHERE  [sb_id] = 5000;
```

Создадим триггер, позволяющий реализовать операцию удаления данных. Обратите внимание, насколько его код короче и проще только что рассмотренных триггеров, реализующих операции вставки и обновления данных. Но, к сожалению, проблем с этим триггером будет намного больше.

MS SQL Решение 3.2.2.a (создание триггера для реализации операции удаления)

```
CREATE TRIGGER [subscriptions_with_text_del]
ON [subscriptions_with_text]
INSTEAD OF DELETE
AS
 DELETE FROM [subscriptions]
 WHERE [sb_id] IN (SELECT [sb_id]
                   FROM   [deleted]);
GO
```

Первая проблема состоит в том, что MS SQL Server наполняет таблицу **deleted** данными **до того**, как передаёт управление триггеру. Поэтому мы никак не можем перехватить ситуацию передачи в **DELETE**-запрос строго числовых данных в полях **sb_subscriber** и **sb_book**, а такая ситуация приводит к ошибке выполнения запроса с резолюцией: невозможно преобразовать {текстовое значение} к {числовому значению}.

Вторая проблема состоит в том, что передача имени читателя или названия книги в виде числа (идентификатора), представленного строкой, приводит к нулевому количеству найденных совпадений (что вполне логично, т.к. представление извлекает не идентификаторы, а имена читателей и названия книг). Передача же полноценных имён читателей и/или названий книг позволяет обнаружить совпадения, но не гарантирует, что мы нашли нужные строки (напомним: у нас могут быть одноимённые читатели и книги с одинаковыми названиями).

Как уже было упомянуто, с первой проблемой мы ничего не можем сделать, вторая же имеет не самое надёжное и не самое красивое, но всё же работающее решение: мы можем получить внутри триггера код запроса, выполнение которого активировало триггер, и проверить, упомянуты ли в этом запросе поля **sb_subscriber** и/или **sb_book**. Если они упомянуты, мы отменим транзакцию.

MS SQL Решение 3.2.2.a (создание триггера для реализации операции удаления; более сложный вариант)

```
CREATE TRIGGER [subscriptions_with_text_del]
ON [subscriptions_with_text]
INSTEAD OF DELETE
AS
 -- Попытка определить, переданы ли в DELETE-запрос
 -- поля sb_subscriber и/или sb_book:
 SET NOCOUNT ON;
 DECLARE @ExecStr VARCHAR(50), @Qry NVARCHAR(255);
 CREATE TABLE #inputbuffer
 (
  [EventType] NVARCHAR(30),
  [Parameters] INT,
  [EventInfo] NVARCHAR(255)
 );

 SET @ExecStr = 'DBCC INPUTBUFFER(' + STR(@@SPID) + ')';
 INSERT INTO #inputbuffer EXEC (@ExecStr);
 SET @Qry = LOWER((SELECT [EventInfo] FROM #inputbuffer));

```

MS SQL Решение 3.2.2.a (создание триггера для реализации операции удаления; более сложный вариант) (продолжение)

```
  -- Для отладки можно раскомментировать следующую строку
  -- и убедиться, что в ней расположен запрос, вызвавший
  -- срабатывание триггера:
  -- PRINT(@Qry);

  IF    ((CHARINDEX ('sb_subscriber', @Qry) > 0)
     OR  (CHARINDEX ('sb_book', @Qry) > 0))
         BEGIN
           RAISERROR ('Deletion from [subscriptions_with_text] view
                      using [sb_subscriber] and/or [sb_book]
                      is prohibited.', 16, 1);
           ROLLBACK;
         END
  SET NOCOUNT OFF;

  -- Здесь выполняется само удаление:
  DELETE FROM [subscriptions]
  WHERE [sb_id] IN (SELECT [sb_id]
                    FROM   [deleted]);
GO
```

Проверим, как работают запросы на удаление. В комментариях к каждому из запросов дано пояснение, в каких случаях он будет выполняться успешно или завершится той или иной ошибкой. Результат будет зависеть от того, какую из двух представленных выше версий триггера вы реализуете.

MS SQL Решение 3.2.2.a (проверка работоспособности операции удаления)

```
-- Запрос 1 (удаление работает):
DELETE FROM [subscriptions_with_text]
WHERE  [sb_id] < 10;

-- Запрос 2 (удаление работает):
DELETE FROM [subscriptions_with_text]
WHERE  [sb_start] = '2012-01-12';

-- Запрос 3 (удаление НЕ работает
-- при обеих версиях триггера):
DELETE FROM [subscriptions_with_text]
WHERE  [sb_book] = 2;

-- Запрос 4 (нулевое количество совпадений
-- в первой версии триггера и отмена транзакции
-- во второй версии триггера):
DELETE FROM [subscriptions_with_text]
WHERE  [sb_book] = '2';

-- Запрос 5 (возможно обнаружение совпадений
-- в первой версии триггера; во второй версии
-- триггера всегда будет отмена транзакции):
DELETE FROM [subscriptions_with_text]
WHERE  [sb_book] = N'Евгений Онегин';
```

Переходим к решению данной задачи для Oracle. Код самого представления — такой же, как для MySQL и MS SQL Server:

Oracle Решение 3.2.2.a (создание представления)

```
1   CREATE VIEW "subscriptions_with_text"
2   AS
3   SELECT "sb_id",
4          "s_name" AS "sb_subscriber",
5          "b_name" AS "sb_book",
6          "sb_start",
7          "sb_finish",
8          "sb_is_active"
9   FROM   "subscriptions"
10         JOIN "subscribers" ON "sb_subscriber" = "s_id"
11         JOIN "books" ON "sb_book" = "b_id"
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

Логика проверки (строки 5-13) схожа в MS SQL Server и Oracle: исходная таблица **subscriptions** содержит в полях **sb_subscriber** и **sb_book** числовые идентификаторы читателя и книги, потому мы обязаны получать вставляемые значения этих полей в числовом виде, следовательно, мы запрещаем операцию вставки нечисловых данных.

Алгоритмически мы выполняем одни и те же действия в обеих СУБД, а разница состоит лишь в функциях по работе с регулярными выражениями и генерации сообщения об ошибке.

Основная часть триггера, отвечающая непосредственно за вставку данных, в Oracle снова получилась чуть более простой, чем в MS SQL Server в силу возможности обрабатывать каждый ряд отдельно.

Механизм управления автоинкрементируемыми первичными ключами (построенный на последовательности (**SEQUENCE**) и отдельном триггере) позволяет нам не заботиться о том, как получить новое значение первичного ключа.

Oracle Решение 3.2.2.a (создание триггера для реализации операции вставки)

```
1   CREATE OR REPLACE TRIGGER "subscriptions_with_text_ins"
2     INSTEAD OF INSERT ON "subscriptions_with_text"
3     FOR EACH ROW
4      BEGIN
5       IF   ((REGEXP_INSTR(:new."sb_subscriber", '[^0-9]') > 0)
6         OR  (REGEXP_INSTR(:new."sb_book", '[^0-9]') > 0))
7       THEN
8        RAISE_APPLICATION_ERROR(-20001, 'Use digital identifiers for
9                                        "sb_subscriber" and "sb_book".
10                                       Do not use subscribers'' names
11                                       or books'' titles');
12       ROLLBACK;
13      END IF;
14      INSERT INTO "subscriptions"
15                  ("sb_id",
16                   "sb_subscriber",
17                   "sb_book",
18                   "sb_start",
19                   "sb_finish",
20                   "sb_is_active")
21      VALUES      (:new."sb_id",
22                   :new."sb_subscriber",
23                   :new."sb_book",
24                   :new."sb_start",
25                   :new."sb_finish",
26                   :new."sb_is_active");
27     END;
```

Проверим, как выполняются следующие запросы на вставку. Вполне ожидаемо запросы 1 и 2 выполняются корректно, а запросы 3-5 приводят к срабатыванию кода в строках 5-14 триггера и отмене транзакции, т.к. передача нечисловых значений для полей **sb_subscriber** и/или **sb_book** запрещена.

Oracle Решение 3.2.2.a (проверка работоспособности операции вставки)

```
1  -- Запрос 1 (вставка выполняется):
2  INSERT INTO "subscriptions_with_text"
3              ("sb_id",
4               "sb_subscriber",
5               "sb_book",
6               "sb_start",
7               "sb_finish",
8               "sb_is_active")
9  VALUES      (5000,
10              1,
11              3,
12              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
13              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
14              'N');
15
16 -- Запрос 2 (вставка выполняется):
17 INSERT INTO "subscriptions_with_text"
18             ("sb_subscriber",
19              "sb_book",
20              "sb_start",
21              "sb_finish",
22              "sb_is_active")
23 VALUES      (1,
24              3,
25              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
26              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
27              'N');
28 -- Запрос 3 (вставка НЕ выполняется):
29 INSERT INTO "subscriptions_with_text"
30             ("sb_subscriber",
31              "sb_book",
32              "sb_start",
33              "sb_finish",
34              "sb_is_active")
35 VALUES      (N'Иванов И.И.',
36              3,
37              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
38              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
39              'N');
40
```

Oracle Решение 3.2.2.a (проверка работоспособности операции вставки) (продолжение)

```
41 -- Запрос 4 (вставка НЕ выполняется):
42 INSERT INTO "subscriptions_with_text"
43             ("sb_subscriber",
44              "sb_book",
45              "sb_start",
46              "sb_finish",
47              "sb_is_active")
48 VALUES      (1,
49              N'Какая-то книга',
50              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
51              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
52              'N');
53
54 -- Запрос 5 (вставка НЕ выполняется):
55 INSERT INTO "subscriptions_with_text"
56             ("sb_subscriber",
57              "sb_book",
58              "sb_start",
59              "sb_finish",
60              "sb_is_active")
61 VALUES      (N'Какой-то читатель',
62              N'Какая-то книга',
63              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
64              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
65              'N');
```

Создадим триггер, позволяющий реализовать операцию обновления данных.

Как и в решении для MS SQL Server, мы должны реагировать на нечисловые значения в полях **sb_subscriber** и **sb_book** только в том случае, если эти значения были явно переданы в запросе, — этим вызвана необходимость более сложного (чем в **INSERT**-триггере) условия в строках 5-8.

В строках 19-30 мы вновь проверяем, пришло ли в наборе новых данных числовое значение полей **sb_subscriber** и **sb_book**, и используем имеющееся в таблице **subscriptions** старое значение, если новое является не числом.

В строках 23 и 29 применён «трюк» с добавлением (конкатенацией) к исходному значению поля пустой строки в юникод-представлении, что приводит к автоматическому преобразованию типа данных к юникод-строке. Это позволяет избежать ошибки компиляции триггера, вызванной несовпадением типов данных полей **sb_subscriber** и **sb_book** в таблице **subscriptions** (**NUMBER**) и представлении **subscriptions_with_text** (**NVARCHAR2**). При этом получившееся строковое представление числа безошибочно автоматически преобразуется к типу **NUMBER** и корректно используется для вставки в таблицу **subscriptions**.

В отличие от MS SQL Server, где триггеру передаётся весь набор обрабатываемых данных, в Oracle наш триггер обрабатывает отдельно каждый обновляемый ряд, и потому мы точно знаем старое и новое значение первичного ключа. Это позволяет избавиться от запрета на обновление значения первичного ключа.

Oracle Решение 3.2.2.a (создание триггера для реализации операции обновления)

```
CREATE OR REPLACE TRIGGER "subscriptions_with_text_upd"
  INSTEAD OF UPDATE ON "subscriptions_with_text"
  FOR EACH ROW
   BEGIN
    IF      ((:old."sb_subscriber" != :new."sb_subscriber")
        AND  (REGEXP_INSTR(:new."sb_subscriber", '[^0-9]') > 0))
        OR  ((:old."sb_book" != :new."sb_book")
        AND  (REGEXP_INSTR(:new."sb_book", '[^0-9]') > 0))
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'Use digital identifiers for
                                     "sb_subscriber" and "sb_book".
                                     Do not use subscribers'' names
                                     or books'' titles');
     ROLLBACK;
    END IF;

    UPDATE "subscriptions"
    SET    "sb_id" = :new."sb_id",
           "sb_subscriber" =
                  CASE
                   WHEN (REGEXP_INSTR(:new."sb_subscriber", '[^0-9]') = 0)
                   THEN :new."sb_subscriber"
                   ELSE "sb_subscriber" || N''
                  END,
           "sb_book" =
                  CASE
                   WHEN (REGEXP_INSTR(:new."sb_book", '[^0-9]') = 0)
                   THEN :new."sb_book"
                   ELSE "sb_book" || N''
                  END,
           "sb_start" = :new."sb_start",
           "sb_finish" = :new."sb_finish",
           "sb_is_active" = :new."sb_is_active"
    WHERE  "sb_id" = :old."sb_id";
  END;
```

Проверим, как работают следующие запросы на обновление данных. Запросы 1-3 и 6 выполнятся успешно (запрос 6 в MS SQL Server не выполняется из-за обновления значения первичного ключа), а запросы 4-5 — нет, т.к. в них происходит попытка передать нечисловые значения имени читателя и названия книги.

Oracle Решение 3.2.2.a (проверка работоспособности операции обновления)

```
-- Запрос 1 (обновление выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_start" = TO_DATE('2021-01-12', 'YYYY-MM-DD')
WHERE  "sb_id" = 101;

-- Запрос 2 (обновление выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_subscriber" = 3
WHERE  "sb_id" = 101;

```

Oracle Решение 3.2.2.a (проверка работоспособности операции обновления) (продолжение)

```
-- Запрос 3 (обновление выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_book" = 4
WHERE  "sb_id" = 101;
-- Запрос 4 (обновление НЕ выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_subscriber" = N'Читатель'
WHERE  "sb_id" = 101;
-- Запрос 5 (обновление НЕ выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_book" = N'Книга'
WHERE  "sb_id" = 101;

-- Запрос 6 (обновление выполняется):
UPDATE "subscriptions_with_text"
SET    "sb_id" = 1001
WHERE  "sb_id" = 101;

```

Создадим триггер, позволяющий реализовать операцию удаления данных. Его код отличается от кода аналогичной триггера для MS SQL Server только логикой определения идентификаторов удаляемых записей.

Oracle Решение 3.2.2.a (создание триггера для реализации операции удаления)

```
CREATE OR REPLACE TRIGGER "subscriptions_with_text_del"
  INSTEAD OF DELETE ON "subscriptions_with_text"
  FOR EACH ROW
   BEGIN
    DELETE FROM "subscriptions"
    WHERE "sb_id" = :old."sb_id";
   END;
```

Oracle (как и MS SQL Server) выполняет операцию поиска соответствующих условию удаления записей **до того**, как передаёт управление триггеру. Поэтому мы никак не можем перехватить ситуацию передачи в **DELETE**-запрос строго числовых данных в полях **sb_subscriber** и **sb_book**, а такая ситуация приводит к ошибке выполнения запроса с резолюцией «некорректное числовое значение».

Вторая проблема (как и в случае с MS SQL Server) состоит в том, что передача имени читателя или названия книги в виде числа (идентификатора), представленного строкой, приводит к нулевому количеству найденных совпадений, а передача полноценных имён читателей и/или названий книг позволяет обнаружить совпадения, но не гарантирует, что мы нашли нужные строки (напомним: у нас могут быть одноимённые читатели и книги с одинаковыми названиями).

И если MS SQL Server позволяет решить вторую проблему через анализ SQL-запроса, активировавшего триггер, то в Oracle не существует способа получить текст SQL запроса в триггерах, реагирующих на выражения модификации данных (DML-триггерах). Таким образом, **DELETE**-триггер в решении данной задачи для Oracle скорее вреден и опасен, чем полезен — он приводит к появлению неожиданных сообщений об ошибках и позволяет случайно удалить лишние данные.

И всё же проверим, как работают запросы на удаление.

Oracle Решение 3.2.2.a (проверка работоспособности операции удаления) (продолжение)

```
-- Запрос 1 (удаление работает):
DELETE FROM "subscriptions_with_text"
WHERE  "sb_id" = 103;

-- Запрос 2 (удаление работает):
DELETE FROM "subscriptions_with_text"
WHERE  "sb_start" = TO_DATE('2011-01-12', 'YYYY-MM-DD');

-- Запрос 3 (удаление НЕ работает:
-- ошибка преобразования типов):
DELETE FROM "subscriptions_with_text"
WHERE  "sb_book" = 2;
-- Запрос 4 (нулевое количество совпадений):
DELETE FROM "subscriptions_with_text"
WHERE  "sb_book" = '2';

-- Запрос 5 (возможно удаление одноимённых, но
-- при этом разных книг):
DELETE FROM "subscriptions_with_text"
WHERE  "sb_book" = N'Евгений Онегин';
```

Решение 3.2.2.b{267}.

Логика выборки данных в этом представлении является упрощённым вариантом решения{77} задачи 2.2.2.b{73}, а сами триггеры в сравнении с предыдущим примером — в разы более простыми.

Т.к. MySQL не позволяет создавать триггеры на представлениях, максимум, что мы можем сделать, это создать само представление, но данные через него модифицировать не получится:

MySQL Решение 3.2.2.b (создание представления)

```
CREATE VIEW `books_with_genres`
AS
SELECT `b_id`,
       `b_name`,
       GROUP_CONCAT(`g_name`) AS `genres`
FROM   `books`
       JOIN `m2m_books_genres` USING(`b_id`)
       JOIN `genres` USING(`g_id`)
GROUP BY `b_id`
```

В MS SQL Server код самого представления выглядит следующим образом (см. пояснения относительно логики получения нужного результата в решении{77} задачи 2.2.2.b{73}):

MS SQL Решение 3.2.2.b (создание представления)

```
CREATE VIEW [books_with_genres]
AS
WITH [prepared_data]
     AS (SELECT [books].[b_id],
                [b_name],
                       [g_name]
         FROM   [books]
                JOIN [m2m_books_genres]
                  ON [books].[b_id] = [m2m_books_genres].[b_id]
                JOIN [genres]
                  ON [m2m_books_genres].[g_id] = [genres].[g_id]
        )
SELECT [outer].[b_id],
       [outer].[b_name],
       STUFF ((SELECT DISTINCT ',' + [inner].[g_name]
               FROM   [prepared_data] AS [inner]
               WHERE  [outer].[b_id] = [inner].[b_id]
               ORDER  BY ',' + [inner].[g_name]
               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
               1, 1, '')
       AS [genres]
FROM   [prepared_data] AS [outer]
GROUP  BY [outer].[b_id],
          [outer].[b_name]
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

MS SQL Решение 3.2.2.b (создание триггера для реализации операции вставки)

```
CREATE TRIGGER [books_with_genres_ins]
ON [books_with_genres]
INSTEAD OF INSERT
AS
 INSERT INTO [genres]
             ([g_name])
 SELECT [genres]
 FROM   [inserted];
GO
```

Да, это — всё. Через такое представление не удастся передать идентификатор жанра (в представлении нет соответствующего поля), равно как по той же причине не удастся реализовать добавление новых книг. А добавление нового жанра действительно реализуется настолько примитивно.

Переходим к решению для Oracle. Создадим представление (см. пояснения относительно логики получения нужного результата в решении{77} задачи 2.2.2.b{73}):

Oracle Решение 3.2.2.b (создание представления)

```
CREATE VIEW "books_with_genres"
AS
SELECT "b_id", "b_name",
       UTL_RAW.CAST_TO_NVARCHAR2
       (
          LISTAGG
          (
              UTL_RAW.CAST_TO_RAW("g_name"),
              UTL_RAW.CAST_TO_RAW(N',')
          )
          WITHIN GROUP (ORDER BY "g_name")
       )
       AS "genres"
       FROM "books"
       JOIN "m2m_books_genres" USING ("b_id")
       JOIN "genres" USING ("g_id")
       GROUP  BY "b_id",
                 "b_name"
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

Oracle Решение 3.2.2.b (создание триггера для реализации операции вставки)

```
CREATE OR REPLACE TRIGGER "books_with_genres_ins"
INSTEAD OF INSERT ON "books_with_genres"
FOR EACH ROW
  BEGIN
    INSERT INTO "genres"
                ("g_name")
    VALUES      (:new."genres");
  END;
```

Как видно из кода триггера, решение для Oracle получилось столь же примитивным в силу причин, описанных выше в решении для MS SQL Server.

Задание 3.2.2.TSK.A: создать представление, извлекающее из таблицы **m2m_books_authors** человекочитаемую (с названиями книг и именами авторов вместо идентификаторов) информацию, и при этом позволяющее модифицировать данные в таблице **m2m_books_authors** (в случае неуникальности названий книг и имён авторов в обоих случаях использовать запись с минимальным значением первичного ключа).

Задание 3.2.2.TSK.B: создать представление, показывающее список книг с их авторами, и при этом позволяющее добавлять новых авторов.

ИСПОЛЬЗОВАНИЕ ТРИГГЕРОВ

АГРЕГАЦИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТРИГГЕРОВ

4.1.1. ПРИМЕР 31: ОБНОВЛЕНИЕ КЭШИРУЮЩИХ ТАБЛИЦ И ПОЛЕЙ

Дополнительным материалом к данному примеру служат решения задач, представленных в примерах 27{222}, 29{254} и 30{267}.

Задача 4.1.1.a{285}: модифицировать схему базы данных «Библиотека» таким образом, чтобы таблица `subscribers` хранила актуальную информацию о дате последнего визита читателя в библиотеку.

Задача 4.1.1.b{294}: создать кэширующую таблицу `averages`, содержащую в любой момент времени следующую актуальную информацию:
а) сколько в среднем книг находится на руках у читателя;
б) за сколько в среднем по времени (в днях) читатель прочитывает книгу;
в) сколько в среднем книг прочитал читатель.

Ожидаемый результат 4.1.1.a.

Таблица `subscribers` содержит дополнительное поле, хранящее актуальную информацию о дате последнего визита читателя в библиотеку:

s_id	s_name	s_last_visit
1	Иванов И.И.	2015-10-07
2	Петров П.П.	NULL
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Ожидаемый результат 4.1.1.b.

Таблица **averages** содержит следующую актуальную в любой момент времени информацию:

books_taken	days_to_read	books_returned
1.2500	46.0000	1.5000

Решение 4.1.1.a{284}.

Для решения этой задачи нужно будет выполнить три шага:

- модифицировать таблицу **subscribers** (добавив туда поле для хранения даты последнего визита читателя);
- проинициализировать значения последних визитов для всех читателей;
- создать триггеры для поддержания этой информации в актуальном состоянии.

Важно! В MySQL триггеры не активируются каскадными операциями, потому изменения в таблице **subscriptions**, вызванные удалением книг, останутся «незаметными» для триггеров на этой таблице. В задании 4.1.1.TSK.D{305} вам предлагается доработать данное решение, устранив эту проблему.

Для MySQL первые два шага выполняются с помощью следующих запросов.

MySQL Решение 4.1.1.a (модификация таблицы и инициализация данных)

```
1   -- Модификация таблицы:
2   ALTER TABLE `subscribers`
3     ADD COLUMN `s_last_visit` DATE NULL DEFAULT NULL AFTER `s_name`;
4
5   -- Инициализация данных:
6   UPDATE `subscribers`
7          LEFT JOIN (SELECT `sb_subscriber`,
8                            MAX(`sb_start`) AS `last_visit`
9                     FROM   `subscriptions`
10                    GROUP  BY `sb_subscriber`) AS `prepared_data`
11                ON `s_id` = `sb_subscriber`
12  SET    `s_last_visit` = `last_visit`;
```

Поскольку значение даты последнего визита читателя зависит от информации, представленной в таблице **subscriptions** (конкретно — от значения поля **sb_start**), именно на этой таблице нам и придётся создавать триггеры.

В принципе, во всех трёх триггерах можно было использовать однотипное решение (**UPDATE** на основе **JOIN**), но для разнообразия в **INSERT**-триггере мы реализуем более простой вариант: проверим, оказалась ли дата добавляемой выдачи больше, чем сохранённая дата последнего визита и, если это так, обновим дату последнего визита.

В **UPDATE**- и **DELETE**-триггерах такое решение не годится: если в результате этих операций окажется, что читатель ни разу не был в библиотеке, значением даты его последнего визита должен стать **NULL**. Такой результат проще всего достигается с помощью **UPDATE** на основе **JOIN**.

Для операций **UPDATE** и **DELETE** критически важно использовать именно **AFTER**-триггеры, т.к. **BEFORE**-триггеры будут работать со «старыми» данными, что приведёт к некорректному определению искомой даты.

Также обратите внимание, что в **UPDATE**-триггере мы должны обновлять дату последнего визита для двух потенциально разных читателей, т.к. при внесении изменения выдача может быть «передана» от одного читателя к другому (мы рассмотрим эту ситуацию в процессе проверки работоспособности полученного решения).

MySQL Решение 4.1.1.a (создание триггеров)

```
-- Удаление старых версий триггеров
-- (удобно в процессе разработки и отладки):
DROP TRIGGER `last_visit_on_subscriptions_ins`;
DROP TRIGGER `last_visit_on_subscriptions_upd`;
DROP TRIGGER `last_visit_on_subscriptions_del`;

-- Переключение разделителя завершения запроса,
-- т.к. сейчас запросом будет создание триггера,
-- внутри которого есть свои, классические запросы:
DELIMITER $$
-- Создание триггера, реагирующего на добавление выдачи книг:
CREATE TRIGGER `last_visit_on_subscriptions_ins`
AFTER INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   IF (SELECT IFNULL(`s_last_visit`, '1970-01-01')
       FROM   `subscribers`
       WHERE  `s_id` = NEW.`sb_subscriber`) < NEW.`sb_start`
   THEN
    UPDATE `subscribers`
    SET    `s_last_visit` = NEW.`sb_start`
    WHERE  `s_id` = NEW.`sb_subscriber`;
   END IF;
  END;
$$

-- Создание триггера, реагирующего на обновление выдачи книг:
CREATE TRIGGER `last_visit_on_subscriptions_upd`
AFTER UPDATE
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   UPDATE `subscribers`
   LEFT JOIN (SELECT `sb_subscriber`,
                     MAX(`sb_start`) AS `last_visit`
                FROM   `subscriptions`
                GROUP  BY `sb_subscriber`) AS `prepared_data`
             ON `s_id` = `sb_subscriber`
   SET    `s_last_visit` = `last_visit`
   WHERE  `s_id` IN (OLD.`sb_subscriber`, NEW.`sb_subscriber`);
  END;
$$

```

MySQL Решение 4.1.1.a (создание триггеров) (продолжение)

```
45 -- Создание триггера, реагирующего на удаление выдачи книг:
46 CREATE TRIGGER `last_visit_on_subscriptions_del`
47 AFTER DELETE
48 ON `subscriptions`
49  FOR EACH ROW
50   BEGIN
51    UPDATE `subscribers`
52    LEFT JOIN (SELECT `sb_subscriber`,
53                      MAX(`sb_start`) AS `last_visit`
54                FROM   `subscriptions`
55                GROUP  BY `sb_subscriber`) AS `prepared_data`
56             ON `s_id` = `sb_subscriber`
57    SET    `s_last_visit` = `last_visit`
58    WHERE  `s_id` = OLD.`sb_subscriber`;
59   END;
60 $$
61
62 -- Восстановление разделителя завершения запросов:
63 DELIMITER ;
64
```

Проверим работоспособность полученного решения. Будем изменять данные в таблице **subscriptions** и отслеживать изменения данных в таблице **subscribers**.

Для начала добавим выдачу книги читателю с идентификатором 2 (ранее он никогда не был в библиотеке):

MySQL Решение 4.1.1.a (проверка работоспособности)

```
1  INSERT INTO `subscriptions`
2  VALUES      (200,
3               2,
4               1,
5               '2019-01-12',
6               '2019-02-12',
7               'N')
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2015-10-07
2	Петров П.П.	2019-01-12
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Теперь эмулируем ситуацию «книга ошибочно записана на другого читателя» и изменим идентификатор читателя в только что добавленной выдаче с 2 на 1. **NULL**-значение даты последнего визита Петрова П.П. корректно восстановилось:

MySQL Решение 4.1.1.a (проверка работоспособности)

```
1  UPDATE `subscriptions`
2  SET    `sb_subscriber` = 1
3  WHERE  `sb_id` = 200
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2019-01-12
2	Петров П.П.	NULL
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Снова добавим выдачу книги Петрову П.П.:

MySQL Решение 4.1.1.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (201,
             2,
             1,
             '2020-01-12',
             '2020-02-12',
             'N')
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2019-01-12
2	Петров П.П.	2020-01-12
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Изменим значение даты ранее откорректированной выдачи книги (которую мы переписали с Петрова П.П. на Иванова И.И.):

MySQL Решение 4.1.1.a (проверка работоспособности)

```
UPDATE `subscriptions`
SET    `sb_start` = '2018-01-12'
WHERE  `sb_id` = 200
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2018-01-12
2	Петров П.П.	2020-01-12
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Удалим эту выдачу:

MySQL Решение 4.1.1.a (проверка работоспособности)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` = 200
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2015-10-07
2	Петров П.П.	2020-01-12
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Удалим единственную выдачу книги Петрову П.П.:

MySQL Решение 4.1.1.a (проверка работоспособности)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` = 201
```

s_id	s_name	s_last_visit
1	Иванов И.И.	2015-10-07
2	Петров П.П.	NULL
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Итак, решение для MySQL полностью готово и корректно работает. Но прежде, чем перейти к решению для MS SQL Server проведём небольшое исследование, наглядно демонстрирующее ответ на вопрос о том, «отменяется ли действие **BEFORE**-триггера в случае ошибки в процессе выполнения операции».

Исследование 4.4.1.EXP.A. Будут ли аннулированы изменения данных, вызванные работой **BEFORE**-триггера, если операция, активировавшая этот триггер, не сможет завершиться успешно?

Изменим вид **INSERT**-триггера с **AFTER** на **BEFORE**:

MySQL Исследование 4.1.1.a (изменение типа триггера)

```
DROP TRIGGER `last_visit_on_subscriptions_ins`;

DELIMITER $$

CREATE TRIGGER `last_visit_on_subscriptions_ins`
BEFORE INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   IF (SELECT IFNULL(`s_last_visit`, '1970-01-01')
       FROM   `subscribers`
       WHERE  `s_id` = NEW.`sb_subscriber`) < NEW.`sb_start`
   THEN
    UPDATE `subscribers`
    SET    `s_last_visit` = NEW.`sb_start`
    WHERE  `s_id` = NEW.`sb_subscriber`;
   END IF;
  END;
$$

DELIMITER ;
```

Выполним последовательно добавление двух выдач книг с разными датами, но одинаковыми значениями первичных ключей:

MySQL Исследование 4.1.1.a (вставка данных)

```
INSERT INTO `subscriptions`
VALUES      (500,
             2,
             1,
             '2020-01-12',
             '2020-02-12',
             'N');

INSERT INTO `subscriptions`
VALUES      (500,
             2,
             1,
             '2021-01-12',
             '2021-02-12',
             'N');
```

Проверим данные в таблице **subscribers**:

s_id	s_name	s_last_visit
1	Иванов И.И.	2015-10-07
2	Петров П.П.	2020-01-12
3	Сидоров С.С.	2014-08-03
4	Сидоров С.С.	2015-10-08

Итак, изменения аннулируются, если операция не может быть завершена успешно. В противном случае значением даты последнего визита Петрова П.П. было бы 2021-01-12.

Переходим к решению поставленной задачи для MS SQL Server. Модифицируем таблицу **subscribers** и проинициализируем добавленное поле данными.

MS SQL Решение 4.1.1.a (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE [subscribers]
  ADD [s_last_visit] DATE NULL DEFAULT NULL;

-- Инициализация данных:
UPDATE [subscribers]
SET    [s_last_visit] = [last_visit]
FROM   [subscribers]
       LEFT JOIN (SELECT [sb_subscriber],
                         MAX([sb_start]) AS [last_visit]
                  FROM   [subscriptions]
                  GROUP  BY [sb_subscriber]) AS [prepared_data]
              ON [s_id] = [sb_subscriber];
```

MS SQL Server поддерживает очень удобный синтаксис создания триггера сразу на нескольких операциях, потому (в отличие от решения для MySQL) мы используем одинаковый код тела триггера для всех трёх случаев (**INSERT**, **UPDATE**, **DELETE**):

MS SQL Решение 4.1.1.a (создание триггеров)

```
CREATE TRIGGER [last_visit_on_subscriptions_ins_upd_del]
ON [subscriptions]
AFTER INSERT, UPDATE, DELETE
AS
 UPDATE [subscribers]
 SET    [s_last_visit] = [last_visit]
 FROM   [subscribers]
        LEFT JOIN (SELECT [sb_subscriber],
                          MAX([sb_start]) AS [last_visit]
                   FROM   [subscriptions]
                   GROUP  BY [sb_subscriber]) AS [prepared_data]
               ON [s_id] = [sb_subscriber];
```

В корректности работы полученного решения вы можете убедиться, выполнив следующие запросы (они выполнены пошагово с пояснениями и показом результатов в решении для MySQL):

MS SQL Решение 4.1.1.a (запросы для проверки работоспособности)

```
SET IDENTITY_INSERT [subscriptions] ON;

-- Добавление выдачи книги читателю с идентификатором 2
-- (ранее он никогда не был в библиотеке):
INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (200,
             2,
             1,
             '2019-01-12',
             '2019-02-12',
             'N');

-- Изменение идентификатора читателя в только что
-- добавленной выдаче с 2 на 1:
UPDATE [subscriptions]
SET    [sb_subscriber] = 1
WHERE  [sb_id] = 200;

-- Ещё одна выдача книги Петрову П.П.
-- (идентификатор читателя = 2):
INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (201,
             2,
             1,
             '2020-01-12',
             '2020-02-12',
             'N');

-- Изменение значения даты ранее откорректированной
-- выдачи книги (которую переписали с Петрова П.П.
-- на Иванова И.И.):
UPDATE [subscriptions]
SET    [sb_start] = '2018-01-12'
WHERE  [sb_id] = 200;
```

MS SQL Решение 4.1.1.a (запросы для проверки работоспособности) (продолжение)

```
47 -- Удаление этой откорректированной выдачи:
48 DELETE FROM [subscriptions]
49 WHERE  [sb_id] = 200;
50
51 -- Удаление единственной выдачи Петрову П.П.:
52 DELETE FROM [subscriptions]
53 WHERE  [sb_id] = 201;
54
55 SET IDENTITY_INSERT [subscriptions] OFF;
```

Переходим к решению для Oracle, которое отличается от решения для MS SQL Server только синтаксическими особенностями реализации той же логики:

Oracle Решение 4.1.1.a (модификация таблицы и инициализация данных)

```
1  -- Модификация таблицы:
2  ALTER TABLE "subscribers"
3    ADD ("s_last_visit" DATE DEFAULT NULL NULL);
4
5  -- Инициализация данных:
6  UPDATE "subscribers" "outer"
7  SET    "s_last_visit" =
8         (
9          SELECT    "last_visit"
10         FROM      "subscribers"
11         LEFT JOIN (SELECT   "sb_subscriber",
12                            MAX("sb_start") AS "last_visit"
13                     FROM     "subscriptions"
14                     GROUP BY "sb_subscriber") "prepared_data"
15         ON         "s_id" = "sb_subscriber"
16         WHERE "outer"."s_id" = "sb_subscriber");
```

Oracle Решение 4.1.1.a (создание триггеров)

```
1  CREATE TRIGGER "last_visit_on_scs_ins_upd_del"
2  AFTER INSERT OR UPDATE OR DELETE
3  ON "subscriptions"
4  BEGIN
5   UPDATE "subscribers" "outer"
6   SET    "s_last_visit" =
7          (
8           SELECT    "last_visit"
9           FROM      "subscribers"
10          LEFT JOIN (SELECT   "sb_subscriber",
11                              MAX("sb_start") AS "last_visit"
12                     FROM     "subscriptions"
13                     GROUP BY "sb_subscriber") "prepared_data"
14          ON         "s_id" = "sb_subscriber"
15          WHERE "outer"."s_id" = "sb_subscriber");
16 END;
```

В данном решении мы использовали возможность Oracle создавать т.н. «триггеры уровня выражения» (statement level triggers), работающие аналогично триггерам MS SQL Server: такой триггер активируется после выполнения всей операции один раз, а не для каждого модифицируемого ряда отдельно, как это происходит, например, в MySQL, где поддерживаются только «триггеры уровня записи» (row level triggers), активирующиеся отдельно для каждого модифицируемого ряда.

В корректности работы полученного решения вы можете убедиться, выполнив следующие запросы (они выполнены пошагово с пояснениями и показом результатов в решении для MySQL):

Oracle Решение 4.1.1.a (запросы для проверки работоспособности)

```
ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE;

-- Добавление выдачи книги читателю с идентификатором 2
-- (ранее он никогда не был в библиотеке):
INSERT INTO "subscriptions"
            ("sb_id",
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
VALUES      (200,
             2,
             1,
             TO_DATE('2019-01-12', 'YYYY-MM-DD'),
             TO_DATE('2019-02-12', 'YYYY-MM-DD'),
             'N');

-- Изменение идентификатора читателя в только что
-- добавленной выдаче с 2 на 1:
UPDATE "subscriptions"
SET    "sb_subscriber" = 1
WHERE  "sb_id" = 200;

-- Ещё одна выдача книги Петрову П.П.
-- (идентификатор читателя = 2):
INSERT INTO "subscriptions"
            ("sb_id",
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
VALUES      (201,
             2,
             1,
             TO_DATE('2020-01-12', 'YYYY-MM-DD'),
             TO_DATE('2020-02-12', 'YYYY-MM-DD'),
             'N');

-- Изменение значения даты ранее откорректированной
-- выдачи книги (которую переписали с Петрова П.П.
-- на Иванова И.И.):
UPDATE "subscriptions"
SET    "sb_start" = TO_DATE('2018-01-12', 'YYYY-MM-DD')
WHERE  "sb_id" = 200;

```

Oracle Решение 4.1.1.a (запросы для проверки работоспособности) (продолжение)

```
48 -- Удаление этой откорректированной выдачи:
49 DELETE FROM "subscriptions"
50 WHERE  "sb_id" = 200;
51
52 -- Удаление единственной выдачи Петрову П.П.:
53 DELETE FROM "subscriptions"
54 WHERE  "sb_id" = 201;
55
56 ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE;
```

На этом решение данной задачи завершено.

Решение 4.1.1.b{284}.

Во многом решение данной задачи похоже на решение{223} задачи 3.1.2.a{223}, которое рекомендуется повторить перед тем, как продолжить чтение.

Как обычно, начнём с MySQL. Создадим агрегирующую таблицу:

MySQL Решение 4.1.1.b (создание агрегирующей таблицы)

```
1  CREATE TABLE `averages`
2  (
3        `books_taken` DOUBLE NOT NULL,
4        `days_to_read` DOUBLE NOT NULL,
5        `books_returned` DOUBLE NOT NULL
6  )
```

Проинициализируем данные в созданной таблице:

MySQL Решение 4.1.1.b (очистка таблицы и инициализация данных)

```
1  -- Очистка таблицы:
2  TRUNCATE TABLE `averages`;
3
4  -- Инициализация данных:
5  INSERT INTO `averages`
6              (`books_taken`,
7               `days_to_read`,
8               `books_returned`)
9  SELECT ( `active_count` / `subscribers_count` )   AS `books_taken`,
10       ( `days_sum` / `inactive_count` )          AS `days_to_read`,
11       ( `inactive_count` / `subscribers_count` ) AS `books_returned`
12 FROM   (SELECT COUNT(`s_id`) AS `subscribers_count`
13         FROM   `subscribers`) AS `tmp_subscribers_count`,
14        (SELECT COUNT(`sb_id`) AS `active_count`
15         FROM   `subscriptions`
16         WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
17        (SELECT COUNT(`sb_id`) AS `inactive_count`
18         FROM   `subscriptions`
19         WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
20        (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
21         FROM   `subscriptions`
22         WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`;
```

Напишем триггеры, модифицирующие данные в агрегирующей таблице. Агрегация происходит на основе информации, представленной в таблицах **subscribers** и **subscriptions**, потому придётся создавать триггеры для обеих этих таблиц.

Чтобы не усложнять решение, мы будем использовать один и тот же код для всех пяти триггеров (на таблице **subscribers** должны быть только **INSERT**- и **DELETE**-триггеры, т.к. обновление этой таблицы не влияет на результаты вычислений, а на таблице **subscriptions** должны быть все три триггера: **INSERT**, **UPDATE**, **DELETE**).

Важно! В MySQL триггеры не активируются каскадными операциями, потому изменения в таблице **subscriptions**, вызванные удалением книг, останутся «незаметными» для триггеров на этой таблице. В задании 4.1.1.TSK.E{305} вам предлагается доработать данное решение, устранив эту проблему.

MySQL Решение 4.1.1.b (триггеры для таблицы subscribers)

```
1   -- Удаление старых версий триггеров
2   -- (удобно в процессе разработки и отладки):
3   DROP TRIGGER `upd_avgs_on_subscribers_ins`;
4   DROP TRIGGER `upd_avgs_on_subscribers_del`;
5
6   -- Переключение разделителя завершения запроса,
7   -- т.к. сейчас запросом будет создание триггера,
8   -- внутри которого есть свои, классические запросы:
9   DELIMITER $$
10
11  -- Создание триггера, реагирующего на добавление читателей:
12  CREATE TRIGGER `upd_avgs_on_subscribers_ins`
13  AFTER INSERT
14  ON `subscribers`
15   FOR EACH ROW
16    BEGIN
17     UPDATE `averages`,
18            (SELECT COUNT(`s_id`) AS `subscribers_count`
19             FROM   `subscribers`) AS `tmp_subscribers_count`,
20            (SELECT COUNT(`sb_id`) AS `active_count`
21             FROM   `subscriptions`
22             WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
23            (SELECT COUNT(`sb_id`) AS `inactive_count`
24             FROM   `subscriptions`
25             WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
26            (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
27             FROM   `subscriptions`
28             WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`
29     SET    `books_taken` = `active_count` / `subscribers_count`,
30            `days_to_read` = `days_sum` / `inactive_count`,
31            `books_returned` = `inactive_count` / `subscribers_count`;
32    END;
33  $$
34
```

MySQL Решение 4.1.1.b (триггеры для таблицы subscribers) (продолжение)

```
35 -- Создание триггера, реагирующего на удаление читателей:
36 CREATE TRIGGER `upd_avgs_on_subscribers_del`
37 AFTER DELETE
38 ON `subscribers`
39  FOR EACH ROW
40   BEGIN
41    UPDATE `averages`,
42           (SELECT COUNT(`s_id`) AS `subscribers_count`
43            FROM   `subscribers`) AS `tmp_subscribers_count`,
44           (SELECT COUNT(`sb_id`) AS `active_count`
45            FROM   `subscriptions`
46            WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
47           (SELECT COUNT(`sb_id`) AS `inactive_count`
48            FROM   `subscriptions`
49            WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
50           (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
51            FROM   `subscriptions`
52            WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`
53    SET    `books_taken` = `active_count` / `subscribers_count`,
45           `days_to_read` = `days_sum` / `inactive_count`,
55           `books_returned` = `inactive_count` / `subscribers_count`;
56   END;
57 $$
58
59 -- Восстановление разделителя завершения запросов:
60 DELIMITER ;
```

Создадим триггеры на таблице **subscriptions**.

MySQL Решение 4.1.1.b (триггеры для таблицы subscriptions)

```
1  -- Удаление старых версий триггеров
2  -- (удобно в процессе разработки и отладки):
3  DROP TRIGGER `upd_avgs_on_subscriptions_ins`;
4  DROP TRIGGER `upd_avgs_on_subscriptions_upd`;
5  DROP TRIGGER `upd_avgs_on_subscriptions_del`;
6
7  -- Переключение разделителя завершения запроса,
8  -- т.к. сейчас запросом будет создание триггера,
9  -- внутри которого есть свои, классические запросы:
10 DELIMITER $$
11
12 -- Создание триггера, реагирующего на добавление выдачи книги:
13 CREATE TRIGGER `upd_avgs_on_subscriptions_ins`
14 AFTER INSERT
15 ON `subscriptions`
16  FOR EACH ROW
17   BEGIN
18    UPDATE `averages`,
19           (SELECT COUNT(`s_id`) AS `subscribers_count`
20            FROM   `subscribers`) AS `tmp_subscribers_count`,
21           (SELECT COUNT(`sb_id`) AS `active_count`
22            FROM   `subscriptions`
23            WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
24           (SELECT COUNT(`sb_id`) AS `inactive_count`
25            FROM   `subscriptions`
26            WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
```

MySQL Решение 4.1.1.b (триггеры для таблицы subscriptions) (продолжение)

```
27            (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
28             FROM   `subscriptions`
29             WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`
30     SET     `books_taken` = `active_count` / `subscribers_count`,
31             `days_to_read` = `days_sum` / `inactive_count`,
32             `books_returned` = `inactive_count` / `subscribers_count`;
33   END;
34 $$
35
36 -- Создание триггера, реагирующего на обновление выдачи книги:
37 CREATE TRIGGER `upd_avgs_on_subscriptions_upd`
38 AFTER UPDATE
39 ON `subscriptions`
40  FOR EACH ROW
41   BEGIN
42    UPDATE `averages`,
43            (SELECT COUNT(`s_id`) AS `subscribers_count`
44             FROM   `subscribers`) AS `tmp_subscribers_count`,
45            (SELECT COUNT(`sb_id`) AS `active_count`
46             FROM   `subscriptions`
47             WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
48            (SELECT COUNT(`sb_id`) AS `inactive_count`
49             FROM   `subscriptions`
50             WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
51            (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
52             FROM   `subscriptions`
53             WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`
45     SET     `books_taken` = `active_count` / `subscribers_count`,
55             `days_to_read` = `days_sum` / `inactive_count`,
56             `books_returned` = `inactive_count` / `subscribers_count`;
57   END;
58 $$
59 -- Создание триггера, реагирующего на удаление выдачи книги:
60 CREATE TRIGGER `upd_avgs_on_subscriptions_del`
61 AFTER DELETE
62 ON `subscriptions`
63  FOR EACH ROW
64   BEGIN
65    UPDATE `averages`,
66            (SELECT COUNT(`s_id`) AS `subscribers_count`
67             FROM   `subscribers`) AS `tmp_subscribers_count`,
68            (SELECT COUNT(`sb_id`) AS `active_count`
69             FROM   `subscriptions`
70             WHERE  `sb_is_active` = 'Y') AS `tmp_active_count`,
71            (SELECT COUNT(`sb_id`) AS `inactive_count`
72             FROM   `subscriptions`
73             WHERE  `sb_is_active` = 'N') AS `tmp_inactive_count`,
74            (SELECT SUM(DATEDIFF(`sb_finish`, `sb_start`)) AS `days_sum`
75             FROM   `subscriptions`
76             WHERE  `sb_is_active` = 'N') AS `tmp_days_sum`
77     SET     `books_taken` = `active_count` / `subscribers_count`,
78             `days_to_read` = `days_sum` / `inactive_count`,
79             `books_returned` = `inactive_count` / `subscribers_count`;
80   END;
81 $$
82
83 -- Восстановление разделителя завершения запросов:
84 DELIMITER ;
```

Проверим работоспособность полученного решения. Будем изменять данные в таблицах **subscribers** и **subscriptions** и отслеживать изменения данных в таблице **averages**.

Исходное состояние таблицы **averages** таково:

books_taken	days_to_read	books_returned
1.25	46	1.5

Добавим читателя:

MySQL Решение 4.1.1.b (проверка работоспособности)

```
INSERT INTO `subscribers`
            (`s_id`,
             `s_name`)
VALUES      (500,
             'Читателев Ч.Ч.')
```

books_taken	days_to_read	books_returned
1	46	1.2

Теперь удалим его:

MySQL Решение 4.1.1.b (проверка работоспособности)

```
DELETE FROM `subscribers`
WHERE  `s_id` = 500
```

books_taken	days_to_read	books_returned
1.25	46	1.5

Добавим две выдачи книги:

MySQL Решение 4.1.1.b (проверка работоспособности)

```
INSERT INTO `subscriptions`
            (`sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`)
VALUES      (200,
             1,
             1,
             '2019-01-12',
             '2019-02-12',
             'N'),
            (201,
             2,
             1,
             '2020-01-12',
             '2020-02-12',
             'N')
```

books_taken	days_to_read	books_returned
1.25	42.25	2

Изменим состояние добавленных выдач с «книга возвращена» на «книга не возвращена»:

MySQL Решение 4.1.1.b (проверка работоспособности)

```
UPDATE `subscriptions`
SET    `sb_is_active` = 'Y'
WHERE  `sb_id` >= 200
```

books_taken	days_to_read	books_returned
1.75	46	1.5

Удалим эти две выдачи книг:

MySQL Решение 4.1.1.b (проверка работоспособности)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` >= 200
```

books_taken	days_to_read	books_returned
1.25	46	1.5

Итак, триггеры для MySQL работают корректно. Переходим к решению для MS SQL Server.

MS SQL Решение 4.1.1.b (создание агрегирующей таблицы)

```
CREATE TABLE [averages]
(
      [books_taken] DOUBLE PRECISION NOT NULL,
      [days_to_read] DOUBLE PRECISION NOT NULL,
      [books_returned] DOUBLE PRECISION NOT NULL
)
```

Проинициализируем данные в созданной таблице. Обратите внимание: здесь снова актуальна проблема преобразования типов данных, т.к. результат деления окажется целочисленным (с потерей части данных), если предварительно не преобразовать полученные значения **COUNT** и **SUM** к дроби.

MS SQL Решение 4.1.1.b (очистка таблицы и инициализация данных)

```
-- Очистка таблицы:
TRUNCATE TABLE [averages];

-- Инициализация данных:
INSERT INTO [averages]
            ([books_taken],
             [days_to_read],
             [books_returned])
SELECT ( [active_count] / [subscribers_count] )   AS [books_taken],
       ( [days_sum] / [inactive_count] )          AS [days_to_read],
       ( [inactive_count] / [subscribers_count] ) AS [books_returned]
FROM   (SELECT CAST(COUNT([s_id]) AS DOUBLE PRECISION)
        AS [subscribers_count]
        FROM   [subscribers]) AS [tmp_subscribers_count],
       (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
        AS [active_count]
        FROM   [subscriptions]
        WHERE  [sb_is_active] = 'Y') AS [tmp_active_count],
       (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
        AS [inactive_count]
        FROM   [subscriptions]
        WHERE  [sb_is_active] = 'N') AS [tmp_inactive_count],
       (SELECT CAST(SUM(DATEDIFF(day, [sb_start], [sb_finish]))
        AS DOUBLE PRECISION) AS [days_sum]
        FROM   [subscriptions]
        WHERE  [sb_is_active] = 'N') AS [tmp_days_sum];
```

Создадим на таблицах **subscribers** и **subscriptions** триггеры, модифицирующие данные в агрегирующей таблице **averages**. Напомним, что MS SQL Server позволяет указывать в триггере сразу несколько активирующих событий, что позволяет нам немного сократить количество написанного кода.

MS SQL Решение 4.1.1.b (триггеры для таблицы subscribers)

```
CREATE TRIGGER [upd_avgs_on_subscribers_ins_del]
ON [subscribers]
AFTER INSERT, DELETE
AS
   UPDATE [averages]
   SET    [books_taken] = [active_count] / [subscribers_count],
          [days_to_read] = [days_sum] / [inactive_count],
          [books_returned] = [inactive_count] / [subscribers_count]
   FROM   (SELECT CAST(COUNT([s_id]) AS DOUBLE PRECISION)
           AS [subscribers_count]
           FROM   [subscribers]) AS [tmp_subscribers_count],
          (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
           AS [active_count]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'Y') AS [tmp_active_count],
          (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
           AS [inactive_count]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'N') AS [tmp_inactive_count],
          (SELECT CAST(SUM(DATEDIFF(day, [sb_start], [sb_finish]))
           AS DOUBLE PRECISION) AS [days_sum]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'N') AS [tmp_days_sum];
```

MS SQL Решение 4.1.1.b (триггеры для таблицы subscriptions)

```
CREATE TRIGGER [upd_avgs_on_subscriptions_ins_upd_del]
ON [subscriptions]
AFTER INSERT, UPDATE, DELETE
AS
   UPDATE [averages]
   SET    [books_taken] = [active_count] / [subscribers_count],
          [days_to_read] = [days_sum] / [inactive_count],
          [books_returned] = [inactive_count] / [subscribers_count]
   FROM   (SELECT CAST(COUNT([s_id]) AS DOUBLE PRECISION)
           AS [subscribers_count]
           FROM   [subscribers]) AS [tmp_subscribers_count],
          (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
           AS [active_count]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'Y') AS [tmp_active_count],
          (SELECT CAST(COUNT([sb_id]) AS DOUBLE PRECISION)
           AS [inactive_count]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'N') AS [tmp_inactive_count],
          (SELECT CAST(SUM(DATEDIFF(day, [sb_start], [sb_finish]))
           AS DOUBLE PRECISION) AS [days_sum]
           FROM   [subscriptions]
           WHERE  [sb_is_active] = 'N') AS [tmp_days_sum];
```

В корректности работы полученного решения вы можете убедиться, выполнив следующие запросы (они выполнены пошагово с пояснениями и показом результатов в решении для MySQL):

MS SQL Решение 4.1.1.b (запросы для проверки работоспособности)

```
SET IDENTITY_INSERT [subscribers] ON;

-- Добавление читателя:
INSERT INTO [subscribers]
            ([s_id],
             [s_name])
VALUES      (500,
             N'Читателев Ч.Ч.');

-- Удаление только что добавленного читателя:
DELETE FROM [subscribers]
WHERE  [s_id] = 500;

SET IDENTITY_INSERT [subscribers] OFF;
SET IDENTITY_INSERT [subscriptions] ON;

-- Добавление двух выдач книг:
INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (200,
             1,
             1,
             '2019-01-12',
             '2019-02-12',
             'N'),
            (201,
             2,
             1,
             '2020-01-12',
             '2020-02-12',
             'N');
```

MS SQL Решение 4.1.1.b (запросы для проверки работоспособности) (продолжение)

```
-- Изменение состояния добавленных выдач с «книга возвращена»
-- на «книга не возвращена»:
UPDATE [subscriptions]
SET    [sb_is_active] = 'Y'
WHERE  [sb_id] >= 200;

-- Удаление только что добавленных выдач книг:
DELETE FROM [subscriptions]
WHERE  [sb_id] >= 200;

SET IDENTITY_INSERT [subscriptions] OFF;
```

Переходим к решению для Oracle, которое отличается от решения для MS SQL Server только синтаксическими особенностями реализации той же логики:

Oracle Решение 4.1.1.b (создание агрегирующей таблицы)

```
CREATE TABLE "averages"
(
      "books_taken" DOUBLE PRECISION NOT NULL,
      "days_to_read" DOUBLE PRECISION NOT NULL,
      "books_returned" DOUBLE PRECISION NOT NULL
)
```

Oracle Решение 4.1.1.b (очистка таблицы и инициализация данных)

```
-- Очистка таблицы:
TRUNCATE TABLE "averages";

-- Инициализация данных:
INSERT INTO "averages"
            ("books_taken",
             "days_to_read",
             "books_returned")
SELECT ( "active_count" / "subscribers_count" )   AS "books_taken",
       ( "days_sum" / "inactive_count" )          AS "days_to_read",
       ( "inactive_count" / "subscribers_count" ) AS "books_returned"
FROM   (SELECT COUNT("s_id")  AS "subscribers_count"
        FROM   "subscribers") "tmp_subscribers_count",
       (SELECT COUNT("sb_id") AS "active_count"
        FROM   "subscriptions"
        WHERE  "sb_is_active" = 'Y') "tmp_active_count",
       (SELECT COUNT("sb_id") AS "inactive_count"
        FROM   "subscriptions"
        WHERE  "sb_is_active" = 'N') "tmp_inactive_count",
       (SELECT SUM("sb_finish" - "sb_start") AS "days_sum"
        FROM   "subscriptions"
        WHERE  "sb_is_active" = 'N') "tmp_days_sum";
```

Несмотря на то, что логика работы триггеров в Oracle полностью идентична подходам, использованным в MySQL и MS SQL Server, в силу синтаксических особенностей данной СУБД сам запрос на обновление данных выглядит несколько необычно. Здесь мы используем оператор `MERGE`, указав как условие объединения `1=1`, т.е. заведомо выполняющееся равенство.

Oracle Решение 4.1.1.b (триггеры для таблицы subscribers)

```
CREATE TRIGGER "upd_avgs_on_sbrs_ins_del"
AFTER INSERT OR DELETE
ON "subscribers"
BEGIN
 MERGE INTO "averages"
 USING
 (
  SELECT ( "active_count" / "subscribers_count" )   AS "books_taken",
         ( "days_sum" / "inactive_count" )          AS "days_to_read",
         ( "inactive_count" / "subscribers_count" ) AS "books_returned"
  FROM   (SELECT COUNT("s_id")  AS "subscribers_count"
          FROM   "subscribers") "tmp_subscribers_count",
         (SELECT COUNT("sb_id") AS "active_count"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'Y') "tmp_active_count",
         (SELECT COUNT("sb_id") AS "inactive_count"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'N') "tmp_inactive_count",
         (SELECT SUM("sb_finish" - "sb_start") AS "days_sum"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'N') "tmp_days_sum"
  ) "tmp" ON (1=1)
 WHEN MATCHED THEN UPDATE
  SET    "averages"."books_taken" = "tmp"."books_taken",
         "averages"."days_to_read" = "tmp"."days_to_read",
         "averages"."books_returned" = "tmp"."books_returned";
END;
```

Oracle Решение 4.1.1.b (триггеры для таблицы subscriptions)

```
CREATE TRIGGER "upd_avgs_on_sbps_ins_upd_del"
AFTER INSERT OR UPDATE OR DELETE
ON "subscriptions"
BEGIN
 MERGE INTO "averages"
 USING
 (
  SELECT ( "active_count" / "subscribers_count" )   AS "books_taken",
         ( "days_sum" / "inactive_count" )          AS "days_to_read",
         ( "inactive_count" / "subscribers_count" ) AS "books_returned"
  FROM   (SELECT COUNT("s_id")  AS "subscribers_count"
          FROM   "subscribers") "tmp_subscribers_count",
         (SELECT COUNT("sb_id") AS "active_count"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'Y') "tmp_active_count",
         (SELECT COUNT("sb_id") AS "inactive_count"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'N') "tmp_inactive_count",
         (SELECT SUM("sb_finish" - "sb_start") AS "days_sum"
          FROM   "subscriptions"
          WHERE  "sb_is_active" = 'N') "tmp_days_sum"
  ) "tmp" ON (1=1)
 WHEN MATCHED THEN UPDATE
  SET    "averages"."books_taken" = "tmp"."books_taken",
         "averages"."days_to_read" = "tmp"."days_to_read",
         "averages"."books_returned" = "tmp"."books_returned";
END;
```

Как и в решении{285} задачи 4.1.1.a{284}, здесь мы использовали возможность Oracle создавать т.н. «триггеры уровня выражения» (statement level triggers): такой триггер активируется после выполнения всей операции один раз, а не для каждого модифицируемого ряда отдельно, как это происходит, например, в MySQL, где поддерживаются только «триггеры уровня записи» (row level triggers), активирующиеся отдельно для каждого модифицируемого ряда.

В корректности работы полученного решения вы можете убедиться, выполнив следующие запросы (они выполнены пошагово с пояснениями и показом результатов в решении для MySQL):

Oracle Решение 4.1.1.b (запросы для проверки работоспособности)

```
ALTER TRIGGER "TRG_subscribers_s_id" DISABLE;
ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE;

-- Добавление читателя:
INSERT INTO "subscribers"
            ("s_id",
             "s_name")
VALUES      (500,
             N'Читателев Ч.Ч.');

-- Удаление только что добавленного читателя:
DELETE FROM "subscribers"
WHERE  "s_id" = 500;

-- Добавление двух выдач книг:
INSERT ALL
 INTO "subscriptions"
            ("sb_id",
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
 VALUES     (200,
             1,
             1,
             TO_DATE('2019-01-12', 'YYYY-MM-DD'),
             TO_DATE('2019-02-12', 'YYYY-MM-DD'),
             'N')
 INTO "subscriptions"
            ("sb_id",
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
 VALUES     (201,
             2,
             1,
             TO_DATE('2020-01-12', 'YYYY-MM-DD'),
             TO_DATE('2020-02-12', 'YYYY-MM-DD'),
             'N')
SELECT 1 FROM "DUAL";

```

Oracle Решение 4.1.1.b (запросы для проверки работоспособности) (продолжение)

```
45 -- Изменение состояния добавленных выдач с «книга возвращена»
46 -- на «книга не возвращена»:
47 UPDATE "subscriptions"
48 SET    "sb_is_active" = 'Y'
49 WHERE  "sb_id" >= 200;
50
51 -- Удаление только что добавленных выдач книг:
52 DELETE FROM "subscriptions"
53 WHERE  "sb_id" >= 200;
54
55 ALTER TRIGGER "TRG_subscribers_s_id" ENABLE;
56 ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE;
```

На этом решение данной задачи завершено.

Задание 4.1.1.TSK.A: модифицировать схему базы данных «Библиотека» таким образом, чтобы таблица **authors** хранила актуальную информацию о дате последней выдачи книги автора читателю.

Задание 4.1.1.TSK.B: создать кэширующую таблицу **best_averages**, содержащую в любой момент времени следующую актуальную информацию:
а) сколько в среднем книг находится на руках у читателей, за время работы с библиотекой прочитавших более 20 книг;
б) за сколько в среднем по времени (в днях) прочитывает книгу читатель, никогда не державший у себя книгу больше двух недель;
в) сколько в среднем книг прочитал читатель, не имеющий просроченных выдач книг.

Задание 4.1.1.TSK.C: оптимизировать MySQL-триггеры из решения{294} задачи 4.1.1.b{284} так, чтобы не выполнять лишних действий там, где в них нет необходимости (подсказка: не в каждом случае нам нужны все собираемые имеющимися запросами данные).

Задание 4.1.1.TSK.D: доработать решение{285} задачи 4.1.1.a{284} для MySQL таким образом, чтобы оно учитывало изменения в таблице **subscriptions**, вызванные операцией каскадного удаления (при удалении книг). Убедиться, что решения для MS SQL Server и Oracle не требуют такой доработки.

Задание 4.1.1.TSK.E: доработать решение{294} задачи 4.1.1.b{284} для MySQL таким образом, чтобы оно учитывало изменения в таблице **subscriptions**, вызванные операцией каскадного удаления (при удалении книг). Убедиться, что решения для MS SQL Server и Oracle не требуют такой доработки.

4.1.2. ПРИМЕР 32: ОБЕСПЕЧЕНИЕ КОНСИСТЕНТНОСТИ ДАННЫХ

Задача 4.1.2.a{306}: модифицировать схему базы данных «Библиотека» таким образом, чтобы таблица **subscribers** хранила информацию о том, сколько в настоящий момент книг выдано каждому из читателей.

Задача 4.1.2.b{320}: модифицировать схему базы данных «Библиотека» таким образом, чтобы таблица **genres** хранила информацию о том, сколько в настоящий момент книг относится к каждому жанру.

Ожидаемый результат 4.1.2.a.

Таблица `subscribers` содержит дополнительное поле, хранящее актуальную информацию о количестве выданных каждому читателю книг:

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Ожидаемый результат 4.1.2.b.

Таблица `genres` содержит дополнительное поле, хранящее актуальную информацию о количестве относящихся к каждому жанру книг:

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	0
5	Классика	4
6	Фантастика	1

Решение 4.1.2.a{305}.

Как и в решении{285} задачи 4.1.1.a{284} здесь нужно будет выполнить три шага:

- модифицировать таблицу `subscribers` (добавив туда поле для хранения количества выданных читателю книг);
- проинициализировать значения количества выданных книг для всех читателей;
- создать триггеры для поддержания этой информации в актуальном состоянии.

В отличие от решения{285} задачи 4.1.1.a{284} здесь значение по умолчанию для нового поля представляет собой не `NULL`, а 0 (потому что «читатель ни разу не приходил в библиотеку» — это «неизвестность», т.е. `NULL`, а «у читателя нет книг» — это вполне чёткое и понятное значение, т.е. 0). Следуя той же логике, в инициализирующем запросе мы используем `JOIN`, а не `LEFT JOIN` — нет никакого смысла обновлять данные для читателей, ни разу не бравших книги.

Итак, для MySQL первые два шага выполняются с помощью следующих запросов.

MySQL Решение 4.1.2.a (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE `subscribers`
  ADD COLUMN `s_books` INT(11) NOT NULL DEFAULT 0 AFTER `s_name`;

-- Инициализация данных:
UPDATE `subscribers`
       JOIN (SELECT `sb_subscriber`,
                    COUNT(`sb_id`) AS `s_has_books`
             FROM   `subscriptions`
             WHERE  `sb_is_active` = 'Y'
             GROUP  BY `sb_subscriber`) AS `prepared_data`
       ON `s_id` = `sb_subscriber`
SET    `s_books` = `s_has_books`;
```

Как видно из инициализирующего запроса, всю необходимую информацию для формирования значения поля **s_books** мы можем взять из таблицы **subscriptions**. На ней мы и будем создавать триггеры.

С **INSERT**- и **DELETE**-триггерами всё просто: если добавляется или удаляется «активная» выдача (поле **sb_is_active** равно **Y**), нужно увеличить или уменьшить на единицу значение счётчика выданных книг у соответствующего читателя.

MySQL Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
DELIMITER $$

-- Реакция на добавление выдачи книги:
CREATE TRIGGER `s_has_books_on_subscriptions_ins`
AFTER INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   IF (NEW.`sb_is_active` = 'Y') THEN
    UPDATE `subscribers`
    SET    `s_books` = `s_books` + 1
    WHERE  `s_id` = NEW.`sb_subscriber`;
   END IF;
  END;
$$

-- Реакция на удаление выдачи книги:
CREATE TRIGGER `s_has_books_on_subscriptions_del`
AFTER DELETE
ON `subscriptions`
 FOR EACH ROW
  BEGIN
   IF (OLD.`sb_is_active` = 'Y') THEN
    UPDATE `subscribers`
    SET    `s_books` = `s_books` - 1
    WHERE  `s_id` = OLD.`sb_subscriber`;
   END IF;
  END;
$$

DELIMITER ;
```

С **UPDATE**-триггером ситуация будет более сложной, т.к. у нас есть два параметра, которые могут как измениться, так и остаться неизменными — идентификатор читателя и состояние выдачи.

MySQL Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
DELIMITER $$

-- Реакция на обновление выдачи книги:
CREATE TRIGGER `s_has_books_on_subscriptions_upd`
AFTER UPDATE
ON `subscriptions`
 FOR EACH ROW
  BEGIN

    -- A) Читатель тот же, Y -> N
    IF ((OLD.`sb_subscriber` = NEW.`sb_subscriber`) AND
        (OLD.`sb_is_active` = 'Y') AND
        (NEW.`sb_is_active` = 'N')) THEN
     UPDATE `subscribers`
     SET    `s_books` = `s_books` - 1
     WHERE  `s_id` = OLD.`sb_subscriber`;
    END IF;

    -- B) Читатель тот же, N -> Y
    IF ((OLD.`sb_subscriber` = NEW.`sb_subscriber`) AND
        (OLD.`sb_is_active` = 'N') AND
        (NEW.`sb_is_active` = 'Y')) THEN
     UPDATE `subscribers`
     SET    `s_books` = `s_books` + 1
     WHERE  `s_id` = OLD.`sb_subscriber`;
    END IF;

    -- C) Читатели разные, Y -> Y
    IF ((OLD.`sb_subscriber` != NEW.`sb_subscriber`) AND
        (OLD.`sb_is_active` = 'Y') AND
        (NEW.`sb_is_active` = 'Y')) THEN
     UPDATE `subscribers`
     SET    `s_books` = `s_books` - 1
     WHERE  `s_id` = OLD.`sb_subscriber`;
     UPDATE `subscribers`
     SET    `s_books` = `s_books` + 1
     WHERE  `s_id` = NEW.`sb_subscriber`;
    END IF;

```

MySQL Решение 4.1.2.a (триггеры для таблицы subscriptions) (продолжение)

```
40      -- D) Читатели разные, Y -> N
41      IF ((OLD.`sb_subscriber` != NEW.`sb_subscriber`) AND
42          (OLD.`sb_is_active` = 'Y') AND
43          (NEW.`sb_is_active` = 'N')) THEN
44       UPDATE `subscribers`
45       SET    `s_books` = `s_books` - 1
46       WHERE  `s_id` = OLD.`sb_subscriber`;
47      END IF;
48
49      -- E) Читатели разные, N -> Y
50      IF ((OLD.`sb_subscriber` != NEW.`sb_subscriber`) AND
51          (OLD.`sb_is_active` = 'N') AND
52          (NEW.`sb_is_active` = 'Y')) THEN
53       UPDATE `subscribers`
54       SET    `s_books` = `s_books` + 1
55       WHERE  `s_id` = NEW.`sb_subscriber`;
56      END IF;
57
58    END;
59  $$
60
61  DELIMITER ;
```

Изобразим рассмотренные ситуации графически.

	Старое значение sb_is_active	Новое значение sb_is_active	Действие	Код действия
Идентификатор читателя остался неизменным	Y	Y	-	
	Y	N	OLD-1	A
	N	Y	OLD+1	B
	N	N	-	
Идентификатор читателя изменился	Y	Y	OLD-1, NEW+1	C
	Y	N	OLD-1	D
	N	Y	NEW+1	E
	N	N	-	

И, наконец, здесь мы приведём решение проблемы, описанной в заданиях 3.2.1.TSK.D{249}, 4.1.1.TSK.D{305}, 4.1.1.TSK.E{305}. Напомним, что MySQL не активирует триггеры каскадными операциями, потому удаление книг (которое приведёт к удалению всех записей о выдачах этих книг) активирует «незаметное» для **DELETE**-триггера на таблице **subscriptions** удаление данных.

Чтобы учесть этот эффект, мы создадим дополнительный триггер на таблице **books**, реагирующий на удаление книг (вставка или обновление данных в таблице **books** не влияет на распределение уже имеющихся выдач книг по тем или иным читателям, потому здесь достаточно создать только **DELETE**-триггер).

Обратите внимание: здесь мы создаём **BEFORE**-триггер, т.к. в момент активации **AFTER**-триггера искомая информация в таблице subscriptions уже будет удалена.

MySQL Решение 4.1.2.a (триггер для таблицы books)

```
DELIMITER $$

-- Реакция на удаление книги:
CREATE TRIGGER `s_has_books_on_books_del`
BEFORE DELETE
ON `books`
 FOR EACH ROW
  BEGIN
   UPDATE `subscribers`
   JOIN (SELECT   `sb_subscriber`,
                  COUNT(`sb_book`) AS `delta`
         FROM     `subscriptions`
         WHERE    `sb_book` = OLD.`b_id`
              AND `sb_is_active` = 'Y'
         GROUP BY `sb_subscriber`) AS `prepared_data`
         ON `s_id` = `sb_subscriber`
   SET   `s_books` = `s_books` - `delta`;
  END;
$$

DELIMITER ;
```

Проверим работоспособность полученного решения. Будем изменять данные в таблицах **books** и **subscriptions** и отслеживать изменения данных в таблице **subscribers**.

Исходное состояние таблицы **subscribers** таково:

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Проверим реакцию на добавление и удаление выдач книг. Добавим Иванову И.И. активную выдачу, а Петрову П.П. неактивную:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (200,
             1,
             1,
             '2011-01-12',
             '2011-02-12',
             'Y'),
            (201,
             2,
             1,
             '2011-01-12',
             '2011-02-12',
             'N')
```

s_id	s_name	s_books
1	Иванов И.И.	1
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Удалим добавленные выдачи:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
DELETE FROM `subscriptions`
WHERE  `sb_id` IN ( 200, 201 )
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Проверим реакцию на обновление выдач книг. Сначала добавим выдачу:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (300,
             1,
             1,
             '2011-01-12',
             '2011-02-12',
             'Y')
```

s_id	s_name	s_books
1	Иванов И.И.	1
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Не меняя идентификатор читателя сделаем выдачу неактивной:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
-- A
UPDATE `subscriptions`
SET    `sb_is_active` = 'N'
WHERE  `sb_id` = 300
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Не меняя идентификатор читателя сделаем выдачу снова активной:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
-- B
UPDATE `subscriptions`
SET    `sb_is_active` = 'Y'
WHERE  `sb_id` = 300
```

s_id	s_name	s_books
1	Иванов И.И.	1
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Изменим идентификатор читателя, не меняя состояние активности выдачи:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
-- C
UPDATE `subscriptions`
SET    `sb_subscriber` = 2
WHERE  `sb_id` = 300
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	1
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Изменим идентификатор читателя и сделаем выдачу неактивной:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
-- D
UPDATE `subscriptions`
SET    `sb_subscriber` = 1,
       `sb_is_active` = 'N'
WHERE  `sb_id` = 300
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Изменим идентификатор читателя и сделаем выдачу активной:

MySQL Решение 4.1.2.a (проверка работоспособности)

```
-- E
UPDATE `subscriptions`
SET    `sb_subscriber` = 2,
       `sb_is_active` = 'Y'
WHERE  `sb_id` = 300
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	1
3	Сидоров С.С.	3
4	Сидоров С.С.	2

Удалим книгу с идентификатором 1 (такая книга сейчас выдана Петрову и обоим Сидоровым по одному экземпляру):

MySQL Решение 4.1.2.a (проверка работоспособности)

```
DELETE FROM `books`
WHERE `b_id` = 1
```

s_id	s_name	s_books
1	Иванов И.И.	0
2	Петров П.П.	0
3	Сидоров С.С.	2
4	Сидоров С.С.	1

Как показало исследование, все операции выполняются корректно и приводят к верным изменениям значений поля **s_books**.

Переходим к решению поставленной задачи для MS SQL Server. Модифицируем таблицу **subscribers** и проинициализируем добавленное поле данными.

MS SQL Решение 4.1.2.a (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE [subscribers]
  ADD [s_books] INT NOT NULL DEFAULT 0;

-- Инициализация данных:
UPDATE [subscribers]
SET    [s_books] = [s_has_books]
FROM   [subscribers]
       JOIN (SELECT [sb_subscriber],
                    COUNT([sb_id]) AS [s_has_books]
             FROM   [subscriptions]
             WHERE  [sb_is_active] = 'Y'
             GROUP  BY [sb_subscriber]) AS [prepared_data]
       ON [s_id] = [sb_subscriber];
```

Логика работы триггеров в MS SQL Server будет иной, т.к. эта СУБД не поддерживает триггеры уровня записи, и нам придётся за один раз обрабатывать все произведённые изменения.

С **INSERT**- и **DELETE**-триггерами, как и в случае с MySQL, будет более-менее просто — нужно выяснить идентификаторы читателей, получивших (или вернувших) книги, количество таких книг по каждому читателю, а затем увеличить или уменьшить счётчики выданных книг соответствующим читателям на соответствующие величины.

MS SQL Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
-- Реакция на добавление выдачи книги:
CREATE TRIGGER [s_has_books_on_subscriptions_ins]
ON [subscriptions]
AFTER INSERT
AS
UPDATE [subscribers]
SET    [s_books] = [s_books] + [s_new_books]
FROM   [subscribers]
       JOIN (SELECT [sb_subscriber],
                    COUNT([sb_id]) AS [s_new_books]
             FROM   [inserted]
                 WHERE  [sb_is_active] = 'Y'
             GROUP  BY [sb_subscriber]) AS [prepared_data]
       ON [s_id] = [sb_subscriber];
GO

```

MS SQL Решение 4.1.2.a (триггеры для таблицы subscriptions) (продолжение)

```
-- Реакция на удаление выдачи книги:
CREATE TRIGGER [s_has_books_on_subscriptions_del]
ON [subscriptions]
AFTER DELETE
AS
UPDATE [subscribers]
SET    [s_books] = [s_books] - [s_old_books]
FROM   [subscribers]
       JOIN (SELECT [sb_subscriber],
                    COUNT([sb_id]) AS [s_old_books]
             FROM   [deleted]
                 WHERE  [sb_is_active] = 'Y'
             GROUP  BY [sb_subscriber]) AS [prepared_data]
       ON [s_id] = [sb_subscriber];

GO
```

UPDATE-триггер получится чуть более сложным, но не настолько, как в MySQL: здесь мы можем посчитать количество сданных и возвращённых книг на основе информации из псевдотаблиц **deleted** и **inserted**, и при этом нам не важно изменение состояния выдач книг — мы лишь считаем (независимо) количество сданных и полученных книг для каждого читателя и изменяем его счётчик книг на эти две величины.

Иными словами, нам не важно, из какого в какое состояние (и от какого к какому читателю) переключается выдача — нас интересует только «у кого удалились активные выдачи», и «кому добавились активные выдачи».

Потому **UPDATE**-триггер будет просто содержать в себе код из **INSERT**-триггера и **DELETE**-триггера. Чуть более элегантным решением была бы реализация в теле триггера такой логики, при которой было бы достаточно выполнить только одну операцию обновления таблицы **subscribers** — в этом и будет состоять задание 4.1.2.TSK.C{331}.

Для проверки работоспособности полученного решения можно использовать запросы, представленные после кода **UPDATE**-триггера (их общая логика и ожидаемая реакция СУБД пояснены в решении для MySQL, но обратите внимание, что здесь мы оперируем чуть большим количеством данных в каждом запросе).

MS SQL Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
-- Реакция на обновление выдачи книги:
CREATE TRIGGER [s_has_books_on_subscriptions_upd]
ON [subscriptions]
AFTER UPDATE
AS
-- (Это, фактически, -- код DELETE-триггера):
UPDATE [subscribers]
SET    [s_books] = [s_books] - [s_old_books]
FROM   [subscribers]
       JOIN (SELECT [sb_subscriber],
                    COUNT([sb_id]) AS [s_old_books]
             FROM   [deleted]
                 WHERE  [sb_is_active] = 'Y'
             GROUP  BY [sb_subscriber]) AS [prepared_data]
       ON [s_id] = [sb_subscriber];
```

MS SQL Решение 4.1.2.a (триггеры для таблицы subscriptions) (продолжение)

```
-- (Это, фактически, -- код INSERT-триггера):
UPDATE [subscribers]
SET    [s_books] = [s_books] + [s_new_books]
FROM   [subscribers]
       JOIN (SELECT [sb_subscriber],
                    COUNT([sb_id]) AS [s_new_books]
             FROM   [inserted]
                 WHERE  [sb_is_active] = 'Y'
             GROUP  BY [sb_subscriber]) AS [prepared_data]
       ON [s_id] = [sb_subscriber];
GO
```

MS SQL Решение 4.1.2.a (проверка работоспособности)

```
SET IDENTITY_INSERT [subscriptions] ON;

-- Добавим Иванову И.И. две активных выдачи,
-- а Петрову П.П. одну активную и одну неактивную:
INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (200,
             1,
             3,
             '2011-01-12',
             '2011-02-12',
             'Y'),
            (201,
             1,
             4,
             '2011-01-12',
             '2011-02-12',
             'Y'),
            (202,
             2,
             3,
             '2011-01-12',
             '2011-02-12',
             'Y'),
            (203,
             2,
             4,
             '2011-01-12',
             '2011-02-12',
             'N');
-- Удалим добавленные выдачи:
DELETE FROM [subscriptions]
WHERE  [sb_id] IN (200, 201, 202, 203);

```

MS SQL Решение 4.1.2.a (проверка работоспособности) (продолжение)

```
-- Проверим реакцию на обновление выдач книг.
-- Сначала добавим две выдачи:
INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
VALUES      (300,
             1,
             3,
             '2011-01-12',
             '2011-02-12',
             'Y'),
            (301,
             1,
             4,
             '2011-01-12',
             '2011-02-12',
             'Y');

-- Не меняя идентификатор читателя сделаем выдачи неактивными:
UPDATE [subscriptions]
SET    [sb_is_active] = 'N'
WHERE  [sb_id] IN (300, 301);

-- Не меняя идентификатор читателя сделаем выдачи снова активными:
UPDATE [subscriptions]
SET    [sb_is_active] = 'Y'
WHERE  [sb_id] IN (300, 301);

-- Изменим идентификатор читателя, не меняя состояние активности выдач:
UPDATE [subscriptions]
SET    [sb_subscriber] = 2
WHERE  [sb_id] IN (300, 301);

-- Изменим идентификатор читателя и сделаем выдачи неактивными:
UPDATE [subscriptions]
SET    [sb_subscriber] = 1,
       [sb_is_active] = 'N'
WHERE  [sb_id] IN (300, 301);

-- Изменим идентификатор читателя и сделаем выдачи активными:
UPDATE [subscriptions]
SET    [sb_subscriber] = 2,
       [sb_is_active] = 'Y'
WHERE  [sb_id] IN (300, 301);

-- Удаление книги с идентификатором 1 (выдана по одному экземпляру
-- Петрову и обоим Сидоровым):
DELETE FROM [books]
WHERE [b_id] = 1;

SET IDENTITY_INSERT [subscriptions] OFF;
```

Переходим к решению поставленной задачи для Oracle. Модифицируем таблицу **subscribers** и проинициализируем добавленное поле данными.

MS SQL Решение 4.1.2.a (модификация таблицы и инициализация данных)

```
1  -- Модификация таблицы:
2  ALTER TABLE "subscribers"
3    ADD ("s_books" INT DEFAULT 0 NOT NULL);
4
5  -- Инициализация данных:
6  UPDATE "subscribers"
7  SET    "s_books" = NVL(
8         (SELECT   COUNT("sb_id") AS "s_has_books"
9          FROM     "subscriptions"
10         WHERE    "sb_is_active" = 'Y'
11              AND "sb_subscriber" = "s_id"
12         GROUP BY "sb_subscriber"), 0);
```

Обратите внимание: из-за синтаксических особенностей Oracle, вынуждающих нас писать такой запрос на обновление, приходится применять функцию **NVL**, потому что коррелирующий подзапрос в строках 7-16 выполнится для каждого ряда таблицы **subscribers**, и в некоторых случаях вернёт **NULL**.

Несмотря на то, что Oracle (как и MS SQL Server) поддерживает триггеры уровня выражения, мы не можем использовать представленную в решении для MS SQL логику, т.к. в Oracle нет псевдотаблиц **inserted** и **updated**. Нам придётся идти по пути решения для MySQL и использовать триггеры уровня записи.

Oracle Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
1  -- Реакция на добавление выдачи книги:
2  CREATE OR REPLACE TRIGGER "s_has_books_on_sbps_ins"
3  AFTER INSERT
4  ON "subscriptions"
5  FOR EACH ROW
6   BEGIN
7     IF (:new."sb_is_active" = 'Y') THEN
8      UPDATE "subscribers"
9      SET    "s_books" = "s_books" + 1
10     WHERE  "s_id" = :new."sb_subscriber";
11    END IF;
12  END;
13
14 -- Реакция на удаление выдачи книги:
15 CREATE OR REPLACE TRIGGER "s_has_books_on_sbps_del"
16 AFTER DELETE
17 ON "subscriptions"
18 FOR EACH ROW
19  BEGIN
20    IF (:old."sb_is_active" = 'Y') THEN
21     UPDATE "subscribers"
22     SET    "s_books" = "s_books" - 1
23     WHERE  "s_id" = :old."sb_subscriber";
24    END IF;
25  END;
```

В **UPDATE**-триггере мы также используем один в один тот же самый код, который был использован в решении для MySQL (там же были рассмотрены и показаны графически все ситуации, которые должен учитывать данный триггер).

Oracle Решение 4.1.2.a (триггеры для таблицы subscriptions)

```
1   -- Реакция на обновление выдачи книги:
2   CREATE OR REPLACE TRIGGER "s_has_books_on_sbps_upd"
3   AFTER UPDATE
4   ON "subscriptions"
5   FOR EACH ROW
6    BEGIN
7      -- A) Читатель тот же, Y -> N
8      IF ((:old."sb_subscriber" = :new."sb_subscriber") AND
9          (:old."sb_is_active" = 'Y') AND
10         (:new."sb_is_active" = 'N')) THEN
11      UPDATE "subscribers"
12      SET    "s_books" = "s_books" - 1
13      WHERE  "s_id" = :old."sb_subscriber";
14     END IF;
15
16     -- B) Читатель тот же, N -> Y
17     IF ((:old."sb_subscriber" = :new."sb_subscriber") AND
18         (:old."sb_is_active" = 'N') AND
19         (:new."sb_is_active" = 'Y')) THEN
20      UPDATE "subscribers"
21      SET    "s_books" = "s_books" + 1
22      WHERE  "s_id" = :old."sb_subscriber";
23     END IF;
24
25     -- C) Читатели разные, Y -> Y
26     IF ((:old."sb_subscriber" != :new."sb_subscriber") AND
27         (:old."sb_is_active" = 'Y') AND
28         (:new."sb_is_active" = 'Y')) THEN
29      UPDATE "subscribers"
30      SET    "s_books" = "s_books" - 1
31      WHERE  "s_id" = :old."sb_subscriber";
32      UPDATE "subscribers"
33      SET    "s_books" = "s_books" + 1
34      WHERE  "s_id" = :new."sb_subscriber";
35     END IF;
36
37     -- D) Читатели разные, Y -> N
38     IF ((:old."sb_subscriber" != :new."sb_subscriber") AND
39         (:old."sb_is_active" = 'Y') AND
40         (:new."sb_is_active" = 'N')) THEN
41      UPDATE "subscribers"
42      SET    "s_books" = "s_books" - 1
43      WHERE  "s_id" = :old."sb_subscriber";
44     END IF;
45
```

Oracle Решение 4.1.2.a (триггеры для таблицы subscriptions) (продолжение)

```
    -- E) Читатели разные, N -> Y
    IF ((:old."sb_subscriber" != :new."sb_subscriber") AND
        (:old."sb_is_active" = 'N') AND
        (:new."sb_is_active" = 'Y')) THEN
     UPDATE "subscribers"
     SET    "s_books" = "s_books" + 1
     WHERE  "s_id" = :new."sb_subscriber";
    END IF;
  END;
```

В итоге код триггеров для Oracle получился полностью идентичным коду триггеров для MySQL, потому и запросы для проверки работоспособности полученного решения также совпадают для обеих СУБД.

См. код самих запросов ниже, а логика их работы с пояснением и демонстрацией изменения содержимого таблицы `subscribers` представлена в решении для MySQL.

Oracle Решение 4.1.2.a (проверка работоспособности)

```
ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE;

-- Добавим Иванову И.И. активную выдачу, а Петрову П.П. неактивную:
INSERT INTO "subscriptions"
VALUES      (200,
             1,
             1,
             TO_DATE('2011-01-12', 'YYYY-MM-DD'),
             TO_DATE('2011-02-12', 'YYYY-MM-DD'),
             'Y');
INSERT INTO "subscriptions"
VALUES      (201,
             2,
             1,
             TO_DATE('2011-01-12', 'YYYY-MM-DD'),
             TO_DATE('2011-02-12', 'YYYY-MM-DD'),
             'N');

-- Удалим добавленные выдачи:
DELETE FROM "subscriptions"
WHERE  "sb_id" IN ( 200, 201 );

-- Проверим реакцию на обновление выдач книг. Сначала добавим выдачу:
INSERT INTO "subscriptions"
VALUES      (300,
             1,
             1,
             TO_DATE('2011-01-12', 'YYYY-MM-DD'),
             TO_DATE('2011-02-12', 'YYYY-MM-DD'),
             'Y');

-- A) Не меняя идентификатор читателя сделаем выдачу неактивной:
UPDATE "subscriptions"
SET    "sb_is_active" = 'N'
WHERE  "sb_id" = 300;

```

Oracle Решение 4.1.2.a (проверка работоспособности) (продолжение)

```
-- B) Не меняя идентификатор читателя сделаем выдачу снова активной:
UPDATE "subscriptions"
SET    "sb_is_active" = 'Y'
WHERE  "sb_id" = 300;

-- C) Изменим идентификатор читателя, не меняя состояние активности выдачи:
UPDATE "subscriptions"
SET    "sb_subscriber" = 2
WHERE  "sb_id" = 300;

-- D) Изменим идентификатор читателя и сделаем выдачу неактивной:
UPDATE "subscriptions"
SET    "sb_subscriber" = 1,
       "sb_is_active" = 'N'
WHERE  "sb_id" = 300;

-- E) Изменим идентификатор читателя и сделаем выдачу активной:
UPDATE "subscriptions"
SET    "sb_subscriber" = 2,
       "sb_is_active" = 'Y'
WHERE  "sb_id" = 300;

-- Удалим книгу с id = 1 (выдана по одной штуке Петрову и обоим Сидоровым):
DELETE FROM [books]
WHERE [b_id] = 1;

ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE;
```

Итак, решение данной задачи получено и проверено для всех трёх СУБД.

Решение 4.1.2.b{305}.

Как и в решении{306} задачи 4.1.2.a{305} здесь нужно будет выполнить те же самые действия — модифицировать таблицу, проинициализировать данные, создать триггеры. И даже код триггеров будет чем-то похож на рассмотренные ранее решения.

Традиционно начинаем с решения для MySQL: модифицируем таблицу и проинициализируем данные.

MySQL Решение 4.1.2.b (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE `genres`
  ADD COLUMN `g_books` INT(11) NOT NULL DEFAULT 0 AFTER `g_name`;

-- Инициализация данных:
UPDATE `genres`
       JOIN (SELECT `g_id`,
                    COUNT(`b_id`) AS `g_has_books`
             FROM   `m2m_books_genres`
             GROUP  BY `g_id`) AS `prepared_data`
       USING (`g_id`)
SET    `g_books` = `g_has_books`;
```

Код всех трёх триггеров будет предельно прост: в **INSERT**-триггере мы увеличиваем счётчик книг у соответствующего жанра, в **DELETE**-триггере — уменьшаем, в **UPDATE**-триггере уменьшаем «старому» жанру и увеличиваем «новому» жанру. Никаких дополнительных проверок и ухищрений здесь не требуется.

MySQL Решение 4.1.2.b (триггеры для таблицы m2m_books_genres)

```
DELIMITER $$

-- Реакция на добавление связи между книгами и жанрами:
CREATE TRIGGER `g_has_books_on_m2m_b_g_ins`
AFTER INSERT
ON `m2m_books_genres`
 FOR EACH ROW
  BEGIN
   UPDATE `genres`
   SET    `g_books` = `g_books` + 1
   WHERE  `g_id` = NEW.`g_id`;
  END;
$$

-- Реакция на обновление связи между книгами и жанрами:
CREATE TRIGGER `g_has_books_on_m2m_b_g_upd`
AFTER UPDATE
ON `m2m_books_genres`
 FOR EACH ROW
  BEGIN
   UPDATE `genres`
   SET    `g_books` = `g_books` - 1
   WHERE  `g_id` = OLD.`g_id`;
   UPDATE `genres`
   SET    `g_books` = `g_books` + 1
   WHERE  `g_id` = NEW.`g_id`;
  END;
$$
```

MySQL Решение 4.1.2.b (триггеры для таблицы m2m_books_genres) (продолжение)

```
-- Реакция на удаление связи между книгами и жанрами:
CREATE TRIGGER `g_has_books_on_m2m_b_g_del`
AFTER DELETE
ON `m2m_books_genres`
 FOR EACH ROW
  BEGIN
   UPDATE `genres`
   SET    `g_books` = `g_books` - 1
   WHERE  `g_id` = OLD.`g_id`;
  END;
$$

DELIMITER ;
```

Поскольку в MySQL триггеры не активируются каскадными операциями, удаление книги (которое приведёт к удалению всех её связей со всеми жанрами) останется незаметным для триггеров на таблице **m2m_books_genres**. Потому мы должны создать триггер на таблице **books**, учитывающий соответствующую ситуацию.

Каждая книга связана с каждым жанром не более одного раза, потому при удалении любой книги нужно на единицу уменьшить счётчик книг у каждого из жанров, с которыми она связана.

MySQL Решение 4.1.2.b (триггер для таблицы books)

```
DELIMITER $$

-- Реакция на удаление книги:
CREATE TRIGGER `g_has_books_on_books_del`
BEFORE DELETE
ON `books`
 FOR EACH ROW
  BEGIN
   UPDATE `genres`
   SET    `g_books` = `g_books` - 1
   WHERE  `g_id` IN (SELECT `g_id`
                     FROM   `m2m_books_genres`
                     WHERE  `b_id` = OLD.`b_id`);
  END;
$$

DELIMITER ;
```

Проверим корректность полученного решения. Будем модифицировать данные в таблицах **m2m_books_genres** и **books** и проверять изменения в таблице **genres**.

Исходное состояние таблицы genres:

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	0
5	Классика	4
6	Фантастика	1

Добавим две связи к жанру «Наука» (идентификатор жанра равен 4):

MySQL Решение 4.1.2.b (проверка работоспособности)

```
INSERT INTO `m2m_books_genres`
            (`b_id`,
             `g_id`)
VALUES      (1, 4),
            (2, 4)
```

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	2
5	Классика	4
6	Фантастика	1

Изменим в этих связях значения идентификаторов книг, не меняя значения идентификаторов жанров:

MySQL Решение 4.1.2.b (проверка работоспособности)

```
UPDATE `m2m_books_genres`
SET    `b_id` = 3
WHERE  `b_id` = 1
   AND `g_id` = 4;

UPDATE `m2m_books_genres`
SET    `b_id` = 4
WHERE  `b_id` = 2
   AND `g_id` = 4;
```

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	2
5	Классика	4
6	Фантастика	1

Изменим в этих связях значения идентификаторов жанров, не меняя значения идентификаторов книг:

MySQL Решение 4.1.2.b (проверка работоспособности)

```
UPDATE `m2m_books_genres`
SET    `g_id` = 5
WHERE  `b_id` = 3
   AND `g_id` = 4;

UPDATE `m2m_books_genres`
SET    `g_id` = 5
WHERE  `b_id` = 4
   AND `g_id` = 4;
```

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	0
5	Классика	6
6	Фантастика	1

Изменим в этих связях значения идентификаторов жанров, и идентификаторов книг одновременно:

MySQL Решение 4.1.2.b (проверка работоспособности)

```
UPDATE `m2m_books_genres`
SET    `b_id` = 1,
       `g_id` = 4
WHERE  `b_id` = 3
   AND `g_id` = 5;

UPDATE `m2m_books_genres`
SET    `b_id` = 2,
       `g_id` = 4
WHERE  `b_id` = 4
   AND `g_id` = 5;
```

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	2
5	Классика	4
6	Фантастика	1

Удалим эти созданные для проверки работоспособности решения связи:

MySQL Решение 4.1.2.b (проверка работоспособности)

```
DELETE FROM `m2m_books_genres`
WHERE  `b_id` = 1
   AND `g_id` = 4;

DELETE FROM `m2m_books_genres`
WHERE  `b_id` = 2
   AND `g_id` = 4;
```

g_id	g_name	g_books
1	Поэзия	2
2	Программирование	3
3	Психология	1
4	Наука	0
5	Классика	4
6	Фантастика	1

Удалим книги с идентификаторами 1 и 2 (обе эти книги одновременно относятся к жанрам «Поэзия» и «Классика»):

MySQL Решение 4.1.2.b (проверка работоспособности)

```
DELETE FROM `books`
WHERE `b_id` IN (1, 2)
```

g_id	g_name	g_books
1	Поэзия	0
2	Программирование	3
3	Психология	1
4	Наука	0
5	Классика	2
6	Фантастика	1

Итак, решение для MySQL завершено и проверено.

Переходим к решению для MS SQL Server. Модифицируем таблицу и проинициализируем данные.

MS SQL Решение 4.1.2.b (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE [genres]
  ADD [g_books] INT NOT NULL DEFAULT 0;

-- Инициализация данных:
UPDATE [genres]
SET    [g_books] = [g_has_books]
FROM   [genres]
       JOIN (SELECT [g_id],
                    COUNT([b_id]) AS [g_has_books]
             FROM   [m2m_books_genres]
             GROUP  BY [g_id]) AS [prepared_data]
         ON [genres].[g_id] = [prepared_data].[g_id];
```

В MS SQL Server триггеры активируются каскадными операциями, потому здесь будет достаточно создать триггеры только на таблице **m2m_books_genres**.

INSERT- и **DELETE**-триггеры достаточно просты: каждый из них подсчитывает количество книг, добавленных к жанру или убранных у жанра, и изменяет счётчик книг у соответствующего жанра на полученное значение.

MS SQL Решение 4.1.2.b (триггеры для таблицы m2m_books_genres)

```
-- Реакция на добавление связи между книгами и жанрами:
CREATE TRIGGER [g_has_books_on_m2m_b_g_ins]
ON [m2m_books_genres]
AFTER INSERT
AS
UPDATE [genres]
SET    [g_books] = [g_books] + [g_new_books]
FROM   [genres]
       JOIN (SELECT [g_id],
                    COUNT([b_id]) AS [g_new_books]
             FROM   [inserted]
             GROUP  BY [g_id]) AS [prepared_data]
         ON [genres].[g_id] = [prepared_data].[g_id];
GO

```

MS SQL Решение 4.1.2.b (триггеры для таблицы m2m_books_genres) (продолжение)

```
16 -- Реакция на обновление связи между книгами и жанрами:
17 CREATE TRIGGER [g_has_books_on_m2m_b_g_upd]
18 ON [m2m_books_genres]
19 AFTER UPDATE
20 AS
21 UPDATE [genres]
22 SET    [g_books] = [g_books] + [delta]
23 FROM   [genres]
24        JOIN (SELECT [g_id],
25                     SUM([delta]) AS [delta]
26              FROM   (SELECT [g_id],
27                             -COUNT([b_id]) AS [delta]
28                      FROM   [deleted]
29                      GROUP  BY [g_id]
30                      UNION
31                      SELECT [g_id],
32                             COUNT([b_id]) AS [delta]
33                      FROM   [inserted]
34                      GROUP  BY [g_id]) AS [raw_deltas]
35              GROUP  BY [g_id]) AS [ready_delta]
36          ON [genres].[g_id] = [ready_delta].[g_id];
37 GO
```

MS SQL Решение 4.1.2.b (триггеры для таблицы m2m_books_genres) (продолжение)

```
38 -- Реакция на удаление связи между книгами и жанрами:
39 CREATE TRIGGER [g_has_books_on_m2m_b_g_del]
40 ON [m2m_books_genres]
41 AFTER DELETE
42 AS
43 UPDATE [genres]
44 SET    [g_books] = [g_books] - [g_old_books]
45 FROM   [genres]
46        JOIN (SELECT [g_id],
47                     COUNT([b_id]) AS [g_old_books]
48              FROM   [deleted]
49              GROUP  BY [g_id]) AS [prepared_data]
50          ON [genres].[g_id] = [prepared_data].[g_id];
51 GO
```

Логика **UPDATE**-триггера чуть более сложная. Чтобы не выполнять два отдельных обновления таблицы **genres**, мы сначала в строках 26-34 запроса получаем «сводную таблицу» по удалённым и добавленным связям между книгами и жанрами. Эта таблица в некоторой гипотетической ситуации может выглядеть так:

g_id	delta
3	4
5	1
1	-3
3	-6
6	2

Со знаком минус представлено количество удалённых связей между книгами и жанрами, со знаком полюс — количество добавленных связей. Обратите внимание на жанр с идентификатором 3, у которого за одну операцию обновления часть связей было удалено, часть добавлено. В строке 25 запроса эти отрицательные и положительные значения суммируются, формируя таким образом итоговую дельту количества связей между жанрами и книгами.

g_id	delta
3	-2
5	1
1	-3
6	2

В строке 22 запроса эти данные используются для изменения значения счётчика связей между жанрами и книгами.

Проверить работоспособность полученного решения можно с помощью следующих запросов (которые подробно рассмотрены в решении для MySQL).

MS SQL Решение 4.1.2.b (проверка работоспособности)

```
1   -- Добавление двух связей к жанру «Наука» (идентификатор жанра равен 4):
2   INSERT INTO [m2m_books_genres]
3               ([b_id],
4                [g_id])
5   VALUES      (1, 4),
6               (2, 4);
7   -- Изменение в добавленных связях значения идентификаторов книг,
8   -- без изменения значения идентификаторов жанров:
9   UPDATE [m2m_books_genres]
10  SET    [b_id] = 3
11  WHERE  [b_id] = 1
12     AND [g_id] = 4;
13
14  UPDATE [m2m_books_genres]
15  SET    [b_id] = 4
16  WHERE  [b_id] = 2
17     AND [g_id] = 4;
18
19  -- Изменение в добавленных связях значения идентификаторов жанров,
20  -- без изменения значения идентификаторов книг:
21  UPDATE [m2m_books_genres]
22  SET    [g_id] = 5
23  WHERE  [b_id] = 3
24     AND [g_id] = 4;
25
26  UPDATE [m2m_books_genres]
27  SET    [g_id] = 5
28  WHERE  [b_id] = 4
29     AND [g_id] = 4;
30
```

MS SQL Решение 4.1.2.b (проверка работоспособности) (продолжение)

```
-- Изменение в добавленных связях значения идентификаторов жанров,
-- и идентификаторов книг одновременно:
UPDATE [m2m_books_genres]
SET    [b_id] = 1,
       [g_id] = 4
WHERE  [b_id] = 3
   AND [g_id] = 5;

UPDATE [m2m_books_genres]
SET    [b_id] = 2,
       [g_id] = 4
WHERE  [b_id] = 4
   AND [g_id] = 5;

-- Удаление ранее созданных связей:
DELETE FROM [m2m_books_genres]
WHERE  [b_id] = 1
   AND [g_id] = 4;

DELETE FROM [m2m_books_genres]
WHERE  [b_id] = 2
   AND [g_id] = 4;

-- Удаление книг с идентификаторами 1 и 2 (обе эти книги одновременно
-- относятся к жанрам «Поэзия» и «Классика»):
DELETE FROM [books]
WHERE [b_id] IN (1, 2);
```

Итак, решение для MySQL завершено и проверено.

Переходим к решению для Oracle. Т.к. данная СУБД не поддерживает псевдотаблицы **inserted** и **deleted**, мы будем опираться на логику решения для MySQL. Все соответствующие подробности этого решения уже описаны выше, потому здесь будет представлен только SQL-код.

Также отметим, что поскольку в Oracle триггеры активируются каскадными операциями, в данном решении (в отличие от решения для MySQL) не потребуется создавать **DELETE**-триггер на таблице **books**.

Oracle Решение 4.1.2.b (модификация таблицы и инициализация данных)

```
-- Модификация таблицы:
ALTER TABLE "genres"
  ADD ("g_books" NUMBER(10) DEFAULT 0 NOT NULL);

-- Инициализация данных:
UPDATE "genres" "outer"
SET    "g_books" =
         NVL((SELECT COUNT("b_id") AS "g_has_books"
         FROM   "m2m_books_genres"
         WHERE "outer"."g_id" = "g_id"
         GROUP  BY "g_id"), 0);
```

Oracle Решение 4.1.2.b (триггеры для таблицы m2m_books_genres)

```
-- Реакция на добавление связи между книгами и жанрами:
CREATE TRIGGER "g_has_bks_on_m2m_b_g_ins"
AFTER INSERT
ON "m2m_books_genres"
 FOR EACH ROW
  BEGIN
   UPDATE "genres"
   SET    "g_books" = "g_books" + 1
   WHERE  "g_id" = :new."g_id";
  END;

-- Реакция на обновление связи между книгами и жанрами:
CREATE TRIGGER "g_has_bks_on_m2m_b_g_upd"
AFTER UPDATE
ON "m2m_books_genres"
 FOR EACH ROW
  BEGIN
   UPDATE "genres"
   SET    "g_books" = "g_books" + 1
   WHERE  "g_id" = :new."g_id";
   UPDATE "genres"
   SET    "g_books" = "g_books" - 1
   WHERE  "g_id" = :old."g_id";
  END;

-- Реакция на удаление связи между книгами и жанрами:
CREATE TRIGGER "g_has_bks_on_m2m_b_g_del"
AFTER DELETE
ON "m2m_books_genres"
 FOR EACH ROW
  BEGIN
   UPDATE "genres"
   SET    "g_books" = "g_books" - 1
   WHERE  "g_id" = :old."g_id";
  END;
```

Проверить работоспособность полученного решения можно с помощью следующих запросов (которые подробно рассмотрены в решении для MySQL).

Oracle Решение 4.1.2.b (проверка работоспособности)

```
-- Добавление двух связей к жанру «Наука» (идентификатор жанра равен 4):
INSERT INTO "m2m_books_genres"
            ("b_id", "g_id")
VALUES      (1, 4);

INSERT INTO "m2m_books_genres"
            ("b_id", "g_id")
VALUES      (2, 4);
```

Oracle Решение 4.1.2.b (проверка работоспособности)

```
-- Изменение в добавленных связях значения идентификаторов книг,
-- без изменения значения идентификаторов жанров:
UPDATE "m2m_books_genres"
SET    "b_id" = 3
WHERE  "b_id" = 1
   AND "g_id" = 4;

UPDATE "m2m_books_genres"
SET    "b_id" = 4
WHERE  "b_id" = 2
   AND "g_id" = 4;

-- Изменение в добавленных связях значения идентификаторов жанров,
-- без изменения значения идентификаторов книг:
UPDATE "m2m_books_genres"
SET    "g_id" = 5
WHERE  "b_id" = 3
   AND "g_id" = 4;

UPDATE "m2m_books_genres"
SET    "g_id" = 5
WHERE  "b_id" = 4
   AND "g_id" = 4;

-- Изменение в добавленных связях значения идентификаторов жанров,
-- и идентификаторов книг одновременно:
UPDATE "m2m_books_genres"
SET    "b_id" = 1,
       "g_id" = 4
WHERE  "b_id" = 3
   AND "g_id" = 5;

UPDATE "m2m_books_genres"
SET    "b_id" = 2,
       "g_id" = 4
WHERE  "b_id" = 4
   AND "g_id" = 5;

-- Удаление ранее созданных связей:
DELETE FROM "m2m_books_genres"
WHERE  "b_id" = 1
   AND "g_id" = 4;

DELETE FROM "m2m_books_genres"
WHERE  "b_id" = 2
   AND "g_id" = 4;

-- Удаление книг с идентификаторами 1 и 2 (обе эти книги одновременно
-- относятся к жанрам «Поэзия» и «Классика»):
DELETE FROM "books"
WHERE "b_id" IN (1, 2);
```

На этом решение данной задачи завершено.

Задание 4.1.2.TSK.A: доработать триггеры из решений{306}, {320} задач 4.1.2.a{305} и 4.1.2.b{305} таким образом, чтобы ни при каких манипуляциях с данными значения полей `s_books` (в таблице `subscribers`) и `g_books` (в таблице `genres`) не могли оказаться отрицательными.

Задание 4.1.2.TSK.B: модифицировать схему базы данных «Библиотека» таким образом, чтобы таблица `subscribers` хранила информацию о том, сколько раз читатель брал в библиотеке книги (этот счётчик должен инкрементироваться каждый раз, когда читателю выдаётся книга; уменьшение значения этого счётчика не предусмотрено).

Задание 4.1.2.TSK.C: оптимизировать код `UPDATE`-триггера из решения{306} задачи 4.1.2.a{305} для MS SQL Server так, чтобы выполнялась одна операция обновления таблицы `subscribers` (а не две отдельных операции, как это реализовано сейчас).

4.2. КОНТРОЛЬ ОПЕРАЦИЙ С ДАННЫМИ С ИСПОЛЬЗОВАНИЕМ ТРИГГЕРОВ

4.2.1. ПРИМЕР 33: КОНТРОЛЬ ОПЕРАЦИЙ МОДИФИКАЦИИ ДАННЫХ

Задача 4.2.1.a{332}: создать триггер, не позволяющий добавить в базу данных информацию о выдаче книги, если выполняется хотя бы одно из условий:

- дата выдачи находится в будущем;
- дата возврата находится в прошлом (только для вставки данных);
- дата возврата меньше даты выдачи.

Задача 4.2.1.b{346}: создать триггер, не позволяющий выдать книгу читателю, у которого на руках находится десять и более книг.

Задача 4.2.1.c{353}: создать триггер, не позволяющий изменять значение поля `sb_is_active` таблицы `subscriptions` со значения `N` на значение `Y`.

Ожидаемый результат 4.2.1.a.

При попытке внести в базу данных изменения, противоречащие условию задачи, операция (транзакция) должна быть отменена. Также должно быть выведено сообщение об ошибке, наглядно поясняющее суть проблемы, например:

- “Date 2038.01.12 for subscription 145 activation is in the future”.
- “Date 1983.01.12 for subscription 155 deactivation is in the past”.
- “Date 2000.01.12 for subscription 165 deactivation is less than date for its activation (2015.01.12)”.

Ожидаемый результат 4.2.1.b.

При попытке внести в базу данных изменения, противоречащие условию задачи, операция (транзакция) должна быть отменена. Также должно быть выведено сообщение об ошибке, наглядно поясняющее суть проблемы, например: “Subscriber Иванов И.И. (id = 1) already has 23 books out of 10 allowed.”

Ожидаемый результат 4.2.1.c.

При попытке внести в базу данных изменения, противоречащие условию задачи, операция (транзакция) должна быть отменена. Также должно быть выведено сообщение об ошибке, наглядно поясняющее суть проблемы, например: “It is prohibited to activate previously deactivated subscriptions (rule violated for subscriptions 34, 89, 12).”

Решение 4.2.1.a{331}.

Для решения данной задачи нам понадобятся только **INSERT**- и **UPDATE**-триггеры, т.к. в процессе удаления данных невозможно нарушить ни одно из контролируемых условий.

Код **INSERT**-триггера для MySQL выглядит следующим образом.

MySQL Решение 4.2.1.a (триггеры для таблицы subscriptions)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `subscriptions_control_ins`
4   AFTER INSERT
5   ON `subscriptions`
6    FOR EACH ROW
7     BEGIN
8
9      -- Блокировка выдач книг с датой выдачи в будущем.
10     IF NEW.`sb_start` > CURDATE()
11      THEN
12       SET @msg = CONCAT('Date ', NEW.`sb_start`, ' for subscription ',
13                  NEW.`sb_id`, ' activation is in the future.');
14       SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
15      END IF;
16
17     -- Блокировка выдач книг с датой возврата в прошлом.
18     IF NEW.`sb_finish` < CURDATE()
19      THEN
20       SET @msg = CONCAT('Date ', NEW.`sb_finish`, ' for subscription ',
21                  NEW.`sb_id`, ' deactivation is in the past.');
22       SIGNAL SQLSTATE '45002' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1002;
23      END IF;
24
```

MySQL Решение 4.2.1.a (триггеры для таблицы subscriptions) (продолжение)

```
25      -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
26      IF NEW.`sb_finish` < NEW.`sb_start`
27       THEN
28        SET @msg = CONCAT('Date ', NEW.`sb_finish`, ' for subscription ',
29                  NEW.`sb_id`,
30                  ' deactivation is less than the date for its activation (',
31                  NEW.`sb_start`, ').');
32        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1003;
33       END IF;
34
35     END;
36  $$
37
38  DELIMITER ;
```

С точки зрения функциональности в данном случае можно было бы использовать и **BEFORE**-триггер, но в случае с **AFTER**-триггером сообщение об ошибке получается более информативным, т.к. уже содержит в себе корректное значение автоинкрементируемого первичного ключа (в **BEFORE**-триггере это значение не определено, и потому в сообщении об ошибке превращается в 0).

Код в строках 25-33 в настоящий момент не нужен — две предшествующих проверки не допустят возникновения проверяемой этим кодом ситуации. Но если в будущем эти проверки будут модифицированы или убраны, код в строках 25-33 будет срабатывать.

Поскольку в MySQL триггер не может явно отменить транзакцию, мы порождаем исключительную ситуацию (строки 14, 22, 23), при возникновении которой отменяется транзакция, активировавшая срабатывание триггера.

Код **UPDATE**-триггера будет даже чуть более простым, т.к. в нём по условию задачи нет необходимости проверять, находится ли дата возврата в прошлом.

При этом код в строках 17-25 (ранее отмеченный как бесполезный для **INSERT**-триггера) здесь будет срабатывать, т.к. по условию задачи при выполнении операции обновления допускается установка в поле **sb_finish** даты из прошлого, что позволяет нарушить третье условие задачи.

MySQL Решение 4.2.1.a (триггеры для таблицы subscriptions)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `subscriptions_control_upd`
4   AFTER UPDATE
5   ON `subscriptions`
6    FOR EACH ROW
7     BEGIN
8
9       -- Блокировка выдач книг с датой выдачи в будущем.
10      IF NEW.`sb_start` > CURDATE()
11       THEN
12        SET @msg = CONCAT('Date ', NEW.`sb_start`, ' for subscription ',
13                  NEW.`sb_id`, ' activation is in the future.');
14        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
15       END IF;
16
```

MySQL Решение 4.2.1.a (триггеры для таблицы subscriptions) (продолжение)

```
    -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
    IF NEW.`sb_finish` < NEW.`sb_start`
     THEN
      SET @msg = CONCAT('Date ', NEW.`sb_finish`, ' for subscription ',
                 NEW.`sb_id`,
                 ' deactivation is less than the date for its activation (',
                 NEW.`sb_start`, ').');
      SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1003;
     END IF;

  END;
$$

DELIMITER ;
```

Проверим работоспособность полученного решения. Будем выполнять следующие запросы и следить за реакцией СУБД.

Попытаемся добавить выдачу книги с датой активации в будущем:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (500,
             1,
             1,
             '2020-01-12',
             '2020-02-12',
             'N')
```

СУБД запретит эту операцию, вернув следующее сообщение об ошибке:

```
Error Code: 1001. Date 2020-01-12 for subscription 500 activation is in the
future.
```

Попытаемся добавить выдачу книги с датой активации в будущем, при этом не указав значение первичного ключа:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
            (`sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`)
VALUES      (NULL,
             3,
             3,
             '2020-01-12',
             '2020-02-12',
             'N');
```

СУБД запретит эту операцию, вернув следующее сообщение об ошибке:

```
Error Code: 1001. Date 2020-01-12 for subscription 501 activation is in the
future.
```

Попытаемся добавить выдачу книги с датой возврата в прошлом:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (502,
             1,
             1,
             '2000-01-12',
             '2000-02-12',
             'N')
```

СУБД запретит эту операцию, вернув следующее сообщение об ошибке:

```
Error Code: 1002. Date 2000-02-12 for subscription 502 deactivation is in
the past.
```

Попытаемся добавить выдачу книги, не нарушая ни одного из условий данной задачи:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
INSERT INTO `subscriptions`
VALUES      (503,
             1,
             1,
             '2000-01-12',
             '2020-02-12',
             'N')
```

СУБД позволит выполнить вставку.

Попытаемся обновить добавленную выдачу книги так, чтобы дата её активации оказалась в будущем:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
UPDATE `subscriptions`
SET    `sb_start` = '2020-01-01'
WHERE  `sb_id` = 503
```

СУБД запретит эту операцию, вернув следующее сообщение об ошибке:

```
Error Code: 1001. Date 2020-01-01 for subscription 503 activation is in the
future.
```

Попытаемся обновить добавленную выдачу книги так, чтобы дата её активации оказалась позже даты возврата:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
UPDATE `subscriptions`
SET    `sb_start` = '2010-01-01',
       `sb_finish` = '2005-01-01'
WHERE  `sb_id` = 503
```

СУБД запретит эту операцию, вернув следующее сообщение об ошибке:

```
Error Code: 1003. Date 2005-01-01 for subscription 503 deactivation is less
than the date for its activation (2010-01-01).
```

Попытаемся обновить добавленную выдачу книги так, чтобы дата её возврата была в прошлом (для операции обновления такое разрешено):

MySQL Решение 4.2.1.a (проверка работоспособности)

```
UPDATE `subscriptions`
SET    `sb_start` = '2005-01-01',
       `sb_finish` = '2006-01-01'
WHERE  `sb_id` = 503
```

СУБД позволит выполнить обновление.

Попытаемся обновить добавленную выдачу книги, не нарушая ни одного из условий задачи:

MySQL Решение 4.2.1.a (проверка работоспособности)

```
1  UPDATE `subscriptions`
2  SET    `sb_start` = '2005-01-01',
3         `sb_finish` = '2010-01-01'
4  WHERE  `sb_id` = 503
```

СУБД позволит выполнить обновление.

Итак, решение данной задачи для MySQL готово и проверено. Переходим к решению для MS SQL Server.

В MS SQL Server нам доступны только триггеры уровня выражения, потому их внутренняя логика будет иной, нежели в MySQL. Также несколько иначе будут выглядеть возвращаемые триггерами сообщения об ошибках, т.к. в них нужно будет отразить информацию обо всех выдачах книг: в одном варианте реализации — только о «плохих», во втором — и о «плохих», и о «хороших».

Первый вариант реализации триггера полностью блокирует операцию, если хотя бы одна из записей нарушает условия задачи. Информация о таких записях аккумулируется в строковой переменной **@bad_records** (пояснение о логике работы функции **STUFF** см. в решении{74} задачи 2.2.2.a{73}).

Далее проверяется длина полученной строки: если она не равна нулю, значит, «плохие» записи обнаружены, и триггер должен отправить клиенту сообщение об ошибке (строки 23, 48, 67) и отменить операцию («откатить транзакцию») (строки 24, 49, 68).

В строках 28-32 мы получаем информацию о количестве записей в псевдотаблицах **inserted** и **deleted**, чтобы затем в строках 42-43 определить, выполнялась ли операция вставки (её признак: нет записей в **deleted**, есть записи в **inserted**).

MS SQL Решение 4.2.1.a (триггеры для таблицы subscriptions, первый вариант решения)

```
1   -- Вариант с полной блокировкой операции.
2   CREATE TRIGGER [subscriptions_control]
3   ON [subscriptions]
4   AFTER INSERT, UPDATE
5   AS
6    -- Переменные для хранения списка "плохих записей" и сообщения об ошибке.
7    DECLARE @bad_records NVARCHAR(max);
8    DECLARE @msg NVARCHAR(max);
9
10   -- Блокировка выдач книг с датой выдачи в будущем.
11   SELECT @bad_records = STUFF((SELECT ', ' + CAST([sb_id] AS NVARCHAR) +
12                          ' (' + CAST([sb_start] AS NVARCHAR) + ')'
13                 FROM  [inserted]
14                 WHERE [sb_start] > CONVERT(date, GETDATE())
15                 ORDER  BY [sb_id]
16                 FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
17                 1, 2, '');
18   IF LEN(@bad_records) > 0
19    BEGIN
20     SET @msg =
21     CONCAT('The following subscriptions'' activation dates are
22     in the future: ', @bad_records);
23     RAISERROR (@msg, 16, 1);
24     ROLLBACK TRANSACTION;
25     RETURN
26    END;
```

MS SQL Решение 4.2.1.a (триггеры для таблицы subscriptions, первый вариант решения) (продолжение)

```
-- Блокировка выдач книг с датой возврата в прошлом.
DECLARE @deleted_records INT;
DECLARE @inserted_records INT;

SELECT @deleted_records = COUNT(*) FROM [deleted];
SELECT @inserted_records = COUNT(*) FROM [inserted];

SELECT @bad_records = STUFF((SELECT ', ' + CAST([sb_id] AS NVARCHAR) +
       ' (' + CAST([sb_start] AS NVARCHAR) + ')'
              FROM  [inserted]
              WHERE [sb_finish] < CONVERT(date, GETDATE())
              ORDER  BY [sb_id]
              FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
              1, 2, '');
IF ((LEN(@bad_records) > 0) AND
    (@deleted_records = 0) AND
    (@inserted_records > 0))
 BEGIN
  SET @msg =
  CONCAT('The following subscriptions'' deactivation dates are
  in the past: ', @bad_records);
  RAISERROR (@msg, 16, 1);
  ROLLBACK TRANSACTION;
  RETURN
 END;

-- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
SELECT @bad_records = STUFF((SELECT ', ' + CAST([sb_id] AS NVARCHAR) +
       ' (act: ' + CAST([sb_start] AS NVARCHAR) + ', deact: ' +
      CAST([sb_finish] AS NVARCHAR) + ')'
              FROM  [inserted]
              WHERE [sb_finish] < [sb_start]
              ORDER  BY [sb_id]
              FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
              1, 2, '');
IF LEN(@bad_records) > 0
 BEGIN
  SET @msg =
  CONCAT('The following subscriptions'' deactivation dates are less
          than activation dates: ', @bad_records);
  RAISERROR (@msg, 16, 1);
  ROLLBACK TRANSACTION;
  RETURN
 END;
GO
```

Второй вариант реализации триггера блокирует только операции с «плохими» записями, а операции с «хорошими» записями выполняет. Также он выводит сообщение с информацией о «хороших» записях и сообщение об ошибке с информацией о «плохих».

Чтобы добиться такого эффекта мы будем использовать **`INSTEAD OF`** триггер, который активируется **вместо** соответствующей операции с данными. Если в теле такого триггера не выполнять никаких действий, то исходная операция с данными не выполнится по определению, и даже нет надобности «откатывать транзакцию». Чтобы операция выполнилась, в теле триггера нужно будет выполнить соответствующий запрос на модификацию данных в таблице.

К сожалению, в таблице **`subscriptions`** есть внешние кличи с каскадным обновлением, потому MS SQL Server не позволит создать **`INSTEAD OF UPDATE`** триггер, но продемонстрировать общую логику решения можно и на одной лишь операции вставки.

Если бы перед нами остро стояла задача применить именно этот вариант решения и для операции обновления, мы могли бы изменить свойства внешних ключей, убрав там операцию каскадного обновления и реализовав её отдельным триггером.

Обратите внимание на то, как в данном варианте решения реализована последовательность действий: поскольку мы не отменяем операцию и не выходим из тела триггера (как это реализовано в строках 24-25, 49-50, 68-69 первого варианта решения), триггер доработает до конца своего тела, что вынуждает нас одновременно учитывать все три условия задачи при принятии решения о том, «хорошая» ли нам попалась запись или «плохая».

В первую очередь мы получаем три списка «плохих» записей — по каждому из условий задачи (строки 14-40). В **`INSTEAD OF`** триггере нам неизвестны значения автоинкрементируемого первичного ключа (**`IDENTITY`**-поля **`sb_id`**), потому мы собираем только сами значения дат. Однако, для реальной вставки данных (строки 81-105) эти значения нам понадобятся: в решении{255} задачи 3.2.1.a{254} подробно объяснена логика их получения и использования.

MS SQL Решение 4.2.1.a (триггер для таблицы subscriptions, второй вариант решения)

```
1   -- Вариант с частичной блокировкой операции.
2   CREATE TRIGGER [subscriptions_control]
3   ON [subscriptions]
4   INSTEAD OF INSERT
5   AS
6    -- Переменные для хранения сообщений и списков записей.
7    DECLARE @bad_records_act_future NVARCHAR(max);
8    DECLARE @bad_records_deact_past NVARCHAR(max);
9    DECLARE @bad_records_act_greater_than_deact NVARCHAR(max);
10
11   DECLARE @good_records NVARCHAR(max);
12   DECLARE @msg NVARCHAR(max);
13
14   -- Блокировка выдач книг с датой выдачи в будущем.
15   SELECT @bad_records_act_future =
16                STUFF((SELECT ', ' + CAST([sb_start] AS NVARCHAR)
17                FROM  [inserted]
18                WHERE [sb_start] > CONVERT(date, GETDATE())
19                ORDER  BY [sb_start]
20                FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
21                1, 2, '');
22
23   -- Блокировка выдач книг с датой возврата в прошлом.
24   SELECT @bad_records_deact_past =
25                STUFF((SELECT ', ' + CAST([sb_finish] AS NVARCHAR)
26                FROM  [inserted]
27                WHERE [sb_finish] < CONVERT(date, GETDATE())
28                ORDER  BY [sb_finish]
29                FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
30                1, 2, '');
31
32   -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
33   SELECT @bad_records_act_greater_than_deact =
34                STUFF((SELECT ', (act: ' + CAST([sb_start] AS NVARCHAR) +
35                       ', deact: ' + CAST([sb_finish] AS NVARCHAR) + ')'
36                FROM  [inserted]
37                WHERE [sb_finish] < [sb_start]
38                ORDER  BY [sb_start], [sb_finish]
39                       FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
40                1, 2, '');
```

MS SQL Решение 4.2.1.a (триггер для таблицы subscriptions, второй вариант решения) (продолжение)

```
41    IF ((LEN(@bad_records_act_future) > 0) OR
42        (LEN(@bad_records_deact_past) > 0) OR
43        (LEN(@bad_records_act_greater_than_deact) > 0))
44     BEGIN
45      SET @msg = 'Some records were NOT inserted!';
46      IF (LEN(@bad_records_act_future) > 0)
47       BEGIN
48        SET @msg = CONCAT(@msg, CHAR(13), CHAR(10),
49        'The following activation dates are in the future: ',
50        @bad_records_act_future);
51       END;
52      IF (LEN(@bad_records_deact_past) > 0)
53       BEGIN
54        SET @msg = CONCAT(@msg, CHAR(13), CHAR(10),
55        'The following deactivation dates are in the past: ',
56        @bad_records_deact_past);
57       END;
58      IF (LEN(@bad_records_act_greater_than_deact) > 0)
59       BEGIN
60        SET @msg = CONCAT(@msg, CHAR(13), CHAR(10),
61        'The following deactivation dates are less than activation dates: ',
62        @bad_records_act_greater_than_deact);
63       END;
64      RAISERROR (@msg, 16, 1);
65     END;
66
67    SELECT @good_records = STUFF((SELECT ', ' +
68                  CAST([sb_start] AS NVARCHAR) + '/' +
69                  CAST([sb_finish] AS NVARCHAR)
70                  FROM  [inserted]
71                  WHERE (([sb_start] <= CONVERT(date, GETDATE())) AND
72                         ([sb_finish] >= CONVERT(date, GETDATE())) AND
73                         ([sb_finish] >= [sb_start]))
74                  ORDER  BY [sb_start], [sb_finish]
75                  FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
76                  1, 2, '');
77
78  IF LEN(@good_records) > 0
79    BEGIN
80     SET IDENTITY_INSERT [subscriptions] ON;
81     INSERT INTO [subscriptions]
82                ([sb_id],
83                 [sb_subscriber],
84                 [sb_book],
85                 [sb_start],
86                 [sb_finish],
87                 [sb_is_active])
88     SELECT ( CASE
89                 WHEN [sb_id] IS NULL
90                   OR [sb_id] = 0 THEN IDENT_CURRENT('subscriptions')
91                                       + IDENT_INCR('subscriptions')
92                                       + ROW_NUMBER() OVER (ORDER BY
93                                                            (SELECT 1))
94                                       - 1
95                                  ELSE [sb_id]
96              END ) AS [sb_id],
97            [sb_subscriber],
```

MS SQL Решение 4.2.1.a (триггер для таблицы subscriptions, второй вариант решения) (окончание)

```
98              [sb_book],
99              [sb_start],
100             [sb_finish],
101             [sb_is_active]
102     FROM    [inserted]
103     WHERE ((([sb_start] <= CONVERT(date, GETDATE())) AND
104                 ([sb_finish] >= CONVERT(date, GETDATE())) AND
105                 ([sb_finish] >= [sb_start]));
106     SET IDENTITY_INSERT [subscriptions] OFF;
107     SET @msg =
108     CONCAT('Subscriptions with the following activation/deactivation
109     dates were inserted successfully: ', @good_records);
110     PRINT @msg;
111   END;
112 GO
```

Код в строках 41-65 проверяет, были ли обнаружены «плохие» записи и, если да, формирует сообщение об ошибке, учитывающее все три условия задачи. Поскольку после отправки сообщения об ошибке (строка 64) мы не откатываем транзакцию и не выходим из тела триггера, выполнение продолжается дальше, и мы получаем возможность произвести вставку в таблицу «хороших» записей.

В строках 67-76 формируется список таких «хороших» записей, данные в которых не нарушают ни одного из условий задачи. Если такие записи были обнаружены, в строках 78-111 мы выполняем их вставку в таблицу и выводим сообщение (просто сообщение, **не** сообщение об ошибке) со списком их дат. Легко догадаться, что эта операция вставки не приводит к повторной активации **`INSTEAD OF`** триггера (иначе мы получили бы бесконечную рекурсию).

Важно! Этот (второй) вариант решения показан в учебных целях для демонстрации возможностей триггеров MS SQL Server. В реальных приложениях такая «частичная» обработка данных (когда часть записей успешно вставляется в таблицу, а часть — нет) может привести к сложнообнаружимым дефектам и иным слабопредсказуемым последствиям.

Итак, оба варианты решений готовы, осталось проверить их работоспособность. Будем выполнять запросы и отслеживать полученные сообщения.

Выполним вставку данных с явно указанными значениями первичного ключа и частью записей, удовлетворяющей условиям задачи, а частью — не удовлетворяющей:

MS SQL Решение 4.2.1.a (проверка работоспособности)

```
1   SET IDENTITY_INSERT [subscriptions] ON;
2   INSERT INTO [subscriptions]
3               ([sb_id],
4                [sb_subscriber],
5                [sb_book],
6                [sb_start],
7                [sb_finish],
8                [sb_is_active])
9   VALUES      (500,
10               3,
11               3,
12               '2020-01-12',
13               '2020-02-12',
14               'N'),
```

MS SQL Решение 4.2.1.a (проверка работоспособности) (продолжение)

```
15              (600,
16                3,
17                4,
18                '2021-01-12',
19                '2021-02-12',
20                'N'),
21              (700,
22                4,
23                4,
24                '2001-01-12',
25                '2021-02-12',
26                'N');
27  SET IDENTITY_INSERT [subscriptions] OFF;
```

Полученные сообщения:

- В первом варианте решения – сообщение об ошибке:

```
The following subscriptions' activation dates are in the future: 500
(2020-01-12), 600 (2021-01-12)
```

- Во втором варианте решения:
  - Сообщение об ошибке:

```
Some records were NOT inserted! The following activation dates are
in the future: 2020-01-12, 2021-01-12
```

  - Информационное сообщение:

```
Subscriptions with the following activation/deactivation dates were
inserted successfully: 2001-01-12/2021-02-12
```

Выполним вставку данных с без указания значений первичного ключа и с частью записей, удовлетворяющей условиям задачи, а частью — не удовлетворяющей:

MS SQL Решение 4.2.1.a (проверка работоспособности)

```
1   INSERT INTO [subscriptions]
2               ([sb_subscriber],
3                [sb_book],
4                [sb_start],
5                [sb_finish],
6                [sb_is_active])
7   VALUES      (3,
8                3,
9                '2020-01-12',
10               '2020-02-12',
11               'N'),
12              (3,
13               4,
14               '2021-01-12',
15               '2021-02-12',
16               'N'),
17              (4,
18               4,
19               '2001-01-12',
20               '2021-02-12',
21               'N')
```

Полученные сообщения:

- В первом варианте решения – сообщение об ошибке:

```
The following subscriptions' deactivation dates are in the past: 704
(2001-01-12), 705 (2002-01-12)
```

- Во втором варианте решения:
  - Сообщение об ошибке:

```
Some records were NOT inserted! The following deactivation dates are
in the past: 2001-02-12, 2002-02-12
```

  - Информационное сообщение:

```
Subscriptions with the following activation/deactivation dates were
inserted successfully: 2001-01-12/2021-02-12
```

Выполним вставку данных, удовлетворяющих всем условиям задачи:

MS SQL Решение 4.2.1.a (проверка работоспособности)

```
1   INSERT INTO [subscriptions]
2               ([sb_subscriber],
3                [sb_book],
4                [sb_start],
5                [sb_finish],
6                [sb_is_active])
7   VALUES      (4,
8                4,
9                '2001-01-12',
10               '2021-02-12',
11               'N')
```

Полученные сообщения:

- В первом варианте решения: никаких сообщений от триггера нет.
- Во втором варианте решения:
  - Сообщение об ошибке: отсутствует.
  - Информационное сообщение:

```
Subscriptions with the following activation/deactivation dates were
inserted successfully: 2001-01-12/2021-02-12
```

Выполним обновление данных с нарушением одного из условия задачи:

MS SQL Решение 4.2.1.a (проверка работоспособности)

```
1   UPDATE [subscriptions]
2   SET    [sb_finish] = '2005-01-01'
3   WHERE  [sb_start] > '2011-01-01'
```

Триггер во втором варианте решения не реагирует на операцию обновления, а от триггера в первом варианте решения поступит следующее сообщение об ошибке:

```
The following subscriptions' deactivation dates are less than activation
dates: 2 (act: 2011-01-12, deact: 2005-01-01), 3 (act: 2012-05-17, deact:
2005-01-01), 42 (act: 2012-06-11, deact: 2005-01-01), 57 (act: 2012-06-11,
deact: 2005-01-01), 61 (act: 2014-08-03, deact: 2005-01-01), 62 (act: 2014-
08-03, deact: 2005-01-01), 86 (act: 2014-08-03, deact: 2005-01-01), 91 (act:
2015-10-07, deact: 2005-01-01), 95 (act: 2015-10-07, deact: 2005-01-01), 99
(act: 2015-10-08, deact: 2005-01-01), 100 (act: 2011-01-12, deact: 2005-01-
01)
```

Выполним обновление данных с соблюдением всех условий задачи:

MS SQL Решение 4.2.1.a (проверка работоспособности)

```
UPDATE [subscriptions]
SET    [sb_finish] = '2002-01-01'
WHERE  [sb_start] = '2001-01-12';
```

Триггер во втором варианте решения не реагирует на операцию обновления, а от триггера в первом варианте решения не поступит никаких сообщений.

Итак, решение данной задачи для MS SQL Server получено и проверено. Переходим к решению для Oracle.

Поскольку Oracle не поддерживает псевдотаблицы **deleted** и **inserted**, мы реализуем ту же логику, что и в решении для MySQL, используя триггеры уровня записи.

Таким образом, отличие в решении для Oracle от решения для MySQL будет только в способе отмена операции (с одновременным выводом сообщения об ошибке): в Oracle для таких задач удобно использовать функцию **RAISE_APPLICATION_ERROR**.

В остальном решения для Oracle и MySQL полностью идентичны.

Oracle Решение 4.2.1.a (триггеры для таблицы subscriptions)

```
-- Реакция на добавление выдачи книги.
CREATE TRIGGER "subscriptions_control_ins"
AFTER INSERT
ON "subscriptions"
 FOR EACH ROW
  BEGIN

   -- Блокировка выдач книг с датой выдачи в будущем.
   IF :new."sb_start" > TRUNC(SYSDATE)
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'Date ' || :new."sb_start" ||
                             ' for subscription ' || :new."sb_id" ||
                             ' activation is in the future.');
    END IF;

   -- Блокировка выдач книг с датой возврата в прошлом.
   IF :new."sb_finish" < TRUNC(SYSDATE)
    THEN
     RAISE_APPLICATION_ERROR(-20002, 'Date ' || :new."sb_finish" ||
                             ' for subscription ' || :new."sb_id" ||
                             ' deactivation is in the past.');
    END IF;

   -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
   IF :new."sb_finish" < :new."sb_start"
    THEN
     RAISE_APPLICATION_ERROR(-20003, 'Date ' || :new."sb_finish" ||
                             ' for subscription ' || :new."sb_id" ||
                             ' deactivation is less than the date
                               for its activation (' ||
                             :new."sb_start" || ').');
    END IF;
  END;

```

Oracle Решение 4.2.1.a (триггеры для таблицы subscriptions) (продолжение)

```
35  -- Реакция на обновление выдачи книги.
36  CREATE TRIGGER "subscriptions_control_upd"
37  AFTER UPDATE
38  ON "subscriptions"
39   FOR EACH ROW
40    BEGIN
41
42     -- Блокировка выдач книг с датой выдачи в будущем.
43     IF :new."sb_start" > TRUNC(SYSDATE)
44      THEN
45       RAISE_APPLICATION_ERROR(-20001, 'Date ' || :new."sb_start" ||
46                               ' for subscription ' || :new."sb_id" ||
47                               ' activation is in the future.');
48      END IF;
49
50     -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
51     IF :new."sb_finish" < :new."sb_start"
52      THEN
53       RAISE_APPLICATION_ERROR(-20003, 'Date ' || :new."sb_finish" ||
54                               ' for subscription ' || :new."sb_id" ||
55                               ' deactivation is less than the date
56                                 for its activation (' ||
57                               :new."sb_start" || ').');
58      END IF;
59
60    END;
```

Проверить работоспособность полученного решения можно с помощью следующих запросов (их логика и ожидаемая реакция триггеров рассмотрены в решении для MySQL).

Oracle Решение 4.2.1.a (проверка работоспособности)

```
1   -- Деактивация триггера, формирующего значение автоинкрементируемого ПК:
2   ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE;
3
4   -- Добавление выдачи книги с датой активации в будущем:
5   INSERT INTO "subscriptions"
6   VALUES      (500,
7                1,
8                1,
9                TO_DATE('2020-01-12', 'YYYY-MM-DD'),
10               TO_DATE('2020-02-12', 'YYYY-MM-DD'),
11               'N');
12
13  -- Активация триггера, формирующего значение автоинкрементируемого ПК:
14  ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE;
15
16  -- Добавление выдачи книги с датой активации в будущем
17  -- (без указания значения первичного ключа):
18  INSERT INTO "subscriptions"
19              ("sb_subscriber",
20               "sb_book",
21               "sb_start",
22               "sb_finish",
23               "sb_is_active")
24  VALUES      (3,
25               3,
```

Oracle Решение 4.2.1.a (проверка работоспособности) (продолжение)

```
              TO_DATE('2020-01-12', 'YYYY-MM-DD'),
              TO_DATE('2020-02-12', 'YYYY-MM-DD'),
              'N');

-- Добавление выдачи книги с датой возврата в прошлом:
INSERT INTO "subscriptions"
            ("sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
VALUES      (1,
             1,
             TO_DATE('2000-01-12', 'YYYY-MM-DD'),
             TO_DATE('2000-02-12', 'YYYY-MM-DD'),
             'N');

-- Добавление выдачи книги без нарушения условий задачи:
INSERT INTO "subscriptions"
            ("sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
VALUES      (1,
             1,
             TO_DATE('2000-01-12', 'YYYY-MM-DD'),
             TO_DATE('2020-02-12', 'YYYY-MM-DD'),
             'N');

-- Обновление добавленной выдачи книги таким образом, чтобы дата
-- её активации оказалась в будущем:
UPDATE "subscriptions"
SET    "sb_start" = TO_DATE('2020-01-01', 'YYYY-MM-DD')
WHERE  "sb_id" = 104;
-- Обновление добавленной выдачи книги таким образом, чтобы
-- дата её активации оказалась позже даты возврата:
UPDATE "subscriptions"
SET    "sb_start" = TO_DATE('2010-01-01', 'YYYY-MM-DD'),
       "sb_finish" = TO_DATE('2005-01-01', 'YYYY-MM-DD')
WHERE  "sb_id" = 104;

-- Обновление добавленной выдачи книги таким образом, чтобы
-- дата её возврата была в прошлом (для операции обновления
-- такое разрешено):
UPDATE "subscriptions"
SET    "sb_start" = TO_DATE('2005-01-01', 'YYYY-MM-DD'),
       "sb_finish" = TO_DATE('2006-01-01', 'YYYY-MM-DD')
WHERE  "sb_id" = 104;

-- Обновление добавленной выдачи книги без нарушения условий задачи:
UPDATE "subscriptions"
SET    "sb_start" = TO_DATE('2005-01-01', 'YYYY-MM-DD'),
       "sb_finish" = TO_DATE('2010-01-01', 'YYYY-MM-DD')
WHERE  "sb_id" = 104;
```

На этом решение данной задачи завершено.

Решение 4.2.1.b{331}.

На примере этой (достаточно простой) задачи продемонстрируем типичное неправильное решение, которое часто первым приходит в голову. Оно состоит в том, чтобы в **AFTER**-триггере проверить, существуют ли читатели, для которых нарушается условие задачи (выборкой по всем читателям) и, если да, «откатить транзакцию». На достаточно объёмной базе данных такое решение может приводить к очень заметному падению производительности.

Правильное же решение состоит в том, чтобы в **BEFORE**-триггере производить проверку выполнения условия задачи только для того читателя (тех читателей — в MS SQL Server), для которого сейчас выполняется операция вставки или обновления записи в таблице subscriptions.

Итак, для всех трёх СУБД представим неправильное и правильное решение и сравним скорость их работы на базе данных «Большая библиотека».

В неправильном решении для MySQL создадим **INSERT**- и **UPDATE**-триггеры с полностью идентичным кодом, в котором будем формировать список читателей, для которых было нарушено условие задачи (недопустимость выдачи более десяти книг).

При крайней неоптимальности с точки зрения производительности у этого решения всё же есть один плюс: оно будет реагировать в том числе и на все нарушения условия задачи, которые были совершены до создания триггера. Однако, если такое поведение нас не устраивает, этот плюс превращается в минус, и представленное решение становится ещё хуже, чем мы думали.

MS SQL Решение 4.2.1.b (неправильное решение)

```
DELIMITER $$

CREATE TRIGGER `sbs_cntrl_10_books_ins_WRONG`
AFTER INSERT
ON `subscriptions`
 FOR EACH ROW
  BEGIN

    SET @msg = IFNULL((SELECT GROUP_CONCAT(
                              CONCAT('(id=', `s_id`, ', ', `s_name`,
                              ', books=', `s_books`, ')') SEPARATOR ', ')
                              AS `list`
                       FROM   (SELECT `s_id`,
                                      `s_name`,
                                      COUNT(`sb_book`) AS `s_books`
                                FROM   `subscribers`
                                       JOIN `subscriptions`
                                       ON `s_id` = `sb_subscriber`
                                WHERE  `sb_is_active` = 'Y'
                                GROUP  BY `sb_subscriber`
                                HAVING `s_books` > 10) AS `prepared_data`),
                  '');

   IF (LENGTH(@msg) > 0)
    THEN
     SET @msg = CONCAT('The following readers have more books
                        than allowed (10 allowed): ', @msg);
     SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
    END IF;

  END;
$$

```

MS SQL Решение 4.2.1.b (неправильное решение) (продолжение)

```
34  CREATE TRIGGER `sbs_cntrl_10_books_upd_WRONG`
35  AFTER UPDATE
36  ON `subscriptions`
37   FOR EACH ROW
38    BEGIN
39
40     SET @msg = IFNULL((SELECT GROUP_CONCAT(
41                               CONCAT('(id=', `s_id`, ', ', `s_name`,
42                               ', books=', `s_books`, ')') SEPARATOR ', ')
43                               AS `list`
44                        FROM   (SELECT `s_id`,
45                                       `s_name`,
46                                       COUNT(`sb_book`) AS `s_books`
47                                FROM   `subscribers`
48                                       JOIN `subscriptions`
49                                       ON `s_id` = `sb_subscriber`
50                                WHERE  `sb_is_active` = 'Y'
51                                GROUP  BY `sb_subscriber`
52                                HAVING `s_books` > 10) AS `prepared_data`),
53                '');
54
55     IF (LENGTH(@msg) > 0)
56      THEN
57       SET @msg = CONCAT('The following readers have more books
58                          than allowed (10 allowed): ', @msg);
59       SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
60      END IF;
61
62    END;
63  $$
64
65  DELIMITER ;
```

MS SQL Решение 4.2.1.b (триггеры для таблицы subscriptions)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `sbs_cntrl_10_books_ins_OK`
4   BEFORE INSERT
5   ON `subscriptions`
6    FOR EACH ROW
7     BEGIN
8
9      SET @msg = IFNULL((SELECT CONCAT('Subscriber ', `s_name`,
10                               ' (id=', `sb_subscriber`, ') already has ',
11                               `sb_books`, ' books out of 10 allowed.')
12                               AS `message`
13                        FROM   (SELECT `sb_subscriber`,
14                                       COUNT(`sb_book`) AS `sb_books`
15                                FROM   `subscriptions`
16                                WHERE  `sb_is_active` = 'Y'
17                                AND `sb_subscriber` = NEW.`sb_subscriber`
18                                GROUP  BY `sb_subscriber`
19                                HAVING `sb_books` >= 10) AS `prepared_data`
20                        JOIN `subscribers`
21                        ON `sb_subscriber` = `s_id`),
22                '');
23
```

MS SQL Решение 4.2.1.b (триггеры для таблицы subscriptions) (продолжение)

```
24      IF (LENGTH(@msg) > 0)
25       THEN
26        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
27       END IF;
28
29     END;
30   $$
31
32   CREATE TRIGGER `sbs_cntrl_10_books_upd_OK`
33   BEFORE UPDATE
34   ON `subscriptions`
35    FOR EACH ROW
36     BEGIN
37
38      SET @msg = IFNULL((SELECT CONCAT('Subscriber ', `s_name`,
39                                ' (id=', `sb_subscriber`, ') already has ',
40                                `sb_books`, ' books out of 10 allowed.')
41                                AS `message`
42                         FROM   (SELECT `sb_subscriber`,
43                                        COUNT(`sb_book`) AS `sb_books`
44                                 FROM   `subscriptions`
45                                 WHERE  `sb_is_active` = 'Y'
46                                 AND `sb_subscriber` = NEW.`sb_subscriber`
47                                 GROUP  BY `sb_subscriber`
48                                 HAVING `sb_books` >= 10) AS `prepared_data`
49                         JOIN `subscribers`
50                         ON `sb_subscriber` = `s_id`),
51                 '');
52
53      IF (LENGTH(@msg) > 0)
54       THEN
55        SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
56       END IF;
57
58     END;
59   $$
60
61   DELIMITER ;
```

В правильном решении для MySQL мы реагируем только на выдачи книг для читателя, идентификатор которого фигурирует в добавляемой/изменяемой записи. Также проверку мы выполняем **перед** тем, как изменения вступят в силу, и потому СУБД даже не придётся их отменять, если операция будет запрещена (что также сэкономит немного времени).

Исследование 4.2.1.EXP.A. Проведём исследование на базе данных «Большая библиотека», сравнив скорость работы представленных неправильного и правильного решений для MySQL.

После выполнения тысячи запросов на вставку данных, нарушающих условие задачи, медианы времени (в секундах) приняли следующие значения:

Неправильное решение	Правильное решение	Разница, раз
51.177	0.009	5686

Переходим к решению для MS SQL Server, в котором также представим два варианта — неправильный и правильный.

Неправильный вариант полностью повторяет аналогичную неверную логику решения для MySQL — мы пытаемся анализировать полный набор информации в базе данных, к тому же делаем это после выполнения операции модификации данных, заставляя СУБД отменять полученные изменения.

MS SQL Решение 4.2.1.b (неправильное решение)

```
CREATE TRIGGER [sbs_cntrl_10_books_ins_upd_WRONG]
ON [subscriptions]
AFTER INSERT, UPDATE
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 SELECT @bad_records = STUFF((SELECT ', ' + [list]
                        FROM  (SELECT CONCAT('(id=', [s_id], ', ',
                                      [s_name], ', books=',
                                      COUNT([sb_book]), ')') AS [list]
                               FROM   [subscribers]
                                      JOIN [subscriptions]
                                      ON [s_id] = [sb_subscriber]
                               WHERE  [sb_is_active] = 'Y'
                               GROUP  BY [s_id], [s_name]
                               HAVING COUNT([sb_book]) > 10)
                               AS [prepared_data]
               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
               1, 2, '');

 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('The following readers have more books
                      than allowed (10 allowed): ', @bad_records);
   RAISERROR (@msg, 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;
GO
```

Правильный вариант решения для MS SQL Server подвержен ограничениям, подробно описанным в решении{332} задачи 4.2.1.a{331} (невозможность создания **INSTEAD OF UPDATE** триггера без отключения операции каскадного обновления на внешних ключах, необходимость вычислять значение первичного ключа), потому здесь мы также ограничимся созданием только **INSERT**-триггера.

Однако, благодаря тому, что мы ограничиваем анализ выполнения условия задачи только перечнем читателей, чьи идентификаторы находятся в псевдотаблице **inserted**, а также выполняем этот анализ до реальной вставки данных, есть шанс существенно повысить скорость работы нашего триггера.

MS SQL Решение 4.2.1.b (триггер для таблицы subscriptions)

```
CREATE TRIGGER [sbs_cntrl_10_books_ins_OK]
ON [subscriptions]
INSTEAD OF INSERT
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

```

MS SQL Решение 4.2.1.b (триггер для таблицы subscriptions) (продолжение)

```
8    SELECT @bad_records = STUFF((SELECT ', ' + [list]
9                         FROM   (SELECT CONCAT('(id=', [s_id], ', ',
10                                       [s_name], ', books=',
11                                       COUNT([sb_book]), ')') AS [list]
12                                FROM   [subscribers]
13                                       JOIN [subscriptions]
14                                       ON [s_id] = [sb_subscriber]
15                                WHERE  [sb_is_active] = 'Y'
16                                       AND [sb_subscriber] IN
17                                           (SELECT [sb_subscriber]
18                                            FROM   [inserted])
19                                GROUP  BY [s_id], [s_name]
20                                HAVING COUNT([sb_book]) >= 10)
21                                AS [prepared_data]
22                  FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
23                  1, 2, '');
24
25   IF (LEN(@bad_records) > 0)
26    BEGIN
27     SET @msg = CONCAT('The following readers have more books
28                        than allowed (10 allowed): ', @bad_records);
29     RAISERROR (@msg, 16, 1);
30     ROLLBACK TRANSACTION;
31     RETURN;
32    END;
33
34   SET IDENTITY_INSERT [subscriptions] ON;
35   INSERT INTO [subscriptions]
36               ([sb_id],
37                [sb_subscriber],
38                [sb_book],
39                [sb_start],
40                [sb_finish],
41                [sb_is_active])
42   SELECT ( CASE
43                WHEN [sb_id] IS NULL
44                  OR [sb_id] = 0 THEN IDENT_CURRENT('subscriptions')
45                                      + IDENT_INCR('subscriptions')
46                                      + ROW_NUMBER() OVER (ORDER BY
47                                                           (SELECT 1))
48                                      - 1
49                                ELSE [sb_id]
50              END ) AS [sb_id],
51          [sb_subscriber],
52          [sb_book],
53          [sb_start],
54          [sb_finish],
55          [sb_is_active]
56   FROM   [inserted];
57   SET IDENTITY_INSERT [subscriptions] OFF;
58  GO
```

Исследование 4.2.1.EXP.B. Проведём исследование на базе данных «Большая библиотека», сравнив скорость работы представленных неправильного и правильного решений для MS SQL Server.

После выполнения тысячи запросов на вставку данных, нарушающих условие задачи, медианы времени (в секундах) приняли следующие значения:

Неправильное решение	Правильное решение	Разница, раз
6.598	2.746	2.4

Получилось не так внушительно, как в случае с MySQL, но всё равно достаточно, чтобы ощутить разницу в производительности неправильного и правильного вариантов решения.

Переходим к решению для Oracle, в котором также представим два варианта — неправильный и правильный. Их внутренняя логика будет полностью эквивалентна решению данной задачи для MySQL, потому сразу переходим к коду.

Oracle Решение 4.2.1.b (неправильное решение)

```
1   CREATE TRIGGER "sbs_ctr_10_bks_ins_upd_WRONG"
2   AFTER INSERT OR UPDATE
3   ON "subscriptions"
4    FOR EACH ROW
5     DECLARE
6      PRAGMA AUTONOMOUS_TRANSACTION;
7      msg NCLOB;
8     BEGIN
9      SELECT NVL((SELECT UTL_RAW.CAST_TO_NVARCHAR2
10                          (
11                           LISTAGG
12                           (
13                            UTL_RAW.CAST_TO_RAW(N'(id=' ||
14                            "sb_subscriber" || N', ' || "s_name" ||
15                            N', books=' || "s_books" || N')'),
16                            UTL_RAW.CAST_TO_RAW(N', ')
17                           )
18                           WITHIN GROUP (ORDER BY "sb_subscriber")
19                          )
20                    FROM  (SELECT "sb_subscriber",
21                                  "s_name",
22                                  COUNT("sb_book") AS "s_books"
23                           FROM   "subscribers"
24                                  JOIN "subscriptions"
25                                  ON "s_id" = "sb_subscriber"
26                           WHERE  "sb_is_active" = 'Y'
27                           GROUP  BY "sb_subscriber", "s_name"
28                           HAVING COUNT("sb_book") > 10) "prepared_data"),
29                '')
30     INTO msg FROM dual;
31
32     IF (LENGTH(msg) > 0)
33      THEN
34       RAISE_APPLICATION_ERROR(-20001, 'The following readers have
35                                        more books than allowed
36                                        (10 allowed): ' || msg);
37      END IF;
38    END;
```

Поскольку Oracle запрещает обращение из триггера к таблице **subscriptions**, в которую производится вставка, мы запускаем выполнение триггера в автономной транзакции (строка 6). Эту особенность придётся учитывать и в правильном решении, которое мы сейчас и рассмотрим.

Oracle Решение 4.2.1.b (триггеры для таблицы subscriptions)

```
CREATE TRIGGER "sbs_ctr_10_bks_ins_upd_OK"
BEFORE INSERT OR UPDATE
ON "subscriptions"
 FOR EACH ROW
  DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
   msg NCLOB;
  BEGIN
   SELECT NVL((SELECT (N'Subscriber ' || "s_name" || N' (id=' ||
                       "sb_subscriber" || N') already has ' ||
                       "sb_books" || N' books out of 10 allowed.')
                      AS "message"
                      FROM   (SELECT "sb_subscriber",
                                     COUNT("sb_book") AS "sb_books"
                              FROM   "subscriptions"
                              WHERE  "sb_is_active" = 'Y'
                              AND "sb_subscriber" = :new."sb_subscriber"
                              GROUP  BY "sb_subscriber"
                              HAVING COUNT("sb_book") >= 10)
                             "prepared_data"
                      JOIN "subscribers"
                      ON "sb_subscriber" = "s_id"),
               '')
   INTO msg FROM dual;

   IF (LENGTH(msg) > 0)
    THEN
     RAISE_APPLICATION_ERROR(-20001, msg);
    END IF;
  END;
```

Остаётся проверить разницу в скорости работы представленных решений.

Исследование 4.2.1.EXP.C. Проведём исследование на базе данных «Большая библиотека», сравнив скорость работы представленных неправильного и правильного решений для Oracle.

После выполнения тысячи запросов на вставку данных, нарушающих условие задачи, медианы времени (в секундах) приняли следующие значения:

Неправильное решение	Правильное решение	Разница, раз
99.667	4.961	20

Если свести все результаты исследований в одну таблицу, получается:

СУБД	Неправильное решение	Правильное решение	Разница, раз
MySQL	51.177	0.009	5686
MS SQL Server	6.598	2.746	2.4
Oracle	99.667	4.961	20

Также отметим, что во всех СУБД в неправильном варианте решения в процессе накопления информации обо всех читателях, для которых нарушается условие задачи, мы рискуем превысить максимально допустимый размер строки, аккумулирующей данную информацию. Так что второй (правильный) вариант решения оказывается не только быстрее, но и надёжнее.

На этом решение данной задачи завершено.

Решение 4.2.1.c{331}.

Поскольку по условию задачи запрещено только изменение значения поля **sb_is_active** с **N** на **Y** у уже существующих записей, нам понадобится только **UPDATE**-триггер.

Задачи такого типа очень просто и удобно решаются с использованием триггеров уровня записи (поддерживаются MySQL и Oracle) — мы используем ключевые слова **old** и **new** для доступа к старому и новому значению поля **sb_is_active**.

В случае с триггерами уровня выражения (только такие триггеры есть в MS SQL Server) придётся использовать запрос не объединение из псевдотаблиц **inserted** и **deleted**, чтобы найти взаимное соответствие старого и нового значений поля **sb_is_active** для каждой записи. Также придётся запретить изменение значения первичного ключа (строки 8-14 кода триггера для MS SQL Server), чтобы иметь возможность гарантированно получать соответствие старого и нового значения контролируемого поля.

В остальном логика решения этой задачи тривиальна: если происходит попытка изменения данных запрещённым по условию задачи образом, мы выводим сообщение об ошибке и «откатываем транзакцию».

Традиционно начнём с кода для MySQL.

MySQL Решение 4.2.1.c (триггер для таблицы subscriptions)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `sbs_cntrl_is_active`
4   BEFORE UPDATE
5   ON `subscriptions`
6    FOR EACH ROW
7     BEGIN
8      IF ((OLD.`sb_is_active` = 'N') AND (NEW.`sb_is_active` = 'Y'))
9       THEN
10       SET @msg = CONCAT('It is prohibited to activate previously
11                          deactivated subscriptions (rule violated
12                          for subscription with id ', NEW.`sb_id`, ').');
13       SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
14      END IF;
15    END;
16  $$
17
18  DELIMITER ;
```

В решении для MS SQL Server можно было бы пойти по более оптимальному с точки зрения производительности пути и сделать **INSTEAD OF** триггер, но в таком случае код триггера стал бы сложнее.

Поскольку в решении{346} задачи 4.2.1.b{331} мы уже рассматривали такую ситуацию, здесь мы пожертвуем производительностью ради краткости и понятности кода самого триггера.

MS SQL Решение 4.2.1.c (триггер для таблицы subscriptions)

```
1   CREATE TRIGGER [sbs_cntrl_is_active]
2   ON [subscriptions]
3   AFTER UPDATE
4   AS
5    DECLARE @bad_records NVARCHAR(max);
6    DECLARE @msg NVARCHAR(max);
7
8    IF (UPDATE([sb_id]))
9     BEGIN
10     RAISERROR ('Please, do NOT update surrogate PK
11                 on table [subscriptions]!', 16, 1);
```

MS SQL Решение 4.2.1.c (триггер для таблицы subscriptions) (продолжение)

```
    ROLLBACK TRANSACTION;
    RETURN;
   END;

  SELECT @bad_records = STUFF((SELECT ', ' +
                                      CAST([inserted].[sb_id] AS NVARCHAR)
                               FROM   [deleted]
                                      JOIN [inserted]
                                      ON [deleted].[sb_id] =
                                         [inserted].[sb_id]
                               WHERE  [deleted].[sb_is_active] = 'N'
                                      AND [inserted].[sb_is_active] = 'Y'
                FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
                1, 2, '');

  IF (LEN(@bad_records) > 0)
   BEGIN
    SET @msg = CONCAT('It is prohibited to activate previously
                       deactivated subscriptions (rule violated for
                       subscriptions with id ', @bad_records, ').');
    RAISERROR (@msg, 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
   END;
GO
```

И, наконец, представим решение для Oracle. Оно отличается от решения для MySQL только синтаксически, т.к. сама логика этих двух решений полностью идентична.

Oracle Решение 4.2.1.c (триггер для таблицы subscriptions)

```
CREATE TRIGGER "sbs_ctr_is_active"
BEFORE UPDATE
ON "subscriptions"
 FOR EACH ROW
  BEGIN
   IF ((:old."sb_is_active" = 'N') AND (:new."sb_is_active" = 'Y'))
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'It is prohibited to activate
                                      previously deactivated subscriptions
                                      (rule violated for subscription with
                                      id ' || :new."sb_id" || ').');
    END IF;
  END;
```

На этом решение данной задачи завершено. Проверить его работоспособность вы можете сами, выполняя запросы на обновление данных в таблице **subscriptions** так, чтобы либо не нарушить, либо нарушить условие задачи.

Задание 4.2.1.TSK.A: создать триггер, не позволяющий добавить в базу данных информацию о выдаче книги, если выполняется хотя бы одно из условий:

- дата выдачи или возврата приходится на воскресенье;
- читатель брал за последние полгода более 100 книг;
- промежуток времени между датами выдачи и возврата менее трёх дней.

Задание 4.2.1.TSK.B: создать триггер, не позволяющий выдать книгу читателю, у которого на руках находится пять и более книг, при условии, что суммарное время, оставшееся до возврата всех выданных ему книг, составляет менее одного месяца.

Задание 4.2.1.TSK.C: переработать решение{353} задачи 4.2.1.c{331} для MS SQL Server, изменив **AFTER**-триггер на **INSTEAD OF** триггер.

4.2.2. ПРИМЕР 34: КОНТРОЛЬ ФОРМАТА И ЗНАЧЕНИЙ ДАННЫХ

Задача 4.2.2.a{355}: создать триггер, допускающий регистрацию в библиотеке только таких читателей, имя которых содержит хотя бы два слова и одну точку.

Задача 4.2.2.b{359}: создать триггер, допускающий регистрацию в библиотеке только книг, изданных не более ста лет назад.

Ожидаемый результат 4.2.2.a:

При попытке внести в базу данных изменения, противоречащие условию задачи, операция (транзакция) должна быть отменена. Также должно быть выведено сообщение об ошибке, наглядно поясняющее суть проблемы, например: «Subscribers name should contain at least two words and one point, but the following name violates this rule: ИвановИИ».

Ожидаемый результат 4.2.2.b:

При попытке внести в базу данных изменения, противоречащие условию задачи, операция (транзакция) должна быть отменена. Также должно быть выведено сообщение об ошибке, наглядно поясняющее суть проблемы, например: «The following issuing year is more than 100 years in the past: 1812».

Решение 4.2.2.a{355}:

В решении{346} задачи 4.2.1.b{331} мы уже рассматривали подробно преимущества **`BEFORE`**- и **`INSTEAD OF`** триггеров перед **`AFTER`**-триггерами в плане производительности, потому здесь не будем повторно приводить те же самые рассуждения, а сразу переходим к сути задачи.

Самое сложное здесь — посчитать слова. И сложность эта — не столько техническая, сколько «философская»: что считать словом? Договоримся, что словом мы будем считать непрерывную последовательность из букв и знаков - (минус) и ' (апостроф). Приняв это допущение, мы можем построить универсальное решение на основе регулярных выражений (для СУБД, которые их поддерживают).

Изучение сути регулярных выражений выходит за рамки этой книги, потому — вот готовый универсальный вариант, который должен сработать в подавляющем большинстве СУБД и языков программирования (да, можно написать более оптимальный и элегантный вариант, но это повышает риск потери универсальности):

```
^[a-zA-Za-яA-ЯёЁ\'-]+([^a-zA-Za-яA-ЯёЁ\'-]+[a-zA-Za-яA-ЯёЁ\'.-]+){1,}$
```

Графическое представление этого регулярного выражения представлено на рисунке 4.a. Буквы «ё» добавлены в символьный класс как отдельный символ потому, что они не входят в диапазон букв русского алфавита.

Если по какой-то причине вы не хотите или не можете использовать регулярные выражения, есть второй способ убедиться, что в строке есть два слова (если допустить, что разделителем слов является пробел): нужно подсчитать количество пробелов в строке, у которой гарантированно удалены т.н. «концевые пробелы» в начале и конце. Если полученное число больше ноля, в строке точно есть как минимум два слова.

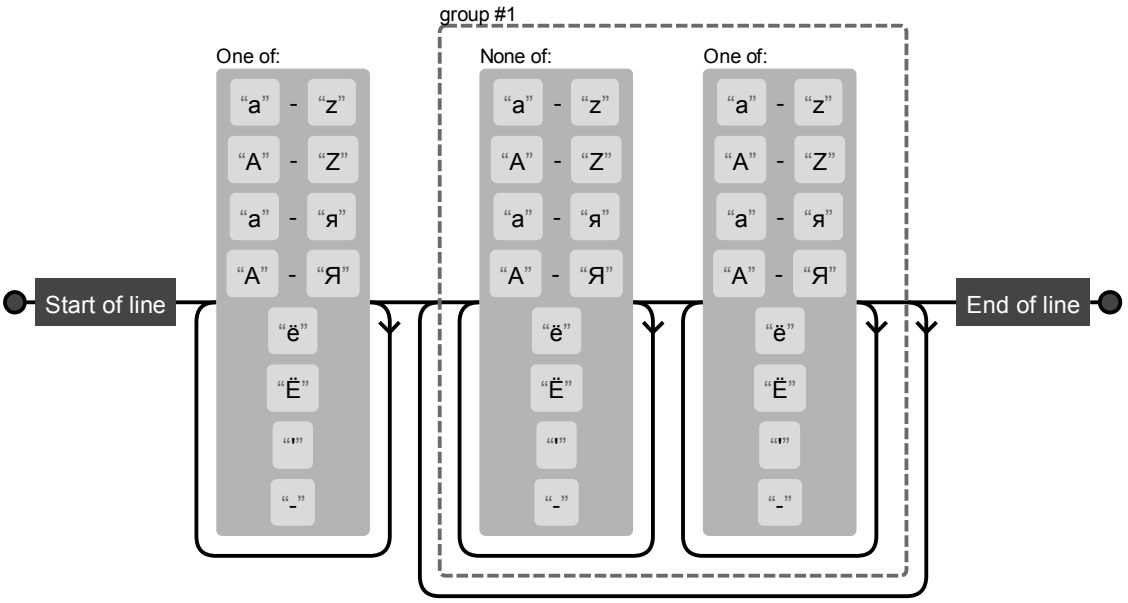


*Рисунок 4.a — **Графическое представление регулярного выражения***[14]

Традиционно начинаем с MySQL. И сразу же сталкиваемся с проблемой: эта СУБД не поддерживает мультибайтовые строки при использовании регулярных выражений[15], а потому нам придётся преобразовывать анализируемые значения к однобайтовой кодировке (например, CP1251). Для русского алфавита это безопасно, но для других языков может привести к искажениям данных и неверной работе механизма регулярных выражений.

MySQL Решение 4.2.2.a (триггеры для таблицы subscribers)

```
DELIMITER $$

CREATE TRIGGER `sbsrs_cntrl_name_ins`
BEFORE INSERT
ON `subscribers`
 FOR EACH ROW
  BEGIN
   IF ((CAST(NEW.`s_name` AS CHAR CHARACTER SET cp1251) REGEXP
        CAST('^[a-zA-Za-яA-ЯёЁ\'-]+([^a-zA-Za-яA-ЯёЁ\'-]+[a-zA-Za-яA-ЯёЁ\'.-]+)
{1,}$' AS CHAR CHARACTER SET cp1251)) = 0)
       OR (LOCATE('.', NEW.`s_name`) = 0)
    THEN
     SET @msg = CONCAT('Subscribers name should contain at
                        least two words and one point, but the following
                        name violates this rule: ', NEW.`s_name`);
     SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
    END IF;
  END;
$$
```

[14] https://regexper.com/#^[a-zA-Z%D0%B0-%D1%8F%D0%90-%D0%AF%D1%91%D0%81\%27-]%2B%28[^a-zA-Z%D0%B0-%D1%8F%D0%90-%D0%AF%D1%91%D0%81\%27-]%2B[a-zA-Z%D0%B0-%D1%8F%D0%90-%D0%AF%D1%91%D0%81\%27-]%2B%29{1%2C}%24

[15] http://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp

MySQL Решение 4.2.2.a (триггеры для таблицы subscribers) (продолжение)

```
20  CREATE TRIGGER `sbsrs_cntrl_name_upd`
21  BEFORE UPDATE
22  ON `subscribers`
23   FOR EACH ROW
24    BEGIN
25     IF ((CAST(NEW.`s_name` AS CHAR CHARACTER SET cp1251) REGEXP
26          CAST('^[a-zA-Za-яA-ЯёЁ\'-]+([^a-zA-Za-яA-ЯёЁ\'-]+[a-zA-Za-яA-ЯёЁ\'.-]+)
27  {1,}$' AS CHAR CHARACTER SET cp1251)) = 0)
28         OR (LOCATE('.', NEW.`s_name`) = 0)
29      THEN
30       SET @msg = CONCAT('Subscribers name should contain at
31                          least two words and one point, but the following
32                          name violates this rule: ', NEW.`s_name`);
33       SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
34      END IF;
35    END;
36  $$
37
38  DELIMITER ;
```

Поскольку MS SQL Server не поддерживает полноценные регулярные выражения, здесь мы используем альтернативное решение на основе подсчёта оставшихся в строке пробелов.

Вторая проблема MS SQL Server и его триггеров уровня выражения состоит в том, что в **UPDATE**-триггере мы обязаны запретить изменение первичного ключа (строки 50-56), иначе мы не сможем гарантированно корректно выполнить в коде триггера операцию обновления данных.

Третья уже знакомая нам проблема MS SQL Server связана с необходимостью вычисления значения автоинкрементируемого первичного ключа (строки 29-38) в **INSERT**-триггере (см. пояснение в решении{332} задачи 3.2.1.a{331}).

Стоит отметить, что если объём данных у нас небольшой и производительность не снижается сколь бы то ни было заметным образом от использования **AFTER**-триггеров, то решение этой задачи можно сделать гораздо более коротким, простым и универсальным (код **INSERT**- и **UPDATE**-триггера будет полностью идентичным). Убедитесь в этом самостоятельно, выполнив задание 4.2.2.TSK.B{361}.

MS SQL Решение 4.2.2.a (триггеры для таблицы subscribers)

```
1   CREATE TRIGGER [sbsrs_cntrl_name_ins]
2   ON [subscribers]
3   INSTEAD OF INSERT
4   AS
5    DECLARE @bad_records NVARCHAR(max);
6    DECLARE @msg NVARCHAR(max);
7
8    SELECT @bad_records = STUFF((SELECT ', ' + [s_name]
9                                 FROM   [inserted]
10                                WHERE
11                                 CHARINDEX(' ', LTRIM(RTRIM([s_name]))) = 0
12                               OR CHARINDEX('.', [s_name]) = 0
13                FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
14                1, 2, '');
15
16  IF (LEN(@bad_records) > 0)
17   BEGIN
18    SET @msg = CONCAT('Subscribers name should contain at least two
19                       words and one point, but the following names
20                       violate this rule: ', @bad_records);
21    RAISERROR (@msg, 16, 1);
22    ROLLBACK TRANSACTION;
23    RETURN;
24   END;
```

MS SQL Решение 4.2.2.a (триггеры для таблицы subscribers) (продолжение)

```
SET IDENTITY_INSERT [subscribers] ON;
INSERT INTO [subscribers]
            ([s_id],
             [s_name])
SELECT ( CASE
             WHEN [s_id] IS NULL
               OR [s_id] = 0 THEN IDENT_CURRENT('subscribers')
                                  + IDENT_INCR('subscribers')
                                  + ROW_NUMBER() OVER (ORDER BY
                                                       (SELECT 1))
                                   - 1
                                ELSE [s_id]
           END ) AS [s_id],
         [s_name]
FROM   [inserted];
SET IDENTITY_INSERT [subscribers] OFF;
GO

CREATE TRIGGER [sbsrs_cntrl_name_upd]
ON [subscribers]
INSTEAD OF UPDATE
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 IF (UPDATE([s_id]))
  BEGIN
   RAISERROR ('Please, do NOT update surrogate PK
               on table [subscribers]!', 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;

 SELECT @bad_records = STUFF((SELECT ', ' + [s_name]
                              FROM   [inserted]
                              WHERE
                               CHARINDEX(' ', LTRIM(RTRIM([s_name]))) = 0
                             OR CHARINDEX('.', [s_name]) = 0
               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
               1, 2, '');

 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('Subscribers name should contain at least two
                      words and one point, but the following names
                      violate this rule: ', @bad_records);
   RAISERROR (@msg, 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;

 UPDATE [subscribers]
 SET    [subscribers].[s_name] = [inserted].[s_name]
 FROM   [subscribers]
        JOIN [inserted]
          ON [subscribers].[s_id] = [inserted].[s_id];
GO
```

Переходим к решению для Oracle, которое полностью повторяет логику решения для MySQL — **BEFORE**-триггер на основе регулярного выражения и функции проверки существования подстроки в строке.

Oracle Решение 4.2.2.a (триггеры для таблицы subscribers)

```
CREATE TRIGGER "sbsrs_cntrl_name_ins_upd"
BEFORE INSERT OR UPDATE
ON "subscribers"
 FOR EACH ROW
  BEGIN
     IF  ((NOT  REGEXP_LIKE(:new."s_name",  '^[a-zA-Za-яA-ЯёЁ''-]+([^a-zA-Za-яA-
ЯёЁ''-]+[a-zA-Za-яA-ЯёЁ''.-]+){1,}$'))
      OR (INSTRC(:new."s_name", '.', 1, 1) = 0))
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'Subscribers name should contain
                                      at least two words and one point,
                                      but the following name violates
                                      this rule: ' || :new."s_name");
    END IF;
  END;
```

На этом решение данной задачи завершено. Убедиться в его корректности вы можете самостоятельно, выполнив запросы к таблице **subscribers** на вставку и обновление данных — как нарушающие условие задачи, так и не нарушающие.

Решение 4.2.2.b{355}.

Поскольку условие данной задачи во многом схоже с предыдущей, реализуем самое простое решение (для MS SQL Server используем **AFTER**-триггер) и ограничимся лишь кодом без подробных пояснений:

MySQL Решение 4.2.2.b (триггеры для таблицы books)

```
DELIMITER $$

CREATE TRIGGER `books_cntrl_year_ins`
BEFORE INSERT
ON `books`
 FOR EACH ROW
  BEGIN
   IF ((YEAR(CURDATE()) - NEW.`b_year`) > 100)
    THEN
     SET @msg = CONCAT('The following issuing year is more than
                        100 years in the past: ', NEW.`b_year`);
     SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
    END IF;
  END;
$$

```

MySQL Решение 4.2.2.b (триггеры для таблицы books) (продолжение)

```
CREATE TRIGGER `books_cntrl_year_upd`
BEFORE UPDATE
ON `books`
 FOR EACH ROW
  BEGIN
   IF ((YEAR(CURDATE()) - NEW.`b_year`) > 100)
    THEN
     SET @msg = CONCAT('The following issuing year is more than
                        100 years in the past: ', NEW.`b_year`);
     SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1001;
    END IF;
  END;
$$

DELIMITER ;
```

MS SQL Решение 4.2.2.b (триггеры для таблицы books)

```
CREATE TRIGGER [books_cntrl_year_ins_upd]
ON [books]
AFTER INSERT, UPDATE
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 SELECT @bad_records = STUFF((SELECT ', ' + CAST([b_year] AS NVARCHAR)
                              FROM   [inserted]
                              WHERE  (YEAR(GETDATE()) - [b_year]) > 100
               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
               1, 2, '');

 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('The following issuing years are more
                      than 100 years in the past: ', @bad_records);
   RAISERROR (@msg, 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;
GO
```

Oracle Решение 4.2.2.b (триггеры для таблицы books)

```
CREATE TRIGGER "books_cntrl_year_ins_upd"
BEFORE INSERT OR UPDATE
ON "books"
 FOR EACH ROW
  BEGIN
   IF ((TO_NUMBER(TO_CHAR(SYSDATE, 'YYYY')) - :new."b_year") > 100)
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'The following issuing year is
                                      more than 100 years in the past: '
                                      || :new."b_year");
   END IF;
  END;
```

На этом решение данной задачи завершено. Убедиться в его корректности вы можете самостоятельно, выполнив запросы к таблице **books** на вставку и обновление данных — как нарушающие условие задачи, так и не нарушающие.

Задание 4.2.2.TSK.A: модифицировать решение{355} задачи 4.2.2.a{355} для MySQL и Oracle так, чтобы в коде триггеров не использовались регулярные выражения.

Задание 4.2.2.TSK.B: переписать решение{355} задачи 4.2.2.a{355} для MS SQL Server с использованием **`AFTER`**-триггеров.

Задание 4.2.2.TSK.C: переписать регулярные выражения в решении{355} задачи 4.2.2.a{355} для MySQL и Oracle так, чтобы:

- исключить необходимость отдельной проверки наличия точки в имени читателя;
- допустить нахождение точки в любом из слов (а не только во втором и далее, как это сделано сейчас).

Задание 4.2.2.TSK.D: создать триггер, допускающий регистрацию в библиотеке только таких автором, имя которых не содержит никаких символов кроме букв, цифр, знаков - (минус), ' (апостроф) и пробелов (не допускается два и более идущих подряд пробела).

4.2.3. ПРИМЕР 35: ПРОЗРАЧНОЕ ИСПРАВЛЕНИЕ ОШИБОК В ДАННЫХ

Задача 4.2.3.a{361}: создать триггер, проверяющий наличие точки в конце имени читателя и добавляющий таковую при её отсутствии.

Задача 4.2.3.b{365}: создать триггер, меняющий дату возврата книги на «два месяца с момента выдачи», если дата возврата меньше даты выдачи или находится в прошлом.

Ожидаемый результат 4.2.3.a.

При выполнении операции модификации данных, нарушающей условие задачи, данные должны быть откорректированы в соответствии с условием задачи. Также должно быть выведено информационное сообщение в стиле: «Value [Иванов И.И] was automatically changed to [Иванов И.И.]».

Ожидаемый результат 4.2.3.b.

При выполнении операции модификации данных, нарушающей условие задачи, данные должны быть откорректированы в соответствии с условием задачи. Также должно быть выведено информационное сообщение в стиле: «Return date 2020.01.01 is less than giveaway date 2021.01.01. Return date changed to 2020.03.01.»

Решение 4.2.3.a{361}.

Приведённый ниже код триггеров для MySQL отличается от множества ранее рассмотренных подобных примеров только значением **`SQLSTATE`**: значения, начинающиеся с 01, сообщают не об ошибках, а о предупреждениях. И это — единственный способ сообщить из MySQL-триггера о выполненном преобразовании некорректного значения поля в корректное.

К сожалению, до версии 5.7.2[16] MySQL хранит предупреждения не по всей текущей сессии, а только по последнему выражению, потому в нашем случае (с использованием MySQL 5.6) мы не увидим этих сообщений. Но сами триггеры при этом корректно выполняют свою работу и корректируют некорректные данные.

MySQL Решение 4.2.3.a (триггеры для таблицы subscribers)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `sbsrs_name_lp_ins`
4   BEFORE INSERT
5   ON `subscribers`
6    FOR EACH ROW
7     BEGIN
8      IF (SUBSTRING(NEW.`s_name`, -1) <> '.')
9       THEN
10       SET @new_value = CONCAT(NEW.`s_name`, '.');
11       SET @msg = CONCAT('Value [', NEW.`s_name`, '] was automatically
12                          changed to [', @new_value ,']');
13       SET NEW.`s_name` = @new_value;
14       SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1000;
15      END IF;
16    END;
17  $$
18  CREATE TRIGGER `sbsrs_name_lp_upd`
19  BEFORE UPDATE
20  ON `subscribers`
21   FOR EACH ROW
22    BEGIN
23     IF (SUBSTRING(NEW.`s_name`, -1) <> '.')
24      THEN
25       SET @new_value = CONCAT(NEW.`s_name`, '.');
26       SET @msg = CONCAT('Value [', NEW.`s_name`, '] was automatically
27                          changed to [', @new_value ,']');
28       SET NEW.`s_name` = @new_value;
29       SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1000;
30      END IF;
31    END;
32  $$
33
34  DELIMITER ;
```

В MS SQL Server так же просто и красиво подменить некорректное значение корректным не получится, т.к. в этой СУБД нет триггеров уровня записи.

Потому нам придётся создавать **INSTEAD OF** триггеры (со всеми вытекающими отсюда проблемами и ограничениями в виде «ручной» генерации значения автоинкрементируемого первичного ключа на вставке данных и запрета изменения значения первичного ключа на обновлении данных).

Зато в MS SQL Server можно очень легко и удобно передавать сообщения из триггеров. Для демонстрации этих возможностей в представленном ниже коде реализовано два варианта поведения:

- В строках 18 и 68: с помощью конструкции PRINT (которая просто выводит текстовое сообщение в консоль).
- В строках 19 и 69: с помощью функции RAISERROR (которая при таких параметрах (см. документацию[17]) генерирует сообщение, не приводящее к остановке операции; к тому же мы не «откатываем» транзакцию в теле триггера).

16 http://dev.mysql.com/doc/refman/5.7/en/show-warnings.html

17 https://msdn.microsoft.com/en-us/library/ms178592.aspx

MySQL Решение 4.2.3.a (триггеры для таблицы subscribers)

```
CREATE TRIGGER [sbsrs_name_lp_ins]
ON [subscribers]
INSTEAD OF INSERT
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 SELECT @bad_records = STUFF((SELECT ', ' + '[' + [s_name] + '] -> [' +
                                    [s_name] + '.]'
                              FROM   [inserted]
                              WHERE  RIGHT([s_name], 1) <> '.'
              FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
              1, 2, '');

 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('Some values were changed: ', @bad_records);
   PRINT @msg;
   RAISERROR (@msg, 16, 0);
  END;
 SET IDENTITY_INSERT [subscribers] ON;
 INSERT INTO [subscribers]
            ([s_id],
             [s_name])
 SELECT ( CASE
             WHEN [s_id] IS NULL
               OR [s_id] = 0 THEN IDENT_CURRENT('subscribers')
                                  + IDENT_INCR('subscribers')
                                  + ROW_NUMBER() OVER (ORDER BY
                                                       (SELECT 1))
                                  - 1
                               ELSE [s_id]
          END ) AS [s_id],
        ( CASE
             WHEN RIGHT([s_name], 1) <> '.'
              THEN CONCAT([s_name], '.')
              ELSE [s_name]
          END ) AS [s_name]
 FROM   [inserted];
 SET IDENTITY_INSERT [subscribers] OFF;
GO

CREATE TRIGGER [sbsrs_name_lp_upd]
ON [subscribers]
INSTEAD OF UPDATE
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 IF (UPDATE([s_id]))
  BEGIN
   RAISERROR ('Please, do NOT update surrogate PK
               on table [subscribers]!', 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;

```

MySQL Решение 4.2.3.a (триггеры для таблицы subscribers) (продолжение)

```
SELECT @bad_records = STUFF((SELECT ', ' + '[' + [s_name] + '] -> [' +
                                    [s_name] + '.]'
                             FROM   [inserted]
                             WHERE  RIGHT([s_name], 1) <> '.'
               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
               1, 2, '');

IF (LEN(@bad_records) > 0)
 BEGIN
  SET @msg = CONCAT('Some values were changed: ', @bad_records);
  PRINT @msg;
  RAISERROR (@msg, 16, 0);
 END;

UPDATE [subscribers]
SET    [subscribers].[s_name] =
         ( CASE
            WHEN RIGHT([inserted].[s_name], 1) <> '.'
            THEN CONCAT([inserted].[s_name], '.')
            ELSE [inserted].[s_name]
           END )
FROM   [subscribers]
       JOIN [inserted]
         ON [subscribers].[s_id] = [inserted].[s_id];
GO
```

Решение для Oracle традиционно повторяет логику решения для MySQL с той лишь разницей, что здесь мы можем только вывести текстовое сообщение (сгенерировать предупреждение, не влияющее на выполнение операции, на текущий момент в Oracle нельзя).

Чтобы увидеть выводимое сообщение необходимо предварительно выполнить команду **`SET SERVEROUTPUT ON`**, включающую показ таких данных.

Oracle Решение 4.2.3.a (триггеры для таблицы subscribers)

```
CREATE TRIGGER "sbsrs_name_lp_ins_upd"
BEFORE INSERT OR UPDATE
ON "subscribers"
 FOR EACH ROW
 DECLARE
  new_value NVARCHAR2(150);
  BEGIN
   IF (SUBSTR(:new."s_name", -1) <> '.')
    THEN
     new_value := CONCAT(:new."s_name", '.');
     DBMS_OUTPUT.PUT_LINE('Value [' || :new."s_name" ||
     '] was automatically changed to [' || new_value || ']');
     :new."s_name" := new_value;
   END IF;
  END;
```

На этом решение данной задачи завершено. Убедиться в его корректности вы можете самостоятельно, выполнив запросы к таблице **`subscribers`** на вставку и обновление данных — как нарушающие условие задачи, так и не нарушающие.

Решение 4.2.3.b{361}.

Общая логика решения данной задачи повторяет логику решения{361} задачи 4.2.3.a{361}, и достойным отдельного упоминания здесь можно считать только следующее:

- в MySQL и MS SQL Server приходится делать два отдельных триггера (MySQL не умеет «объединять» объявление триггеров для нескольких операций, а в MS SQL Server различается внутреннее поведение **INSERT**- и **UPDATE**-триггера), в то время как в Oracle получается компактный одинаковый код, актуальный для обеих операций;
- в MS SQL Server без доработки модели БД можно создать только **INSTEAD OF INSERT** триггер, а для создания **INSTEAD OF UPDATE** триггера придётся отключить каскадное обновление на внешних ключах таблицы **subscriptions** и реализовать соответствующие операции по обеспечению ссылочной целостности в самом триггере;
- отображаемые триггерами сообщения об автоматической корректировке значения поля **sb_finish** в представленной ниже реализации могут содержать начальное и конечное значение даты в разных форматах (чтобы этого избежать, нужно явно приводить оба значения к одинаковому формату даты).

В остальном все представленные далее триггеры содержат лишь вариации на тему рассмотренных ранее операций.

MySQL Решение 4.2.3.b (триггеры для таблицы subscriptions)

```
1   DELIMITER $$
2
3   CREATE TRIGGER `sbscs_date_tm_ins`
4   BEFORE INSERT
5   ON `subscriptions`
6    FOR EACH ROW
7     BEGIN
8      IF (NEW.`sb_finish` < NEW.`sb_start`) OR (NEW.`sb_finish` < CURDATE())
9       THEN
10       SET @new_value = DATE_ADD(CURDATE(), INTERVAL 2 MONTH);
11       SET @msg = CONCAT('Value [', NEW.`sb_finish`, '] was automatically
12                         changed to [', @new_value ,']');
13       SET NEW.`sb_finish` = @new_value;
14       SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1000;
15      END IF;
16    END;
17  $$
18
19  CREATE TRIGGER `sbscs_date_tm_upd`
20  BEFORE UPDATE
21  ON `subscriptions`
22   FOR EACH ROW
23    BEGIN
24     IF (NEW.`sb_finish` < NEW.`sb_start`) OR (NEW.`sb_finish` < CURDATE())
25      THEN
26       SET @new_value = DATE_ADD(CURDATE(), INTERVAL 2 MONTH);
27       SET @msg = CONCAT('Value [', NEW.`sb_finish`, '] was automatically
28                         changed to [', @new_value ,']');
29       SET NEW.`sb_finish` = @new_value;
30       SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = @msg, MYSQL_ERRNO = 1000;
31      END IF;
32    END;
33  $$
34
35  DELIMITER ;
```

MS SQL Решение 4.2.3.b (триггеры для таблицы subscriptions)

```
CREATE TRIGGER [sbscs_date_tm_ins]
ON [subscriptions]
INSTEAD OF INSERT
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 SELECT @bad_records =
        STUFF((SELECT ', ' + '[' + CAST([sb_finish] AS NVARCHAR) +
                     '] -> [' + FORMAT(DATEADD(month, 2, GETDATE()),
                                       'yyyy-MM-dd') + ']'
                FROM   [inserted]
                WHERE  ([sb_finish] < [sb_start]) OR
                             ([sb_finish] < GETDATE())
        FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
        1, 2, '');

 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('Some values were changed: ', @bad_records);
   PRINT @msg;
   RAISERROR (@msg, 16, 0);
  END;
 SET IDENTITY_INSERT [subscriptions] ON;
 INSERT INTO [subscriptions]
            ([sb_id],
             [sb_subscriber],
             [sb_book],
             [sb_start],
             [sb_finish],
             [sb_is_active])
 SELECT ( CASE
             WHEN [sb_id] IS NULL
               OR [sb_id] = 0 THEN IDENT_CURRENT('subscriptions')
                                   + IDENT_INCR('subscriptions')
                                   + ROW_NUMBER() OVER (ORDER BY
                                                        (SELECT 1))
                                   - 1
                                ELSE [sb_id]
          END ) AS [sb_id],
          [sb_subscriber],
              [sb_book],
              [sb_start],
        ( CASE
             WHEN (([sb_finish] < [sb_start]) OR
                   ([sb_finish] < GETDATE()))
             THEN DATEADD(month, 2, GETDATE())
             ELSE [sb_finish]
          END ) AS [sb_finish],
         [sb_is_active]
 FROM   [inserted];
 SET IDENTITY_INSERT [subscriptions] OFF;
GO

```

MS SQL Решение 4.2.3.b (триггеры для таблицы subscriptions) (продолжение)

```
-- Внимание! Чтобы этот триггер можно было создать, необходимо
-- отключить каскадное обновление на внешних
-- ключах таблицы [subscriptions].
-- Правда, тогда придётся доработать триггер так, чтобы с его
-- помощью обеспечивать ссылочную целостность.
CREATE TRIGGER [sbscs_date_tm_upd]
ON [subscriptions]
INSTEAD OF UPDATE
AS
 DECLARE @bad_records NVARCHAR(max);
 DECLARE @msg NVARCHAR(max);

 IF (UPDATE([sb_id]))
  BEGIN
   RAISERROR ('Please, do NOT update surrogate PK
               on table [subscriptions]!', 16, 1);
   ROLLBACK TRANSACTION;
   RETURN;
  END;

 SELECT @bad_records =
        STUFF((SELECT ', ' + '[' + CAST([sb_finish] AS NVARCHAR) +
                      '] -> [' + FORMAT(DATEADD(month, 2, GETDATE()),
                                        'yyyy-MM-dd') + ']'
               FROM   [inserted]
               WHERE  ([sb_finish] < [sb_start]) OR
                      ([sb_finish] < GETDATE())
        FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
        1, 2, '');
 IF (LEN(@bad_records) > 0)
  BEGIN
   SET @msg = CONCAT('Some values were changed: ', @bad_records);
   PRINT @msg;
   RAISERROR (@msg, 16, 0);
  END;

 UPDATE [subscriptions]
 SET    [subscriptions].[sb_subscriber] = [inserted].[sb_subscriber],
        [subscriptions].[sb_book] = [inserted].[sb_book],
            [subscriptions].[sb_start] = [inserted].[sb_start],
            [subscriptions].[sb_finish] =
          ( CASE
             WHEN (([inserted].[sb_finish] < [inserted].[sb_start]) OR
                         ([inserted].[sb_finish] < GETDATE()))
             THEN DATEADD(month, 2, GETDATE())
             ELSE [inserted].[sb_finish]
            END ),
        [subscriptions].[sb_is_active] = [inserted].[sb_is_active]
 FROM   [subscriptions]
        JOIN [inserted]
          ON [subscriptions].[sb_id] = [inserted].[sb_id];
GO
```

Oracle Решение 4.2.3.b (триггеры для таблицы subscriptions)

```
1   CREATE TRIGGER "sbscs_date_tm_ins_upd"
2   BEFORE INSERT OR UPDATE
3   ON "subscriptions"
4    FOR EACH ROW
5    DECLARE
6     new_value DATE;
7     BEGIN
8      IF (:new."sb_finish" < :new."sb_start") OR (:new."sb_finish" < SYSDATE)
9       THEN
10       new_value := ADD_MONTHS(2, SYSDATE);
11       DBMS_OUTPUT.PUT_LINE('Value [' || :new."sb_finish" ||
12       '] was automatically changed to [' ||
13       TO_CHAR(new_value, 'YYYY-MM-DD') || ']');
14       :new."sb_finish" := new_value;
15      END IF;
16    END;
```

На этом решение данной задачи завершено. Убедиться в его корректности вы можете самостоятельно, выполнив запросы к таблице **subscriptions** на вставку и обновление данных — как нарушающие условие задачи, так и не нарушающие.

Задание 4.2.3.TSK.A: доработать решение{365} задачи 4.2.3.b{361} таким образом, чтобы исходные и автоматически полученные скорректированные значения даты в сообщениях, выводимых триггерами, всегда гарантированно представляли дату в одинаковом формате (в текущей реализации формат исходного и полученного значения может различаться).

Задание 4.2.3.TSK.B: доработать решение{365} задачи 4.2.3.b{361} для MS SQL Server таким образом, чтобы получить возможность создания **INSTEAD OF UPDATE** триггера и в то же время не потерять каскадное обновление внешних ключей таблицы **subscriptions** (иными словами: отключить каскадное обновление на самих внешних ключах и реализовать его «вручную» в **INSTEAD OF UPDATE** триггере).

Задание 4.2.3.TSK.C: создать триггер, корректирующий название книги таким образом, чтобы оно удовлетворяло следующим условиям:

- не допускается наличие пробелов в начале и конце названия;
- не допускается наличие повторяющихся пробелов;
- первая буква в названии всегда должна быть заглавной.

Задание 4.2.3.TSK.D: создать триггер, меняющий дату выдачи книги на текущую, если указанная в SQL-запросе дата выдачи книги меньше текущей на полгода и более.

ИСПОЛЬЗОВАНИЕ ХРАНИМЫХ ФУНКЦИЙ И ПРОЦЕДУР

ИСПОЛЬЗОВАНИЕ ХРАНИМЫХ ФУНКЦИЙ

5.1.1. ПРИМЕР 36: ВЫБОРКА И МОДИФИКАЦИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ХРАНИМЫХ ФУНКЦИЙ

Задача 5.1.1.a{370}: создать хранимую функцию, получающую на вход даты выдачи и возврата книги и возвращающую разницу между этими датами в днях, а также слова « [OK]», « [NOTICE]», « [WARNING]», соответственно, если разница в днях составляет менее десяти, от десяти до тридцати и более тридцати дней. рмацию обо всех читателях.

Задача 5.1.1.b{372}: создать хранимую функцию, возвращающую список свободных значений автоинкрементируемых первичных ключей в указанной таблице (свободными считаются значения первичного ключа, которые отсутствуют в таблице, и при этом меньше максимального используемого значения; например, если в таблице есть первичные ключи 1, 3, 8, то свободными считаются 2, 4, 5, 6, 7).

Задача 5.1.1.c{385}: создать хранимую функцию, актуализирующую данные в таблице `books_statistics` (см. задачу 3.1.2.a{223}) и возвращающую число, показывающее изменение количества фактически имеющихся в библиотеке книг.

Ожидаемый результат 5.1.1.a.

Результат выполнения запроса, извлекающего идентификатор, даты выдачи и возврата и результат работы функции для случаев, когда книга не возвращена, должен выглядеть так:

sb_id	sb_start	sb_finish	rdns
3	2012-05-17	2012-07-17	61 WARNING
62	2014-08-03	2014-10-03	61 WARNING
86	2014-08-03	2014-09-03	31 WARNING
91	2015-10-07	2015-03-07	-214 OK
99	2015-10-08	2025-11-08	3684 WARNING

Ожидаемый результат 5.1.1.b.

Допускается три формата представления результата:

- таблица из двух колонок, в которых хранятся значения начала и конца диапазонов «свободных ключей»;
- таблица из одной колонки с перечислением всех имеющихся значений «свободных ключей»;
- строка с перечислением всех имеющихся значений «свободных ключей».

Ожидаемый результат 5.1.1.c.

При вызове функции данные в таблице `books_statistics` актуализируются, функция возвращает разницу между предыдущим и новым значением количества фактически имеющихся в библиотеке книг.

Решение 5.1.1.a[{369}].

Хранимые процедуры и функции в общем случае могут быть крайне сложными и неочевидными, а их синтаксическим особенностям в документации к каждой СУБД посвящены десятки страниц. Потому в рассматриваемых задачах осознанно сказано создать такие хранимые процедуры, которые можно с минимальными отличиями реализовать во всех трёх СУБД.

Одну «теоретическую особенность» всё же упомянем: во всех трёх представленных ниже решениях данной задачи функции объявлены как детерминированные (в MS SQL Server этот эффект достигается с помощью конструкции `WITH SCHEMABINDING`). Это означает, что каждый раз при вызове с одинаковыми параметрами на одинаковых наборах данных (для текущей задачи это не актуально, но если бы мы обращались к данным в таблицах БД, это было бы важно) такие функции будут возвращать одинаковые значения. Указание такого свойства хранимой функции позволяет СУБД более эффективно использовать механизмы внутренней оптимизации и повысить производительность.

В остальном логика решения проста: мы получаем на вход две даты, вычисляем их разницу и сохраняем результат в переменную, на основе значения этой переменной определяем текстовую часть сообщения, затем возвращаем результат конкатенации этой переменной и полученного сообщения.

И это — всё, дальше — только сам код.

Решение для MySQL:

MySQL Решение 5.1.1.a

```
DELIMITER $$
CREATE FUNCTION READ_DURATION_AND_STATUS(start_date DATE, finish_date DATE)
RETURNS VARCHAR(150) DETERMINISTIC
BEGIN
 DECLARE days INT;
 DECLARE message VARCHAR(150);
 SET days = DATEDIFF(finish_date, start_date);
 CASE
  WHEN (days<10) THEN SET message = ' OK';
  WHEN ((days>=10) AND (days<=30)) THEN SET message = ' NOTICE';
  WHEN (days>30) THEN SET message = ' WARNING';
 END CASE;
 RETURN CONCAT(days, message);
END$$

DELIMITER ;
```

Для проверки корректности полученного решения нужно использовать следующий запрос:

MySQL Решение 5.1.1.a (проверка работоспособности)

```
SELECT `sb_id`,
       `sb_start`,
       `sb_finish`,
       READ_DURATION_AND_STATUS(`sb_start`, `sb_finish`) AS `rdns`
FROM   `subscriptions`
WHERE  `sb_is_active` = 'Y'
```

Решение для MS SQL Server:

MS SQL Решение 5.1.1.a

```
CREATE FUNCTION READ_DURATION_AND_STATUS(@start_date DATE,
                                         @finish_date DATE)
RETURNS NVARCHAR(150)
WITH SCHEMABINDING
AS
BEGIN
 DECLARE @days INT;
 DECLARE @message NVARCHAR(150);

 SET @days = DATEDIFF(day, @start_date, @finish_date);
 SET @message =
  CASE
   WHEN (@days<10) THEN ' OK'
   WHEN ((@days>=10) AND (@days<=30)) THEN ' NOTICE'
   WHEN (@days>30) THEN ' WARNING'
  END;

 RETURN CONCAT(@days, @message);
END;
GO
```

Для проверки корректности полученного решения нужно использовать следующий запрос (обратите внимание на необходимость обращения к функции по её полному имени, включающему имя схемы):

MS SQL Решение 5.1.1.a (проверка работоспособности)

```
SELECT [sb_id],
       [sb_start],
       [sb_finish],
       dbo.READ_DURATION_AND_STATUS([sb_start], [sb_finish]) AS [rdns]
FROM   [subscriptions]
WHERE  [sb_is_active] = 'Y'
```

Решение для Oracle:

Oracle Решение 5.1.1.a

```
CREATE FUNCTION READ_DURATION_AND_STATUS(start_date IN DATE,
                                         finish_date IN DATE)
 RETURN NVARCHAR2
 DETERMINISTIC
 IS
 days NUMBER(10);
 message NVARCHAR2(150);
BEGIN
 SELECT (finish_date - start_date) INTO days FROM dual;
 SELECT CASE
         WHEN (days<10) THEN ' OK'
         WHEN ((days>=10) AND (days<=30)) THEN ' NOTICE'
         WHEN (days>30) THEN ' WARNING'
        END
 INTO message FROM dual;

 RETURN CONCAT(days, message);
END;
```

Для проверки корректности полученного решения нужно использовать следующий запрос:

Oracle Решение 5.1.1.a (проверка работоспособности)

```
SELECT "sb_id",
       "sb_start",
       "sb_finish",
       READ_DURATION_AND_STATUS("sb_start", "sb_finish") AS "rdns"
FROM   "subscriptions"
WHERE  "sb_is_active" = 'Y'
```

На этом решение данной задачи завершено.

Решение 5.1.1.b[369].

Логику решения данной задачи удобнее всего рассматривать на примере таблицы **subscriptions** (там есть свободные значения первичного ключа). Для большей наглядности предварительно добавим две выдачи книг — со значениями первичного ключа 200 и 202.

Посмотрим, что должно получиться.

sb_id	Свободные значения
	1 — 1
2	
3	
	4 — 41
42	
	43 — 56
57	
	58 — 60
61	
62	
	63 — 85
86	
	87 — 90
91	
	92 — 94
95	
	96 — 98
99	
100	
	101 — 199
200	
	201 — 201
202	

К сожалению, хранимые функции в MySQL на текущий момент имеют ряд жёстких ограничений, два из которых сильно влияют на решение данной задачи:

- внутри хранимых функций нельзя выполнять динамические SQL-запросы (т.е. мы не сможем передать имя таблицы, и функцию придётся жёстко привязывать к конкретной таблице БД);
- хранимые функции не могут возвращать таблицы (это ограничение частично можно обойти, возвращая множество значений в виде строки, которую затем можно будет обработать встроенными функциями MySQL, например, **`FIND_IN_SET`**).

Таким образом, результат работы функции примет следующий вид:

```
1,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
32,33,34,35,36,37,38,39,40,41,43,44,45,46,47,48,49,50,51,52,53,54,55,56,58,59,60,
63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,87,88,89,90,
92,93,94,96,97,98,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,
117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,
137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,
157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,
177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,
197,198,199,201
```

Итак, решение данной задачи для MySQL выглядит следующим образом.

MySQL Решение 5.1.1.b

```
DROP FUNCTION IF EXISTS GET_FREE_KEYS_IN_SUBSCRIPTIONS;
DELIMITER $$
CREATE FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS() RETURNS VARCHAR(21845)
BEGIN
 DECLARE start_value INT DEFAULT 0;
 DECLARE stop_value  INT DEFAULT 0;
 DECLARE done        INT DEFAULT 0;
 DECLARE free_keys_string VARCHAR(21845) DEFAULT '';
 DECLARE free_keys_cursor CURSOR FOR
  SELECT `start`,
         `stop`
  FROM   (SELECT `min_t`.`sb_id` + 1                    AS `start`,
                 (SELECT MIN(`sb_id`) - 1
                  FROM   `subscriptions` AS `x`
                  WHERE  `x`.`sb_id` > `min_t`.`sb_id`) AS `stop`
          FROM   `subscriptions` AS `min_t`
          UNION
          SELECT 1                                      AS `start`,
                 (SELECT MIN(`sb_id`) - 1
                  FROM   `subscriptions` AS `x`
                  WHERE  `sb_id` > 0)                   AS `stop`
         ) AS `data`
  WHERE  `stop` >= `start`
  ORDER  BY `start`,
            `stop`;
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

 OPEN free_keys_cursor;
 BEGIN
 read_loop: LOOP
  FETCH free_keys_cursor INTO start_value, stop_value;
  IF done THEN
   LEAVE read_loop;
  END IF;
  for_loop: LOOP
    SET free_keys_string = CONCAT(free_keys_string, start_value, ',');
    SET start_value := start_value + 1;
    IF start_value <= stop_value THEN
      ITERATE for_loop;
    END IF;
    LEAVE for_loop;
  END LOOP for_loop;
 END LOOP read_loop;
 END;

 CLOSE free_keys_cursor;
 RETURN SUBSTRING(free_keys_string, 1, CHAR_LENGTH(free_keys_string) - 1);
END;
$$
DELIMITER ;
```

Данная функция возвращает строку с максимальной длиной 21845 символов (предел для **VARCHAR** в UTF-кодировках).

В строках 5-9 объявляются переменные:

- **start_value** — начало последовательности «свободных ключей»;
- **stop_value** — конец последовательности «свободных ключей»;
- **done** — признак того, что курсор выбрал все данные из результата выполнения запроса;
- **free_keys_string** — строка для накопления и возврата результата работы функции;
- **free_keys_cursor** — курсор для построкового доступа к результатам выполнения запроса, ищущего начало и конец последовательностей «свободных ключей».

В строках 10-25 содержится запрос, являющийся «сердцем» все функции, но к нему мы вернёмся чуть позже.

В строке 26 объявляется обработчик ситуации **NOT FOUND** для курсора, объявленного в строке 9 (такой обработчик срабатывает при достижении конца набора данных, или когда данных нет вообще).

В строке 28 происходит открытие курсора, т.е. выполнение представленного в строках 10-25 запроса и предоставление доступа к строкам результата его выполнения.

В строках 30-43 находится цикл, выполняющийся для каждой строки результата выполнения запроса:

- в строке 31 очередной набор данных из результата выполнения запроса помещается в переменные **start_value** и **stop_value**;
- в строках 32-34 проверяется, удалось ли получить данные (если не удалось — происходит выход из цикла);
- в строках 35-42 находится вложенный цикл, представляющий собой SQL-реализацию классического цикла FOR: все значения от **start_value** до **stop_value** пошагово накапливаются в строке **free_keys_string**;

В строке 46 происходит закрытие курсора, и в строке 47 мы возвращаем результат работы функции (предварительно убрав последнюю запятую).

Теперь рассмотрим отдельно запрос в строках 10-25. Его суть выражена в строках 12-16:

- мы берём **значение_ключа+1** (логично, что само существующее значение ключа не может быть началом диапазона «свободных ключей», потому мы делаем предположение, что такой диапазон начинается со следующего значения) — результат помещается в поле **start**;
- затем мы ищем минимальное значение ключа, которое больше найденного при получении значения поля **start** значения ключа, и вычитаем из него единицу (логично, что существующее значение не может быть концом диапазона «свободных ключей», потому мы делаем предположение, что такой диапазон заканчивается предыдущим значением) — результат помещается в поле **stop**;
- условие в строке 23 позволяет отличить реально существующие диапазоны «свободных значений» от ложных срабатываний (в реально существующих диапазонах верхняя граница не может быть меньше нижней).

UNION-секция в строках 17-21 нужна для обнаружения «свободных диапазонов» в начале последовательности значений первичного ключа (от 1 до первого реально существующего значения).

Если выполнить этот запрос без условия **WHERE** (строка 23), мы получим следующий набор данных (серым фоном отмечены «ложные срабатывания», от которых как раз и позволяет избавиться условие **WHERE**):

start	stop
1	1
3	2
4	41
43	56
58	60
62	61
63	85
87	90
92	94
96	98
100	99
101	199
201	201
203	NULL

Получить результат рабоы функции нможно следующим запросом.

MySQL Решение 5.1.1.b (запрос для получения результата работы функции)

```
SELECT GET_FREE_KEYS_IN_SUBSCRIPTIONS()
```

Итак, решение для MySQL завершено. Переходим к MS SQL Server. Логика основного запроса, а также логика работы с курсорами и циклами только что была рассмотрена, потому здесь мы не будем повторять эти пояснения.

MS SQL Server тоже не поддерживает динамический SQL внутри хранимых функций, зато здесь мы можем возвращать из хранимой функции таблицы. Благодаря этому, в первом варианте решения мы лишь «обернём» в функцию запрос, возвращающий начальные и конечные значения диапазонов свободных ключей.

MS SQL Решение 5.1.1.b (первый вариант решения)

```
CREATE FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS()
RETURNS @free_keys TABLE
   (
    [start] INT,
    [stop]  INT
   )
AS
BEGIN
 INSERT @free_keys
  SELECT [start],
         [stop]
  FROM   (SELECT [min_t].[sb_id] + 1                    AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [x].[sb_id] > [min_t].[sb_id]) AS [stop]
          FROM   [subscriptions] AS [min_t]
          UNION
          SELECT 1                                      AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [sb_id] > 0)                   AS [stop]
         ) AS [data]
  WHERE  [stop] >= [start]
  ORDER  BY [start],
            [stop]
 RETURN
END;
GO
```

Получить результат рабоы функции нможно следующим запросом.

MS SQL Решение 5.1.1.b (запрос для получения результата работы функции)

```
SELECT * FROM GET_FREE_KEYS_IN_SUBSCRIPTIONS()
```

Во втором варианте решения мы возвратим таблицу с одним полем, которое будет содержать полный перечень свободных ключей. Это решение очень похоже на решение для MySQL с тем лишь отличием, что во вложенным цикле мы не накапливаем значения свободных ключей в строковой переменной, а помещаем их в результирующую таблицу.

MS SQL Решение 5.1.1.b (второй вариант решения)

```
CREATE FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS()
RETURNS @free_keys TABLE
(
 [key] INT
)
AS
BEGIN
 DECLARE @start_value INT;
 DECLARE @stop_value INT;
 DECLARE free_keys_cursor CURSOR LOCAL FAST_FORWARD FOR
  SELECT [start],
         [stop]
  FROM   (SELECT [min_t].[sb_id] + 1                    AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [x].[sb_id] > [min_t].[sb_id]) AS [stop]
          FROM   [subscriptions] AS [min_t]
          UNION
          SELECT 1                                      AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [sb_id] > 0)                   AS [stop]
         ) AS [data]
  WHERE  [stop] >= [start]
  ORDER  BY [start],
            [stop];

 OPEN free_keys_cursor;
 FETCH NEXT FROM free_keys_cursor INTO @start_value, @stop_value;
 WHILE @@FETCH_STATUS = 0
  BEGIN
   WHILE @start_value <= @stop_value
    BEGIN
       INSERT INTO @free_keys([key]) VALUES (@start_value);
       SET @start_value = @start_value + 1;
    END;
   FETCH NEXT FROM free_keys_cursor INTO @start_value, @stop_value;
  END;
 CLOSE free_keys_cursor;
 DEALLOCATE free_keys_cursor;

 RETURN
END;
GO
```

Для получения результата работы функции здесь, как и в первом варианте решения, необходимо выполнить запрос следующего вида.

MS SQL Решение 5.1.1.b (запрос для получения результата работы функции)

```
SELECT * FROM GET_FREE_KEYS_IN_SUBSCRIPTIONS()
```

И, наконец, реализуем третий вариант решения, который полностью повторяет логику решения для MySQL.

MS SQL Решение 5.1.1.b (третий вариант решения)

```
CREATE FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS()
RETURNS VARCHAR(max)
AS
BEGIN
 DECLARE @start_value INT;
 DECLARE @stop_value INT;
 DECLARE @free_keys_string VARCHAR(max);
 DECLARE free_keys_cursor CURSOR LOCAL FAST_FORWARD FOR
  SELECT [start],
         [stop]
  FROM   (SELECT [min_t].[sb_id] + 1                    AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [x].[sb_id] > [min_t].[sb_id]) AS [stop]
          FROM   [subscriptions] AS [min_t]
          UNION
          SELECT 1                                      AS [start],
                 (SELECT MIN([sb_id]) - 1
                  FROM   [subscriptions] AS [x]
                  WHERE  [sb_id] > 0)                   AS [stop]
         ) AS [data]
  WHERE  [stop] >= [start]
  ORDER  BY [start],
            [stop];

 OPEN free_keys_cursor;
 FETCH NEXT FROM free_keys_cursor INTO @start_value, @stop_value;
 WHILE @@FETCH_STATUS = 0
  BEGIN
   WHILE @start_value <= @stop_value
    BEGIN
       SET @free_keys_string = CONCAT(@free_keys_string,
                                      @start_value, ',');
       SET @start_value = @start_value + 1;
    END;
   FETCH NEXT FROM free_keys_cursor INTO @start_value, @stop_value;
  END;
 CLOSE free_keys_cursor;
 DEALLOCATE free_keys_cursor;

 RETURN LEFT(@free_keys_string, LEN(@free_keys_string) - 1);
END;
GO
```

Получить результат работы функции можно следующим запросом.

MS SQL Решение 5.1.1.b (запрос для получения результата работы функции)

```
SELECT dbo.GET_FREE_KEYS_IN_SUBSCRIPTIONS()
```

Итак, решение для MS SQL Server завершено. Переходим к Oracle.

Здесь решение хоть и базируется на всё том же основном запросе (рассмотренном в решении для MySQL), но технологически является более сложным. К тому же Oracle — единственная из трёх СУБД, позволяющая выполнять внутри хранимых функций динамические SQL-запросы, что позволит нам в полной мере выполнить условие исходной задачи и создать универсальную функцию, возвращающую информацию по свободным ключам любой таблицы.

В первую очередь (из соображений единообразия) реализуем хранимую функцию по аналогии с решением для MS SQL Server: функция жёстко привязана к одной таблице, а на выходе возвращает таблицу с двумя полями, хранящими начало и конец диапазонов свободных ключей.

Oracle позволяет реализовывать хранимые функции, возвращающие таблицы, двумя способами (подробности можно узнать в официальной документации или, например, здесь[18]), которые мы и рассмотрим:

- с предварительной подготовкой всех данных внутри функции и последующей передачей их в вызывающий код;
- с мгновенной передачей данных (по мере их готовности) в вызывающий код (т.н. pipelined-функции).

Первый вариант решения: функция по-прежнему ориентируется только на одну таблицу и возвращает все данные после их полной подготовки.

Oracle Решение 5.1.1.b (первый вариант решения)

```
-- Удаление старых версий типов данных:
DROP TYPE "t_tf_free_keys_table";
/
DROP TYPE "t_tf_free_keys_row";
/
-- Создание типа данных, описывающего ряд итоговой таблицы:
CREATE TYPE "t_tf_free_keys_row" AS OBJECT (
  "start" NUMBER,
  "stop"  NUMBER
);
/
-- Создание типа данных, описывающего итоговую таблицу:
CREATE TYPE "t_tf_free_keys_table" IS TABLE OF "t_tf_free_keys_row";
/

-- Сама функция:
DROP FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS;
CREATE OR REPLACE FUNCTION GET_FREE_KEYS_IN_SUBSCRIPTIONS
RETURN "t_tf_free_keys_table"
AS
 result_tab  "t_tf_free_keys_table" := "t_tf_free_keys_table"();
 CURSOR free_keys_cursor IS
  SELECT "start",
         "stop"
  FROM   (SELECT "min_t"."sb_id" + 1                    AS "start",
                 (SELECT MIN("sb_id") - 1
                  FROM   "subscriptions" "x"
                  WHERE  "x"."sb_id" > "min_t"."sb_id") AS "stop"
```

[18] http://stackoverflow.com/questions/21171349/difference-between-table-function-and-pipelined-function

Oracle Решение 5.1.1.b (первый вариант решения) (продолжение)

```
29            FROM   "subscriptions" "min_t"
30            UNION
31            SELECT 1                                          AS "start",
32                   (SELECT MIN("sb_id") - 1
33                    FROM   "subscriptions" "x"
34                    WHERE  "sb_id" > 1)                      AS "stop"
35            FROM dual
36           ) "data"
37    WHERE  "stop" >= "start"
38    ORDER  BY "start",
39              "stop";
40   BEGIN
41    FOR one_row IN free_keys_cursor
42    LOOP
43     result_tab.extend;
44     result_tab(result_tab.last) :=
45                     "t_tf_free_keys_row"(one_row."start", one_row."stop");
46    END LOOP;
47
48    RETURN result_tab;
49   END;
50  /
```

Прежде, чем приступить к рассмотрению кода самих функций, отметим, что Oracle требует создания специальных типов данных, позволяющих хранимым функциям возвращать таблицы (строки 1-14 всех представленных решений посвящены именно этой подзадаче).

Ключевые отличия реализации данной функции в Oracle (по сравнению с MS SQL Server) заключены в логике работы с курсором и формирования итогового результата.

Во-первых, здесь поддерживается вполне полноценный цикл FOR (строки 41-46).

Во-вторых, для формирования итогового набора данных нам нужно выполнять две операции: добавлять в набор данных новый элемент (строка 43) и наполнять его реальными данными (строки 44-45).

В остальном здесь нет принципиальных отличий от реализации для MS SQL Server.

Для получения результата работы функции в этом варианте решения необходимо выполнить запрос следующего вида.

Oracle Решение 5.1.1.b (запрос для получения результата работы функции)

```
1   SELECT * FROM TABLE(GET_FREE_KEYS_IN_SUBSCRIPTIONS)
```

Второй вариант решения: функция возвращает все данные после их полной подготовки, но уже принимает имя таблицы и имя её первичного ключа.

Здесь в строках 29-47 происходит формирование значения текстовой переменной, которая представляет собой SQL-запрос, сформированный с учётом полученных через параметры функции имени таблицы и её первичного ключа.

Второе незначительное отличие заключается в том, что вместо «обычного курсора» (работающего для готовых статических SQL-запросов) мы используем т.н. **`REF CURSOR`**, который может применяться для динамического SQL.

Oracle Решение 5.1.1.b (второй вариант решения)

```
1   -- Удаление старых версий типов данных:
2   DROP TYPE "t_tf_free_keys_table";
3   /
4   DROP TYPE "t_tf_free_keys_row";
5   /
```

Oracle Решение 5.1.1.b (второй вариант решения) (продолжение)

```
-- Создание типа данных, описывающего ряд итоговой таблицы:
CREATE TYPE "t_tf_free_keys_row" AS OBJECT (
  "start" NUMBER,
  "stop"  NUMBER
);
/
-- Создание типа данных, описывающего итоговую таблицу:
CREATE TYPE "t_tf_free_keys_table" IS TABLE OF "t_tf_free_keys_row";
/

-- Сама функция:
DROP FUNCTION GET_FREE_KEYS;
CREATE OR REPLACE FUNCTION GET_FREE_KEYS (table_name IN VARCHAR2,
                                          pk_name IN VARCHAR2)
RETURN "t_tf_free_keys_table"
AS
 result_tab  "t_tf_free_keys_table" := "t_tf_free_keys_table"();
 TYPE type_free_keys_cursor IS REF CURSOR;
 free_keys_cursor type_free_keys_cursor;
 start_value NUMBER;
 stop_value NUMBER;
 final_query VARCHAR2(1024);
BEGIN
  final_query := 'SELECT "start",
                         "stop"
                  FROM   (SELECT "min_t"."' || pk_name ||
                          '" + 1                                    AS "start",
                  (SELECT MIN("' || pk_name || '") - 1
                   FROM   "' || table_name || '" "x"
                   WHERE  "x"."' || pk_name || '" > "min_t"."' ||
                                                 pk_name || '") AS "stop"
           FROM   "' || table_name || '" "min_t"
           UNION
           SELECT 1                                                 AS "start",
                  (SELECT MIN("' || pk_name || '") - 1
                   FROM   "' || table_name || '" "x"
                   WHERE  "' || pk_name || '" > 0)                  AS "stop"
           FROM dual
          ) "data"
    WHERE  "stop" >= "start"
    ORDER  BY "start",
              "stop"';

  OPEN free_keys_cursor FOR final_query;
  LOOP
   FETCH free_keys_cursor INTO start_value, stop_value;
   EXIT WHEN free_keys_cursor%NOTFOUND;
   result_tab.extend;
   result_tab(result_tab.last) :=
                            "t_tf_free_keys_row"(start_value, stop_value);
  END LOOP;

  CLOSE free_keys_cursor;
  RETURN result_tab;
END;
/
```

Получить результат работы функции можно следующим запросом.

Oracle Решение 5.1.1.b (запрос для получения результата работы функции)

```
1  SELECT * FROM TABLE(GET_FREE_KEYS('subscriptions', 'sb_id'))
```

Третий вариант решения: функция всё также принимает имя таблицы и первичного ключа, но возвращает табличные данные без предварительной полной генерации (экономится память).

Здесь иначе выглядит тело цикла работы с курсором: вместо того, чтобы формировать новый элемент коллекции данных, мы извлекаем данные в переменные (строка 49), проверяем успех операции и выходим из цикла, если данных больше нет (строка 50), и передаём данные в вызывающий код, если они есть (строка 51).

Oracle Решение 5.1.1.b (третий вариант решения)

```
1   -- Удаление старых версий типов данных:
2   DROP TYPE "t_tf_free_keys_table";
3   /
4   DROP TYPE "t_tf_free_keys_row";
5   /
6   -- Создание типа данных, описывающего ряд итоговой таблицы:
7   CREATE TYPE "t_tf_free_keys_row" AS OBJECT (
8     "start" NUMBER,
9     "stop"  NUMBER
10  );
11  /
12  -- Создание типа данных, описывающего итоговую таблицу:
13  CREATE TYPE "t_tf_free_keys_table" IS TABLE OF "t_tf_free_keys_row";
14  /
15  -- Сама функция:
16  DROP FUNCTION GET_FREE_KEYS;
17  CREATE OR REPLACE FUNCTION GET_FREE_KEYS (table_name IN VARCHAR2,
18                                            pk_name IN VARCHAR2)
19  RETURN "t_tf_free_keys_table" PIPELINED
20  AS
21   TYPE type_free_keys_cursor IS REF CURSOR;
22   free_keys_cursor type_free_keys_cursor;
23   start_value NUMBER;
24   stop_value NUMBER;
25   final_query VARCHAR2(1024);
26   BEGIN
27    final_query := 'SELECT "start",
28                           "stop"
29                    FROM   (SELECT "min_t"."' || pk_name ||
30                             '" + 1                                    AS "start",
31                    (SELECT MIN("' || pk_name || '") - 1
32                     FROM   "' || table_name || '" "x"
33                     WHERE  "x"."' || pk_name || '" > "min_t"."' ||
34                                                 pk_name || '") AS "stop"
35             FROM   "' || table_name || '" "min_t"
36             UNION
37             SELECT 1                                                  AS "start",
38                    (SELECT MIN("' || pk_name || '") - 1
39                     FROM   "' || table_name || '" "x"
40                     WHERE  "' || pk_name || '" > 0)                   AS "stop"
41             FROM dual
42            ) "data"
43     WHERE  "stop" >= "start"
44     ORDER  BY "start",
45               "stop"';
46
```

Oracle Решение 5.1.1.b (третий вариант решения) (продолжение)

```
47    OPEN free_keys_cursor FOR final_query;
48    LOOP
49     FETCH free_keys_cursor INTO start_value, stop_value;
50     EXIT WHEN free_keys_cursor%NOTFOUND;
51     PIPE ROW("t_tf_free_keys_row"(start_value, stop_value));
52    END LOOP;
53
54    CLOSE free_keys_cursor;
55    RETURN;
56  END;
57  /
```

Получить результат работы функции можно следующим запросом.

Oracle Решение 5.1.1.b (запрос для получения результата работы функции)

```
1   SELECT * FROM TABLE(GET_FREE_KEYS('subscriptions', 'sb_id'))
```

Второй и третий варианты решений, даже будучи универсальными в плане возможности работы с любой таблицей, обладают одним небольшим недостатком — помимо имени обрабатываемой таблицы необходимо также передавать имя её первичного ключа. Это, конечно, мелочь, но от неё достаточно просто избавиться.

В представленном далее решении осознанно (для упрощения логики) не выполняются проверки на: существование первичного ключа, тип данных первичного ключа, простоту первичного ключа (состоит ли он из одного поля, или же является составным, т.е. состоит из нескольких полей) и т.д.

Информацию о первичном ключе таблицы можно получить запросом вида[19]:

Oracle Решение 5.1.1.b (получение информации о первичном ключе таблицы)

```
1   SELECT cols.table_name,
2          cols.column_name,
3          cols.position,
4          cons.status,
5          cons.owner
6   FROM   all_constraints cons,
7          all_cons_columns cols
8   WHERE  cols.table_name = 'TABLE_NAME'
9          AND cons.constraint_type = 'P'
10         AND cons.constraint_name = cols.constraint_name
11         AND cons.owner = cols.owner
12  ORDER  BY cols.table_name,
13            cols.position
```

Поскольку нас будет интересовать только ситуация с простым первичным ключом и только в таблице, относящейся к текущему пользователю (от имени которого установлено соединение), мы добавим два ограничения: `cons.owner = USER` — показывать только данные по текущему пользователю, `ROWNUM = 1` — показывать только первое поле (на случай составного первичного ключа).

[19] http://stackoverflow.com/questions/5353522/how-to-query-for-a-primary-key-in-oracle-11g

Четвёртый вариант решения: доработка третьего варианта с автоматическим определением имени первичного ключа таблицы.

В строках 30-37 формируется запрос, с помощью которого будет определено имя первичного ключа обрабатываемой таблицы, а в строке 42 он выполняется, а его результат помещается в переменную **pk_name**.

Если вы захотите раскомментировать отладочный вывод (строки 40, 45, 68), не забудьте выполнить команду **SET SERVEROUTPUT ON** (иначе выводимые данные не будут отображаться).

Oracle Решение 5.1.1.b (четвёртый вариант решения)

```
1   -- Удаление старых версий типов данных:
2   DROP TYPE "t_tf_free_keys_table";
3   /
4   DROP TYPE "t_tf_free_keys_row";
5   /
6   -- Создание типа данных, описывающего ряд итоговой таблицы:
7   CREATE TYPE "t_tf_free_keys_row" AS OBJECT (
8     "start" NUMBER,
9     "stop"  NUMBER
10  );
11  /
12  -- Создание типа данных, описывающего итоговую таблицу:
13  CREATE TYPE "t_tf_free_keys_table" IS TABLE OF "t_tf_free_keys_row";
14  /
15
16  -- Сама функция:
17  DROP FUNCTION GET_FREE_KEYS;
18  CREATE OR REPLACE FUNCTION GET_FREE_KEYS (table_name IN VARCHAR2)
19  RETURN "t_tf_free_keys_table" PIPELINED
20  AS
21   TYPE type_free_keys_cursor IS REF CURSOR;
22   free_keys_cursor type_free_keys_cursor;
23   start_value NUMBER;
24   stop_value NUMBER;
25   pk_name VARCHAR2(1024);
26   getpk_query VARCHAR2(1024);
27   final_query VARCHAR2(1024);
28  BEGIN
29
30    getpk_query := 'SELECT cols.column_name FROM all_constraints cons,
31    all_cons_columns cols
32    WHERE cols.table_name = ''' || table_name || '''
33    AND cons.constraint_type = ''P''
34    AND cons.constraint_name = cols.constraint_name
35    AND cons.owner = cols.owner
36    AND cons.owner = USER
37    AND ROWNUM = 1';
38
39    -- Раскомментируйте, чтобы увидеть весь запрос для получения имени ПК:
40    -- DBMS_OUTPUT.PUT_LINE(getpk_query);
41
42    EXECUTE IMMEDIATE getpk_query INTO pk_name;
43
44    -- Раскомментируйте, чтобы увидеть имя ПК:
45    -- DBMS_OUTPUT.PUT_LINE(pk_name);
46
```

Oracle Решение 5.1.1.b (четвёртый вариант решения) (продолжение)

```
47    final_query := 'SELECT "start",
48                           "stop"
49                    FROM   (SELECT "min_t"."' || pk_name ||
50                            '" + 1                                  AS "start",
51                    (SELECT MIN("' || pk_name || '") - 1
52                     FROM   "' || table_name || '" "x"
53                     WHERE  "x"."' || pk_name || '" > "min_t"."' ||
54                                                    pk_name || '") AS "stop"
55             FROM   "' || table_name || '" "min_t"
56             UNION
57             SELECT 1                                               AS "start",
58                    (SELECT MIN("' || pk_name || '") - 1
59                     FROM   "' || table_name || '" "x"
60                     WHERE  "' || pk_name || '" > 0)                AS "stop"
61             FROM dual
62            ) "data"
63     WHERE  "stop" >= "start"
64     ORDER  BY "start",
65               "stop"';
66
67    -- Раскомментируйте, чтобы увидеть финальный запрос:
68    -- DBMS_OUTPUT.PUT_LINE(final_query);
69
70    OPEN free_keys_cursor FOR final_query;
71    LOOP
72     FETCH free_keys_cursor INTO start_value, stop_value;
73     EXIT WHEN free_keys_cursor%NOTFOUND;
74     PIPE ROW("t_tf_free_keys_row"(start_value, stop_value));
75    END LOOP;
76
77    CLOSE free_keys_cursor;
78    RETURN;
79  END;
80  /
```

Получить результат работы функции можно следующим запросом.

Oracle Решение 5.1.1.b (запрос для получения результата работы функции)

```
1   SELECT * FROM TABLE(GET_FREE_KEYS('subscriptions'))
```

На этом решение данной задачи завершено.

Реализовать ещё два варианта поведения функции, в которых она возвращает таблицу из одного поля со списком ключей или строку со списком ключей вам предлагается самостоятельно в задании 5.1.1.TSK.C{388}.

Решение 5.1.1.c{369}.

Поскольку решение данной задачи базируется на решении{223} задачи 3.1.2.a{223}), нам остаётся только «обернуть» в функцию запрос на обновление данных. Исходное и конечное значение количества книг в библиотеке мы будем получать обычным **SELECT**-запросом до и после обновления данных.

Также отметим, что:

- решения данной задачи для MS SQL Server не существует, т.к. эта СУБД не позволяет выполнять операции модификации данных в хранимых функциях;

- решение данной задачи для Oracle придётся реализовывать через удаление ранее созданного материализованного представления и создания агрегирующей таблицы по аналогии с MySQL, т.к. в противном случае задача не имеет смысла — данные в материализованном представлении итак находятся в актуальном состоянии, а наша функция всегда будет возвращать значение 0.

В самой задаче 3.1.2.a[223] требовалось создать представление, хранящее в себе фактические значения агрегированных данных, но MySQL не поддерживает такие представления, что оказывается очень кстати для решения данной задачи: в случае MySQL у нас роль такого представления играет реальная таблица, данные которой мы и будем обновлять.

MySQL Решение 5.1.1.c (создание таблицы и инициализация данных)

```
-- Создание таблицы:
CREATE TABLE `books_statistics`
(
      `total` INTEGER UNSIGNED NOT NULL,
      `given` INTEGER UNSIGNED NOT NULL,
      `rest`  INTEGER UNSIGNED NOT NULL
);

-- Инициализация данных:
INSERT INTO `books_statistics`
            (`total`,
             `given`,
             `rest`)
SELECT IFNULL(`total`, 0),
       IFNULL(`given`, 0),
       IFNULL(`total` - `given`, 0) AS `rest`
FROM   (SELECT (SELECT SUM(`b_quantity`)
                FROM   `books`)              AS `total`,
               (SELECT COUNT(`sb_book`)
                FROM   `subscriptions`
                WHERE  `sb_is_active` = 'Y') AS `given`)
       AS `prepared_data`;
```

Внутри кода функции остаётся лишь написать запрос на обновление данных. Получить результат работы функции можно следующим запросом.

MySQL Решение 5.1.1.c (запрос для получения результата работы функции)

```
SELECT BOOKS_DELTA()
```

MySQL Решение 5.1.1.c (код функции)

```
DROP FUNCTION IF EXISTS BOOKS_DELTA;
DELIMITER $$
CREATE FUNCTION BOOKS_DELTA() RETURNS INT
BEGIN
 DECLARE old_books_count INT DEFAULT 0;
 DECLARE new_books_count INT DEFAULT 0;

```

MySQL Решение 5.1.1.c (код функции) (продолжение)

```
  SET old_books_count := (SELECT `total` FROM `books_statistics`);

  UPDATE `books_statistics`
  JOIN
  (SELECT IFNULL(`total`, 0) AS `total`,
          IFNULL(`given`, 0) AS `given`,
          IFNULL(`total` - `given`, 0) AS `rest`
   FROM   (SELECT (SELECT SUM(`b_quantity`)
                   FROM   `books`)              AS `total`,
                  (SELECT COUNT(`sb_book`)
                   FROM   `subscriptions`
                   WHERE  `sb_is_active` = 'Y') AS `given`)
          AS `prepared_data`) AS `src`
  SET `books_statistics`.`total` = `src`.`total`,
      `books_statistics`.`given` = `src`.`given`,
      `books_statistics`.`rest` = `src`.`rest`;

  SET new_books_count := (SELECT `total` FROM `books_statistics`);

  RETURN (new_books_count - old_books_count);
END;
$$
DELIMITER ;
```

Решение для MySQL готово, а т.к. решения для MS SQL Server не существует, сразу переходим к Oracle, где полностью воспроизведём логику решения для MySQL.

Oracle Решение 5.1.1.c (создание таблицы и инициализация данных)

```
-- Создание таблицы:
CREATE TABLE "books_statistics"
(
      "total" NUMBER(10),
      "given" NUMBER(10),
      "rest"  NUMBER(10)
);

-- Инициализация данных:
INSERT INTO "books_statistics"
           ("total",
            "given",
            "rest")
SELECT "total",
       "given",
       ("total" - "given") AS "rest"
FROM   (SELECT SUM("b_quantity") AS "total"
        FROM   "books")
  JOIN (SELECT COUNT("sb_book") AS "given"
        FROM   "subscriptions"
        WHERE  "sb_is_active" = 'Y')
  ON 1 = 1;
```

Oracle Решение 5.1.1.c (код функции)

```
1  CREATE OR REPLACE FUNCTION BOOKS_DELTA RETURN NUMBER IS
2   PRAGMA AUTONOMOUS_TRANSACTION;
3   old_books_count NUMBER;
4   new_books_count NUMBER;
5  BEGIN
6   SELECT "total" INTO old_books_count FROM "books_statistics";
7   COMMIT;
8
9   UPDATE "books_statistics"
10  SET ("total", "given", "rest") =
11  (SELECT "total",
12          "given",
13          ("total" - "given") AS "rest"
14   FROM   (SELECT SUM("b_quantity") AS "total"
15           FROM   "books")
16     JOIN (SELECT COUNT("sb_book") AS "given"
17           FROM   "subscriptions"
18           WHERE  "sb_is_active" = 'Y')
19     ON 1 = 1);
20  COMMIT;
21
22  SELECT "total" INTO new_books_count FROM "books_statistics";
23  COMMIT;
24
25  RETURN (new_books_count - old_books_count);
26 END;
```

Поскольку у Oracle есть ряд ограничений на модификацию данных из кода хранимых функций, нам нужно реализовать две идеи:

- выполнять функцию в автономной транзакции (строка 2);
- подтверждать транзакцию в теле функции после выполнения каждого запроса (строки 7, 20, 23), т.к. в противном случае возникает ситуация взаимной блокировки между запросами на чтение и обновление данных).

Получить результат выполнения функции можно запросом.

Oracle Решение 5.1.1.c (запрос для получения результата работы функции)

```
1  SELECT BOOKS_DELTA FROM dual
```

На этом решение данной задачи завершено.

Задание 5.1.1.TSK.A: создать хранимую функцию, получающую на вход идентификатор читателя и возвращающую список идентификаторов книг, которые он уже прочитал и вернул в библиотеку.

Задание 5.1.1.TSK.B: создать хранимую функцию, возвращающую список первого диапазона свободных значений автоинкрементируемых первичных ключей в указанной таблице (например, если в таблице есть первичные ключи 1, 4, 8, то первый свободный диапазон — это значения 2 и 3).

Задание 5.1.1.TSK.C: дополнить решение{372} задачи 5.1.1.b{369} для Oracle двумя вариантами реализации хранимой функции, в которых:

- функция возвращает таблицу из одного поля, в котором хранится весь список значений свободных ключей;
- функция возвращает строку, в которой через запятую перечислены все значения свободных ключей.

Задание 5.1.1.TSK.D: создать хранимую функцию, актуализирующую данные в таблице **`subscriptions_ready`** (см. задачу 3.1.2.b{223}) и возвращающую число, показывающее изменение количества выдач книг.

5.1.2. ПРИМЕР 37: КОНТРОЛЬ ОПЕРАЦИЙ С ДАННЫМИ С ИСПОЛЬЗОВАНИЕМ ХРАНИМЫХ ФУНКЦИЙ

Задача 5.1.2.a{389}: создать хранимую функцию, автоматизирующую проверку условий задачи 4.2.1.a{331}, т.е. возвращающую значение 1 (все условия выполнены) или -1, -2, -3 (если хотя бы одно условие нарушено, модуль числа соответствует номеру условия) в зависимости от того, выполняются ли следующие условия:

- дата выдачи книги не может находиться в будущем;
- дата возврата книги не может находиться в прошлом (только в случае вставки данных);
- дата возврата книги не может быть меньше даты выдачи книги.

Задача 5.1.2.b{392}: создать хранимую функцию, автоматизирующую проверку условий задачи 4.2.2.a{355}, т.е. возвращающую 1, если имя читателя содержит хотя бы два слова и одну точку, и 0, если это условие нарушено.

Ожидаемый результат 5.1.2.a.

Функция возвращает 1, если все условия задачи выполнены, и 0, если хотя бы одно условие нарушено.

Ожидаемый результат 5.1.2.a.

Функция возвращает 1, если условие задачи выполнено, и 0, если оно нарушено.

Решение 5.1.2.a{389}.

Во всех трёх СУБД логика решения данной задачи будет одинакова: мы передадим в функцию три параметра (дату выдачи книги, дату возврата книги, признак обновления данных), проверим в теле функции выполнение условий и вернём соответствующее число.

MySQL Решение 5.1.2.a (код функции)

```
1  DELIMITER $$
2  CREATE FUNCTION CHECK_SUBSCRIPTION_DATES(sb_start DATE,
3                                           sb_finish DATE,
4                                           is_insert INT)
5  RETURNS INT
6  DETERMINISTIC
7  BEGIN
8   DECLARE result INT DEFAULT 1;
9
```

MySQL Решение 5.1.2.a (код функции) (продолжение)

```
  -- Блокировка выдач книг с датой выдачи в будущем
  IF (sb_start > CURDATE())
   THEN
    SET result = -1;
   END IF;

  -- Блокировка выдач книг с датой возврата в прошлом.
  IF ((sb_finish < CURDATE()) AND (is_insert = 1))
   THEN
    SET result = -2;
   END IF;
  -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
  IF (sb_finish < sb_start)
   THEN
    SET result = -3;
   END IF;

  RETURN result;
END;
$$
DELIMITER ;
```

MySQL Решение 5.1.2.a (код для проверки работы функции)

```
SELECT CHECK_SUBSCRIPTION_DATES('2025-01-01', '2026-01-01', 1);
SELECT CHECK_SUBSCRIPTION_DATES('2025-01-01', '2026-01-01', 0);

SELECT CHECK_SUBSCRIPTION_DATES('2005-01-01', '2006-01-01', 1);
SELECT CHECK_SUBSCRIPTION_DATES('2005-01-01', '2006-01-01', 0);

SELECT CHECK_SUBSCRIPTION_DATES('2005-01-01', '2004-01-01', 1);
SELECT CHECK_SUBSCRIPTION_DATES('2005-01-01', '2004-01-01', 0);
```

MS SQL Решение 5.1.2.a (код функции)

```
CREATE FUNCTION CHECK_SUBSCRIPTION_DATES(@sb_start DATE,
                                         @sb_finish DATE,
                                         @is_insert INT)
RETURNS INT
WITH SCHEMABINDING
AS
BEGIN
 DECLARE @result INT = 1;

 -- Блокировка выдач книг с датой выдачи в будущем
 IF (@sb_start > CONVERT(date, GETDATE()))
  BEGIN
   SET @result = -1;
  END;

 -- Блокировка выдач книг с датой возврата в прошлом.
 IF ((@sb_finish < CONVERT(date, GETDATE())) AND (@is_insert = 1))
  BEGIN
   SET @result = -2;
  END;

```

MS SQL Решение 5.1.2.a (код функции) (продолжение)

```
22   -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
23   IF (@sb_finish < @sb_start)
24    BEGIN
25     SET @result = -3;
26    END;
27
28   RETURN @result;
29  END;
```

MS SQL Решение 5.1.2.a (код для проверки работы функции)

```
1   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2025-01-01', '2026-01-01', 1);
2   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2025-01-01', '2026-01-01', 0);
3
4   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2005-01-01', '2006-01-01', 1);
5   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2005-01-01', '2006-01-01', 0);
6
7   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2005-01-01', '2004-01-01', 1);
8   SELECT dbo.CHECK_SUBSCRIPTION_DATES('2005-01-01', '2004-01-01', 0);
```

Oracle Решение 5.1.2.a (код функции)

```
1   CREATE OR REPLACE FUNCTION CHECK_SUBSCRIPTION_DATES (sb_start DATE,
2                                                        sb_finish DATE,
3                                                        is_insert INT)
4   RETURN NUMBER DETERMINISTIC IS
5    result_value NUMBER := 1;
6   BEGIN
7    -- Блокировка выдач книг с датой выдачи в будущем
8    IF (sb_start > TRUNC(SYSDATE))
9     THEN
10     result_value := -1;
11   END IF;
12
13   -- Блокировка выдач книг с датой возврата в прошлом.
14   IF ((sb_finish < TRUNC(SYSDATE)) AND (is_insert = 1))
15    THEN
16     result_value := -2;
17   END IF;
18
19   -- Блокировка выдач книг с датой возврата меньшей, чем дата выдачи.
20   IF (sb_finish < sb_start)
21    THEN
22     result_value := -3;
23   END IF;
24
25   RETURN result_value;
26  END;
```

Oracle Решение 5.1.2.a (код для проверки работы функции)

```
SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2025-01-01',  'YYYY-MM-DD'),
TO_DATE('2026-01-01', 'YYYY-MM-DD'), 1) FROM dual;
SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2025-01-01',  'YYYY-MM-DD'),
TO_DATE('2026-01-01', 'YYYY-MM-DD'), 0) FROM dual;

SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2005-01-01',  'YYYY-MM-DD'),
TO_DATE('2006-01-01', 'YYYY-MM-DD'), 1) FROM dual;
SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2005-01-01',  'YYYY-MM-DD'),
TO_DATE('2006-01-01', 'YYYY-MM-DD'), 0) FROM dual;

SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2005-01-01',  'YYYY-MM-DD'),
TO_DATE('2004-01-01', 'YYYY-MM-DD'), 1) FROM dual;
SELECT  CHECK_SUBSCRIPTION_DATES(TO_DATE('2005-01-01',  'YYYY-MM-DD'),
TO_DATE('2004-01-01', 'YYYY-MM-DD'), 0) FROM dual;
```

Сам код функций примитивен и не нуждается в пояснениях, но стоит отметить отличие логики представленного здесь решения от логики решения{332} задачи 4.2.1.a{331}.

Ранее мы останавливали операцию («откатывали» транзакцию) по факту обнаружения первого же нарушения условия, потому получить сообщение о том, что дата возврата меньше даты выдачи, было крайне тяжело: для выполнения этого условия нужно, чтобы не сработали две предыдущие проверки — на то, что дата выдачи находится в будущем, и что дата возврата находится в прошлом.

Здесь же мы в решениях для всех трёх СУБД возвращаем результат лишь в самом конце тела функции, потому провал каждой следующей проверки будет аннулировать признак провала предыдущей проверки.

Если такое поведение окажется неудобным, всегда можно переписать код, возвращая результат по факту провала первой же проверки, обнаружившей нарушение условия задачи.

На этом решение данной задачи завершено.

Решение 5.1.2.b{389}.

По сравнению с предыдущей задачей здесь всё будет ещё проще: нужно просто «обернуть» в функцию две проверки, на основе результата которых возвратить 1 или 0.

Только в решении для MS SQL Server будет одна необычная деталь: мы вынуждены использовать промежуточную переменную и возвращать её значение в самом конце тела функции, т.к. данная СУБД требует, чтобы последним выражением внутри хранимой функции был оператор **RETURN**.

MySQL Решение 5.1.2.b (код функции)

```
DROP FUNCTION IF EXISTS CHECK_SUBSCRIBER_NAME;
DELIMITER $$
CREATE FUNCTION CHECK_SUBSCRIBER_NAME(subscriber_name VARCHAR(150)) RETURNS
INT DETERMINISTIC
BEGIN
   IF ((CAST(subscriber_name AS CHAR CHARACTER SET cp1251) REGEXP
            CAST('^[a-zA-Za-яA-ЯёЁ\'-]+([^a-zA-Za-яA-ЯёЁ\'-]+[a-zA-Za-яA-
ЯёЁ\'.-]+){1,}$' AS CHAR CHARACTER SET cp1251)) = 0)
      OR (LOCATE('.', subscriber_name) = 0)
    THEN
     RETURN 0;
    ELSE
     RETURN 1;
    END IF;
END;
$$
DELIMITER ;
```

MySQL Решение 5.1.2.b (код для проверки работы функции)

```
SELECT CHECK_SUBSCRIBER_NAME('Иванов');
SELECT CHECK_SUBSCRIBER_NAME('Иванов И');
SELECT CHECK_SUBSCRIBER_NAME('Иванов И.');
```

MS SQL Решение 5.1.2.b (код функции)

```
CREATE FUNCTION CHECK_SUBSCRIBER_NAME(@subscriber_name NVARCHAR(150))
RETURNS INT
WITH SCHEMABINDING
AS
BEGIN
 DECLARE @result INT = -1;

 IF ((CHARINDEX(' ', LTRIM(RTRIM(@subscriber_name))) = 0) OR
     (CHARINDEX('.', @subscriber_name) = 0))
  BEGIN
   SET @result = 0;
  END
 ELSE
  BEGIN
   SET @result = 1;
  END;

 RETURN @result;
END;
GO
```

MS SQL Решение 5.1.2.b (код для проверки работы функции)

```
SELECT dbo.CHECK_SUBSCRIBER_NAME('Иванов');
SELECT dbo.CHECK_SUBSCRIBER_NAME('Иванов И');
SELECT dbo.CHECK_SUBSCRIBER_NAME('Иванов И.');
```

Oracle Решение 5.1.2.b (код функции)

```
CREATE OR REPLACE
FUNCTION CHECK_SUBSCRIBER_NAME (subscriber_name NVARCHAR2)
RETURN NUMBER DETERMINISTIC IS
BEGIN
  IF ((NOT REGEXP_LIKE(subscriber_name, '^[a-zA-Za-яA-ЯёЁ''-]+([^a-zA-Za-
яA-ЯёЁ''-]+[a-zA-Za-яA-ЯёЁ''.-]+){1,}$'))
       OR (INSTRC(subscriber_name, '.', 1, 1) = 0))
     THEN
      RETURN 0;
     ELSE
      RETURN 1;
     END IF;
END;
```

Oracle Решение 5.1.2.b (код для проверки работы функции)

```
SELECT CHECK_SUBSCRIBER_NAME(N'Иванов') FROM dual;
SELECT CHECK_SUBSCRIBER_NAME(N'Иванов И') FROM dual;
SELECT CHECK_SUBSCRIBER_NAME(N'Иванов И.') FROM dual;
```

На этом решение данной задачи завершено.

Задание 5.1.2.TSK.A: переписать решения{332}, {355} задач 4.2.1.a{331} и 4.2.2.a{355} с использованием хранимых функций, созданных в решениях{389}, {392} задач 5.1.2.a{389} и 5.1.2.b{389} соответственно.

Задание 5.1.2.TSK.B: создать хранимую функцию, автоматизирующую проверку условий задачи 4.2.1.b{331}, т.е. возвращающую 1, если у читателя на руках сейчас менее десяти книг, и 0 в противном случае.

Задание 5.1.2.TSK.C: создать хранимую функцию, автоматизирующую проверку условий задачи 4.2.2.b{355}, т.е. возвращающую 1, если книга издана менее ста лет назад, и 0 в противном случае.

5.2. ИСПОЛЬЗОВАНИЕ ХРАНИМЫХ ПРОЦЕДУР

5.2.1. ПРИМЕР 38: ВЫПОЛНЕНИЕ ДИНАМИЧЕСКИХ ЗАПРОСОВ С ПОМОЩЬЮ ХРАНИМЫХ ПРОЦЕДУР

Задача 5.2.1.a{395}: создать хранимую процедуру, устраняющую промежутки в последовательности значений первичного ключа для заданной таблицы (например, если значения первичного ключа были равны 4, 7, 9, то после выполнения хранимой процедуры они станут равны 1, 2, 3).

Задача 5.2.1.b{402}: создать хранимую процедуру, формирующую список представлений, триггеров и внешних ключей для указанной таблицы.

Ожидаемый результат 5.2.1.a.

После выполнения хранимой процедуры, в которую первыми двумя параметрами передано имя обрабатываемой таблицы и её первичного ключа, значения первичного ключа в таблице принимают вид 1, 2, 3, … (т.е. начинаются с 1 и идут без пропусков), а сама хранимая процедура возвращает информацию о том, сколько значений первичного ключа было изменено.

Ожидаемый результат 5.2.1.b.

После выполнения хранимой процедуры, в которую первым параметром передано имя обрабатываемой таблицы, формируется и возвращается результирующая таблица вида:

object_type	object_name
foreign_key	FK_1
foreign_key	FK_2
trigger	TRG_1
trigger	TRG_2
view	VIEW_1
view	VIEW_2

Решение 5.2.1.a{394}.

Традиционно начнём решение задачи с MySQL. В отличие от хранимых функций в хранимые процедуры данной СУБД позволяют формировать и выполнять динамические SQL-запросы.

Прежде, чем начать рассмотрение кода хранимой процедуры, сделаем два важных замечания:

- выполнять динамические запросы и помещать результаты их работы в переменные можно только с использованием т.н. «сессионных переменных[20]» (имена которых начинаются со знака @);
- имена переменных, в которые помещается результат выполнения запроса, не должны совпадать с именами параметров хранимой процедуры (и, в некоторых случаях, с именами полей, возвращаемых запросом[21]).

MySQL Решение 5.2.1.a (код процедуры)

```
1   DROP PROCEDURE COMPACT_KEYS;
2   DELIMITER $$
3
4   CREATE PROCEDURE COMPACT_KEYS (IN table_name VARCHAR(150),
5                                  IN pk_name VARCHAR(150),
6                                  OUT keys_changed INT)
7   BEGIN
8    SET keys_changed = 0;
9    SELECT
10   CONCAT('Point 1. table_name = ', table_name, ', pk_name = ',
11          pk_name, ', keys_changed = ', IFNULL(keys_changed, 'NULL'));
12
13   SET @empty_key_query =
14   CONCAT('SELECT MIN(`empty_key`) AS `empty_key` INTO @empty_key_value
15          FROM   (SELECT `left`.`', pk_name, '` + 1 AS `empty_key`
16                  FROM   `', table_name, '` AS `left`
17                         LEFT OUTER JOIN `', table_name, '` AS `right`
18                                      ON `left`.`', pk_name,
19                                          '` + 1 = `right`.`', pk_name, '`
20           WHERE  `right`.`', pk_name, '` IS NULL
21           UNION
22           SELECT 1 AS `empty_key`
23           FROM   `', table_name, '`
24           WHERE  NOT EXISTS(SELECT `', pk_name, '`
25                            FROM   `', table_name, '`
26                            WHERE  `', pk_name, '` = 1)) AS `prepared_data`
27          WHERE  `empty_key` < (SELECT MAX(`', pk_name, '`)
28                                FROM   `', table_name, '`)');
29
```

20 http://stackoverflow.com/questions/1009954/mysql-variable-vs-variable-whats-the-difference

21 http://dba.stackexchange.com/questions/112285/select-into-variable-results-in-null-or-idk

MySQL Решение 5.2.1.a (код процедуры) (продолжение)

```
30   SET @max_key_query =
31       CONCAT('SELECT MAX(`', pk_name, '`)
32               INTO @max_key_value FROM `', table_name, '`');
33   SELECT CONCAT('Point 2. empty_key_query = ', @empty_key_query,
34                 'max_key_query = ', @max_key_query);
35
36   PREPARE empty_key_stmt FROM @empty_key_query;
37   PREPARE max_key_stmt FROM @max_key_query;
38
39   while_loop: LOOP
40    EXECUTE empty_key_stmt;
41    SELECT CONCAT('Point 3. @empty_key_value = ', @empty_key_value);
42
43    IF (@empty_key_value IS NULL)
44     THEN LEAVE while_loop;
45    END IF;
46
47    EXECUTE max_key_stmt;
48    SET @update_key_query =
49        CONCAT('UPDATE `', table_name, '` SET `', pk_name,
50        '` = @empty_key_value WHERE `', pk_name, '` = ', @max_key_value);
51    SELECT CONCAT('Point 4. @update_key_query = ', @update_key_query);
52
53    PREPARE update_key_stmt FROM @update_key_query;
54    EXECUTE update_key_stmt;
55    DEALLOCATE PREPARE update_key_stmt;
56
57    SET keys_changed = keys_changed + 1;
58    ITERATE while_loop;
59   END LOOP while_loop;
60
61   DEALLOCATE PREPARE max_key_stmt;
62   DEALLOCATE PREPARE empty_key_stmt;
63  END;
64  $$
65  DELIMITER ;
```

Несмотря на общую громоздкость и кажущуюся сложность, логика работы данной процедуры очень проста. Рассмотрим её детально.

Запросы в строках 10-11, 33-34, 41 и 51 представлены исключительно для отладки и наглядности, и могут быть удалены.

В строке 8 происходит инициализация значения выходного параметра, который будет накапливать в себе информацию о количестве изменений, внесённых в таблицу.

В строках 13-28 на основе переданных в хранимую процедуру имён обрабатываемой таблицы и её первичного ключа формируется текст SQL-запроса, который будет искать первое свободное значение в последовательности значений первичного ключа. Рассмотрим этот запрос отдельно (на примере таблицы `subscriptions`).

MySQL Решение 5.2.1.a (текст запроса, выполняющего поиск первого свободного значения первичного ключа)

```
1   SELECT MIN(`empty_key`) AS `empty_key`
2     FROM   (SELECT `left`.`sb_id` + 1 AS `empty_key`
3             FROM   `subscriptions` AS `left`
4                    LEFT OUTER JOIN `subscriptions` AS `right`
5                                 ON `left`.`sb_id` + 1 = `right`.`sb_id`
6             WHERE  `right`.`sb_id` IS NULL
7             UNION
8             SELECT 1 AS `empty_key`
9             FROM   `subscriptions`
10            WHERE  NOT EXISTS(SELECT `sb_id`
11                              FROM   `subscriptions`
12                              WHERE  `sb_id` = 1)) AS `prepared_data`
13    WHERE  `empty_key` < (SELECT MAX(`sb_id`)
14                          FROM   `subscriptions`)
```

Основная секция запроса в строках 2-4 ищет отсутствующие значения первичного ключа, следующие сразу за реально существующими.

Дополнительная **UNION**-секция в строках 8-12 проверяет наличие свободных значений первичного ключа в самом начале последовательности (начиная с 1).

В строке 1 выбирается минимальное значение из набора найденных.

И, наконец, в строках 13-14 применяется условие, гарантирующее, что обнаруженное свободное значение не превышает реально существующее максимальное значение первичного ключа.

Возвращаемся к коду хранимой процедуры.

В строках 30-32 формируется текст запроса, определяющего максимальное значение первичного ключа в обрабатываемой таблице — именно оно на каждом шаге цикла будет меняться на первое найденное свободное значение.

В строках 36-37 на основе текстового представления SQL-запросов, в которые подставлены имена обрабатываемой таблицы и её первичного ключа, создаются исполнимые выражения.

В строках 39-59 представлен цикл, который выполняется до тех пор, пока существует хотя бы одно свободное значение первичного ключа:

- в строке 40 происходит выполнение основного запроса, производящего поиск первого (минимального) свободного значения первичного ключа и помещающего это значение в переменную **@empty_key_value**;
- в строках 43-45 происходит проверка полученного значения на равенство **NULL** (если условие выполнено, то больше свободных значений нет, и происходит выход из цикла);
- в строке 47 выполняется запрос, помещающий текущее максимальное значение первичного ключа в переменную **@max_key_value** (мы вынуждены использовать промежуточную переменную, чтобы обойти ограничение MySQL на использование одной и той же таблицы одновременно в **UPDATE**- и **SELECT**-частях запроса);
- в строках 48-50 формируется текстовое представления запроса на обновление значения первичного ключа;
- для этого запроса в строках 53, 54, 55 соответственно формируется исполнимое выражение, происходит выполнение запроса, исполнимое выражение освобождается (т.к. на следующем шаге цикла текст запроса уже будет иным);
- в строке 57 происходит наращивание счётчика обновлённых значений первичного ключа;
- в строке 58 происходит переход на следующую итерацию цикла.

После завершения цикла нам остаётся только освободить ранее подготовленные исполнимые выражения, что и происходит в строках 61-62.

Выполнить полученную хранимую процедуру и узнать, сколько значений первичного ключа было изменено, можно следующими запросами (в первом случае будет возвращено значение 9, во втором — 0, т.к. в таблице **books** нет свободных значений первичного ключа).

MySQL Решение 5.2.1.a (код для выполнения хранимой процедуры и получения результата её работы)

```
1  CALL COMPACT_KEYS ('subscriptions', 'sb_id', @keys_changed);
2  SELECT @keys_changed;
3
4  CALL COMPACT_KEYS ('books', 'b_id', @keys_changed);
5  SELECT @keys_changed;
```

Если рассмотреть по шагам (для таблицы **subscriptions** их будет девять) работу этой хранимой процедуры, получится следующая картина:

	Исх.	**Шаг 1**	**Шаг 2**	**Шаг 3**	**Шаг 4**	**Шаг 5**	**Шаг 6**	**Шаг 7**	**Шаг 8**	**Шаг 9**
Значения первичного ключа	2	1	1	1	1	1	1	1	1	1
	3	2	2	2	2	2	2	2	2	2
	42	3	3	3	3	3	3	3	3	3
	57	42	4	4	4	4	4	4	4	4
	61	57	42	5	5	5	5	5	5	5
	62	61	57	42	6	6	6	6	6	6
	86	62	61	57	42	7	7	7	7	7
	91	86	62	61	57	42	8	8	8	8
	95	91	86	62	61	57	42	9	9	9
	99	95	91	86	62	61	57	42	10	10
	100	99	95	91	86	62	61	57	42	11
Своб.	**1**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**NULL**
Обн.	**100**	**99**	**95**	**91**	**86**	**62**	**61**	**57**	**42**	**11**

Для дополнительного изучения логики работы представленного решения рекомендуется создать рассмотренную хранимую процедуру, не удаляя отладочный вывод, и проследить на его основе логику работы и изменения значения переменных внутри процедуры.

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Общая логика решения будет очень похожа на только что рассмотренное для MySQL за исключением одной непреодолимой проблемы: MS SQL Server не позволяет обновлять **IDENTITY**-поля в таблице (включение **IDENTITY_INSERT** позволяет лишь вставлять значения в такие поля, но не обновлять их).

Мы могли бы обойти это ограничение через удаление старого ряда таблицы и вставку нового (с подменённым значением первичного ключа), но такое решение приведёт к фатальным последствиям, если на модифицируемый первичный ключ ссылаются внешние ключи других таблиц.

Никакого простого универсального решения для отключения и повторного включения **IDENTITY**-свойства у поля средствами SQL-запросов нет. Единственный более-менее доступный вариант — это выключение и повторное включение этого свойства через графический интерфейс MS SQL Server Management Studio.

Таким образом, в нашей хранимой процедуре мы будем проверять, является ли предназначенное для обработки поле **IDENTITY**-полем, и запрещать выполнение операции, если является (строки 14-22 кода хранимой процедуры).

MS SQL Решение 5.2.1.a (код процедуры)

```
CREATE PROCEDURE COMPACT_KEYS
                 @table_name NVARCHAR(150),
                 @pk_name NVARCHAR(150),
                 @keys_changed INT OUTPUT
WITH EXECUTE AS OWNER
AS
 DECLARE @empty_key_query NVARCHAR(1000) = '';
 DECLARE @max_key_query NVARCHAR(1000) = '';
 DECLARE @empty_key_value INT = NULL;
 DECLARE @max_key_value INT = NULL;
 DECLARE @update_key_query NVARCHAR(1000) = '';
 DECLARE @error_message NVARCHAR(1000) = '';

 IF (COLUMNPROPERTY(OBJECT_ID(@table_name), @pk_name, 'IsIdentity') = 1)
  BEGIN
   SET @keys_changed = -1;
   SET @error_message = CONCAT('Remove identity property for column [',
                               @pk_name,'] of table [', @table_name,
                               '] via MS SQL Server Management Studio.');
   RAISERROR (@error_message, 16, 1);
   RETURN -1;
  END;

 SET @keys_changed = 0;

 PRINT(CONCAT('Point 1. @table_name = ', @table_name, ', @pk_name = ',
        @pk_name, ', @keys_changed = ', ISNULL(@keys_changed, 'NULL')));

 SET @empty_key_query =
 CONCAT('SET @empty_k_v = (SELECT MIN([empty_key]) AS [empty_key]
        FROM   (SELECT [left].[', @pk_name, '] + 1 AS [empty_key]
                FROM   [', @table_name, '] AS [left]
                       LEFT OUTER JOIN [', @table_name, '] AS [right]
                                    ON [left].[', @pk_name,
                                        '] + 1 = [right].[', @pk_name, ']
         WHERE  [right].[', @pk_name, '] IS NULL
         UNION
         SELECT 1 AS [empty_key]
         FROM   [', @table_name, ']
         WHERE  NOT EXISTS(SELECT [', @pk_name, ']
                           FROM   [', @table_name, ']
                           WHERE  [', @pk_name, '] = 1)
        ) AS [prepared_data]
        WHERE  [empty_key] < (SELECT MAX([', @pk_name, '])
                              FROM   [', @table_name, ']))');

 SET @max_key_query =
     CONCAT('SET @max_k_v = (SELECT MAX([', @pk_name, ']) FROM [',
             @table_name, '])');

 PRINT(CONCAT('Point 2. @empty_key_query = ', @empty_key_query,
               CHAR(13), CHAR(10), '@max_key_query = ', @max_key_query));
```

MS SQL Решение 5.2.1.a (код процедуры) (продолжение)

```
53   WHILE (1 = 1)
54    BEGIN
55     EXECUTE sp_executesql @empty_key_query,
56                           N'@empty_k_v INT OUT',
57                           @empty_key_value OUTPUT;
58
59     IF (@empty_key_value IS NULL)
60      BREAK;
61
62     EXECUTE sp_executesql @max_key_query,
63                           N'@max_k_v INT OUT',
64                           @max_key_value OUTPUT;
65
66     SET @update_key_query =
67         CONCAT('UPDATE [', @table_name, '] SET [', @pk_name,
68                '] = ', @empty_key_value, ' WHERE [', @pk_name, '] = ',
69                @max_key_value);
70
71     PRINT(CONCAT('Point 3. @update_key_query = ', @update_key_query));
72
73     EXECUTE sp_executesql @update_key_query;
74
75     SET @keys_changed = @keys_changed + 1;
76
77    END;
78  GO
```

SQL-запросы для определения первого свободного значения первичного ключа, максимального значения первичного ключа и обновления значения первичного ключа аналогичны решению для MySQL.

Небольшое отличие состоит в том, как получить в переменную результат выполнения динамического запроса: вместо **`SELECT ... INTO ...`** используется **`SET ... = (SELECT ...)`**, а при выполнении динамического SQL с помощью **`sp_executesql`** передаются дополнительные параметры, позволяющие поместить результат выполнения запроса в указанную переменную (строки 55-57, 62-64).

Поскольку MS SQL Server не поддерживает do … while циклы, мы вынуждены использовать бесконечный **`WHILE`**-цикл (строки 53-77), внутри которого будем проверять условие выхода и (при его выполнении) принудительно завершать цикл (строки 59-60).

Последнее отличие от MySQL состоит в способе вывода отладочной информации: в MS SQL Server мы можем использовать конструкцию **`PRINT`** (строки 26-27, 51-52, 71).

Проверить работоспособность полученного решения можно следующими запросами.

MS SQL Решение 5.2.1.a (код для выполнения хранимой процедуры и получения результата её работы)

```
1   DECLARE @res INT;
2   EXECUTE COMPACT_KEYS 'subscriptions', 'sb_id', @res OUTPUT;
3   SELECT @res;
4   GO
5
6   DECLARE @res INT;
7   EXECUTE COMPACT_KEYS 'books', 'b_quantity', @res OUTPUT;
8   SELECT @res;
9   GO
10
11  SELECT * FROM [books] ORDER BY [b_quantity];
```

В первом случае мы получим сообщение об ошибке с просьбой убрать **IDENTITY**-свойство с поля **sb_id** таблицы **subscriptions**, во втором случае процедура выполнится (да, устранение свободных значений в поле, хранящем количество книг, лишено всякого здравого смысла, но для проверки работоспособности хранимой процедуры такой вариант годится).

На этом решение для MS SQL Server завершено.

Переходим к Oracle. Характерных для MS SQL Server проблем здесь нет (даже нет необходимости отключать триггеры, обеспечивающие автоинкрементацию первичного ключа при вставке, т.к. там именно **INSERT**-триггеры, а мы будем выполнять **UPDATE**).

Oracle Решение 5.2.1.a (код процедуры)

```
CREATE PROCEDURE COMPACT_KEYS (table_name IN VARCHAR, pk_name IN VARCHAR,
                               keys_changed OUT NUMBER) AS
 empty_key_query VARCHAR(1000) := '';
 max_key_query VARCHAR(1000) := '';
 empty_key_value NUMBER := NULL;
 max_key_value NUMBER := NULL;
 update_key_query VARCHAR(1000) := '';
BEGIN
 keys_changed := 0;

 DBMS_OUTPUT.PUT_LINE('Point 1. table_name = ' || table_name ||
 ' || pk_name = ' || pk_name || ', keys_changed = ' || keys_changed);

 empty_key_query :=
 'SELECT MIN("empty_key") AS "empty_key"
  FROM   (SELECT "left"."' || pk_name || '" + 1 AS "empty_key"
          FROM   "' || table_name || '" "left"
                 LEFT OUTER JOIN "' || table_name || '" "right"
                              ON "left"."' || pk_name ||
                              '" + 1 = "right"."' || pk_name || '"
          WHERE  "right"."' || pk_name || '" IS NULL
          UNION
          SELECT 1 AS "empty_key"
          FROM   "' || table_name || '"
          WHERE  NOT EXISTS(SELECT "' || pk_name || '"
                            FROM   "' || table_name || '"
                            WHERE  "' || pk_name ||
                            '" = 1)) "prepared_data"
  WHERE  "empty_key" < (SELECT MAX("' || pk_name || '")
                        FROM   "' || table_name || '")';

 max_key_query :=
 'SELECT MAX("' || pk_name || '") FROM "' || table_name || '"';

 DBMS_OUTPUT.PUT_LINE('Point 2. empty_key_query = ' || empty_key_query ||
 CHR(13) || CHR(10) || ' max_key_query = ' || max_key_query);

 LOOP
  EXECUTE IMMEDIATE empty_key_query INTO empty_key_value;
  EXIT WHEN empty_key_value IS NULL;
  EXECUTE IMMEDIATE max_key_query INTO max_key_value;
  update_key_query :=
      'UPDATE "' || table_name || '" SET "' || pk_name ||
      '" = ' || TO_CHAR(empty_key_value) || ' WHERE "' || pk_name ||
      '" = ' || TO_CHAR(max_key_value);
  DBMS_OUTPUT.PUT_LINE('Point 3. update_key_query = ' || update_key_query);
  EXECUTE IMMEDIATE update_key_query;
  keys_changed := keys_changed + 1;
END LOOP;
END;
/
```

Получается, что единственные два отличия решения для Oracle от решения для MySQL состоят в способе вывода отладочной информации (строки 11-12, 35-36, 46) и синтаксисе описания логики выхода из цикла (строка 40).

Проверить работоспособность полученного решения можно следующими запросами (не забудьте предварительно включить отображение получаемых от сервера сообщений запросом **SET SERVEROUTPUT ON**).

Oracle Решение 5.2.1.a (код для выполнения хранимой процедуры и получения результата её работы)

```
1  DECLARE
2   keys_changed_in_table NUMBER;
3  BEGIN
4   COMPACT_KEYS('books', 'b_id', keys_changed_in_table);
5   DBMS_OUTPUT.PUT_LINE('Keys changed: ' || keys_changed_in_table);
6
7   COMPACT_KEYS('subscriptions', 'sb_id', keys_changed_in_table);
8   DBMS_OUTPUT.PUT_LINE('Keys changed: ' || keys_changed_in_table);
9  END;
```

На этом решение данной задачи завершено.

Решение 5.2.1.b{394}.

В решении{395} задачи 5.2.1.a{394} мы уже рассмотрели логику формирования и выполнения динамических SQL-запросов в хранимых процедурах. Отличие решения этой задачи будет в том, что результатом работы хранимой процедуры будет не изменение в БД и возвращение числа правок, а возвращение таблицы.

Решение для MySQL выглядит следующим образом.

MySQL Решение 5.2.1.b (код процедуры)

```
1  DELIMITER $$
2  CREATE PROCEDURE SHOW_TABLE_OBJECTS (IN table_name VARCHAR(150))
3  BEGIN
4   SET @query_text = '
5   SELECT \'foreign_key\'   AS `object_type`,
6          `constraint_name` AS `object_name`
7   FROM   `information_schema`.`table_constraints`
8   WHERE  `table_schema` = DATABASE()
9          AND `table_name` = \'_FP_TABLE_NAME_PLACEHOLDER_\'
10         AND `constraint_type` = \'FOREIGN KEY\'
11  UNION
12  SELECT \'trigger\'    AS `object_type`,
13         `trigger_name` AS `object_name`
14  FROM   `information_schema`.`triggers`
15  WHERE  `event_object_schema` = DATABASE()
16         AND `event_object_table` = \'_FP_TABLE_NAME_PLACEHOLDER_\'
17  UNION
18  SELECT \'view\'     AS `object_type`,
19         `table_name` AS `object_name`
20  FROM   `information_schema`.`views`
21  WHERE  `table_schema` = DATABASE()
22         AND `view_definition` LIKE \'%`_FP_TABLE_NAME_PLACEHOLDER_`%\'';
23  SET @query_text = REPLACE(@query_text,
24                            '_FP_TABLE_NAME_PLACEHOLDER_', table_name);
25
```

MySQL Решение 5.2.1.b (код процедуры) (продолжение)

```
1   DELIMITER $$
2   CREATE PROCEDURE SHOW_TABLE_OBJECTS (IN table_name VARCHAR(150))
3   BEGIN
4    SET @query_text = '
5    SELECT \'foreign_key\'   AS `object_type`,
6           `constraint_name` AS `object_name`
7    FROM   `information_schema`.`table_constraints`
8    WHERE  `table_schema` = DATABASE()
9           AND `table_name` = \'_FP_TABLE_NAME_PLACEHOLDER_\'
10          AND `constraint_type` = \'FOREIGN KEY\'
11   UNION
12   SELECT \'trigger\'    AS `object_type`,
13          `trigger_name` AS `object_name`
14   FROM   `information_schema`.`triggers`
15   WHERE  `event_object_schema` = DATABASE()
16          AND `event_object_table` = \'_FP_TABLE_NAME_PLACEHOLDER_\'
17   UNION
18   SELECT \'view\'     AS `object_type`,
19          `table_name` AS `object_name`
20   FROM   `information_schema`.`views`
21   WHERE  `table_schema` = DATABASE()
22          AND `view_definition` LIKE \'%`_FP_TABLE_NAME_PLACEHOLDER_`%\'';
23   SET @query_text = REPLACE(@query_text,
24                             '_FP_TABLE_NAME_PLACEHOLDER_', table_name);
25
26   PREPARE query_stmt FROM @query_text;
27   EXECUTE query_stmt;
28   DEALLOCATE PREPARE query_stmt;
29  END;
30  $$
31  DELIMITER ;
```

Здесь (для разнообразия) текст итогового запроса мы получаем не с использованием функции **CONCAT**, а путём замены плейсхолдера **_FP_TABLE_NAME_PLACEHOLDER_** на реальное имя таблицы в заранее подготовленном полном тексте запроса.

Логика же получения самого списка искомых объектов полностью тривиальная для внешних ключей и триггеров (см. текст запроса в строках 5-16), и только для представлений мы должны анализировать их исходный код, чтобы обнаружить упоминание там имени таблицы, полученной как параметр нашей процедуры (т.к. представления не ассоциируются напрямую с таблицами, а являются независимыми объектами).

Проверить работоспособность полученного решения можно следующим запросом.

MySQL Решение 5.2.1.b (код для выполнения хранимой процедуры и получения результата её работы)

```
1   CALL SHOW_TABLE_OBJECTS('subscriptions')
```

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Здесь логика решения полностью совпадает с решением для MySQL, за исключением того факта, что информацию о триггерах приходится извлекать из **[sys].[triggers]** как аналога **`information_schema`.`triggers`**.

MS SQL Решение 5.2.1.b (код процедуры)

```
CREATE PROCEDURE SHOW_TABLE_OBJECTS
                 @table_name NVARCHAR(150)
WITH EXECUTE AS OWNER
AS
 DECLARE @query_text NVARCHAR(1000) = '';
 SET @query_text =
 'SELECT ''foreign_key''   AS [object_type],
         [constraint_name] AS [object_name]
  FROM   [information_schema].[table_constraints]
  WHERE  [table_catalog] = DB_NAME()
         AND [table_name] = ''_FP_TABLE_NAME_PLACEHOLDER_''
         AND [constraint_type] = ''FOREIGN KEY''
  UNION
  SELECT ''trigger'' AS [object_type],
         [name]      AS [object_name]
  FROM   [sys].[triggers]
  WHERE  OBJECT_NAME([parent_id]) = ''_FP_TABLE_NAME_PLACEHOLDER_''
  UNION
  SELECT ''view''     AS [object_type],
         [table_name] AS [object_name]
  FROM   [information_schema].[views]
  WHERE  [table_catalog] = DB_NAME()
         AND [view_definition] LIKE ''%[_FP_TABLE_NAME_PLACEHOLDER_]%''';

 SET @query_text = REPLACE(@query_text, '_FP_TABLE_NAME_PLACEHOLDER_',
                                        @table_name);

 EXECUTE sp_executesql @query_text;
GO
```

Проверить работоспособность полученного решения можно следующим запросом.

MS SQL Решение 5.2.1.b (код для выполнения хранимой процедуры и получения результата её работы)

```
EXECUTE SHOW_TABLE_OBJECTS 'subscriptions';
```

На этом решение для MS SQL Server завершено.

Переходим к Oracle. И вот здесь уже появятся радикальные отличия. В основе решения всё равно будет лежать тот же запрос, который мы использовали для MySQL и MS SQL Server, но в Oracle существует одна очень неприятная проблема, усложняющая решение в разы.

Текст представления (в котором мы ищем упоминание интересующей нас таблицы) хранится в поле типа **LONG** (это — не «длинное целое», это устаревший, но всё ещё используемый иногда текстовый тип данных[22]), и данные этого типа невозможно ни использовать в выражениях типа **LIKE**, ни преобразовать простым способом к другому типу (например, **VARCHAR2**).

Единственный более-менее адекватный способ извлечения **LONG**-данных с конвертацией к **VARCHAR2** — создание хранимой функции. Существует универсальное решение[23], но для простоты мы реализуем вариант, привязанный к конкретному источнику данных.

В представленном ниже коде мы создаём хранимую функцию, возвращающую таблицу (принцип создания и использования таких функций рассмотрен в решении{372} задачи 5.1.1.b{369}). Ключевая идея здесь состоит в том, что поле **TEXT** объекта **all_views_row** объявлено как **VARCHAR2**, и именно такой тип данных будет в выходной таблице. А с **VARCHAR2**-данными уже можно выполнять операции сравнения.

---

22 https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/datatypes.htm#LNPLS346

23 https://asktom.oracle.com/pls/apex/f?p=100:11:0::NO::P11_QUESTION_ID:839298816582

В данном конкретном случае функция сначала готовит полный набор данных, и затем возвращает его в виде сформированной таблицы, а **PIPLINED**-решение будет представлено далее.

Oracle Решение 5.2.1.b (код функции для преобразования LONG в VARCHAR2)

```
CREATE OR REPLACE TYPE "all_views_row" AS OBJECT
(
   "VIEW_NAME" VARCHAR2(500),
   "TEXT"      VARCHAR2(32767)
);
/
CREATE TYPE "all_views_table" IS TABLE OF "all_views_row";
/

CREATE OR REPLACE FUNCTION ALL_VIEWS_VARCHAR2
RETURN "all_views_table"
AS
 result_table  "all_views_table" := "all_views_table"();
 CURSOR all_views_table_cursor IS
  SELECT VIEW_NAME,
         TEXT
  FROM   ALL_VIEWS
  WHERE  OWNER = USER;
BEGIN
  FOR one_row IN all_views_table_cursor
  LOOP
   result_table.extend;
   result_table(result_table.last) :=
                   "all_views_row"(one_row."VIEW_NAME", one_row."TEXT");
  END LOOP;
  RETURN result_table;
END;
/
```

Теперь, когда проблема с применением выражения **LIKE** к тексту представления решена, остаётся только создать хранимую процедуру по аналогии с решениями для MySQL и MS SQL Server.

Oracle Решение 5.2.1.b (код процедуры)

```
CREATE OR REPLACE PROCEDURE SHOW_TABLE_OBJECTS(table_name IN VARCHAR2,
                                               final_rc OUT SYS_REFCURSOR)
IS
 query_text VARCHAR2(1000);
BEGIN
 query_text := '
  SELECT ''foreign_key'' AS "object_type",
         CONSTRAINT_NAME AS "object_name"
  FROM ALL_CONSTRAINTS
  WHERE OWNER = USER
  AND TABLE_NAME = ''_FP_TABLE_NAME_PLACEHOLDER_''
  AND CONSTRAINT_TYPE = ''R''
  UNION
  SELECT ''trigger''  AS "object_type",
         TRIGGER_NAME AS "object_name"
  FROM ALL_TRIGGERS
  WHERE OWNER = USER
  AND TABLE_NAME = ''_FP_TABLE_NAME_PLACEHOLDER_''
  UNION
  SELECT ''view''    AS "object_type",
         "VIEW_NAME" AS "object_name"
  FROM TABLE(ALL_VIEWS_VARCHAR2)
  WHERE "TEXT" LIKE ''%"_FP_TABLE_NAME_PLACEHOLDER_"%''';

```

Oracle Решение 5.2.1.b (код процедуры) (продолжение)

```
25    query_text := REPLACE(query_text, '_FP_TABLE_NAME_PLACEHOLDER_',
26                                      table_name);
27
28   OPEN final_rc FOR query_text;
29  END;
30  /
```

Но одно неудобство остаётся: чтобы получить результат работы такой процедуры, придётся использовать достаточно нетривиальный код:

Oracle Решение 5.2.1.b (код для выполнения хранимой процедуры и получения результата её работы)

```
1   DECLARE
2     rc SYS_REFCURSOR;
3     object_type VARCHAR2(500);
4     object_name VARCHAR2(500);
5   BEGIN
6     SHOW_TABLE_OBJECTS('subscriptions', rc);
7
8     LOOP
9       FETCH rc INTO object_type, object_name;
10      EXIT WHEN rc%NOTFOUND;
11      DBMS_OUTPUT.PUT_LINE(object_type || ' | ' || object_name);
12    END LOOP;
13    CLOSE rc;
14  END;
```

И даже с таким нетривиальным кодом мы получаем результат в виде текста, а хотелось бы получить полноценную таблицу. Это возможно, но нам придётся отказаться от хранимой процедуры и реализовать хранимую функцию.

Чуть выше мы создавали такую функцию для конвертации типа данных поля, в котором хранится текст представления. Сейчас мы доработаем её, получив в виде одной функции законченное решение, полностью удовлетворяющее условиям текущей задачи.

Заодно мы изменим тип функции на **PIPELINED**, что снизит нагрузку на оперативную память и немного повысит производительность.

Oracle Решение 5.2.1.b (альтернативное решение в виде одной хранимой функции)

```
1   CREATE OR REPLACE TYPE "show_table_objects_row" AS OBJECT
2   (
3      "field_a"  VARCHAR2(500),
4      "field_b"  VARCHAR2(32767)
5   );
6   /
7   CREATE TYPE "show_table_objects_table"
8   IS TABLE OF "show_table_objects_row";
9   /
10
11  CREATE OR REPLACE FUNCTION SHOW_TABLE_OBJECTS_FNC(table_name IN VARCHAR2)
12  RETURN "show_table_objects_table" PIPELINED
13  AS
14   TYPE type_rc IS REF CURSOR;
15   rc type_rc;
16   field_a    VARCHAR2(500);
17   field_b    VARCHAR2(32767);
18   query_text VARCHAR2(1000);
```

Oracle Решение 5.2.1.b (альтернативное решение в виде одной хранимой функции) (продолжение)

```
19  BEGIN
20   query_text := '
21    SELECT ''foreign_key'' AS "object_type",
22           CONSTRAINT_NAME AS "object_name"
23    FROM ALL_CONSTRAINTS
24    WHERE OWNER = USER
25    AND TABLE_NAME = ''_FP_TABLE_NAME_PLACEHOLDER_''
26    AND CONSTRAINT_TYPE = ''R''
27    UNION
28    SELECT ''trigger''  AS "object_type",
29           TRIGGER_NAME AS "object_name"
30    FROM ALL_TRIGGERS
31    WHERE OWNER = USER
32    AND TABLE_NAME = ''_FP_TABLE_NAME_PLACEHOLDER_''';
33    query_text := REPLACE(query_text, '_FP_TABLE_NAME_PLACEHOLDER_',
34                                      table_name);
35
36    OPEN rc FOR query_text;
37    LOOP
38     FETCH rc INTO field_a, field_b;
39     EXIT WHEN rc%NOTFOUND;
40     PIPE ROW("show_table_objects_row"(field_a, field_b));
41    END LOOP;
42    CLOSE rc;
43
44   query_text := '
45    SELECT VIEW_NAME,
46           TEXT
47    FROM   ALL_VIEWS
48    WHERE  OWNER = USER';
49
50    OPEN rc FOR query_text;
51    LOOP
52     FETCH rc INTO field_a, field_b;
53     EXIT WHEN rc%NOTFOUND;
54     IF (INSTR(field_b, '"' || table_name || '"') > 0)
55      THEN
56       PIPE ROW("show_table_objects_row"('view', field_a));
57      END IF;
58    END LOOP;
59    CLOSE rc;
60
61    RETURN;
62  END;
```

Ключевая идея этой функции состоит в том, чтобы передавать на выход данные, полученные в отдельности из двух разных запросов.

Первый запрос (строки 20-42) просто выбирает данные по внешним ключам и триггерам без каких-то особых сложностей и нюансов: в переменную **field_a** помещается строковая константа (**'foreign_key'**, **'trigger'**), в переменную **field_b** — имя соответствующего внешнего ключа или триггера. Затем эти переменные используются для инициализации полей объекта **show_table_objects_row**.

Во втором запросе (строки 44-59) мы с использованием тех же переменных и того же объекта, что и в первом запросе, делаем следующее:

- сначала в переменные **field_a** и **field_b** помещаются имя и текст представления соответственно (строка 52);
- затем значение переменной **field_b** используется для проверки того факта, что текст представления содержит имя анализируемой таблицы (строка 54), и больше это значение переменной **field_b** нам не нужно и нигде не используется;
- наконец (в строке 56) мы используем текстовую константу 'view' и имя представления, хранящееся в переменной **field_b**, для инициализации полей объекта **show_table_objects_row**.

Теперь мы можем получить необходимый нам результат в виде таблицы.

Oracle	Решение 5.2.1.b (код для выполнения хранимой функции и получения результата в виде таблицы)

```
1   SELECT * FROM TABLE(SHOW_TABLE_OBJECTS_FNC('subscriptions'));
```

На этом решение данной задачи завершено.

Задание 5.2.1.TSK.A: создать хранимую процедуру, обновляющую все поля типа **DATE** (если такие есть) всех записей указанной таблицы на значение текущей даты.

Задание 5.2.1.TSK.B: создать хранимую процедуру, формирующую список таблиц и их внешних ключей, зависящих от указанной в параметре функции таблицы.

5.2.2. ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ С ПОМОЩЬЮ ХРАНИМЫХ ПРОЦЕДУР

Задача 5.2.2.a{408}: создать хранимую процедуру, запускаемую по расписанию каждый час и обновляющую данные в агрегирующей таблице **books_statistics** (см. задачу 3.1.2.a{223}).

Задача 5.2.2.b{413}: создать хранимую процедуру, запускаемую по расписанию каждый день и оптимизирующую (дефрагментирующую, компактифицирующую) все таблицы базы данных.

Ожидаемый результат 5.2.2.a.

В начале каждого часа запускается созданная хранимая процедура. Данные в таблице **books_statistics** приводятся в актуальное состояние.

Ожидаемый результат 5.2.2.b.

В начале каждых суток запускается созданная хранимая процедура. Все таблицы базы данных приводятся в оптимизированное состояние.

Решение 5.2.2.a{408}.

Основной запрос (строки 14-29), выполняющий обновление данных в таблице **books_statistics**, построен на основе решения{223} задачи 3.1.2.a{223}.

Однако для повышения надёжности мы будем проверять существование таблицы **books_statistics**, которую собираемся обновить. Эта проверка реализована в строках 5-13. Если искомая таблица не существует, мы завершаем работу хранимой процедуры, возвращая соответствующее сообщение об ошибке.

MySQL Решение 5.2.2.a (код процедуры)

```
1  DELIMITER $$
2  CREATE PROCEDURE UPDATE_BOOKS_STATISTICS()
3  BEGIN
4
5   IF (NOT EXISTS(SELECT *
6                  FROM `information_schema`.`tables`
7                  WHERE `table_schema` = DATABASE()
8                        AND `table_name` = 'books_statistics'))
9    THEN
10    SIGNAL SQLSTATE '45001'
11    SET MESSAGE_TEXT = 'The `books_statistics` table is missing.',
12    MYSQL_ERRNO = 1001;
13   END IF;
14  UPDATE `books_statistics`
15  JOIN
16  (SELECT IFNULL(`total`, 0) AS `total`,
17          IFNULL(`given`, 0) AS `given`,
18          IFNULL(`total` - `given`, 0) AS `rest`
19   FROM   (SELECT (SELECT SUM(`b_quantity`)
20                   FROM   `books`)              AS `total`,
21                  (SELECT COUNT(`sb_book`)
22                   FROM   `subscriptions`
23                   WHERE  `sb_is_active` = 'Y') AS `given`)
24          AS `prepared_data`
25  ) AS `src`
26  SET
27   `books_statistics`.`total` = `src`.`total`,
28   `books_statistics`.`given` = `src`.`given`,
29   `books_statistics`.`rest` = `src`.`rest`;
30 END;
31 $$
32 DELIMITER ;
```

Проверим работоспособность (предварительно можно выполнить отдельный запрос на обновление данных в таблице **books_statistics** и установить все значения в ноль).

MySQL Решение 5.2.2.a (запуск и проверка работоспособности)

```
1  CALL UPDATE_BOOKS_STATISTICS;
```

Установка запуска полученной хранимой процедуры по расписанию выглядит следующим образом. Предварительно в строке 1 мы включаем планировщик задач MySQL (для того, чтобы он продолжал работать и после перезапуска MySQL, необходимо добавить строку **event_scheduler = on** в файл настроек MySQL my.ini).

MySQL Решение 5.2.2.a (установка запуска по расписанию)

```
1  SET GLOBAL event_scheduler = ON;
2
3  CREATE EVENT `update_books_statistics_hourly`
4  ON SCHEDULE
5  EVERY 1 HOUR
6  STARTS DATE(NOW()) + INTERVAL (HOUR(NOW())+1) HOUR + INTERVAL 1 MINUTE
7  ON COMPLETION PRESERVE
8  DO
9  CALL UPDATE_BOOKS_STATISTICS;
```

Убедиться, что соответствующая задача добавлена в планировщик, можно выполнив следующий запрос:

MySQL Решение 5.2.2.a (просмотр расписания)

```
1  SELECT * FROM `information_schema`.`events`
```

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Логика работы хранимой процедуры здесь полностью эквивалентна решению для MySQL: мы проверяем наличие таблицы **books_statistics** в строках 3-10 (и завершаем работу хранимой процедуры, если таблицы нет), а в строках 12-30 выполняем запрос, обновляющий данные.

Пока всё выглядит достаточно просто и тривиально, но добавление задачи в планировщик в данной СУБД реализовано куда более сложным образом.

MS SQL Решение 5.2.2.a (код процедуры)

```
1  CREATE PROCEDURE UPDATE_BOOKS_STATISTICS
2  AS
3   IF (NOT EXISTS(SELECT *
4                  FROM [information_schema].[tables]
5                  WHERE [table_catalog] = DB_NAME()
6                    AND [table_name] = 'books_statistics'))
7    BEGIN
8     RAISERROR ('The [books_statistics] table is missing.', 16, 1);
9     RETURN;
10   END;
11
12  UPDATE [books_statistics]
13  SET
14   [books_statistics].[total] = [src].[total],
15   [books_statistics].[given] = [src].[given],
16   [books_statistics].[rest] = [src].[rest]
17   FROM [books_statistics]
18  JOIN
19  (SELECT ISNULL([total], 0) AS [total],
20          ISNULL([given], 0) AS [given],
21          ISNULL([total] - [given], 0) AS [rest]
22   FROM   (SELECT (SELECT SUM([b_quantity])
23                   FROM   [books])              AS [total],
24                  (SELECT COUNT([sb_book])
25                   FROM   [subscriptions]
26                   WHERE  [sb_is_active] = 'Y') AS [given])
27          AS [prepared_data]
28  ) AS [src]
29  ON 1=1;
30 GO
```

Проверим работоспособность (предварительно можно выполнить отдельный запрос на обновление данных в таблице **books_statistics** и установить все значения в ноль).

MS SQL Решение 5.2.2.a (запуск и проверка работоспособности)

```
EXECUTE UPDATE_BOOKS_STATISTICS
```

Установка запуска полученной хранимой процедуры по расписанию в MS SQL Server не только выглядит куда более сложно, но и не даст эффекта, если вы используете Express Edition этой СУБД: в этой версии отсутствует компонент SQL Server Agent, который и отвечает за выполнение запланированных задач.

По каждой приведённой далее команде в официальной документации написано очень много (перед каждой командой специально приведены ссылка на соответствующий раздел документации), но если выразить простыми словами алгоритм, получится следующее. Необходимо:

- Создать задачу.
- Создать шаг задачи, описывающий запуск нашей хранимой процедуры.
- Создать расписание, в котором указать требуемую периодичность выполнения.
- Прикрепить ранее созданную задачу к только что созданному расписанию.
- Передать созданную задачу на обработку в SQL Server Agent.

MS SQL Решение 5.2.2.a (установка запуска по расписанию)

```
USE msdb ;
GO
-- https://msdn.microsoft.com/en-us/library/ms182079.aspx
EXEC dbo.sp_add_job
    @job_name = N'Hourly [books_statistics] update';
GO
-- https://msdn.microsoft.com/en-us/library/ms187358.aspx
EXEC sp_add_jobstep
    @job_name = N'Hourly [books_statistics] update',
    @step_name = N'Execute UPDATE_BOOKS_STATISTICS stored procedure',
    @subsystem = N'TSQL',
    @command = N'EXECUTE UPDATE_BOOKS_STATISTICS',
    @database_name = N'library_ex_2015_mod';
GO
-- https://msdn.microsoft.com/en-us/library/ms187320.aspx
EXEC dbo.sp_add_schedule
    @schedule_name = N'UpdateBooksStatistics',
    @freq_type = 4,
    @freq_interval = 4,
    @freq_subday_type = 8,
    @freq_subday_interval = 1,
    @active_start_time = 000100 ;
USE msdb ;
GO
-- https://msdn.microsoft.com/en-us/library/ms186766.aspx
EXEC sp_attach_schedule
   @job_name = N'Hourly [books_statistics] update',
   @schedule_name = N'UpdateBooksStatistics';
GO
-- https://msdn.microsoft.com/en-us/library/ms178625.aspx
EXEC dbo.sp_add_jobserver
    @job_name = N'Hourly [books_statistics] update';
GO
```

Убедиться, что соответствующая задача добавлена в планировщик, можно выполнив следующий запрос (он покажет список задач даже в MS SQL Server Express Edition):

MS SQL Решение 5.2.2.a (просмотр расписания)

```
SELECT * FROM msdb.dbo.sysschedules
```

В данном обсуждении[24] представлен очень хороший готовый SQL-скрипт для получения в удобной форме информации обо всех запланированных в MS SQL Server задачах.

На этом решение для MS SQL Server завершено.

Переходим к Oracle. Логика работы хранимой процедуры здесь полностью эквивалентна решениям для MySQL и MS SQL Server: мы проверяем наличие таблицы **books_statistics** в строках 5-15 (и завершаем работу хранимой процедуры, если таблицы нет), а в строках 17-27 выполняем запрос, обновляющий данные.

Oracle Решение 5.2.2.a (код процедуры)

```
1   CREATE OR REPLACE PROCEDURE UPDATE_BOOKS_STATISTICS
2   AS
3    rows_count NUMBER;
4   BEGIN
5    SELECT COUNT(1) INTO rows_count
6    FROM ALL_TABLES
7    WHERE OWNER = USER
8    AND TABLE_NAME = 'books_statistics';
9
10   IF (rows_count = 0)
11    THEN
12     RAISE_APPLICATION_ERROR(-20001,
13                             'The "books_statistics" table is missing.');
14     RETURN;
15    END IF;
16
17   UPDATE "books_statistics"
18     SET ("total", "given", "rest") =
19     (SELECT NVL("total", 0) AS "total",
20             NVL("given", 0) AS "given",
21             NVL("total" - "given", 0) AS "rest"
22      FROM   (SELECT (SELECT SUM("b_quantity")
23                      FROM   "books")              AS "total",
24                     (SELECT COUNT("sb_book")
25                      FROM   "subscriptions"
26                      WHERE  "sb_is_active" = 'Y') AS "given"
27             FROM dual) "prepared_data");
28  END;
29  /
```

Проверим работоспособность (предварительно можно выполнить отдельный запрос на обновление данных в таблице **books_statistics** и установить все значения в ноль).

Oracle Решение 5.2.2.a (запуск и проверка работоспособности)

```
EXECUTE UPDATE_BOOKS_STATISTICS
```

Установка запуска полученной хранимой процедуры по расписанию выглядит следующим образом.

---

[24] http://www.sqlservercentral.com/Forums/Topic410557-116-1.aspx

Oracle Решение 5.2.2.a (установка запуска по расписанию)

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name            =>  'hourly_update_books_statistics',
    job_type            =>  'STORED_PROCEDURE',
    job_action          =>  'UPDATE_BOOKS_STATISTICS',
    start_date          =>  '01-APR-16 1.00.00 AM',
    repeat_interval     =>  'FREQ=HOURLY;INTERVAL=1',
    auto_drop           =>  FALSE,
    enabled             =>  TRUE);
END;
```

Убедиться, что соответствующая задача добавлена в планировщик, можно выполнив следующий запрос:

Oracle Решение 5.2.2.a (просмотр расписания)

```
SELECT * FROM ALL_SCHEDULER_JOBS WHERE OWNER=USER
```

На этом решение данной задачи завершено.

Решение 5.2.2.b[408].

Решение этой задачи одновременно является очень простым и очень сложным, т.к. нам нужно получить список таблиц базы данных и… что-то с ними сделать.

Вопрос оптимизации производительности баз данных и СУБД заслуживает отдельной книги, потому здесь мы пойдём по пути наименьшего сопротивления и будем считать, что:

- Для MySQL будет достаточно выполнить **`OPTIMIZE`** для всех таблиц.
- Для MS SQL Server достаточно выполнить **`REORGANIZE`** или **`REBUILD`** для всех кластерных индексов (что приводит к оптимизации соответствующей таблицы, на которых построен индекс).
- Для Oracle будет достаточно выполнить **`SHRINK SPACE COMPACT CASCADE`** для всех таблиц.

Ещё раз особо подчеркнём: решение данной задачи носит исключительно демонстрационный характер и не должно рассматриваться как рекомендация по универсальной оптимизации производительности. В некоторых случаях выполнение показанных ниже действий может снизить производительность базы данных, потому обязательно внимательно изучите официальную документацию по соответствующей СУБД и профессиональные рекомендации по оптимизации производительности в той или иной реальной ситуации.

Традиционно начинаем с MySQL.

MySQL Решение 5.2.2.b (код процедуры)

```
DELIMITER $$
CREATE PROCEDURE OPTIMIZE_ALL_TABLES()
BEGIN
 DECLARE done INT DEFAULT 0;
 DECLARE tbl_name VARCHAR(200) DEFAULT '';
 DECLARE all_tables_cursor CURSOR FOR
  SELECT `table_name`
  FROM   `information_schema`.`tables`
  WHERE  `table_schema` = DATABASE()
  AND    `table_type` = 'BASE TABLE';
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

```

MySQL Решение 5.2.2.b (код процедуры) (продолжение)

```
13   OPEN all_tables_cursor;
14
15   tables_loop: LOOP
16    FETCH all_tables_cursor INTO tbl_name;
17    IF done THEN
18     LEAVE tables_loop;
19    END IF;
20
21    SET @table_opt_query = CONCAT('OPTIMIZE TABLE `', tbl_name, '`');
22    PREPARE table_opt_stmt FROM @table_opt_query;
23    EXECUTE table_opt_stmt;
24    DEALLOCATE PREPARE table_opt_stmt;
25
26   END LOOP tables_loop;
27
28   CLOSE all_tables_cursor;
29  END;
30  $$
31
32  DELIMITER ;
```

Запрос в строках 7-10 позволяет получить список таблиц текущей базы данных (часть условия в строке 10 позволяет отличить таблицы от представлений), после чего:

- для полученного списка таблиц открывается курсор (строка 13);
- для всех строк курсора выполняется цикл (строки 15-26);
- в теле цикла формируется (строка 21) и выполняется (строки 22-23) запрос, реализующий оптимизацию таблицы.

Запустить полученную хранимую процедуру можно следующим образом.

MySQL Решение 5.2.2.b (запуск и проверка работоспособности)

```
1   CALL OPTIMIZE_ALL_TABLES
```

Теперь остаётся создать событие, запускаемое планировщиком задач MySQL по определённому расписанию. Это реализуется следующим кодом.

MySQL Решение 5.2.2.b (установка запуска по расписанию)

```
1   SET GLOBAL event_scheduler = ON;
2
3   CREATE EVENT `optimize_all_tables_daily`
4   ON SCHEDULE
5   EVERY 1 DAY
6   STARTS DATE(NOW()) + INTERVAL 1 HOUR
7   ON COMPLETION PRESERVE
8   DO
9   CALL OPTIMIZE_ALL_TABLES;
```

Убедиться, что событие добавлено, можно следующим запросом.

MySQL Решение 5.2.2.b (просмотр расписания)

```
1   SELECT * FROM `information_schema`.`events`
```

Если рассмотреть избранные поля результата такого запроса, получается следующая картина. Здесь также представлено событие, активирующее раз в час хранимую процедуру, созданную в процессе решения{408} задачи 5.2.2.a{408}.

EVENT_NAME	EVENT_TYPE	INTERVAL_ VALUE	INTERVAL_ FIELD	STARTS	ENDS	STATUS	ON_ COMPLETION
update_books_ statistics_hourly	RECURRING	1	HOUR	2016-04-27 14:01:00	NULL	ENABLED	PRESERVE
optimize_all_tables_ daily	RECURRING	1	DAY	2016-04-27 01:00:00	NULL	ENABLED	PRESERVE

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Концепция представленного ниже решения основана на этой рекомендации[25].

Итак, мы должны будем получить список кластерных индексов текущей базы данных, выяснить степень их фрагментации (`avg_fragmentation_in_percent`) и, в зависимости от этого значения, выполнить реорганизацию (`REORGANIZE`) или перестроение (`REBUILD`) индекса.

Для начала напишем запрос, предоставляющий информацию по всем индексам, соответствующим таблицам, полям и т.д. К сожалению, готового решения MS SQL Server не предоставляет, потому придётся всё делать «вручную».

MS SQL Решение 5.2.2.b (получение данных по всем индексам)

```
SELECT [tables].[name]                                AS [table_name],
       [indexes].[name]                               AS [index_name],
       [indexes].[type]                               AS [index_type],
       [stats].[index_type_desc]                      AS [index_type_desc],
       [indexes].[object_id]                          AS [index_object_id],
       [columns].[name]                               AS [column_name],
       [stats].[avg_fragmentation_in_percent]   AS [avg_fragm_perc],
       [stats].[avg_page_space_used_in_percent] AS [avg_space_perc]
FROM   sys.indexes AS [indexes]
       INNER JOIN sys.index_columns AS [index_columns]
               ON [indexes].[object_id] = [index_columns].[object_id]
                  AND [indexes].[index_id] = [index_columns].[index_id]
       INNER JOIN sys.columns AS [columns]
               ON [index_columns].[object_id] = [columns].[object_id]
                  AND [index_columns].[column_id] = [columns].[column_id]
       INNER JOIN sys.tables AS [tables]
               ON [indexes].[object_id] = [tables].[object_id]
       INNER JOIN sys.dm_db_index_physical_stats(DB_ID(DB_NAME()),
                                                 NULL, NULL, NULL,
                                                 'SAMPLED') AS [stats]
               ON [indexes].[object_id] = [stats].[object_id]
                     AND [indexes].[index_id] = [stats].[index_id]
ORDER  BY [tables].[name],
          [indexes].[name],
          [indexes].[index_id],
          [index_columns].[index_column_id]
```

25 http://blog.sqlauthority.com/2010/01/12/sql-server-fragmentation-detect-fragmentation-and-eliminate-fragmentation/

В результате выполнение такого запроса мы получим следующие данные.

table_name	index_name	index_type	index_type_desc	index_object_id	column_name	avg_fragm_perc	avg_space_perc
authors	PK_authors	1	CLUSTERED INDEX	245575913	a_id	0	3.22461082283173
books	PK_books	1	CLUSTERED INDEX	277576027	b_id	0	5.72028663207314
genres	PK_genres	1	CLUSTERED INDEX	309576141	g_id	0	2.59451445515196
genres	UQ_genres_g_name	2	NONCLUSTERED INDEX	309576141	g_name	0	2.37212750185322
m2m_books_authors	PK_m2m_books_authors	1	CLUSTERED INDEX	341576255	b_id	0	1.86557944156165
m2m_books_authors	PK_m2m_books_authors	1	CLUSTERED INDEX	341576255	a_id	0	1.86557944156165
m2m_books_genres	PK_m2m_books_genres	1	CLUSTERED INDEX	373576369	b_id	0	2.28564368668149
m2m_books_genres	PK_m2m_books_genres	1	CLUSTERED INDEX	373576369	g_id	0	2.28564368668149
subscribers	PK_subscribers	1	CLUSTERED INDEX	405576483	s_id	0	1.95206325673338
subscriptions	PK_subscriptions	1	CLUSTERED INDEX	437576597	sb_id	0	5.16431924882629

Этот результат хорош тем, что удобен для просмотра человеком, но внутри хранимой процедуры нам будут нужны только три поля: имя таблицы, имя индекса, процент фрагментации. К тому же составные кластерные индексы (например, **`PK_m2m_books_authors`**) должны быть представлены только один раз (а не по разу на каждое из входящих в состав индекса полей, как это есть сейчас).

Упростим запрос так, чтобы оставить только необходимые данные.

MS SQL Решение 5.2.2.b (получение краткого набора данных по всем индексам)

```
SELECT DISTINCT
       [tables].[name]                                AS [table_name],
       [indexes].[name]                               AS [index_name],
       [stats].[avg_fragmentation_in_percent]         AS [avg_fragm_perc]
FROM   sys.indexes AS [indexes]
       INNER JOIN sys.tables AS [tables]
               ON [indexes].[object_id] = [tables].[object_id]
       INNER JOIN sys.dm_db_index_physical_stats(DB_ID(DB_NAME()),
                                                 NULL, NULL, NULL,
                                                 'SAMPLED') AS [stats]
               ON [indexes].[object_id] = [stats].[object_id]
                  AND [indexes].[index_id] = [stats].[index_id]
WHERE  [indexes].[type] = 1
ORDER  BY [tables].[name],
          [indexes].[name]
```

В результате выполнение такого запроса мы получим следующие данные.

table_name	index_name	avg_fragm_perc
authors	PK_authors	0
books	PK_books	0
genres	PK_genres	0
m2m_books_authors	PK_m2m_books_authors	0
m2m_books_genres	PK_m2m_books_genres	0
subscribers	PK_subscribers	0
subscriptions	PK_subscriptions	0

Теперь используем полученный запрос в теле хранимой процедуры. Проходя по всем рядам, возвращаемым курсором (цикл в строках 29-56), мы будем анализировать значение **avg_fragm_perc** и либо выполнять одно из двух действий по оптимизации, либо не выполнять никаких действий.

MS SQL Решение 5.2.2.b (код процедуры)

```
1   CREATE PROCEDURE OPTIMIZE_ALL_TABLES
2   AS
3   BEGIN
4    DECLARE @table_name NVARCHAR(200);
5    DECLARE @index_name NVARCHAR(200);
6    DECLARE @avg_fragm_perc DOUBLE PRECISION;
7    DECLARE @query_text NVARCHAR(2000);
8    DECLARE indexes_cursor CURSOR LOCAL FAST_FORWARD FOR
9     SELECT DISTINCT
10           [tables].[name]                          AS [table_name],
11           [indexes].[name]                         AS [index_name],
12           [stats].[avg_fragmentation_in_percent]   AS [avg_fragm_perc]
13    FROM   sys.indexes AS [indexes]
14           INNER JOIN sys.tables AS [tables]
15                   ON [indexes].[object_id] = [tables].[object_id]
16           INNER JOIN sys.dm_db_index_physical_stats(DB_ID(DB_NAME()),
17                                                     NULL, NULL, NULL,
18                                                     'SAMPLED') AS [stats]
19                   ON [indexes].[object_id] = [stats].[object_id]
20                          AND [indexes].[index_id] = [stats].[index_id]
21    WHERE  [indexes].[type] = 1
22    ORDER  BY [tables].[name],
23              [indexes].[name];
24
25   OPEN indexes_cursor;
26   FETCH NEXT FROM indexes_cursor INTO @table_name,
27                                       @index_name,
28                                       @avg_fragm_perc;
29  WHILE @@FETCH_STATUS = 0
30    BEGIN
31     IF (@avg_fragm_perc >= 5.0) AND (@avg_fragm_perc <= 30.0)
32      BEGIN
33       SET @query_text = CONCAT('ALTER INDEX [', @index_name,
34                                '] ON [', @table_name, '] REORGANIZE');
35       PRINT CONCAT('Index [', @index_name,'] on [', @table_name,
36                    '] will be REORGANIZED...');
37       EXECUTE sp_executesql @query_text;
38      END;
39     IF (@avg_fragm_perc > 30.0)
40      BEGIN
41       SET @query_text = CONCAT('ALTER INDEX [', @index_name,'] ON [',
42                                @table_name, '] REBUILD');
43       PRINT CONCAT('Index [', @index_name,'] on [', @table_name,
44                    '] will be REBUILT...');
45       EXECUTE sp_executesql @query_text;
46      END;
47     IF (@avg_fragm_perc < 5.0)
48      BEGIN
49       PRINT CONCAT('Index [', @index_name,'] on [', @table_name,
50                    '] needs no optimization...');
51      END;
52
```

MS SQL Решение 5.2.2.b (код процедуры) (продолжение)

```
53      FETCH NEXT FROM indexes_cursor INTO @table_name,
54                                          @index_name,
55                                          @avg_fragm_perc;
56     END;
57    CLOSE indexes_cursor;
58    DEALLOCATE indexes_cursor;
59  END;
60  GO
```

Запустить полученную хранимую процедуру можно следующим образом.

MS SQL Решение 5.2.2.b (запуск и проверка работоспособности)

```
1   EXECUTE OPTIMIZE_ALL_TABLES
```

Логика установки запуска задачи по расписанию была рассмотрена в решении{408} задачи 5.2.2.a{408}, потому здесь просто приведём код, с помощью которого выполняется эта операция.

MS SQL Решение 5.2.2.b (установка запуска по расписанию)

```
1   USE msdb ;
2   GO
3   -- https://msdn.microsoft.com/en-us/library/ms182079.aspx
4   EXEC dbo.sp_add_job
5       @job_name = N'Hourly [books_statistics] update';
6   GO
7   -- https://msdn.microsoft.com/en-us/library/ms187358.aspx
8   EXEC sp_add_jobstep
9       @job_name = N'Hourly [books_statistics] update',
10      @step_name = N'Execute UPDATE_BOOKS_STATISTICS stored procedure',
11      @subsystem = N'TSQL',
12      @command = N'EXECUTE UPDATE_BOOKS_STATISTICS',
13      @database_name = N'library_ex_2015_mod';
14  GO
15  -- https://msdn.microsoft.com/en-us/library/ms187320.aspx
16  EXEC dbo.sp_add_schedule
17      @schedule_name = N'UpdateBooksStatistics',
18      @freq_type = 4,
19      @freq_interval = 4,
20      @freq_subday_type = 8,
21      @freq_subday_interval = 1,
22      @active_start_time = 000105 ;
23  USE msdb ;
24  GO
25  -- https://msdn.microsoft.com/en-us/library/ms186766.aspx
26  EXEC sp_attach_schedule
27     @job_name = N'Hourly [books_statistics] update',
28     @schedule_name = N'UpdateBooksStatistics';
29  GO
30  -- https://msdn.microsoft.com/en-us/library/ms178625.aspx
31  EXEC dbo.sp_add_jobserver
32      @job_name = N'Hourly [books_statistics] update';
33  GO
```

Убедиться, что событие добавлено, можно следующим запросом.

MS SQL Решение 5.2.2.b (просмотр расписания)

```
1   SELECT * FROM msdb.dbo.sysschedules
```

Если рассмотреть избранные поля результата такого запроса, получается следующая картина. Здесь также представлено событие, активирующее раз в час хранимую процедуру, созданную в процессе решения{408} задачи 5.2.2.a{408}.

Name	enabled	freq_ type	freq_ interval	freq_subday_ type	freq_subday_ interval	active_start_ date	active_start_ time
UpdateBooksStatistics	1	4	4	8	1	20160427	000105
DailyOptimizeAllTables	1	4	4	1	1	20160427	000100

На этом решение для MS SQL Server завершено.

Переходим к Oracle. Код создания нужной нам хранимой процедуры выглядит следующим образом.

Oracle Решение 5.2.2.b (код процедуры)

```
CREATE OR REPLACE PROCEDURE OPTIMIZE_ALL_TABLES
AS
 table_name VARCHAR(150) := '';
 query_text VARCHAR(1000) := '';
 CURSOR tables_cursor IS
  SELECT TABLE_NAME AS "table_name"
  FROM   ALL_TABLES
  WHERE OWNER=USER;
BEGIN
 FOR one_row IN tables_cursor
 LOOP
  query_text := 'ALTER TABLE "' || one_row."table_name" ||
                '" ENABLE ROW MOVEMENT';
  DBMS_OUTPUT.PUT_LINE('Enabling row movement for "' ||
                       one_row."table_name" || '"...');
  EXECUTE IMMEDIATE query_text;

  query_text := 'ALTER TABLE "' || one_row."table_name" ||
                '" SHRINK SPACE COMPACT CASCADE';
  DBMS_OUTPUT.PUT_LINE('Performing SHRINK SPACE COMPACT CASCADE on "' ||
                       one_row."table_name" || '"...');
  EXECUTE IMMEDIATE query_text;

  query_text := 'ALTER TABLE "' || one_row."table_name" ||
                '" DISABLE ROW MOVEMENT';
  DBMS_OUTPUT.PUT_LINE('Disabling row movement for "' ||
                       one_row."table_name" || '"...');
  EXECUTE IMMEDIATE query_text;
 END LOOP;
END;
/
```

Концепция представленного решения основана на этой рекомендации[26]. Итак, мы должны будем получить список всех таблиц, а затем для каждой из них сначала разрешить перемещение рядов, потом выполнить компактификацию и, наконец, снова запретить перемещение рядов. В отличие от MS SQL Server все запросы здесь совершенно тривиальны.

Запустить полученную хранимую процедуру можно следующим образом.

Oracle Решение 5.2.2.b (запуск и проверка работоспособности)

```
SET SERVEROUTPUT ON;
EXECUTE OPTIMIZE_ALL_TABLES;
```

[26] https://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:17312316112393#1765387500346472492

Теперь остаётся создать событие, запускаемое планировщиком задач Oracle по определённому расписанию. Это реализуется следующим кодом.

Oracle Решение 5.2.2.b (установка запуска по расписанию)

```
1  BEGIN
2    DBMS_SCHEDULER.CREATE_JOB (
3      job_name            =>  'daily_optimize_all_tables',
4      job_type            =>  'STORED_PROCEDURE',
5      job_action          =>  'OPTIMIZE_ALL_TABLES',
6      start_date          =>  '25-APR-16 4.00.00 PM',
7      repeat_interval     =>  'FREQ=DAILY;INTERVAL=1',
8      auto_drop           =>  FALSE,
9      enabled             =>  TRUE);
10 END;
```

Убедиться, что событие добавлено, можно следующим запросом.

Oracle Решение 5.2.2.b (просмотр расписания)

```
1  SELECT * FROM ALL_SCHEDULER_JOBS WHERE OWNER=USER
```

Если рассмотреть избранные поля результата такого запроса, получается следующая картина. Здесь также представлено событие, активирующее раз в час хранимую процедуру, созданную в процессе решения{408} задачи 5.2.2.a{408}.

JOB_NAME	JOB_ STYLE	JOB_TYPE	JOB_ ACTION	START_DATE	REPEAT_ INTERVAL	ENABLED	STATE
DAILY_ OPTIMIZE_ ALL_TABLES	REGULAR	STORED_ PROCEDURE	OPTIMIZE_ ALL_ TABLES	25-APR-27 04.00.00.000000000 PM -03:00	FREQ=DAILY; INTERVAL=1	TRUE	SCHE- DULED
HOURLY_ UPDATE_ BOOKS_ STATISTICS	REGULAR	STORED_ PROCEDURE	UPDATE_ BOOKS_ STATISTICS	22-APR-27 04.00.00.000000000 PM -03:00	FREQ=HOURLY; INTERVAL=1	TRUE	SCHE- DULED

На этом решение данной задачи завершено.

Задание 5.2.2.TSK.A: создать хранимую процедуру, запускаемую по расписанию каждые 12 часов и обновляющую данные в агрегирующей таблице **`subscriptions_ready`** (см. задачу 3.1.2.b{223}).

Задание 5.2.2.TSK.B: создать хранимую процедуру, запускаемую по расписанию раз в неделю и оптимизирующую (дефрагментирующую, компактифицирующую) все таблицы базы данных, в которых находится не менее одного миллиона записей.

5.2.3. ПРИМЕР 40: УПРАВЛЕНИЕ СТРУКТУРАМИ БАЗЫ ДАННЫХ С ПОМОЩЬЮ ХРАНИМЫХ ПРОЦЕДУР

Задача 5.2.3.a{421}: создать хранимую процедуру, автоматически создающую и наполняющую данными агрегирующую таблицу **`books_statistics`** (см. задачу 3.1.2.a{223}).

Задача 5.2.3.b{425}: создать хранимую процедуру, автоматически создающую и наполняющую данными агрегирующую таблицу **`tables_rc`**, содержащую информацию о количестве записей во всех таблицах базы данных в формате (**`имя_таблицы, количество_записей`**).

Ожидаемый результат 5.2.3.a.

При вызове хранимой процедуры таблица **books_statistics** создаётся и наполняется данными. Если таблица уже существует на момент вызова хранимой процедуры, данные в ней обновляются (приводятся в актуальное состояние).

Пример содержимого таблицы **books_statistics** см. в задаче 3.1.2.a{223}.

Ожидаемый результат 5.2.3.b.

При вызове хранимой процедуры таблица **tables_rc** создаётся и наполняется данными. Если таблица уже существует на момент вызова хранимой процедуры, данные в ней обновляются (приводятся в актуальное состояние).

Пример содержимого таблицы **tables_rc**:

table_name	rows_count
authors	7
books	7
books_statistics	1
genres	6
m2m_books_authors	9
m2m_books_genres	11
subscribers	4
subscriptions	11
tables_rc	8

Решение 5.2.3.a{420}.

В решении{385} задачи 5.1.1.c{369} уже содержатся готовые запросы для создания таблицы **books_statistics**, наполнения её данными и обновления её данных. Сейчас нам остаётся только разместить эти запросы внутри хранимой процедуры и добавить проверку существования самой таблицы **books_statistics**.

Решения для MySQL и MS SQL Server будут полностью идентичными, а в Oracle нам придётся все запросы выполнять через **EXECUTE IMMEDIATE**, т.к. в случае отсутствия таблицы **books_statistics** эта СУБД автоматически считает хранимую процедуру некорректной, если в ней напрямую прописаны запросы, обращающиеся к этой таблице.

Теперь остаётся только привести код.

Решение для MySQL выглядит следующим образом.

MySQL Решение 5.2.3.a (код процедуры)

```
1  DELIMITER $$
2  CREATE PROCEDURE CREATE_BOOKS_STATISTICS()
3  BEGIN
4
```

MySQL Решение 5.2.3.a (код процедуры) (продолжение)

```
5    IF NOT EXISTS
6     (SELECT `table_name`
7      FROM   `information_schema`.`tables`
8      WHERE  `table_schema` = DATABASE()
9      AND    `table_type` = 'BASE TABLE'
10     AND    `table_name` = 'books_statistics')
11   THEN
12    CREATE TABLE `books_statistics`
13     (
14        `total` INTEGER UNSIGNED NOT NULL,
15        `given` INTEGER UNSIGNED NOT NULL,
16        `rest`  INTEGER UNSIGNED NOT NULL
17     );
18    INSERT INTO `books_statistics`
19                (`total`,
20                 `given`,
21                 `rest`)
22    SELECT IFNULL(`total`, 0),
23           IFNULL(`given`, 0),
24           IFNULL(`total` - `given`, 0) AS `rest`
25    FROM   (SELECT (SELECT SUM(`b_quantity`)
26                    FROM   `books`)              AS `total`,
27                   (SELECT COUNT(`sb_book`)
28                    FROM   `subscriptions`
29                    WHERE  `sb_is_active` = 'Y') AS `given`)
30           AS `prepared_data`;
31   ELSE
32    UPDATE `books_statistics`
33    JOIN
34    (SELECT IFNULL(`total`, 0) AS `total`,
35            IFNULL(`given`, 0) AS `given`,
36            IFNULL(`total` - `given`, 0) AS `rest`
37     FROM   (SELECT (SELECT SUM(`b_quantity`)
38                     FROM   `books`)              AS `total`,
39                    (SELECT COUNT(`sb_book`)
40                     FROM   `subscriptions`
41                     WHERE  `sb_is_active` = 'Y') AS `given`)
42            AS `prepared_data`) AS `src`
43    SET `books_statistics`.`total` = `src`.`total`,
44        `books_statistics`.`given` = `src`.`given`,
45        `books_statistics`.`rest` = `src`.`rest`;
46   END IF;
47  END;
48  $$
49  DELIMITER ;
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

MySQL Решение 5.2.3.a (проверка работоспособности)

```
1   DROP TABLE `books_statistics`;
2   CALL CREATE_BOOKS_STATISTICS;
3   SELECT * FROM `books_statistics`;
```

Решение для MS SQL Server выглядит следующим образом.

MS SQL Решение 5.2.3.a (код процедуры)

```
CREATE PROCEDURE CREATE_BOOKS_STATISTICS
AS
BEGIN
 IF NOT EXISTS
  (SELECT [name]
   FROM   sys.tables
   WHERE  [name] = 'books_statistics')
 BEGIN
  CREATE TABLE [books_statistics]
  (
   [total] INTEGER NOT NULL,
   [given] INTEGER NOT NULL,
   [rest]  INTEGER NOT NULL
  );
  INSERT INTO [books_statistics]
              ([total],
               [given],
               [rest])
  SELECT ISNULL([total], 0) AS [total],
         ISNULL([given], 0) AS [given],
         ISNULL([total] - [given], 0) AS [rest]
  FROM   (SELECT (SELECT SUM([b_quantity])
                  FROM   [books])              AS [total],
                 (SELECT COUNT([sb_book])
                  FROM   [subscriptions]
                  WHERE  [sb_is_active] = 'Y') AS [given])
         AS [prepared_data];

  END
   ELSE
  BEGIN
   UPDATE [books_statistics]
   SET
    [books_statistics].[total] = [src].[total],
    [books_statistics].[given] = [src].[given],
    [books_statistics].[rest] = [src].[rest]
    FROM [books_statistics]
   JOIN
   (SELECT ISNULL([total], 0) AS [total],
           ISNULL([given], 0) AS [given],
           ISNULL([total] - [given], 0) AS [rest]
    FROM   (SELECT (SELECT SUM([b_quantity])
                    FROM   [books])              AS [total],
                   (SELECT COUNT([sb_book])
                    FROM   [subscriptions]
                    WHERE  [sb_is_active] = 'Y') AS [given])
           AS [prepared_data]
   ) AS [src]
   ON 1=1;
  END;
END;
GO
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

MS SQL Решение 5.2.3.a (проверка работоспособности)

```
DROP TABLE [books_statistics];
EXECUTE CREATE_BOOKS_STATISTICS;
SELECT * FROM [books_statistics];
```

Решение для Oracle выглядит следующим образом. Обратите внимание на то, как реализована проверка существования таблицы **books_statistics**: т.к. Oracle не поддерживает вариант с **IF NOT EXISTS**, мы вынуждены поместить в переменную количество найденных рядов и затем проверить его значение в блоке **IF**.

Oracle Решение 5.2.3.a (код процедуры)

```
CREATE OR REPLACE PROCEDURE CREATE_BOOKS_STATISTICS
AS
 table_found NUMBER(1) :=0;
BEGIN

 SELECT COUNT(1) INTO table_found
 FROM   ALL_TABLES
 WHERE  OWNER=USER
 AND    TABLE_NAME = 'books_statistics';

 IF (table_found = 0)
 THEN
  EXECUTE IMMEDIATE 'CREATE TABLE "books_statistics"
   (
       "total" NUMBER(10),
       "given" NUMBER(10),
       "rest"  NUMBER(10)
   )';

  EXECUTE IMMEDIATE 'INSERT INTO "books_statistics"
              ("total",
               "given",
               "rest")
   SELECT "total",
          "given",
          ("total" - "given") AS "rest"
   FROM   (SELECT SUM("b_quantity") AS "total"
           FROM   "books")
     JOIN (SELECT COUNT("sb_book") AS "given"
           FROM   "subscriptions"
           WHERE  "sb_is_active" = ''Y'')
     ON 1 = 1';

  ELSE
    EXECUTE IMMEDIATE 'UPDATE "books_statistics"
    SET ("total", "given", "rest") =
    (SELECT "total",
            "given",
            ("total" - "given") AS "rest"
```

Oracle Решение 5.2.3.a (код процедуры) (продолжение)

```
40       FROM   (SELECT SUM("b_quantity") AS "total"
41               FROM   "books")
42         JOIN (SELECT COUNT("sb_book") AS "given"
43               FROM   "subscriptions"
44               WHERE  "sb_is_active" = ''Y'')
45         ON 1 = 1)';
46  END IF;
47 END;
48 /
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

Oracle Решение 5.2.3.a (проверка работоспособности)

```
1  DROP TABLE "books_statistics";
2  EXECUTE CREATE_BOOKS_STATISTICS;
3  SELECT * FROM "books_statistics";
```

На этом решение данной задачи завершено.

Решение 5.2.3.b{420}.

Логика решения этой задачи представляет собой комбинацию подходов, представленных в решениях{413}, {421} задач 5.2.2.b{408} и 5.2.3.a{420}. Мы будем:

- проверять существование целевой таблицы **tables_rc**;
- создавать её, если таковой не обнаружилось;
- получать список таблиц базы данных и для каждой из них выполнять требуемую операцию — получение количества рядов и добавление этого количества вместе с именем анализируемой таблицы в агрегирующую таблицу **tables_rc**.

Важно отметить, что здесь для упрощения кода мы выполняем операцию **TRUNCATE** с последующим добавлением рядов. Это избавляет нас от необходимости реализовывать алгоритм с добавлением информации о новых таблицах, удалением информации о старых и обновлением информации о существующих.

Однако в реальных проектных задачах некий код может не ожидать, что в таблице **tables_rc** нет данных (что наблюдается в промежуток времени между выполнением операции **TRUNCATE** и проходом по циклу наполнения), и это может привести к возникновению ошибок.

Решение для MySQL выглядит следующим образом.

MySQL Решение 5.2.3.b (код процедуры)

```
1  DELIMITER $$
2  CREATE PROCEDURE CACHE_TABLES_RC()
3  BEGIN
4   DECLARE done INT DEFAULT 0;
5   DECLARE tbl_name VARCHAR(200) DEFAULT '';
6   DECLARE all_tables_cursor CURSOR FOR
7    SELECT `table_name`
8    FROM   `information_schema`.`tables`
9    WHERE  `table_schema` = DATABASE()
10   AND    `table_type` = 'BASE TABLE';
11  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
12
```

MySQL Решение 5.2.3.b (код процедуры) (продолжение)

```
13   IF NOT EXISTS
14    (SELECT `table_name`
15     FROM   `information_schema`.`tables`
16     WHERE  `table_schema` = DATABASE()
17     AND    `table_type` = 'BASE TABLE'
18     AND    `table_name` = 'tables_rc')
19   THEN
20    CREATE TABLE `tables_rc`
21    (
22     `table_name` VARCHAR(200),
23     `rows_count` INT
24    );
25   END IF;
26
27   TRUNCATE TABLE `tables_rc`;
28
29   OPEN all_tables_cursor;
30   tables_loop: LOOP
31    FETCH all_tables_cursor INTO tbl_name;
32    IF done THEN
33     LEAVE tables_loop;
34    END IF;
35
36    SET @table_rc_query = CONCAT('SELECT COUNT(1) INTO @tbl_rc FROM `',
37                                 tbl_name, '`');
38    PREPARE table_opt_stmt FROM @table_rc_query;
39    EXECUTE table_opt_stmt;
40    DEALLOCATE PREPARE table_opt_stmt;
41
42    INSERT INTO `tables_rc` (`table_name`,
43                             `rows_count`)
44                     VALUES (tbl_name,
45                             @tbl_rc);
46
47   END LOOP tables_loop;
48
49   CLOSE all_tables_cursor;
50  END;
51  $$
52  DELIMITER ;
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

MySQL Решение 5.2.3.b (проверка работоспособности)

```
1   CALL CACHE_TABLES_RC;
2   SELECT * FROM `tables_rc`;
```

Решение для MS SQL Server выглядит следующим образом.

MS SQL Решение 5.2.3.b (код процедуры)

```
1   CREATE PROCEDURE CACHE_TABLES_RC
2   AS
3   BEGIN
4    DECLARE @table_name NVARCHAR(200);
5    DECLARE @table_rows INT;
6    DECLARE @query_text NVARCHAR(2000);
7    DECLARE tables_cursor CURSOR LOCAL FAST_FORWARD FOR
```

MS SQL Решение 5.2.3.b (код процедуры) (продолжение)

```
8      SELECT [name]
9      FROM   sys.tables;
10
11    IF NOT EXISTS
12     (SELECT [name]
13      FROM   sys.tables
14      WHERE  [name] = 'tables_rc')
15    BEGIN
16     CREATE TABLE [tables_rc]
17     (
18      [table_name] VARCHAR(200),
19      [rows_count] INT
20     );
21    END;
22
23    TRUNCATE TABLE [tables_rc];
24
25    OPEN tables_cursor;
26    FETCH NEXT FROM tables_cursor INTO @table_name;
27    WHILE @@FETCH_STATUS = 0
28     BEGIN
29      SET @query_text = CONCAT('SELECT @cnt = COUNT(1) FROM [',
30                               @table_name, ']');
31      EXECUTE sp_executesql @query_text, N'@cnt INT OUT', @table_rows OUTPUT;
32
33      INSERT INTO [tables_rc] ([table_name],
34                               [rows_count])
35                       VALUES (@table_name,
36                                       @table_rows);
37
38      FETCH NEXT FROM tables_cursor INTO @table_name;
39     END;
40    CLOSE tables_cursor;
41    DEALLOCATE tables_cursor;
42  END;
43  GO
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

MS SQL Решение 5.2.3.b (проверка работоспособности)

```
1   EXECUTE CACHE_TABLES_RC;
2   SELECT * FROM [tables_rc];
```

Решение для Oracle выглядит следующим образом.

Oracle Решение 5.2.3.b (код процедуры)

```
1   CREATE OR REPLACE PROCEDURE CACHE_TABLES_RC
2   AS
3    table_name VARCHAR(150) := '';
4    table_rows NUMBER(10) := 0;
5    table_found NUMBER(1) :=0;
6    query_text VARCHAR(1000) := '';
7    CURSOR tables_cursor IS
8     SELECT TABLE_NAME AS "table_name"
9     FROM   ALL_TABLES
10    WHERE OWNER=USER;
```

Oracle Решение 5.2.3.b (код процедуры) (продолжение)

```
CREATE OR REPLACE PROCEDURE CACHE_TABLES_RC
AS
 table_name VARCHAR(150) := '';
 table_rows NUMBER(10) := 0;
 table_found NUMBER(1) :=0;
 query_text VARCHAR(1000) := '';
 CURSOR tables_cursor IS
  SELECT TABLE_NAME AS "table_name"
  FROM   ALL_TABLES
  WHERE OWNER=USER;
BEGIN
 SELECT COUNT(1) INTO table_found
 FROM   ALL_TABLES
 WHERE  OWNER=USER
 AND    TABLE_NAME = 'tables_rc';

 IF (table_found = 0)
 THEN
  EXECUTE IMMEDIATE 'CREATE TABLE "tables_rc"
  ("table_name" VARCHAR(200),
   "rows_count" NUMBER(10))';
 END IF;
 EXECUTE IMMEDIATE 'TRUNCATE TABLE "tables_rc"';

 FOR one_row IN tables_cursor
 LOOP
  query_text := 'SELECT COUNT(1) FROM "' || one_row."table_name" ||
                '"';
  EXECUTE IMMEDIATE query_text INTO table_rows;

  query_text := 'INSERT INTO "tables_rc" ("table_name", "rows_count")
                 VALUES (''' || one_row."table_name" || ''', ' ||
                 table_rows || ')';
  EXECUTE IMMEDIATE query_text;
 END LOOP;
END;
/
```

Для проверки корректности полученного решения можно выполнить следующие запросы.

Oracle Решение 5.2.3.b (проверка работоспособности)

```
EXECUTE CACHE_TABLES_RC;
SELECT * FROM "tables_rc";
```

Задание 5.2.3.TSK.A: создать хранимую процедуру, автоматически создающую и наполняющую данными таблицу **arrears**, в которой должны быть представлены идентификаторы и имена читателей, у которых до сих пор находится на руках хотя бы одна книга, по которой дата возврата установлена в прошлом относительно текущей даты. Эта таблица должна быть связана с таблицей **subscriptions** связью «один к одному».

Задание 5.2.3.TSK.B: создать хранимую процедуру, удаляющую все индексы (кроме первичных ключей), построенные на таблицах текущей базы данных и включающие в себя более одного поля.

Задание 5.2.3.TSK.C: создать хранимую процедуру, удаляющую все представления, для которых **SELECT COUNT(1) FROM представление** возвращает значение меньше десяти.

ИСПОЛЬЗОВАНИЕ ТРАНЗАКЦИЙ

УПРАВЛЕНИЕ НЕЯВНЫМИ И ЯВНЫМИ ТРАНЗАКЦИЯМИ

6.1.1. ПРИМЕР 41: УПРАВЛЕНИЕ НЕЯВНЫМИ ТРАНЗАКЦИЯМИ

Задача 6.1.1.a{430}: продемонстрировать поведение СУБД при выполнении операций модификации данных в случаях, когда режим автоподтверждения неявных транзакций включён и выключен.

Задача 6.1.1.b{434}: создать хранимую процедуру, выполняющую следующие действия:

- определяющую, включён ли режим автоподтверждения неявных транзакций;
- выключающую этот режим, если требуется (если в процедуру передан соответствующий параметр с соответствующим значением);
- выполняющую вставку N записей в таблицу `subscribers` (N передаётся в процедуру соответствующим параметром);
- восстанавливающую исходное значение режима автоподтверждения неявных транзакций (если оно было изменено);
- возвращающую время, затраченное на выполнение вставки.

Ожидаемый результат 6.1.1.a.

При включённом режиме автоподтверждения неявных транзакций модификация данных немедленно фиксируется, при выключенном режиме автоподтверждения неявных транзакций модификация данных не вступает в силу до момента явного подтверждения транзакции.

Ожидаемый результат 6.1.1.b.

Хранимая процедура выводит отладочные сообщения по всем шагам описанного в задании алгоритма и после завершения своей работы возвращает информацию о количестве затраченного на выполнение операции вставки времени.

Решение 6.1.1.a{429}.

Режим автоподтверждения неявных транзакций актуален для MySQL[27] и MS SQL Server[28, 29] (и не актуален для Oracle[30], где данное поведение полностью отдано на усмотрение клиентского ПО) в случае, когда операции не обрамляются явным образом выражениями по запуску и подтверждению или отмене транзакций.

MySQL по умолчанию работает с включённым автоподтверждением неявных транзакций, т.е. любые изменения данных сразу же вступают в силу. За изменение данного поведения отвечает параметр **`autocommit`** (которым можно управлять локально на протяжении сессии или глобально, изменив соответствующую настройку в конфигурационном файле).

Для решения данной задачи в MySQL необходимо использовать следующий набор запросов.

MySQL Решение 6.1.1.a

```
1  -- Автоподтверждение выключено:
2  SET autocommit = 0;
3
4  SELECT COUNT(*)
5  FROM   `subscribers`; -- 4
6
7  INSERT INTO `subscribers`
8              (`s_name`)
9  VALUES      ('Иванов И.И.');
10
11 SELECT COUNT(*)
12 FROM   `subscribers`; -- 5
13
14 ROLLBACK;
15
16 SELECT COUNT(*)
17 FROM   `subscribers`; -- 4
18
19 -- Автоподтверждение включено:
20 SET autocommit = 1;
21
22 SELECT COUNT(*)
23 FROM   `subscribers`; -- 4
24
25 INSERT INTO `subscribers`
26             (`s_name`)
27 VALUES      ('Иванов И.И.');
28
```

27 http://dev.mysql.com/doc/refman/5.6/en/commit.html

28 https://technet.microsoft.com/en-us/library/ms190230%28v=sql.105%29.aspx

29 https://msdn.microsoft.com/en-us/library/ms187807.aspx

30 https://asktom.oracle.com/pls/apex/f?p=100:11:0%3A%3A%3A%3AP11_QUESTION_ID:314816776423

MySQL Решение 6.1.1.a (продолжение)

```
29 SELECT COUNT(*)
30 FROM   `subscribers`; -- 5
31
32 ROLLBACK;
33
34 SELECT COUNT(*)
35 FROM   `subscribers`; -- 5
```

В строках 1-17 запросы выполняются в режиме отключённого автоподтверждения неявных транзакций: именно поэтому отмена транзакции в строке 14 проходит успешно и вставка данных, выполненная в строках 7-9, аннулируется.

В строках 19-35 запросы выполняются в режиме включённого автоподтверждения неявных транзакций, и потому отмена транзакции в строке 32 ни на что не влияет: вставка данных, выполненная в строках 25-27, остаётся в силе.

MS SQL Server (как и MySQL) по умолчанию работает с включённым автоподтверждением неявных транзакций, т.е. любые изменения данных сразу же вступают в силу. За изменение данного поведения отвечает параметр **IMPLICIT_TRANSACTIONS** (которым в общем случае можно управлять только локально на протяжении сессии; общие идеи по управлению этим параметром на уроне настроек описаны здесь[31]).

Для решения данной задачи в MS SQL Server необходимо использовать следующий набор запросов. Обратите внимание, что параметр **IMPLICIT_TRANSACTIONS** в MS SQL Server по своей логике противоположен параметру **autocommit** в MySQL (т.е. для выключения автоподтверждения неявных транзакций необходимо выполнить команду **SET IMPLICIT_TRANSACTIONS ON**).

MS SQL Решение 6.1.1.a

```
1  -- Автоподтверждение выключено:
2  SET IMPLICIT_TRANSACTIONS ON;
3
4  SELECT COUNT(*)
5  FROM   [subscribers]; -- 4
6
7  INSERT INTO [subscribers]
8              ([s_name])
9  VALUES      (N'Иванов И.И.');
10
11 SELECT COUNT(*)
12 FROM   [subscribers]; -- 5
13
14 ROLLBACK;
15
16 SELECT COUNT(*)
17 FROM   [subscribers]; -- 4
18
19 -- Автоподтверждение включено:
20 SET IMPLICIT_TRANSACTIONS OFF;
21
22 SELECT COUNT(*)
23 FROM   [subscribers]; -- 4
24
```

31 https://msdn.microsoft.com/en-us/library/ms176031%28SQL.90%29.aspx

MS SQL Решение 6.1.1.a (продолжение)

```
25 INSERT INTO [subscribers]
26             ([s_name])
27 VALUES      (N'Иванов И.И.');
28
29 SELECT COUNT(*)
30 FROM   [subscribers]; -- 5
31
32 ROLLBACK; -- Ошибка! Нет соответствующей транзакции, которую
33           -- можно было бы отменить.
34
35 SELECT COUNT(*)
36 FROM   [subscribers]; -- 5
```

В строках 1-17 запросы выполняются в режиме отключённого автоподтверждения неявных транзакций: именно поэтому отмена транзакции в строке 14 проходит успешно и вставка данных, выполненная в строках 7-9, аннулируется.

В строках 19-36 запросы выполняются в режиме включённого автоподтверждения неявных транзакций, и потому отмена транзакции в строке 32 ни на что не влияет: вставка данных, выполненная в строках 25-27, остаётся в силе.

В MS SQL Server существует одна важная особенность, которую необходимо учитывать. Если в режиме **IMPLICIT_TRANSACTIONS ON** использовать выражение **BEGIN TRANSACTION**, СУБД читает созданную транзакцию вложенной (**@@TRANCOUNT** принимает значение 2) и для успешного подтверждения её выполнения необходимо использовать выражение **COMMIT TRANSACTION** дважды. В противном случае вы рискуете или получить «подвисшую» транзакцию (которая так и не завершена), или потерять результаты модификации данных (если закроете соединение с СУБД). При этом **ROLLBACK TRANSACTION** работает в обоих режимах одинаково, отменяя все транзакции вне зависимости от глубины их вложенности.

Эта проблема усугубляется тем, что при отладке запросов в средствах наподобие MS SQL Server Management Studio вы, как правило, работаете в рамках одного и того же соединения, и вместо «подвисшей» транзакции получаете продолжение предыдущей (не закрытой ранее). Потому в большинстве случаев при отладке всё работает правильно, а в реальных приложениях поведение становится неверным.

Продемонстрируем только что описанное поведение MS SQL Server.

MS SQL Решение 6.1.1.a (демонстрация особенности работы MS SQL Server)

```
1  -- Режим по умолчанию
2  SET IMPLICIT_TRANSACTIONS OFF;
3  PRINT @@TRANCOUNT; -- 0
4  -- Старт первой ("родительской") транзакции
5  BEGIN TRANSACTION;
6   PRINT @@TRANCOUNT; -- 1
7   -- Старт второй ("дочерней") транзакции
8   BEGIN TRANSACTION;
9    PRINT @@TRANCOUNT; -- 2
10  -- Подтверждение второй ("дочерней") транзакции
11  COMMIT TRANSACTION;
12  PRINT @@TRANCOUNT; -- 1
13 -- Подтверждение первой ("родительской") транзакции
14 COMMIT TRANSACTION;
15 PRINT @@TRANCOUNT; -- 0
16
```

MS SQL Решение 6.1.1.a (демонстрация особенности работы MS SQL Server) (продолжение)

```
-- Режим "неявных транзакций"
SET IMPLICIT_TRANSACTIONS ON;
PRINT @@TRANCOUNT; -- 0
-- Старт первой ("родительской") транзакции
BEGIN TRANSACTION;
 PRINT @@TRANCOUNT; -- 2
 -- Старт второй ("дочерней") транзакции
 BEGIN TRANSACTION;
  PRINT @@TRANCOUNT; -- 3
 -- Подтверждение второй ("дочерней") транзакции
 COMMIT TRANSACTION;
 PRINT @@TRANCOUNT; -- 2
-- Подтверждение первой ("родительской") транзакции
COMMIT TRANSACTION;
PRINT @@TRANCOUNT; -- 1
-- Необходим ещё и этот COMMIT
COMMIT TRANSACTION;
PRINT @@TRANCOUNT; -- 0

-- Режим по умолчанию
SET IMPLICIT_TRANSACTIONS OFF;
-- Старт первой ("родительской") транзакции
BEGIN TRANSACTION;
 PRINT @@TRANCOUNT; -- 1
 -- Старт второй ("дочерней") транзакции
 BEGIN TRANSACTION;
  PRINT @@TRANCOUNT; -- 2
-- Отмена всех транзакций
ROLLBACK TRANSACTION;
PRINT @@TRANCOUNT; -- 0

-- Режим "неявных транзакций"
SET IMPLICIT_TRANSACTIONS ON;
-- Старт первой ("родительской") транзакции
BEGIN TRANSACTION;
 PRINT @@TRANCOUNT; -- 2
 -- Старт второй ("дочерней") транзакции
 BEGIN TRANSACTION;
  PRINT @@TRANCOUNT; -- 3
-- Отмена всех транзакций
ROLLBACK TRANSACTION;
PRINT @@TRANCOUNT; -- 0
```

Oracle (в отличие от MySQL и MS SQL Server) не оперирует такими понятиями, как «неявная транзакция» и её автоподтверждение. Эта СУБД лишь автоматически подтверждает текущую транзакцию в случае, если выполняется выражение, модифицирующее структуру базы данных.

Однако клиентское ПО, организующее взаимодействие с Oracle, может иметь свои собственные настройки, отвечающие за автоматическое подтверждение транзакций, не обрамлённых явно выражениями по запуску и подтверждению или отмене.

В таком средстве как Oracle SQL Developer, например, соответствующий эффект достигается выполнением команды **`SET AUTOCOMMIT ON / OFF`** (эффект которой эквивалентен изменению параметра **`autocommit`** в MySQL).

Для решения данной задачи в Oracle необходимо использовать следующий набор запросов.

Oracle Решение 6.1.1.a

```
1   -- Автоподтверждение выключено:
2   SET AUTOCOMMIT OFF;
3
4   SELECT COUNT(*)
5   FROM   "subscribers"; -- 4
6
7   INSERT INTO "subscribers"
8               ("s_name")
9   VALUES      (N'Иванов И.И.');
10
11  SELECT COUNT(*)
12  FROM   "subscribers"; -- 5
13
14  ROLLBACK;
15
16  SELECT COUNT(*)
17  FROM   "subscribers"; -- 4
18
19  -- Автоподтверждение включено:
20  SET AUTOCOMMIT ON;
21
22  SELECT COUNT(*)
23  FROM   "subscribers"; -- 4
24
25  INSERT INTO "subscribers"
26              ("s_name")
27  VALUES      (N'Иванов И.И.');
28
29  SELECT COUNT(*)
30  FROM   "subscribers"; -- 5
31
32  ROLLBACK;
33
34  SELECT COUNT(*)
35  FROM   "subscribers"; -- 5
```

В строках 1-17 запросы выполняются в режиме отключённого автоподтверждения неявных транзакций: именно поэтому отмена транзакции в строке 14 проходит успешно и вставка данных, выполненная в строках 7-9, аннулируется.

В строках 19-35 запросы выполняются в режиме включённого автоподтверждения неявных транзакций, и потому отмена транзакции в строке 32 ни на что не влияет: вставка данных, выполненная в строках 25-27, остаётся в силе.

На этом решение данной задачи завершено.

Решение 6.1.1.b{429}.

Данная задача призвана не только напомнить принципы работы с хранимыми процедурами и логику управления автоподтверждением неявных транзакций, она также демонстрирует разницу в производительности СУБД в ситуациях, когда при выполнении множества операций моди-

фикации данных каждая из них вступает в силу по-отдельности, и когда такие операции фиксируются по факту выполнения всей их группы целиком.

Традиционно мы начинаем решение с MySQL и сразу рассмотрим код.

MySQL Решение 6.1.1.b (код процедуры)

```
DELIMITER $$
CREATE PROCEDURE TEST_INSERT_SPEED(IN records_count INT,
                                   IN use_autocommit INT,
                                   OUT total_time TIME(6))
BEGIN
 DECLARE counter INT DEFAULT 0;

 SET @old_autocommit = (SELECT @@autocommit);
 SELECT CONCAT('Old autocommit value = ', @old_autocommit);
 SELECT CONCAT('New autocommit value = ', use_autocommit);

 IF (use_autocommit != @old_autocommit)
 THEN
  SELECT CONCAT('Switching autocommit to ', use_autocommit);
  SET autocommit = use_autocommit;
 ELSE
  SELECT 'No changes in autocommit mode needed.';
 END IF;

 SELECT CONCAT('Starting insert of ', records_count, ' records...');
 SET @start_time = (SELECT NOW(6));
 WHILE counter < records_count DO
  INSERT INTO   `subscribers`
                (`s_name`)
         VALUES (CONCAT('New subscriber ', (counter + 1)));
  SET counter = counter + 1;
 END WHILE;
 SET @finish_time = (SELECT NOW(6));
 SELECT CONCAT('Finished insert of ', records_count, ' records...');

 IF ((SELECT @@autocommit) = 0)
 THEN
  SELECT 'Current autocommit mode is 0. Performing explicit commit.';
  COMMIT;
 END IF;

 IF (use_autocommit != @old_autocommit)
 THEN
  SELECT CONCAT('Switching autocommit back to ', @old_autocommit);
  SET autocommit = @old_autocommit;
 ELSE
  SELECT 'No changes in autocommit mode were made. No restore needed.';
 END IF;

 SET total_time = (SELECT TIMEDIFF(@finish_time, @start_time));
 SELECT CONCAT('Time used: ', total_time);

 SELECT total_time;
END;
$$
DELIMITER ;
```

В строке 8 происходит определение текущего значения автоподтверждения неявных транзакций (в MySQL эту информацию можно извлечь из переменной **@@autocommit**).

В строках 12-18 происходит проверка необходимости изменения режима автоподтверждения неявных транзакций и само изменение (если это необходимо). В строках 37-34 происходит повторная проверка и возврат исходного значения, если оно было изменено.

Определение затраченного на выполнение операции вставки времени происходит за счёт получения текущего времени до (строка 21) и после (строка 28) выполнения цикла вставки (строки 22-27), а затем вычисления разности этих значений (строка 45).

В строках 31-35 проверяется текущее значение режима автоподтверждения неявных транзакций и подтверждение выполняется явным образом в строке 34, если автоподтверждение выключено (здесь нас не интересует, было ли оно выключено изначально или в процессе выполнения нашей процедуры).

Теперь остаётся только вернуть значение затраченного на выполнение цикла вставки времени как результат работы хранимой процедуры (строка 48).

Для проверки работоспособности и оценки производительности MySQL в двух режимах работы с неявными транзакциями можно использовать следующие запросы.

MySQL Решение 6.1.1.b (код для проверки работоспособности)

```
1  CALL TEST_INSERT_SPEED(100000, 1, @tmp);
2  SELECT @tmp;
3
4  CALL TEST_INSERT_SPEED(100000, 0, @tmp);
5  SELECT @tmp;
```

Вы можете самостоятельно произвести соответствующее исследование производительности. Здесь лишь отметим, что отключение автоподтверждения неявных транзакций может ускорить данную операцию вставки в десятки раз.

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Внутренняя логика хранимой процедуры будет очень похожа на решение для MySQL, и главным отличием будет лишь способ[32] определения режима автоподтверждения неявных транзакций. Т.к. данная СУБД не предоставляет эту информацию явным образом, мы попытаемся определить её косвенно.

Такое определение основано на информации об уровне вложенности текущей транзакции (**@@TRANCOUNT**) и настройках текущего соединения (**@@OPTIONS**). В строках 12-31 кода хранимой процедуры мы рассматриваем все возможные интересующие нас сочетания значений этих параметров, выводим отладочную информацию и определяем, включён ли режим подтверждения неявных транзакций.

В строках 36-45 мы определяем необходимость изменения режима автоподтверждения и меняем его, если это требуется.

В строках 47-57 совершенно аналогично с решением для MySQL выполняется цикл вставки указанного количества записей.

В строках 59-64 проверяется, в каком режиме запущена хранимая процедура (в случае с MySQL мы ориентировались на текущее значение переменной **@@autocommit**, но т.к. в MS SQL Server её нет, а определение текущего режима довольно нетривиально (см. строки 11-31), мы полагаем, что работа идёт в том режиме, который указан при вызове хранимой процедуры).

Остаётся только восстановить исходное значение **IMPLICIT_TRANSACTIONS** (строки 65-74) и определить, сколько времени заняло выполнение цикла вставки записей (строки 76-80). Такая громоздкая конструкция с использованием функций **CONVERT**, **DATEADD**, **DATEDIFF** необходима для того, чтобы получить на выходе затраченное время в удобной для человека форме.

[32] http://stackoverflow.com/questions/2919018/in-sql-server-how-do-i-know-what-transaction-mode-im-currently-using

Итак, вот код процедуры.

MS SQL Решение 6.1.1.b (код процедуры)

```
CREATE PROCEDURE TEST_INSERT_SPEED @records_count INT,
                                   @use_autocommit INT,
                                   @total_time TIME OUTPUT
AS
BEGIN
 DECLARE @counter INT = 0;
 DECLARE @old_autocommit INT = 0;
 DECLARE @start_time TIME;
 DECLARE @finish_time TIME;

 IF (@@TRANCOUNT = 0 AND (@@OPTIONS & 2 = 0))
  BEGIN
   PRINT 'IMPLICIT_TRANSACTIONS = OFF, no transaction is running.';
   SET @old_autocommit = 1;
  END
 ELSE IF (@@TRANCOUNT = 0 AND (@@OPTIONS & 2 = 2))
  BEGIN
   PRINT 'IMPLICIT_TRANSACTIONS = ON, no transaction is running.';
   SET @old_autocommit = 0;
  END
 ELSE IF (@@OPTIONS & 2 = 0)
  BEGIN
   PRINT 'IMPLICIT_TRANSACTIONS = OFF, explicit transaction is running.';
   SET @old_autocommit = 1;
  END
 ELSE
  BEGIN
   PRINT 'IMPLICIT_TRANSACTIONS = ON, implicit or explicit transaction
          is running.';
   SET @old_autocommit = 0;
  END;

 PRINT CONCAT('Old autocommit value = ', @old_autocommit);
 PRINT CONCAT('New autocommit value = ', @use_autocommit);

 IF (@use_autocommit != @old_autocommit)
  BEGIN
   PRINT CONCAT('Switching autocommit to ', @use_autocommit);
   IF (@use_autocommit = 1)
    SET IMPLICIT_TRANSACTIONS OFF;
   ELSE
    SET IMPLICIT_TRANSACTIONS ON;
  END
 ELSE
  PRINT 'No changes in autocommit mode needed.';

 PRINT CONCAT('Starting insert of ', @records_count, ' records...');
 SET @start_time = GETDATE();
 WHILE (@counter < @records_count)
  BEGIN
   INSERT INTO [subscribers]
               ([s_name])
        VALUES (CONCAT('New subscriber ', (@counter + 1)));
   SET @counter = @counter + 1;
  END;
```

MS SQL Решение 6.1.1.b (код процедуры) (продолжение)

```
56   SET @finish_time = GETDATE();
57   PRINT CONCAT('Finished insert of ', @records_count, ' records...');
58
59   IF (@use_autocommit = 0)
60    BEGIN
61     PRINT 'Current autocommit mode is 0 (IMPLICIT_TRANSACTIONS = ON).
62            Performing explicit commit.';
63     COMMIT;
64    END;
65   IF (@use_autocommit != @old_autocommit)
66    BEGIN
67     PRINT CONCAT('Switching autocommit back to ', @old_autocommit);
68     IF (@old_autocommit = 1)
69      SET IMPLICIT_TRANSACTIONS OFF;
70     ELSE
71      SET IMPLICIT_TRANSACTIONS ON;
72    END
73   ELSE
74    PRINT 'No changes in autocommit mode needed.';
75
76   SET @total_time = CONVERT(VARCHAR(12),
77                             DATEADD(ms,
78                                  DATEDIFF(ms, @start_time, @finish_time),
79                                     0),
80                             114);
81   PRINT CONCAT('Time used: ', @total_time);
82   RETURN;
83  END;
84  GO
```

Для проверки работоспособности и оценки производительности MS SQL Server в двух режимах работы с неявными транзакциями можно использовать следующие запросы.

MS SQL Решение 6.1.1.b (код для проверки работоспособности)

```
1   DECLARE @t TIME;
2   SET IMPLICIT_TRANSACTIONS ON
3   EXECUTE TEST_INSERT_SPEED 10, 1, @t OUTPUT;
4   PRINT CONCAT ('Stored procedure has returned the following value: ', @t);
5
6   DECLARE @t TIME;
7   SET IMPLICIT_TRANSACTIONS ON
8   EXECUTE TEST_INSERT_SPEED 10, 0, @t OUTPUT;
9   PRINT CONCAT ('Stored procedure has returned the following value: ', @t);
```

На этом решение для MySQL завершено.

Переходим к Oracle. Поскольку данная СУБД вообще не оперирует таким понятием как «автоподтверждение неявных транзакций», мы можем работать лишь в одном из двух режимов (который явно выбираем и реализуем сами):

- выполнение подтверждения транзакции после каждой операции (аналог включённого автоподтверждения неявных транзакций);
- выполнение подтверждения транзакции после серии операций (аналог выключенного автоподтверждения неявных транзакций)

Учитывая этот факт, мы реализуем решение для Oracle по аналогии с MySQL и MS SQL Server, но без определения текущего режима автоподтверждения транзакций.

Поскольку взаимодействие Oracle с клиентским ПО всегда происходит в режиме транзакции (т.е. выполнение любого выражения по выборке или модификации данных происходит внутри транзакции), в начале работы нашей хранимой процедуры мы должны выполнить операцию **COMMIT** (строка 12), чтобы завершить текущую транзакцию (если она есть).

В строках 16-27 мы выполняем цикл вставки, в котором мы можем явно инициировать подтверждение вставки каждой отдельной записи (строка 24), если нам нужно эмулировать режим автоподтверждения неявных транзакций. Если такая эмуляция не нужна, то все выполненные в цикле вставки подтверждаются как набор операций (строка 35).

Oracle Решение 6.1.1.b (код процедуры)

```
1  CREATE OR REPLACE PROCEDURE TEST_INSERT_SPEED(records_count IN INT,
2                                                use_autocommit IN INT,
3                                                total_time OUT NVARCHAR2)
4  AS
5   counter INT := 0;
6   start_time TIMESTAMP;
7   finish_time TIMESTAMP;
8   diff_time INTERVAL DAY TO SECOND;
9  BEGIN
10  DBMS_OUTPUT.PUT_LINE('Autocommit value = ' || use_autocommit);
11  DBMS_OUTPUT.PUT_LINE('Committing previous transaction...');
12  COMMIT;
13  DBMS_OUTPUT.PUT_LINE('Starting insert of ' || records_count ||
14                       ' records...');
15  start_time := CURRENT_TIMESTAMP;
16  WHILE (counter < records_count)
17   LOOP
18    INSERT INTO "subscribers"
19                ("s_name")
20         VALUES (CONCAT('New subscriber ', (counter + 1)));
21    IF (use_autocommit = 1)
22     THEN
23      DBMS_OUTPUT.PUT_LINE('Committing small transaction...');
24      COMMIT;
25     END IF;
26    counter := counter + 1;
27   END LOOP;
28  finish_time := CURRENT_TIMESTAMP;
29  DBMS_OUTPUT.PUT_LINE('Finished insert of ' || records_count ||
30                       ' records...');
31
32  IF (use_autocommit = 0)
33  THEN
34   DBMS_OUTPUT.PUT_LINE('Committing one big transaction...');
35   COMMIT;
36  END IF;
37
38  diff_time := finish_time - start_time;
39  total_time := TO_CHAR(EXTRACT(hour FROM diff_time)) || ':' ||
40                TO_CHAR(EXTRACT(minute FROM diff_time)) || ':' ||
41                TO_CHAR(EXTRACT(second FROM diff_time ), 'fm00.000000' );
42
43  DBMS_OUTPUT.PUT_LINE('Time used: ' || total_time);
44 END;
```

Для проверки работоспособности и оценки производительности Oracle в двух режимах работы с неявными транзакциями можно использовать следующие запросы.

Oracle Решение 6.1.1.b (код для проверки работоспособности)

```
1   DECLARE
2    t NVARCHAR2(100);
3   BEGIN
4    TEST_INSERT_SPEED(10, 1, t);
5    DBMS_OUTPUT.PUT_LINE('Stored procedure has returned
6                          The following value: ' || t);
7   END;
8
9   DECLARE
10   t NVARCHAR2(100);
11  BEGIN
12   TEST_INSERT_SPEED(10, 0, t);
13   DBMS_OUTPUT.PUT_LINE('Stored procedure has returned
14                         The following value: ' || t);
15  END;
```

На этом решение данной задачи завершено.

Задание 6.1.1.TSK.A: сравнить скорость работы представленной в решении{434} задачи 6.1.1.b{429} хранимой процедуры при вставке в обоих режимах автоподтверждения неявных транзакций для 10, 100, 1000, 10000, 100000 записей во всех трёх СУБД.

6.1.2. ПРИМЕР 42: УПРАВЛЕНИЕ ЯВНЫМИ ТРАНЗАКЦИЯМИ

Задача 6.1.2.a{441}: создать хранимую процедуру, которая:

- добавляет каждому читателю три случайных книги с датой выдачи, равной текущей дате, и датой возврата, равной «текущая дата плюс месяц»;
- отменяет совершённые действия, если по итогу выполнения операции хотя бы у одного читателя на руках окажется более десяти книг.

Задача 6.1.2.b{446}: создать хранимую процедуру, которая:

- изменяет все даты возврата книг на «плюс три месяца»;
- отменяет совершённое действие, если по итогу выполнения операции среднее время чтения книги превысит 4 месяца.

Ожидаемый результат 6.1.2.a.

Хранимая процедура выполняет указанные в условии задачи действия, в конце своей работы сигнализируя о фиксации или отмене внесённых изменений.

Ожидаемый результат 6.1.2.b.

Хранимая процедура выполняет указанные в условии задачи действия, в конце своей работы сигнализируя о фиксации или отмене внесённых изменений.

Решение 6.1.2.a{440}.

Решение данной задачи для MySQL будет примечательно рассмотрением способа работы с несколькими курсорами внутри одной хранимой процедуры.

Для получения решения мы будем должны:

- запустить транзакцию (строка 5);
- открыть курсор для извлечения идентификаторов всех читателей (строка 15);
- для каждого идентификатора читателя выполнить вложенный цикл (строки 16-55), в котором:
  - открыть курсор для извлечения трёх идентификаторов случайных книг (строка 13);
  - для каждого полученного идентификатора книги произвести вставку в таблицу выдач книг (строки 33-51);
  - закрыть курсор для извлечения трёх идентификаторов случайных книг (строка 52);
- закрыть курсор для извлечения идентификаторов всех читателей (строка 56);
- проверить, было ли нарушено условие о недопустимости нахождения на руках у одного читателя более десяти книг (строки 58-70) и:
  - если условие было нарушено, отменить транзакцию (строка 66);
  - если условие не было нарушено, подтвердить транзакцию (строка 69).

Несмотря на громоздкость синтаксиса и длительное описание, сам алгоритм тривиален: это обычный вложенный цикл. Что в этой задаче представляет интерес, так это уже упомянутая ранее работа с двумя курсорами.

Обратите внимание, что конструкция **`DECLARE CONTINUE HANDLER FOR NOT FOUND SET`** не подразумевает указание имени курсора, т.е. предполагается, что он один. В ситуациях, когда необходимо использовать несколько курсоров, используются т.н. «блоки кода», ограничивающие область видимости переменных. В нашем случае таких блоков два, второй вложен в первый, и расположены они в строках 7-57 и 22-53 соответственно).

MySQL Решение 6.1.2.a (код процедуры)

```
1  DELIMITER $$
2  CREATE PROCEDURE THREE_RANDOM_BOOKS()
3  BEGIN
4   SELECT 'Starting transaction...';
5   START TRANSACTION;
6
7   USERS: BEGIN
8    DECLARE s_id_value INT DEFAULT 0;
9    DECLARE subscribers_done INT DEFAULT 0;
10   DECLARE subscribers_cursor CURSOR FOR
11    SELECT `s_id`
12    FROM   `subscribers`;
13   DECLARE CONTINUE HANDLER FOR NOT FOUND SET subscribers_done = 1;
14
15   OPEN subscribers_cursor;
16   read_users_loop: LOOP
17    FETCH subscribers_cursor INTO s_id_value;
18    IF subscribers_done THEN
19     LEAVE read_users_loop;
20    END IF;
21
```

MySQL Решение 6.1.2.a (код процедуры) (продолжение)

```
22     BOOKS: BEGIN
23      DECLARE b_id_value INT DEFAULT 0;
24      DECLARE books_done INT DEFAULT 0;
25      DECLARE books_cursor CURSOR FOR
26       SELECT `b_id`
27       FROM   `books`
28       ORDER BY RAND()
29       LIMIT 3;
30      DECLARE CONTINUE HANDLER FOR NOT FOUND SET books_done = 1;
31      OPEN books_cursor;
32
33      read_books_loop: LOOP
34      FETCH books_cursor INTO b_id_value;
35       IF books_done THEN
36        LEAVE read_books_loop;
37       END IF;
38
39       INSERT INTO `subscriptions`
40                   (`sb_subscriber`,
41                    `sb_book`,
42                    `sb_start`,
43                    `sb_finish`,
44                    `sb_is_active`)
45            VALUES (s_id_value,
46                    b_id_value,
47                    NOW(),
48                    NOW() + INTERVAL 1 MONTH,
49                    'Y');
50
51      END LOOP read_books_loop;
52      CLOSE books_cursor;
53     END BOOKS;
54
55    END LOOP read_users_loop;
56    CLOSE subscribers_cursor;
57   END USERS;
58   IF EXISTS (SELECT 1
59              FROM  `subscriptions`
60              WHERE `sb_is_active`='Y'
61              GROUP BY `sb_subscriber`
62              HAVING COUNT(1)>10
63              LIMIT 1)
64    THEN
65     SELECT 'Rolling transaction back...';
66     ROLLBACK;
67    ELSE
68     SELECT 'Committing transaction...';
69     COMMIT;
70   END IF;
71
72 END;
73 $$
74 DELIMITER ;
```

Обратите внимание на имена переменных, в которые извлекаются значения полей `` `s_id` `` и `` `b_id` ``: `s_id_value` и `b_id_value`. Часть `_value` туда добавлена не случайно, т.к. если имена таких переменных будут совпадать с именами полей таблицы, MySQL не будет извлекать в них данные.

Для проверки работоспособности полученного решения можно использовать следующие запросы. Если вы выполните их на исходном наборе данных базы данных «Библиотека», то дважды операция завершится успешно, а третий и последующие вызовы будут завершаться отменой транзакции.

MySQL Решение 6.1.2.a (код для проверки работоспособности)

```
1   CALL THREE_RANDOM_BOOKS();
2   SELECT * FROM `subscriptions`;
```

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Общая логика решения для данной СУБД совпадает с логикой решения для MySQL, но поскольку работать с вложенными курсорами здесь приходится иначе, снова повторим алгоритм действий со ссылками на соответствующие фрагменты кода.

Итак, для получения решения мы будем должны:

- запустить транзакцию (строка 17);
- открыть курсор для извлечения идентификаторов всех читателей (строка 19);
- для каждого идентификатора читателя выполнить вложенный цикл (строки 23-49), в котором:
  - открыть курсор для извлечения трёх идентификаторов случайных книг (строка 25);
  - для каждого полученного идентификатора книги произвести вставку в таблицу выдач книг (строки 28-44);
  - закрыть курсор для извлечения трёх идентификаторов случайных книг (строка 45);
- закрыть курсор для извлечения идентификаторов всех читателей (строка 50);
- проверить, было ли нарушено условие о недопустимости нахождения на руках у одного читателя более десяти книг (строки 53-66) и:
  - если условие было нарушено, отменить транзакцию (строка 60);
  - если условие не было нарушено, подтвердить транзакцию (строка 65).

В MS SQL Server нет необходимости использовать отдельные блоки кода для каждого курсора. Вместо этого мы сохраняем значение параметра `@@FETCH_STATUS` (предоставляющего информацию о последней операции извлечения данных из курсора) в отдельной переменной для каждого из циклов (строки 21, 27, 43, 48), а затем используем эти переменные для организации работы циклов.

MS SQL Решение 6.1.2.a (код процедуры)

```
1   CREATE PROCEDURE THREE_RANDOM_BOOKS
2   AS
3   BEGIN
4    DECLARE @s_id_value INT;
5    DECLARE @b_id_value INT;
6    DECLARE subscribers_cursor CURSOR LOCAL FAST_FORWARD FOR
7     SELECT [s_id]
8     FROM   [subscribers];
9    DECLARE books_cursor CURSOR LOCAL FAST_FORWARD FOR
10    SELECT TOP 3 [b_id]
11    FROM         [books]
12    ORDER BY     NEWID();
13   DECLARE @fetch_subscribers_cursor INT;
14   DECLARE @fetch_books_cursor INT;
15
```

MS SQL Решение 6.1.2.a (код процедуры) (продолжение)

```
16   PRINT 'Starting transaction...';
17   BEGIN TRANSACTION;
18
19   OPEN subscribers_cursor;
20   FETCH NEXT FROM subscribers_cursor INTO @s_id_value;
21   SET @fetch_subscribers_cursor = @@FETCH_STATUS;
22
23   WHILE @fetch_subscribers_cursor = 0
24    BEGIN
25     OPEN books_cursor;
26     FETCH NEXT FROM books_cursor INTO @b_id_value;
27     SET @fetch_books_cursor = @@FETCH_STATUS;
28     WHILE @fetch_books_cursor = 0
29      BEGIN
30       INSERT INTO [subscriptions]
31                   ([sb_subscriber],
32                    [sb_book],
33                    [sb_start],
34                    [sb_finish],
35                    [sb_is_active])
36            VALUES (@s_id_value,
37                    @b_id_value,
38                    GETDATE(),
39                    DATEADD(month, 1, GETDATE()),
40                    N'Y');
41
42       FETCH NEXT FROM books_cursor INTO @b_id_value;
43       SET @fetch_books_cursor = @@FETCH_STATUS;
44      END;
45     CLOSE books_cursor;
46
47     FETCH NEXT FROM subscribers_cursor INTO @s_id_value;
48     SET @fetch_subscribers_cursor = @@FETCH_STATUS;
49    END;
50   CLOSE subscribers_cursor;
51   DEALLOCATE subscribers_cursor;
52   DEALLOCATE books_cursor;
53   IF EXISTS (SELECT TOP 1 1
54              FROM  [subscriptions]
55              WHERE [sb_is_active]='Y'
56              GROUP BY [sb_subscriber]
57              HAVING COUNT(1)>10)
58    BEGIN
59     PRINT 'Rolling transaction back...';
60     ROLLBACK TRANSACTION;
61    END
62   ELSE
63    BEGIN
64     PRINT 'Committing transaction...';
65     COMMIT TRANSACTION;
66    END;
67
68 END;
69 GO
```

Для проверки работоспособности полученного решения можно использовать следующие запросы.

MS SQL Решение 6.1.2.a (код для проверки работоспособности)

```
1  EXECUTE THREE_RANDOM_BOOKS;
2  SELECT * FROM [subscriptions];
```

На этом решение для MS SQL Server завершено.

Переходим к Oracle. Несмотря на то, что мы уже дважды рассматривали алгоритм решения, здесь мы повторим его снова — в том числе для того, чтобы прослеживая отсылки к коду вы увидели, насколько просто и элегантно реализуется работа с вложенными курсорами в Oracle.

Итак, для получения решения мы будем должны:

- завершить предыдущую транзакцию (строка 17) (напомним, что «запустить транзакцию» в Oracle невозможно, т.к. транзакция всегда активируется первой операцией модификации данных);
- создать цикл для прохода по рядам курсора для извлечения идентификаторов всех читателей (строки 19-35), и внутри этого цикла:
  - создать цикл для прохода по рядам курсора для извлечения трёх идентификаторов случайных книг (строки 21-34);
  - для каждого полученного идентификатора книги произвести вставку в таблицу выдач книг (строки 23-33);
- проверить, было ли нарушено условие о недопустимости нахождения на руках у одного читателя более десяти книг (строки 37-52) и:
  - если условие было нарушено, отменить транзакцию (строка 48);
  - если условие не было нарушено, подтвердить транзакцию (строка 51).

Небольшое неудобство в этом решении вызывает только необходимость выяснять существование записей, нарушающих условие задачи, через промежуточную переменную и подзапрос (строки 37-43), что связано с невозможностью применения в Oracle конструкции **`IF EXISTS`**. В остальном весь код хранимой процедуры выглядит не сложнее примитивного примера на любом распространённом языке программирования.

Oracle Решение 6.1.2.a (код процедуры)

```
1  CREATE OR REPLACE PROCEDURE THREE_RANDOM_BOOKS
2  AS
3   counter INT := 0;
4   CURSOR subscribers_cursor IS
5    SELECT "s_id"
6    FROM   "subscribers";
7   CURSOR books_cursor IS
8    SELECT "b_id"
9    FROM
10     (SELECT "b_id"
11      FROM   "books"
12      ORDER BY DBMS_RANDOM.VALUE)
13   WHERE ROWNUM <= 3;
14
15 BEGIN
16  DBMS_OUTPUT.PUT_LINE('Committing previous transaction...');
17  COMMIT;
18
```

Oracle Решение 6.1.2.a (код процедуры) (продолжение)

```
19  FOR one_subscriber IN subscribers_cursor
20  LOOP
21   FOR one_book IN books_cursor
22   LOOP
23    INSERT INTO "subscriptions"
24                ("sb_subscriber",
25                 "sb_book",
26                 "sb_start",
27                 "sb_finish",
28                 "sb_is_active")
29         VALUES (one_subscriber."s_id",
30                 one_book."b_id",
31                 SYSDATE,
32                 ADD_MONTHS(SYSDATE, 1),
33                 'Y');
34   END LOOP;
35  END LOOP;
36
37  SELECT COUNT(1) INTO counter
38  FROM
39  (SELECT COUNT(1)
40   FROM "subscriptions"
41   WHERE "sb_is_active"='Y'
42   GROUP BY "sb_subscriber"
43   HAVING COUNT(1)>10);
44
45  IF (counter > 0)
46   THEN
47    DBMS_OUTPUT.PUT_LINE('Rolling transaction back...');
48    ROLLBACK;
49   ELSE
50    DBMS_OUTPUT.PUT_LINE('Committing transaction...');
51    COMMIT;
52  END IF;
53
54 END;
```

Для проверки работоспособности полученного решения можно использовать следующие запросы.

Oracle Решение 6.1.2.a (код для проверки работоспособности)

```
1  SET SERVEROUTPUT ON;
2  EXECUTE THREE_RANDOM_BOOKS;
3  SELECT * FROM "subscriptions";
```

На этом решение данной задачи завершено.

Решение 6.1.2.b{440}.

В отличие от решения{441} задачи 6.1.2.a{440}, здесь нам даже не понадобятся курсоры. Потому решение сведётся к серии простых действий:

- выполнить изменения;
- проверить, нарушено ли условие задачи, и:
  - отменить изменения, если нарушено;
  - подтвердить изменения, если не нарушено.

Остаётся только рассмотреть код хранимых процедур. Отличия будут только в способе вычисления интервалов дат и (в Oracle) запуске транзакции. В остальном решения для всех трёх СУБД полностью эквивалентны.

Решение для MySQL выглядит следующим образом.

MySQL Решение 6.1.2.b (код процедуры)

```
DELIMITER $$
CREATE PROCEDURE CHANGE_DATES()
BEGIN
 SELECT 'Starting transaction...';
 START TRANSACTION;

 UPDATE `subscriptions`
    SET `sb_finish` = DATE_ADD(`sb_finish`, INTERVAL 3 MONTH);

 SET @avg_read = (SELECT AVG(DATEDIFF(`sb_finish`, `sb_start`))
                    FROM `subscriptions`);

 IF (@avg_read > 120)
  THEN
   SELECT 'Rolling transaction back...';
   ROLLBACK;
  ELSE
   SELECT 'Committing transaction...';
   COMMIT;
 END IF;

END;
$$
DELIMITER ;
```

Для проверки работоспособности полученного решения можно использовать следующий запрос.

MySQL Решение 6.1.2.b (код для проверки работоспособности)

```
CALL CHANGE_DATES()
```

Решение для MS SQL Server выглядит следующим образом.

MS SQL Решение 6.1.2.b (код процедуры)

```
CREATE PROCEDURE CHANGE_DATES
AS
BEGIN
 DECLARE @avg_read DOUBLE PRECISION;

 PRINT 'Starting transaction...';
 BEGIN TRANSACTION;

 UPDATE [subscriptions]
    SET [sb_finish] = DATEADD(month, 3, [sb_finish]);

```

MS SQL Решение 6.1.2.b (код процедуры) (продолжение)

```
12   SET @avg_read = (SELECT AVG(DATEDIFF (month, [sb_start], [sb_finish]))
13                       FROM [subscriptions]);
14
15   IF (@avg_read > 4)
16    BEGIN
17     PRINT 'Rolling transaction back...';
18     ROLLBACK TRANSACTION;
19    END
20   ELSE
21    BEGIN
22     PRINT 'Committing transaction...';
23     COMMIT TRANSACTION;
24    END;
25
26  END;
27  GO
```

Для проверки работоспособности полученного решения можно использовать следующий запрос.

MS SQL Решение 6.1.2.b (код для проверки работоспособности)

```
1   EXECUTE CHANGE_DATES
```

Решение для Oracle выглядит следующим образом.

Oracle Решение 6.1.2.b (код процедуры)

```
1   CREATE OR REPLACE PROCEDURE CHANGE_DATES
2   AS
3    avg_read NUMBER(5, 3) := 0.0;
4
5   BEGIN
6    DBMS_OUTPUT.PUT_LINE('Committing previous transaction...');
7    COMMIT;
8
9    UPDATE "subscriptions"
10      SET "sb_finish" = ADD_MONTHS("sb_finish", 3);
11
12   SELECT AVG(MONTHS_BETWEEN("sb_finish", "sb_start")) INTO avg_read
13   FROM "subscriptions";
14
15   IF (avg_read > 4.0)
16    THEN
17     DBMS_OUTPUT.PUT_LINE('Rolling transaction back...');
18     ROLLBACK;
19    ELSE
20     DBMS_OUTPUT.PUT_LINE('Committing transaction...');
21     COMMIT;
22   END IF;
23
24  END;
```

Для проверки работоспособности полученного решения можно использовать следующий запрос.

Oracle Решение 6.1.2.b (код для проверки работоспособности)

```
1  EXECUTE CHANGE_DATES
```

На этом решение данной задачи завершено.

Задание 6.1.2.TSK.A: создать хранимую процедуру, которая:

- добавляет каждой книге два случайных жанра;
- отменяет совершённые действия, если в процессе работы хотя бы одна операция вставки завершилась ошибкой в силу дублирования значения первичного ключа таблицы `m2m_books_genres` (т.е. у такой книги уже был такой жанр).

Задание 6.1.2.TSK.B: создать хранимую процедуру, которая:

- увеличивает значение поля `b_quantity` для всех книг в два раза;
- отменяет совершённое действие, если по итогу выполнения операции среднее количество экземпляров книг превысит значение 50.

6.2. КОНКУРИРУЮЩИЕ ТРАНЗАКЦИИ

6.2.1. ПРИМЕР 43: УПРАВЛЕНИЕ УРОВНЕМ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

Задача 6.2.1.a{450}: написать запросы, которые, будучи выполненными параллельно, обеспечивали бы следующий эффект:

- первый запрос должен добавлять ко всем датам возврата книг один день и не зависеть от запросов на чтение из таблицы `subscriptions` (не ждать их завершения);
- второй запрос должен читать все даты возврата книг из таблицы `subscriptions` и не зависеть от первого запроса (не ждать его завершения).

Задача 6.2.1.b{452}: написать два запроса, каждый из которых будет считать количество выданных каждому читателю книг, но при этом:

- один запрос должен выполняться максимально быстро (даже ценой предоставления не совсем достоверных данных);
- другой запрос должен предоставлять гарантированно достоверные данные (даже ценой большого времени выполнения).

Ожидаемый результат 6.2.1.a.

При любом варианте запуска («первый, потом второй» или «второй, потом первый») запросы работают параллельно, и ни один из них не ожидает завершения другого.

Ожидаемый результат 6.2.1.b.

Первый запрос никогда не ожидает завершения каких бы то ни было других запросов, второй запрос может быть поставлен в очередь ожидания.

Решение 6.2.1.a{449}.

В MySQL при использовании механизма доступа InnoDB запросы на обновление данных по умолчанию имеют более высокий приоритет, чем запросы на чтение данных.

Если в момент запуска обновления уже выполняется операция чтения, обновление будет ожидать её завершения только в одном случае: если она запущена в транзакции с уровнем изолированности **`SERIALIZABLE`**. Учитывая, что уровнем изолированности транзакций по умолчанию является **`REPEATABLE READ`**, с первой частью задачи у нас нет особых проблем: обновление начнётся сразу же.

Теперь нужно добиться такой работы СУБД, при которой запрос на чтение будет выполняться параллельно с запросом на обновление. И это — тоже не проблема, если не запускать его в **`SERIALIZABLE`**-режиме. Остаётся решить, хотим ли мы получать «сырые данные» (изменения, ещё не вступившие в силу) или же хотим получать только данные, сохранённые в базе данных при подтверждении транзакции? В первом случае запрос на чтение нужно выполнять в транзакции с уровнем изолированности **`READ UNCOMMITTED`**, во втором — в транзакции с уровнем изолированности **`READ COMMITTED`** или **`REPEATABLE READ`**.

Остаётся лишь реализовать данную идею в коде. Следующие два блока кода необходимо выполнять в отдельных соединениях с СУБД (отдельных сессиях), потому обязательно удостоверьтесь, что запросы в первых строках каждого из блоков возвращают **разные** значения идентификаторов сессий. В MySQL Workbench вы можете открыть несколько копий одного соединения с СУБД[33], и они будут работать в разных сессиях.

MySQL Решение 6.2.1.a (первый блок)

```
1  SELECT CONNECTION_ID();
2  SET autocommit = 0;
3  START TRANSACTION;
4  UPDATE `subscriptions`
5  SET    `sb_finish` = DATE_ADD(`sb_finish`, INTERVAL 1 DAY);
6  -- SELECT SLEEP(10);
7  COMMIT;
```

MySQL Решение 6.2.1.a (второй блок)

```
1  SELECT CONNECTION_ID();
2  SET autocommit = 0;
3  SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
4  START TRANSACTION;
5  SELECT `sb_finish`
6  FROM   `subscriptions`;
7  -- SELECT SLEEP(10);
8  COMMIT;
```

Раскомментировав строку с **`SELECT SLEEP(10)`** в соответствующем блоке кода, мы проэмулируем его долгое выполнение, что позволит нам не спеша несколько раз выполнить второй блок (в котором эта строка останется закомментированной) и посмотреть на результат.

На этом решение данной задачи для MySQL завершено, однако для лучшего понимания логики работы транзакций настоятельно рекомендуется провести серию экспериментов, изменяя во втором блоке кода уровень изолированности транзакций и наблюдая за изменением поведения СУБД.

---

33 https://dev.mysql.com/doc/workbench/en/wb-mysql-connections-new.html

Переходим к MS SQL Server. Логика поведения данной СУБД почти совпадает с логикой MySQL, но есть и отличия:

- уровнем изолированности транзакций в MS SQL по умолчанию является **READ COMMITTED** (это не влияет на решение данной задачи);
- при выполнении запроса на чтение в транзакции с уровнем изолированности **READ COMMITTED** MS SQL Server в отличие от MySQL не вернёт мгновенно текущие актуальные данные, а будет ждать завершения конкурирующих транзакций, выполняющих модификацию данных (из этого следует, что для соблюдения условия задачи мы обязаны выполнять запрос на чтение в транзакции с уровнем изолированности **READ UNCOMMITTED**).

Рассмотрим код. Следующие два блока кода необходимо выполнять в отдельных соединениях с СУБД (отдельных сессиях), потому обязательно удостоверьтесь, что запросы в первых строках каждого из блоков возвращают **разные** значения идентификаторов сессий. В MS SQL Server Management Studio отдельные окна для выполнения SQL-запросов будут работать в отдельных сессиях[34].

MS SQL Решение 6.2.1.a (первый блок)

```
SELECT @@SPID;
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN TRANSACTION;
SELECT [sb_finish]
FROM   [subscriptions];
-- WAITFOR DELAY '00:00:10';
COMMIT TRANSACTION;
```

MS SQL Решение 6.2.1.a (второй блок)

```
SELECT @@SPID;
SET IMPLICIT_TRANSACTIONS ON;
BEGIN TRANSACTION;
UPDATE [subscriptions]
SET    [sb_finish] = DATEADD(day, 1, [sb_finish]);
-- WAITFOR DELAY '00:00:10';
COMMIT TRANSACTION;
```

Раскомментировав строку с **WAITFOR DELAY '00:00:10'** в соответствующем блоке кода, мы проэмулируем его долгое выполнение, что позволит нам не спеша несколько раз выполнить второй блок (в котором эта строка останется закомментированной) и посмотреть на результат.

На этом решение данной задачи для MS SQL Server завершено.

Переходим к Oracle. Продолжая аналогию с только рассмотренными решениями для MySQL и MS SQL, отметим, что:

- уровень изолированности транзакций в Oracle по умолчанию — **READ COMMITTED** (как и в MS SQL Server);
- в отличие от MySQL и MS SQL Server в Oracle **нет** уровня изолированности транзакций **READ UNCOMMITTED**;
- операции чтения и модификации данных в Oracle не блокируют друг друга[35], потому решение текущей задачи сводится к простому выполнению необходимых запросов (но для сохранения единообразия мы будем придерживаться того же набора команд, что был использован в MySQL и MS SQL Server).

---

34 https://msdn.microsoft.com/en-us/library/ms174195.aspx

35 http://www.oracle.com/technetwork/issue-archive/2010/10-jan/o65asktom-082389.html

Рассмотрим код. Следующие два блока кода необходимо выполнять в отдельных соединениях с СУБД (отдельных сессиях), потому обязательно удостоверьтесь, что запросы во вторы строках каждого из блоков возвращают **разные** значения идентификаторов сессий. В Oracle SQL Developer открыть новое окно для выполнения запросов в отдельной сессии можно клавиатурной комбинацией Ctrl+Shift+N.

В первых строках обоих блоков кода выполняется операция `COMMIT`, чтобы гарантировать выполнение дальнейших запросов в новой отдельной транзакции.

Oracle Решение 6.2.1.a (первый блок)

```
1  COMMIT;
2  SELECT SYS_CONTEXT('userenv', 'sessionid')
3  FROM   DUAL;
4  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
5  SELECT "sb_finish"
6  FROM   "subscriptions" ORDER BY "sb_finish" ASC;
7  -- EXEC DBMS_LOCK.SLEEP(10);
8  COMMIT;
```

Oracle Решение 6.2.1.a (второй блок)

```
1  COMMIT;
2  SELECT SYS_CONTEXT('userenv', 'sessionid')
3  FROM   DUAL;
4  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
5  SELECT "sb_finish"
6  FROM   "subscriptions" ORDER BY "sb_finish" ASC;
7  -- EXEC DBMS_LOCK.SLEEP(10);
8  COMMIT;
```

Раскомментировав строку с `EXEC DBMS_LOCK.SLEEP(10)` в соответствующем блоке кода, мы проэмулируем его долгое выполнение, что позволит нам не спеша несколько раз выполнить второй блок (в котором эта строка останется закомментированной) и посмотреть на результат.

На этом решение данной задачи завершено.

Решение 6.2.1.b{449}.

Решение данной задачи подчиняется общей логике разделения уровней изолированности транзакций:

- чем уровень ниже, тем больше у СУБД возможностей выполнить запрос параллельно с другими, но тем выше вероятность получить некорректный результат;
- чум уровень выше, тем меньше у СУБД возможностей выполнить запрос параллельно с другими, но тем ниже вероятность получить некорректный результат;
- в MySQL и MS SQL Server самым низким уровнем является `READ UNCOMMITTED`, в Oracle — `READ COMMITTED`;
- во всех трёх СУБД самым высоким уровнем является `SERIALIZABLE`.

Учитывая эти факты, нам остаётся только написать код для выполнения одного и того же запроса на самом низком и самом высоком уровнях изолированности транзакций, а также подготовить проверочный код, который позволит увидеть разницу в работе этих двух вариантов выполнения основного кода.

Код для MySQL выглядит следующим образом.

MySQL Решение 6.2.1.b (максимально быстрое выполнение, возможны некорректные данные)

```
1  SELECT CONNECTION_ID();
2  SET autocommit = 0;
3  SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4  START TRANSACTION;
5  SELECT `sb_subscriber`,
6         COUNT(`sb_book`) AS `sb_has_books`
7  FROM   `subscriptions`
8  WHERE  `sb_is_active` = 'Y'
9  GROUP BY `sb_subscriber`;
10 COMMIT;
```

MySQL Решение 6.2.1.b (максимально корректные данные, возможно долгое выполнение)

```
1  SELECT CONNECTION_ID();
2  SET autocommit = 0;
3  SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
4  START TRANSACTION;
5  SELECT `sb_subscriber`,
6         COUNT(`sb_book`) AS `sb_has_books`
7  FROM   `subscriptions`
8  WHERE  `sb_is_active` = 'Y'
9  GROUP BY `sb_subscriber`;
10 COMMIT;
```

MySQL Решение 6.2.1.b (проверочный код)

```
1  SELECT CONNECTION_ID();
2  SET autocommit = 0;
3  START TRANSACTION;
4  UPDATE `subscriptions`
5  SET    `sb_is_active` =
6         CASE
7          WHEN `sb_is_active` = 'Y' THEN 'N'
8          WHEN `sb_is_active` = 'N' THEN 'Y'
9         END;
10 SELECT SLEEP(10);
11 COMMIT;
```

Код для MS SQL Server выглядит следующим образом.

MS SQL Решение 6.2.1.b (максимально быстрое выполнение, возможны некорректные данные)

```
1  SELECT @@SPID;
2  SET IMPLICIT_TRANSACTIONS ON;
3  SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
4  BEGIN TRANSACTION;
5  SELECT [sb_subscriber],
6         COUNT([sb_book]) AS [sb_has_books]
7  FROM   [subscriptions]
8  WHERE  [sb_is_active] = 'Y'
9  GROUP BY [sb_subscriber];
10 COMMIT TRANSACTION;
```

MS SQL Решение 6.2.1.b (максимально корректные данные, возможно долгое выполнение)

```
SELECT @@SPID;
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;
SELECT [sb_subscriber],
       COUNT([sb_book]) AS [sb_has_books]
FROM   [subscriptions]
WHERE  [sb_is_active] = 'Y'
GROUP BY [sb_subscriber];
COMMIT TRANSACTION;
```

MS SQL Решение 6.2.1.b (проверочный код)

```
SELECT @@SPID;
SET IMPLICIT_TRANSACTIONS ON;
BEGIN TRANSACTION;
UPDATE [subscriptions]
SET    [sb_is_active] =
       CASE
        WHEN [sb_is_active] = 'Y' THEN 'N'
        WHEN [sb_is_active] = 'N' THEN 'Y'
       END;
WAITFOR DELAY '00:00:10';
COMMIT TRANSACTION;
```

Код для Oracle выглядит следующим образом.

Oracle Решение 6.2.1.b (максимально быстрое выполнение, возможны некорректные данные)

```
COMMIT;
SELECT SYS_CONTEXT('userenv','sessionid') FROM DUAL;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT "sb_subscriber",
       COUNT("sb_book") AS "sb_has_books"
FROM   "subscriptions"
WHERE  "sb_is_active" = 'Y'
GROUP BY "sb_subscriber";
COMMIT;
```

Oracle Решение 6.2.1.b (максимально корректные данные, возможно долгое выполнение)

```
COMMIT;
SELECT SYS_CONTEXT('userenv','sessionid') FROM DUAL;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT "sb_subscriber",
       COUNT("sb_book") AS "sb_has_books"
FROM   "subscriptions"
WHERE  "sb_is_active" = 'Y'
GROUP BY "sb_subscriber";
COMMIT;
```

Oracle Решение 6.2.1.b (проверочный код)

```
1  COMMIT;
2  SELECT SYS_CONTEXT('userenv','sessionid') FROM DUAL;
3  UPDATE "subscriptions"
4  SET    "sb_is_active" =
5         CASE
6          WHEN "sb_is_active" = 'Y' THEN 'N'
7          WHEN "sb_is_active" = 'N' THEN 'Y'
8         END;
9  EXEC DBMS_LOCK.SLEEP(10);
10 COMMIT;
```

Для всех трёх СУБД проверочный код необходимо выполнять в отдельной сессии (см. пояснения в решении{450} задачи 6.2.1.a{449}), при этом основной код надо выполнять до начала работы проверочного, во время его работы и после его завершения — это позволит наглядно увидеть, какие данные и в какой момент времени СУБД будет извлекать из базы данных.

Ещё один вариант поведения СУБД можно увидеть, заменив в проверочном коде последнюю команду с `COMMIT` на `ROLLBACK`.

Обратите особое внимание на отличие поведения Oracle от MySQL и MS SQL Server: даже в `SERIALIZABLE`-режиме запрос вернёт результаты без задержки.

На этом решение данной задачи завершено.

Задание 6.2.1.TSK.A: написать запросы, которые, будучи выполненными параллельно, обеспечивали бы следующий эффект:

- первый запрос должен считать количество выданных на руки и возвращённых в библиотеку книг и не зависеть от запросов на обновление таблицы `subscriptions` (не ждать их завершения);
- второй запрос должен инвертировать значения поля `sb_is_active` таблицы `subscriptions` с Y на N и наоборот и не зависеть от первого запроса (не ждать его завершения).

Задание 6.2.1.TSK.B: написать запросы, которые, будучи выполненными параллельно, обеспечивали бы следующий эффект:

- первый запрос должен считать количество выданных на руки и возвращённых в библиотеку книг;
- второй запрос должен инвертировать значения поля `sb_is_active` таблицы `subscriptions` с Y на N и наоборот для читателей с нечётными идентификаторами, после чего делать паузу в десять секунд и отменять данное изменение (отменять транзакцию).

Исследовать поведение все трёх СУБД при выполнении первого запроса до, во время и после завершения выполнения второго запроса, повторив этот эксперимент для всех поддерживаемых конкретной СУБД уровней изолированности транзакций.

6.2.2. ПРИМЕР 44: ВЗАИМОДЕЙСТВИЕ КОНКУРИРУЮЩИХ ТРАНЗАКЦИЙ

Задача 6.2.2.a{456}: продемонстрировать во всех трёх СУБД все аномалии конкурентного доступа для всех возможных комбинаций уровней изолированности транзакций.

Задача 6.2.2.b{483}: продемонстрировать во всех трёх СУБД ситуацию гарантированного получения взаимной блокировки транзакций и реакцию СУБД на такую ситуацию.

Ожидаемый результат 6.2.2.a.

Поскольку решение данной задачи и является ожидаемым результатом, см. решение{456} 6.2.2.a.

Ожидаемый результат 6.2.2.b.

Поскольку решение данной задачи и является ожидаемым результатом, см. решение{483} 6.2.2.b.

Решение 6.2.2.a{456}.

К аномалиям конкурентного доступа относятся:

- грязное чтение (dirty read) — чтение промежуточного состояния данных до того, как модифицирующая их транзакция будет подтверждена или отменена;
- потерянное обновление (lost update) — модификация одной и той же информации двумя и более транзакциями, при которой в силу вступают изменения, выполненные транзакцией, которая была подтверждена последней (а изменения, выполненные остальными транзакциями, теряются);
- неповторяющееся чтение (non-repeatable read) — получение различных результатов выполнения одного и того же запроса на чтение в рамках одной транзакции;
- фантомное чтение (phantom read) — временное появление (исчезновение) в наборе данных, с которым работает транзакция, тех или иных записей в силу их изменения другой транзакцией.

Для удобства навигации приведём таблицу, показывающую номера страниц, с которых начинается рассмотрение той или иной аномалии в каждой СУБД.

	Грязное чтение	Потерянное обновление	Неповторяющееся чтение	Фантомное чтение
MySQL	{435}	{437}	{440}	{442}
MS SQL Server	{445}	{447}	{450}	{452}
Oracle	{455}	{457}	{459}	{461}

Также отметим, что поскольку протоколы исследований будут выглядеть однотипно во всех СУБД, для экономии места мы ниже приведём их только для MySQL, причём в рамках исследования каждой аномалии конкурентного доступа для первой транзакции покажем только один уровень изолированности, а для второй — все поддерживаемые данной СУБД уровни изолированности.

Традиционно начинаем с MySQL. Данная СУБД поддерживает четыре уровня изолированности транзакций, комбинации которых мы и рассмотрим:

- `READ UNCOMMITTED`;
- `READ COMMITTED`;
- `REPEATABLE READ`;
- `SERIALIZABLE`.

Для выполнения эксперимента используем командный файл:

```
start cmd.exe /c "mysql -uПОЛЬЗЬВАТЕЛЬ -pПАРОЛЬ БАЗА_ДАННЫХ < a.sql & pause"
start cmd.exe /c "mysql -uПОЛЬЗЬВАТЕЛЬ -pПАРОЛЬ БАЗА_ДАННЫХ < b.sql & pause"
```

Грязное чтение в MySQL может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MySQL Решение 6.2.2.a (код для исследования аномалии грязного чтения)

```
-- Транзакция A:
SELECT CONCAT('Tr A ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr A START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';


SELECT SLEEP(5);


SELECT CONCAT('Tr A UPDATE: ',
              CURTIME());
UPDATE `subscriptions`
SET    `sb_is_active` =
 CASE
  WHEN `sb_is_active` = 'Y' THEN 'N'
  WHEN `sb_is_active` = 'N' THEN 'Y'
 END
WHERE  `sb_id` = 2;


SELECT SLEEP(20);


SELECT CONCAT('Tr A ROLLBACK: ',
              CURTIME());
ROLLBACK;
```

```
-- Транзакция B:
SELECT CONCAT('Tr B ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr B START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';
SELECT CONCAT('Tr B SELECT 1: ',
              CURTIME());
SELECT `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` = 2;


SELECT SLEEP(10);


SELECT CONCAT('Tr B SELECT 2: ',
              CURTIME());
SELECT `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` = 2;
SELECT CONCAT('Tr B COMMIT: ',
              CURTIME());
COMMIT;
```

Приведём пример журнала выполнения этого кода для ситуации, когда транзакция A выполняется на уровне изолированности **`READ UNCOMMITTED`** и конкурирует с транзакцией B, последовательно выполняемой во всех поддерживаемых MySQL уровнях изолированности.

A: READ UNCOMMITTED	B: READ UNCOMMITTED
Tr A ID = 13 Tr A START: 17:21:01 in READ-UNCOMMITTED Tr A UPDATE: 17:21:06 Tr A ROLLBACK: 17:21:26	Tr B ID = 14 Tr B START: 17:21:01 in READ-UNCOMMITTED Tr B SELECT 1: 17:21:01 sb_is_active = Y Tr B SELECT 2: 17:21:11 **sb_is_active = N** Tr B COMMIT: 17:21:11

A: READ UNCOMMITTED	B: READ COMMITTED
Tr A ID = 15 Tr A START: 17:38:06 in READ-UNCOMMITTED Tr A UPDATE: 17:38:12 Tr A ROLLBACK: 17:38:32	Tr B ID = 16 Tr B START: 17:38:06 in READ-COMMITTED Tr B SELECT 1: 17:38:06 sb_is_active = Y Tr B SELECT 2: 17:38:16 **sb_is_active = Y** Tr B COMMIT: 17:38:16

A: READ UNCOMMITTED	B: REPEATABLE READ
Tr A ID = 18 Tr A START: 17:42:37 in READ-UNCOMMITTED Tr A UPDATE: 17:42:42 Tr A ROLLBACK: 17:43:02	Tr B ID = 17 Tr B START: 17:42:37 in REPEATABLE-READ Tr B SELECT 1: 17:42:37 sb_is_active = Y Tr B SELECT 2: 17:42:47 **sb_is_active = Y** Tr B COMMIT: 17:42:47

A: READ UNCOMMITTED	B: SERIALIZABLE
Tr A ID = 20 Tr A START: 17:48:19 in READ-UNCOMMITTED Tr A UPDATE: 17:48:24 Tr A ROLLBACK: 17:48:49	Tr B ID = 19 Tr B START: 17:48:19 in SERIALIZABLE Tr B SELECT 1: 17:48:19 sb_is_active = Y Tr B SELECT 2: 17:48:29 **sb_is_active = Y** Tr B COMMIT: 17:48:29

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения транзакции B
	READ COMMITTED	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения транзакции B
	REPEATABLE READ	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения транзакции B
	SERIALIZABLE	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения транзакции B

Потерянное обновление MySQL может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MySQL Решение 6.2.2.a (код для исследования аномалии потерянного обновления)

```
-- Транзакция A:
SELECT CONCAT('Tr A ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr A START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';
SELECT CONCAT('Tr A, SELECT: ',
              CURTIME());
SELECT `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` = 2;
```

```
-- Транзакция B:
SELECT CONCAT('Tr B ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr B START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';


SELECT SLEEP(5);
```

MySQL Решение 6.2.2.a (код для исследования аномалии потерянного обновления) (продолжение)

```
20                                          SELECT CONCAT('Tr B, SELECT: ',
21                                                 CURTIME());
22  SELECT SLEEP(10);                       SELECT `sb_is_active`
23                                          FROM   `subscriptions`
24                                          WHERE  `sb_id` = 2;
25  SELECT CONCAT('Tr A UPDATE: ',
26                CURTIME());
27  UPDATE `subscriptions`
28  SET    `sb_is_active` = 'Y'             SELECT SLEEP(10);
29  WHERE  `sb_id` = 2;
30  SELECT CONCAT('Tr A COMMIT: ',
31                CURTIME());
32  COMMIT;
33                                          SELECT CONCAT('Tr B UPDATE: ',
34                                                 CURTIME());
35                                          UPDATE `subscriptions`
36  SELECT SLEEP(10);                       SET    `sb_is_active` = 'N'
37                                          WHERE  `sb_id` = 2;
38                                          SELECT CONCAT('Tr B COMMIT: ',
39                                                 CURTIME());
40                                          COMMIT;
41  SELECT CONCAT('After A, SELECT: ',      SELECT CONCAT('After B, SELECT: ',
42                CURTIME());                      CURTIME());
43  SELECT `sb_is_active`                   SELECT `sb_is_active`
44  FROM   `subscriptions`                  FROM   `subscriptions`
45  WHERE  `sb_id` = 2;                     WHERE  `sb_id` = 2;
```

Приведём пример журнала выполнения этого кода для ситуации, когда транзакция A выполняется на уровне изолированности **`READ COMMITTED`** и конкурирует с транзакцией B, последовательно выполняемой во всех поддерживаемых MySQL уровнях изолированности.

A: READ COMMITTED	B: READ UNCOMMITTED
Tr A ID = 77 Tr A START: 19:12:19 in READ-COMMITTED Tr A, SELECT: 19:12:19 sb_is_active = N Tr A UPDATE: 19:12:29 Tr A COMMIT: 19:12:29 After A, SELECT: 19:12:39 **sb_is_active = N**	Tr B ID = 76 Tr B START: 19:12:19 in READ-UNCOMMITTED Tr B, SELECT: 19:12:24 sb_is_active = N Tr B UPDATE: 19:12:34 Tr B COMMIT: 19:12:34 After B, SELECT: 19:12:34 **sb_is_active = N**

A: READ COMMITTED	B: READ COMMITTED
Tr A ID = 80 Tr A START: 19:14:43 in READ-COMMITTED Tr A, SELECT: 19:14:43 sb_is_active = N Tr A UPDATE: 19:14:53 Tr A COMMIT: 19:14:53 After A, SELECT: 19:15:03 **sb_is_active = N**	Tr B ID = 79 Tr B START: 19:14:43 in READ-COMMITTED Tr B, SELECT: 19:14:48 sb_is_active = N Tr B UPDATE: 19:14:58 Tr B COMMIT: 19:14:58 After B, SELECT: 19:14:58 **sb_is_active = N**

A: READ COMMITTED	B: REPEATABLE READ
Tr A ID = 83 Tr A START: 19:17:00 in READ-COMMITTED Tr A, SELECT: 19:17:00 sb_is_active = N Tr A UPDATE: 19:17:10 Tr A COMMIT: 19:17:10 After A, SELECT: 19:17:20 **sb_is_active = N**	Tr B ID = 82 Tr B START: 19:17:00 in REPEATABLE-READ Tr B, SELECT: 19:17:05 sb_is_active = N Tr B UPDATE: 19:17:15 Tr B COMMIT: 19:17:15 After B, SELECT: 19:17:15 **sb_is_active = N**

A: READ COMMITTED	B: SERIALIZABLE
Tr A ID = 86 Tr A START: 19:19:07 in READ-COMMITTED Tr A, SELECT: 19:19:07 sb_is_active = N Tr A UPDATE: 19:19:17 Tr A COMMIT: 19:19:17 After A, SELECT: 19:19:27 **sb_is_active = N**	Tr B ID = 85 Tr B START: 19:19:06 in SERIALIZABLE Tr B, SELECT: 19:19:11 sb_is_active = N Tr B UPDATE: 19:19:21 Tr B COMMIT: 19:19:21 After B, SELECT: 19:19:21 **sb_is_active = N**

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно
	READ COMMITTED	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно
	REPEATABLE READ	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно
	SERIALIZABLE	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Возможна взаимная блокировка с отменой транзакции B

Если в данном эксперименте убрать чтение информации перед её обновлением (строки 15-19 для транзакции A, и строки 20-24 для транзакции B), то при любой комбинации уровней изолированности результат будет одним и тем же: изменения, выполненные транзакцией A, будут утеряны.

Чтобы получить другой вариант поведения СУБД, необходимо явно блокировать читаемые записи (**SELECT ... LOCK IN SHARE MODE** или **SELECT ... FOR UPDATE**)[36] в первой операции чтения. В данном случае это не было сделано, чтобы продемонстрировать наиболее типичное поведение MySQL. Но если добавить указанные блокировки, поведение MySQL изменится и примет следующий вид.

Итоговые результаты взаимодействия транзакций при использовании **LOCK IN SHARE MODE** для первой операции чтения. Важно отметить, что в некоторых случаях взаимная блокировка нарушает работу обеих транзакций, но в большинстве случаев СУБД отменяет транзакцию B, позволяя транзакции A успешно выполниться.

[36] http://dev.mysql.com/doc/refman/5.6/en/innodb-locking-reads.html

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций
	READ COMMITTED	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций
	REPEATABLE READ	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций
	SERIALIZABLE	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций	Взаимная блокировка транзакций

Итоговые результаты взаимодействия транзакций при использовании `FOR UPDATE` для первой операции чтения.

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A
	READ COMMITTED	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A
	REPEATABLE READ	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A
	SERIALIZABLE	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A	Обновление транзакции A утеряно, транзакция B ждёт завершения A

Очевидно, что использованием различных комбинаций способов выполнения (или вовсе невыполнение) первой операции чтения в начале каждой транзакции можно получить ещё больше вариантов поведения СУБД.

Неповторяющееся чтение в MySQL может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MySQL Решение 6.2.2.a (код для исследования аномалии неповторяющегося чтения)

```
-- Транзакция A:
SELECT CONCAT('Tr A ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr A START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';


SELECT SLEEP(5);


SELECT CONCAT('Tr A UPDATE: ',
              CURTIME());
UPDATE `subscriptions`
SET    `sb_is_active` =
 CASE
  WHEN `sb_is_active` = 'Y' THEN 'N'
  WHEN `sb_is_active` = 'N' THEN 'Y'
 END
WHERE  `sb_id` = 2;
SELECT CONCAT('Tr A COMMIT: ',
              CURTIME());
COMMIT;
```

```
-- Транзакция B:
SELECT CONCAT('Tr B ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr B START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';
SELECT CONCAT('Tr A SELECT-1: ',
              CURTIME());

SELECT `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` = 2;




SELECT SLEEP(10);




SELECT CONCAT('Tr A SELECT-2: ',
              CURTIME());

SELECT `sb_is_active`
FROM   `subscriptions`
WHERE  `sb_id` = 2;
SELECT CONCAT('Tr B COMMIT: ',
              CURTIME());
COMMIT;
```

Приведём пример журнала выполнения этого кода для ситуации, когда транзакция A выполняется на уровне изолированности **`REPEATABLE READ`** и конкурирует с транзакцией B, последовательно выполняемой во всех поддерживаемых MySQL уровнях изолированности.

A: REPEATABLE READ	B: READ UNCOMMITTED
Tr A ID = 151 Tr A START: 20:24:12 in REPEATABLE-READ Tr A UPDATE: 20:24:17 Tr A COMMIT: 20:24:17	Tr B ID = 152 Tr B START: 20:24:12 in READ-UNCOMMITTED Tr B SELECT-1: 20:24:12 **sb_is_active = N** Tr B SELECT-2: 20:24:22 **sb_is_active = Y** Tr B COMMIT: 20:24:22

A: REPEATABLE READ	B: READ COMMITTED
Tr A ID = 153 Tr A START: 20:25:29 in REPEATABLE-READ Tr A UPDATE: 20:25:34 Tr A COMMIT: 20:25:34	Tr B ID = 154 Tr B START: 20:25:29 in READ-COMMITTED Tr B SELECT-1: 20:25:29 **sb_is_active = Y** Tr B SELECT-2: 20:25:39 **sb_is_active = N** Tr B COMMIT: 20:25:39

A: REPEATABLE READ	B: REPEATABLE READ
Tr A ID = 156 Tr A START: 20:26:24 in REPEATABLE-READ Tr A UPDATE: 20:26:29 Tr A COMMIT: 20:26:29	Tr B ID = 155 Tr B START: 20:26:24 in REPEATABLE-READ Tr B SELECT-1: 20:26:24 **sb_is_active = N** Tr B SELECT-2: 20:26:34 **sb_is_active = N** Tr B COMMIT: 20:26:34

A: REPEATABLE READ	B: SERIALIZABLE
Tr A ID = 157 Tr A START: 20:27:15 in REPEATABLE-READ Tr A UPDATE: 20:27:20 Tr A COMMIT: 20:27:25	Tr B ID = 158 Tr B START: 20:27:15 in SERIALIZABLE Tr B SELECT-1: 20:27:15 sb_is_active = Y Tr B SELECT-2: 20:27:25 sb_is_active = Y Tr B COMMIT: 20:27:25

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	READ COMMITTED	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	REPEATABLE READ	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	SERIALIZABLE	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B

Чтобы получить другой вариант поведения СУБД, необходимо явно блокировать читаемые записи (**`SELECT ... LOCK IN SHARE MODE`** *или* **`SELECT ... FOR UPDATE`**)[37] в первой операции чтения. В данном случае это не было сделано, чтобы продемонстрировать наиболее типичное поведение MySQL. Проверить же остальные случаи реакции СУБД вам предлагается самостоятельно в задании 6.2.2.TSK.D{485}.

[37] http://dev.mysql.com/doc/refman/5.6/en/innodb-locking-reads.html

Фантомное чтение в MySQL может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MySQL Решение 6.2.2.a (код для исследования аномалии фантомного чтения)

```
-- Транзакция A:
SELECT CONCAT('Tr A ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr A START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';


SELECT SLEEP(5);


SELECT CONCAT('Tr A INSERT: ',
              CURTIME());
INSERT INTO `subscriptions`
            (`sb_id`,
             `sb_subscriber`,
             `sb_book`,
             `sb_start`,
             `sb_finish`,
             `sb_is_active`)
     VALUES (1000,
             1,
             1,
             '2025-01-12',
             '2026-02-12',
             'N');
SELECT SLEEP(10);


SELECT CONCAT('Tr A ROLLBACK: ',
              CURTIME());
ROLLBACK;
```

```
-- Транзакция B:
SELECT CONCAT('Tr B ID = ',
              CONNECTION_ID());
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL {УРОВЕНЬ};
START TRANSACTION;
SELECT CONCAT('Tr B START: ',
              CURTIME(), ' in ');
SELECT `VARIABLE_VALUE`
FROM   `information_schema`.
       `session_variables`
WHERE  `VARIABLE_NAME` =
       'tx_isolation';
SELECT CONCAT('Tr B COUNT-1: ',
              CURTIME());
SELECT COUNT(*)
FROM   `subscriptions`
WHERE  `sb_id` > 500;


SELECT SLEEP(10);


SELECT CONCAT('Tr B COUNT-2: ',
              CURTIME());
SELECT COUNT(*)
FROM   `subscriptions`
WHERE  `sb_id` > 500;
SELECT SLEEP(15);


SELECT CONCAT('Tr B COUNT-3: ',
              CURTIME());
SELECT COUNT(*)
FROM   `subscriptions`
WHERE  `sb_id` > 500;
SELECT CONCAT('Tr B COMMIT: ',
              CURTIME());
COMMIT;
```

Приведём пример журнала выполнения этого кода для ситуации, когда транзакция A выполняется на уровне изолированности **SERIALIZABLE** и конкурирует с транзакцией B, последовательно выполняемой во всех поддерживаемых MySQL уровнях изолированности.

A: SERIALIZABLE	B: READ UNCOMMITTED
Tr A ID = 198 Tr A START: 21:08:32 in SERIALIZABLE Tr A INSERT: 21:08:37 Tr A ROLLBACK: 21:08:47	Tr B ID = 197 Tr B START: 21:08:32 in READ-UNCOMMITTED Tr B COUNT-1: 21:08:32 **COUNT(*) = 0** Tr B COUNT-2: 21:08:42 **COUNT(*) = 1** Tr B COUNT-3: 21:08:57 **COUNT(*) = 0** Tr B COMMIT: 21:08:57

A: SERIALIZABLE	B: READ COMMITTED
Tr A ID = 200 Tr A START: 21:09:34 in SERIALIZABLE Tr A INSERT: 21:09:39 Tr A ROLLBACK: 21:09:49	Tr B ID = 199 Tr B START: 21:09:34 in READ-COMMITTED Tr B COUNT-1: 21:09:34 **COUNT(*) = 0** Tr B COUNT-2: 21:09:44 **COUNT(*) = 0** Tr B COUNT-3: 21:09:59 **COUNT(*) = 0** Tr B COMMIT: 21:09:59

A: SERIALIZABLE	B: REPEATABLE READ
Tr A ID = 201 Tr A START: 21:10:28 in SERIALIZABLE Tr A INSERT: 21:10:33 Tr A ROLLBACK: 21:10:43	Tr B ID = 202 Tr B START: 21:10:28 in REPEATABLE-READ Tr B COUNT-1: 21:10:28 **COUNT(*) = 0** Tr B COUNT-2: 21:10:38 **COUNT(*) = 0** Tr B COUNT-3: 21:10:53 **COUNT(*) = 0** Tr B COMMIT: 21:10:53

A: SERIALIZABLE	B: SERIALIZABLE
Tr A ID = 203 Tr A START: 21:11:29 in SERIALIZABLE Tr A INSERT: 21:11:34 Tr A ROLLBACK: 21:11:44	Tr B ID = 204 Tr B START: 21:11:29 in SERIALIZABLE Tr B COUNT-1: 21:11:29 **COUNT(*) = 0** Tr B COUNT-2: 21:11:39 **COUNT(*) = 0** Tr B COUNT-3: 21:11:59 **COUNT(*) = 0** Tr B COMMIT: 21:11:59

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B			
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», COUNT-3 ждёт завершения транзакции A
	READ COMMITTED	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», COUNT-3 ждёт завершения транзакции A
	REPEATABLE READ	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», COUNT-3 ждёт завершения транзакции A
	SERIALIZABLE	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», COUNT-3 ждёт завершения транзакции A

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Данная СУБД поддерживает пять уровней изолированности транзакций, комбинации которых мы и рассмотрим:

- **`READ UNCOMMITTED`**;
- **`READ COMMITTED`**;
- **`REPEATABLE READ`**;
- **`SNAPSHOT`**;
- **`SERIALIZABLE`**.

Для выполнения эксперимента используем командный файл:

```
start cmd.exe /c "sqlcmd -S КОМПЬЮТЕР\СЕРВЕР -i a.sql & pause"
start cmd.exe /c "sqlcmd -S КОМПЬЮТЕР\СЕРВЕР -i b.sql & pause"
```

Для упрощения кода приведённых далее запросов создадим функции **`GET_ISOLATION_LEVEL`** и **`GET_CT`**, возвращающие, соответственно, значение текущего уровня изолированности транзакции и значение текущего времени.

MS SQL Решение 6.2.2.a (код и запрос для проверки работоспособности сервисных функций)

```
CREATE FUNCTION GET_ISOLATION_LEVEL()
RETURNS NVARCHAR(50)
BEGIN
 DECLARE @IsolationLevel NVARCHAR(50);
 SET @IsolationLevel = (
 SELECT CASE [transaction_isolation_level]
 WHEN 0 THEN 'Unspecified'
 WHEN 1 THEN 'Read Uncommitted'
 WHEN 2 THEN 'Read Committed'
 WHEN 3 THEN 'Repeatable Read'
 WHEN 4 THEN 'Serializable'
 WHEN 5 THEN 'Snapshot' END AS TRANSACTION_ISOLATION_LEVEL
 FROM [sys].[dm_exec_sessions]
 WHERE [session_id] = @@SPID);
 RETURN @IsolationLevel;
END;
GO

```

MS SQL Решение 6.2.2.a (код и запрос для проверки работоспособности сервисных функций) (продолжение)

```
CREATE FUNCTION GET_CT()
RETURNS NVARCHAR(50)
BEGIN
 DECLARE @CT NVARCHAR(50);
 SET @CT = CONVERT(NVARCHAR(12), GETDATE(), 114);
 RETURN @CT;
END;
GO

PRINT dbo.GET_ISOLATION_LEVEL();
PRINT dbo.GET_CT();
```

Обратите внимание на два важных момента:

- для обеспечения работоспособности уровня изолированности транзакции **SNAPSHOT** необходимо выполнить команду **ALTER DATABASE [имя_базы_данных] SET ALLOW_SNAPSHOT_ISOLATION ON**;
- в представленном ниже коде мы будем дважды подтверждать каждую транзакцию, показывая текущий уровень вложенности транзакций (**@@TRANCOUNT**), что вызвано особенностью работы MS SQL Server в режиме **IMPLICIT_TRANSACTIONS ON**: в этом режиме начало транзакции переводит **@@TRANCOUNT** в значение 2, а не в 1.

Грязное чтение в MS SQL Server может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MS SQL Решение 6.2.2.a (код для исследования аномалии грязного чтения)

`-- Транзакция A:`
`PRINT CONCAT('Tr A ID = ', @@SPID);`
`SET IMPLICIT_TRANSACTIONS ON;`
`SET TRANSACTION ISOLATION`
`LEVEL {УРОВЕНЬ};`
	`-- Транзакция B:`
	`PRINT CONCAT('Tr B ID = ', @@SPID);`
	`SET IMPLICIT_TRANSACTIONS ON;`
	`SET TRANSACTION ISOLATION`
	`LEVEL {УРОВЕНЬ};`

`BEGIN TRANSACTION;`
`PRINT CONCAT('Tr A START: ',`
`dbo.GET_CT(), ' in ',`
`dbo.GET_ISOLATION_LEVEL());`
	`BEGIN TRANSACTION;`
	`PRINT CONCAT('Tr B START: ',`
	`dbo.GET_CT(), ' in ',`
	`dbo.GET_ISOLATION_LEVEL());`

	`PRINT CONCAT('Tr B SELECT-1: ',`
	`dbo.GET_CT());`
	`SELECT [sb_is_active]`
	`FROM [subscriptions]`
	`WHERE [sb_id] = 2;`

`WAITFOR DELAY '00:00:05';`	

`PRINT CONCAT('Tr A UPDATE: ',`
`dbo.GET_CT());`
`UPDATE [subscriptions]`
`SET [sb_is_active] =`
`CASE`
`WHEN [sb_is_active] = 'Y' THEN 'N'`
`WHEN [sb_is_active] = 'N' THEN 'Y'`
`END`
`WHERE [sb_id] = 2;`
	`WAITFOR DELAY '00:00:10';`

	`PRINT CONCAT('Tr B SELECT-2: ',`
	`dbo.GET_CT());`
	`SELECT [sb_is_active]`
	`FROM [subscriptions]`
	`WHERE [sb_id] = 2;`
	`PRINT CONCAT('Tr B COMMIT: ',`
	`dbo.GET_CT());`
	`COMMIT;`
	`PRINT CONCAT('TrC = ', @@TRANCOUNT);`
	`COMMIT;`
	`PRINT CONCAT('TrC = ', @@TRANCOUNT);`

`WAITFOR DELAY '00:00:20';`	

MS SQL Решение 6.2.2.a (код для исследования аномалии грязного чтения) (продолжение)

```
35  PRINT CONCAT('Tr A ROLLBACK: ',
36  dbo.GET_CT());
37  ROLLBACK;
38  PRINT CONCAT('TrC = ', @@TRANCOUNT);
```

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B				
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SNAPSHOT	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение, SELECT-2 в транзакции B ждёт завершения A	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B
	READ COMMITTED	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение, SELECT-2 в транзакции B ждёт завершения A	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B
	REPEATABLE READ	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение, SELECT-2 в транзакции B ждёт завершения A	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B
	SNAPSHOT	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение, SELECT-2 в транзакции B ждёт завершения A	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B
	SERIALIZABLE	Транзакция B успевает прочитать незафиксированное значение	Транзакция B оба раза читает исходное (корректное) значение, SELECT-2 в транзакции B ждёт завершения A	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A ждёт завершения B

Потерянное обновление MS SQL Server может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MS SQL Решение 6.2.2.a (код для исследования аномалии потерянного обновления)

```
-- Транзакция A:
PRINT CONCAT('Tr A ID = ', @@SPID);
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION
LEVEL {УРОВЕНЬ};
BEGIN TRANSACTION;
PRINT CONCAT('Tr A START: ',
dbo.GET_CT(), ' in ',
dbo.GET_ISOLATION_LEVEL());
PRINT CONCAT('Tr A SELECT: ',
dbo.GET_CT());
SELECT [sb_is_active]
FROM   [subscriptions]
WHERE  [sb_id] = 2;


WAITFOR DELAY '00:00:10';



PRINT CONCAT('Tr A UPDATE: ',
dbo.GET_CT());
UPDATE [subscriptions]
SET    [sb_is_active] = 'Y'
WHERE  [sb_id] = 2;
PRINT CONCAT('Tr A COMMIT: ',
dbo.GET_CT());
COMMIT;
PRINT CONCAT('TrC = ', @@TRANCOUNT);
COMMIT;
PRINT CONCAT('TrC = ', @@TRANCOUNT);



WAITFOR DELAY '00:00:10';



PRINT CONCAT('Tr A SELECT AFTER: ',
dbo.GET_CT());
SELECT [sb_is_active]
FROM   [subscriptions]
WHERE  [sb_id] = 2;
```

```
-- Транзакция B:
PRINT CONCAT('Tr B ID = ', @@SPID);
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION
LEVEL {УРОВЕНЬ};
BEGIN TRANSACTION;
PRINT CONCAT('Tr B START: ',
dbo.GET_CT(), ' in ',
dbo.GET_ISOLATION_LEVEL());


WAITFOR DELAY '00:00:05';


PRINT CONCAT('Tr B SELECT: ',
dbo.GET_CT());
SELECT [sb_is_active]
FROM   [subscriptions]
WHERE  [sb_id] = 2;


WAITFOR DELAY '00:00:10';



PRINT CONCAT('Tr B UPDATE: ',
dbo.GET_CT());
UPDATE [subscriptions]
SET    [sb_is_active] = 'N'
WHERE  [sb_id] = 2;
PRINT CONCAT('Tr B COMMIT: ',
dbo.GET_CT());
COMMIT;
PRINT CONCAT('TrC = ', @@TRANCOUNT);
COMMIT;
PRINT CONCAT('TrC = ', @@TRANCOUNT);
PRINT CONCAT('Tr B SELECT AFTER: ',
dbo.GET_CT());
SELECT [sb_is_active]
FROM   [subscriptions]
WHERE  [sb_id] = 2;
```

Итоговые результаты взаимодействия транзакций таковы.

Уровень изолированности транзакции A	Уровень изолированности транзакции B: READ UNCOMMITTED	Уровень изолированности транзакции B: READ COMMITTED	Уровень изолированности транзакции B: REPEATABLE READ	Уровень изолированности транзакции B: SNAPSHOT	Уровень изолированности транзакции B: SERIALIZABLE
READ UNCOMMITTED	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A сохранено, UPDATE в транзакции A ждёт завершения B	Обновление транзакции A сохранено, транзакция B отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Обновление транзакции A сохранено, UPDATE в транзакции A ждёт завершения B
READ COMMITTED	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Обновление транзакции A сохранено, UPDATE в транзакции A ждёт завершения B	Обновление транзакции A сохранено, транзакция B отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Обновление транзакции A сохранено, UPDATE в транзакции A ждёт завершения B
REPEATABLE READ	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Взаимная блокировка транзакций	Обновление транзакции A сохранено, транзакция B отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Взаимная блокировка транзакций
SNAPSHOT	Обновление транзакции A утеряно	Обновление транзакции A утеряно	Транзакция A отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Обновление транзакции A сохранено, транзакция B отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Транзакция A отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)
SERIALIZABLE	Обновление транзакции A утеряно, COMMIT в транзакции A ждёт завершения B	Обновление транзакции A утеряно, COMMIT в транзакции A ждёт завершения B	Взаимная блокировка транзакций	Обновление транзакции A сохранено, транзакция B отменена (попытка обновить заблокированную запись в режиме SNAPSHOT)	Взаимная блокировка транзакций

В учебных целях рассмотрим, что было бы, если бы в коде обеих транзакций мы «забыли» дописать второй **`COMMIT`** (см. подобранности в решении{430} задачи 6.1.1.a{429}).

Итоговые **ошибочные** результаты взаимодействия транзакций приняли бы следующий вид.

		Уровень изолированности транзакции B				
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SNAPSHOT	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно
	READ COMMITTED	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно
	REPEATABLE READ	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Взаимная блокировка транзакций
	SNAPSHOT	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно
	SERIALIZABLE	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Взаимная блокировка транзакций	Обновление транзакции A утеряно, UPDATE в транзакции B ждёт завершения сессии A	Взаимная блокировка транзакций

Обратите внимание на формулировку «**`UPDATE`** в транзакции B ждёт завершения **сессии** A». Здесь имеется в виду именно вся сессия взаимодействия с СУБД, а не просто транзакция. Из-за «забытого» **`COMMIT`** обе транзакции фактически завершаются именно в момент закрытия сессии с СУБД.

Чтобы получить ещё один вариант поведения СУБД, необходимо явно блокировать читаемые записи (**`UPDLOCK`**) в первой операции (чтении). В данном случае это не было сделано, чтобы продемонстрировать наиболее типичное поведение MS SQL Server. Проверить же остальные случаи реакции СУБД вам предлагается самостоятельно в задании 6.2.2.TSK.E{485}.

Неповторяющееся чтение в MS SQL Server может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MS SQL Решение 6.2.2.a (код для исследования аномалии неповторяющегося чтения)

1	`-- Транзакция A:`	`-- Транзакция B:`
2	`PRINT CONCAT('Tr A ID = ', @@SPID);`	`PRINT CONCAT('Tr B ID = ', @@SPID);`
3	`SET IMPLICIT_TRANSACTIONS ON;`	`SET IMPLICIT_TRANSACTIONS ON;`
4	`SET TRANSACTION ISOLATION`	`SET TRANSACTION ISOLATION`
5	`LEVEL {УРОВЕНЬ};`	`LEVEL {УРОВЕНЬ};`
6	`BEGIN TRANSACTION;`	`BEGIN TRANSACTION;`
7	`PRINT CONCAT('Tr A START: ',`	`PRINT CONCAT('Tr B START: ',`
8	`dbo.GET_CT(), ' in ',`	`dbo.GET_CT(), ' in ',`
9	`dbo.GET_ISOLATION_LEVEL());`	`dbo.GET_ISOLATION_LEVEL());`
10		`PRINT CONCAT('Tr B SELECT-1: ',`
11		`dbo.GET_CT());`
12	`WAITFOR DELAY '00:00:05';`	`SELECT [sb_is_active]`
13		`FROM [subscriptions]`
14		`WHERE [sb_id] = 2;`
15	`PRINT CONCAT('Tr A UPDATE: ',`	
16	`dbo.GET_CT());`	
17	`UPDATE [subscriptions]`	
18	`SET [sb_is_active] =`	
19	`CASE`	
20	`WHEN [sb_is_active] = 'Y' THEN 'N'`	`WAITFOR DELAY '00:00:10';`
21	`WHEN [sb_is_active] = 'N' THEN 'Y'`	
22	`END`	
23	`WHERE [sb_id] = 2;`	
24	`PRINT CONCAT('Tr A COMMIT: ',`	
25	`dbo.GET_CT());`	
26	`COMMIT;`	
27	`PRINT CONCAT('TrC = ', @@TRANCOUNT);`	
28	`COMMIT;`	
29	`PRINT CONCAT('TrC = ', @@TRANCOUNT);`	
30		`PRINT CONCAT('Tr B SELECT-2: ',`
31		`dbo.GET_CT());`
32		`SELECT [sb_is_active]`
33		`FROM [subscriptions]`
34		`WHERE [sb_id] = 2;`
35		`PRINT CONCAT('Tr B COMMIT: ',`
36		`dbo.GET_CT());`
37		`COMMIT;`
38		`PRINT CONCAT('TrC = ', @@TRANCOUNT);`
39		`COMMIT;`
40		`PRINT CONCAT('TrC = ', @@TRANCOUNT);`

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B				
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SNAPSHOT	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	READ COMMITTED	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	REPEATABLE READ	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	SNAPSHOT	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B
	SERIALIZABLE	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные, транзакция A ждёт завершения B

Чтобы получить ещё один вариант поведения СУБД, необходимо явно блокировать читаемые записи (**`UPDLOCK`**) в первой операции чтения. В данном случае это не было сделано, чтобы продемонстрировать наиболее типичное поведение MS SQL Server. Проверить же остальные случаи реакции СУБД вам предлагается самостоятельно в задании 6.2.2.TSK.E[485].

Фантомное чтение в MS SQL Server может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

MS SQL Решение 6.2.2.a (код для исследования аномалии фантомного чтения)

```
1  -- Транзакция A:
2  PRINT CONCAT('Tr A ID = ', @@SPID);
3  SET IMPLICIT_TRANSACTIONS ON;
4  SET TRANSACTION ISOLATION
5  LEVEL {УРОВЕНЬ};
6  BEGIN TRANSACTION;
7  PRINT CONCAT('Tr A START: ',
8  dbo.GET_CT(), ' in ',
9  dbo.GET_ISOLATION_LEVEL());
```

```
-- Транзакция B:
PRINT CONCAT('Tr B ID = ', @@SPID);
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION
LEVEL {УРОВЕНЬ};
BEGIN TRANSACTION;
PRINT CONCAT('Tr B START: ',
dbo.GET_CT(), ' in ',
dbo.GET_ISOLATION_LEVEL());
```

MS SQL Решение 6.2.2.a (код для исследования аномалии фантомного чтения) (продолжение)

```
10                                          PRINT CONCAT('Tr B COUNT-1: ',
11                                          dbo.GET_CT());
12  WAITFOR DELAY '00:00:05';               SELECT COUNT(*)
13                                          FROM   [subscriptions]
14                                          WHERE  [sb_id] > 500;
15  PRINT CONCAT('Tr A INSERT: ',
16  dbo.GET_CT());
17
18  SET IDENTITY_INSERT
19  [subscriptions] ON;
20
21  INSERT INTO [subscriptions]
22              ([sb_id],
23               [sb_subscriber],
24               [sb_book],                 WAITFOR DELAY '00:00:10';
25               [sb_start],
26               [sb_finish],
27               [sb_is_active])
28      VALUES (1000,
29               1,
30               1,
31               '2025-01-12',
32               '2026-02-12',
33               'N');
34
35  SET IDENTITY_INSERT
36  [subscriptions] OFF;
37                                          PRINT CONCAT('Tr B COUNT-2: ',
38                                          dbo.GET_CT());
39  WAITFOR DELAY '00:00:10';
40                                          SELECT COUNT(*)
41                                          FROM   [subscriptions]
42                                          WHERE  [sb_id] > 500;
43  PRINT CONCAT('Tr A ROLLBACK: ',
44  dbo.GET_CT());                          WAITFOR DELAY '00:00:15';
45  ROLLBACK;
46  PRINT CONCAT('TrC = ', @@TRANCOUNT);
47                                          PRINT CONCAT('Tr B COUNT-3: ',
48                                          dbo.GET_CT());
49                                          SELECT COUNT(*)
50                                          FROM   [subscriptions]
51                                          WHERE  [sb_id] > 500;
52                                          PRINT CONCAT('Tr B COMMIT: ',
53                                          dbo.GET_CT());
54                                          COMMIT;
55                                          PRINT CONCAT('TrC = ', @@TRANCOUNT);
56                                          COMMIT;
57                                          PRINT CONCAT('TrC = ', @@TRANCOUNT);
```

Итоговые результаты взаимодействия транзакций таковы.

		Уровень изолированности транзакции B				
		READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SNAPSHOT	SERIALIZABLE
Уровень изолированности транзакции A	READ UNCOMMITTED	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», INSERT в транзакции A ждёт завершения B
	READ COMMITTED	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», INSERT в транзакции A ждёт завершения B
	REPEATABLE READ	Транзакция B успевает обработать «фантомную запись», INSERT в транзакции A ждёт завершения B	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», INSERT в транзакции A ждёт завершения B
	SNAPSHOT	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», INSERT в транзакции A ждёт завершения B
	SERIALIZABLE	Транзакция B успевает обработать «фантомную запись»	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи», её COUNT-2 ждёт завершения A	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи», INSERT в транзакции A ждёт завершения B

На этом решение для MS SQL Server завершено.

Переходим к Oracle. Данная СУБД поддерживает два уровня изолированности транзакций и два режима транзакций, комбинации которых мы и рассмотрим:

- Уровни изолированности:
  - `READ COMMITTED`;
  - `SERIALIZABLE`.
- Режимы:
  - `READ ONLY` (фактически, это — тоже уровень изолированности, эквивалентный `REPEATABLE READ` и/или `SERIALIZABLE` в других СУБД);
  - `READ WRITE`.

Для выполнения эксперимента используем командный файл:

```
start cmd.exe /c "echo exit | sqlplus ПОЛЬЗОВАТЕЛЬ/ПАРОЛЬ@КОМПЬЮТЕР @a.sql & pause"
start cmd.exe /c "echo exit | sqlplus ПОЛЬЗОВАТЕЛЬ/ПАРОЛЬ@КОМПЬЮТЕР @b.sql & pause"
```

Для упрощения кода приведённых далее запросов создадим функции **GET_IDS_AND_ISOLATION_LEVEL** и **GET_CT**, возвращающие, соответственно, значение параметров транзакции и значение текущего времени.

Oracle Решение 6.2.2.a (код и запрос для проверки работоспособности сервисных функций)

```
CREATE FUNCTION GET_IDS_AND_ISOLATION_LEVEL
 RETURN NVARCHAR2
 IS
 session_id NUMBER(10);
 session_sn NUMBER(10);
 session_isolation_level NVARCHAR2(150);
 trans_id VARCHAR(100);
BEGIN
 trans_id := DBMS_TRANSACTION.LOCAL_TRANSACTION_ID(FALSE);
 SELECT "session".sid AS "session_id",
        "session".serial# AS "session_sn",
        CASE BITAND("transaction".flag, POWER(2, 28))
          WHEN 0 THEN 'READ COMMITTED'
          ELSE 'SERIALIZABLE'
        END AS "session_isolation_level"
 INTO   session_id,
        session_sn,
        session_isolation_level
 FROM   v$transaction "transaction"
        JOIN v$session "session"
          ON "transaction".addr = "session".taddr
             AND "session".sid = SYS_CONTEXT('USERENV', 'SID');

 RETURN 'ID = ' || session_id || ', SN = ' || session_sn || ', in ' ||
        session_isolation_level;
END;

CREATE FUNCTION GET_CT
 RETURN NVARCHAR2
 IS
BEGIN
 RETURN TO_CHAR(SYSTIMESTAMP,'HH24:MI:SS.FF');
END;

SELECT GET_CT FROM DUAL;
SELECT GET_IDS_AND_ISOLATION_LEVEL FROM DUAL;
```

К сожалению, в Oracle 11gR2 не существует документированного способа различить **READ ONLY** и **READ WRITE** транзакции, равно как не существует и способа комбинировать эти параметры с указанием уровня изолированности. И если вторую проблему можно обойти отдельной настройкой параметров сессии и транзакции, с первой проблемой (пока?) ничего нельзя поделать.

Поскольку Oracle (как и MySQL) не поддерживает вложенные транзакции, здесь (в отличие от MS SQL Server) мы не будем следить за текущим уровнем транзакций.

Грязное чтение в Oracle не существует, но проверить это можно выполнением в двух отдельных сессиях следующих блоков кода:

Oracle Решение 6.2.2.a (код для исследования аномалии грязного чтения)

```
1   -- Транзакция A:
2   ALTER SESSION SET
3   ISOLATION_LEVEL = {УРОВЕНЬ};
4   SET TRANSACTION {РЕЖИМ};
5   SELECT 'Tr A: ' ||
6   GET_IDS_AND_ISOLATION_LEVEL
7   FROM DUAL;
8   SELECT 'Tr A START: ' ||
9   GET_CT FROM DUAL;
10
11
12  EXEC DBMS_LOCK.SLEEP(5);
13
14
15  SELECT 'Tr A UPDATE: ' ||
16  GET_CT FROM DUAL;
17  UPDATE "subscriptions"
18  SET    "sb_is_active" =
19   CASE
20    WHEN "sb_is_active" = 'Y' THEN 'N'
21    WHEN "sb_is_active" = 'N' THEN 'Y'
22   END
23  WHERE  "sb_id" = 2;
24
25
26
27
28  EXEC DBMS_LOCK.SLEEP(20);
29
30
31
32  SELECT 'Tr A ROLLBACK: ' ||
33  GET_CT FROM DUAL;
34  ROLLBACK;
```

```
-- Транзакция B:
ALTER SESSION SET
ISOLATION_LEVEL = {УРОВЕНЬ};
SET TRANSACTION {РЕЖИМ};
SELECT 'Tr B: ' ||
GET_IDS_AND_ISOLATION_LEVEL
FROM DUAL;
SELECT 'Tr B START: ' ||
GET_CT FROM DUAL;
SELECT 'Tr B SELECT-1: ' ||
GET_CT FROM DUAL;
SELECT "sb_is_active"
FROM   "subscriptions"
WHERE  "sb_id" = 2;


EXEC DBMS_LOCK.SLEEP(10);


SELECT 'Tr B SELECT-2: ' ||
GET_CT FROM DUAL;
SELECT "sb_is_active"
FROM   "subscriptions"
WHERE  "sb_id" = 2;
SELECT 'Tr B COMMIT: ' ||
GET_CT FROM DUAL;
COMMIT;
```

Итоговые результаты взаимодействия транзакций таковы.

			Уровень изолированности транзакции B			
			READ COMMITTED		SERIALIZABLE	
			READ ONLY	READ WRITE	READ ONLY	READ WRITE
Уровень изолированности транзакции A	READ COMMITTED	READ ONLY	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)
		READ WRITE	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение
	SERIALIZABLE	READ ONLY	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)	Транзакция B оба раза читает исходное (корректное) значение, UPDATE в транзакции A запрещён (R/O)
		READ WRITE	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение	Транзакция B оба раза читает исходное (корректное) значение

Потерянное обновление Oracle может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

Oracle Решение 6.2.2.a (код для исследования аномалии потерянного обновления)

```
1   -- Транзакция A:
2   ALTER SESSION SET
3   ISOLATION_LEVEL = {УРОВЕНЬ};
4   SET TRANSACTION {РЕЖИМ};
5   SELECT 'Tr A: ' ||
6   GET_IDS_AND_ISOLATION_LEVEL
7   FROM DUAL;
8   SELECT 'Tr A START: ' ||
9   GET_CT FROM DUAL;
10  SELECT 'Tr A SELECT: ' ||
11  GET_CT FROM DUAL;
12  SELECT "sb_is_active"
13  FROM   "subscriptions"
14  WHERE  "sb_id" = 2;
15
16
17  EXEC DBMS_LOCK.SLEEP(10);
18
19
20
21  SELECT 'Tr A UPDATE: ' ||
22  GET_CT FROM DUAL;
23
24  UPDATE "subscriptions"
25  SET    "sb_is_active" = 'Y'
26  WHERE  "sb_id" = 2;
27  SELECT 'Tr A COMMIT: ' ||
28  GET_CT FROM DUAL;
29  COMMIT;
```

```
-- Транзакция B:
ALTER SESSION SET
ISOLATION_LEVEL = {УРОВЕНЬ};
SET TRANSACTION {РЕЖИМ};
SELECT 'Tr B: ' ||
GET_IDS_AND_ISOLATION_LEVEL
FROM DUAL;
SELECT 'Tr B START: ' ||
GET_CT FROM DUAL;



EXEC DBMS_LOCK.SLEEP(5);



SELECT 'Tr B SELECT: ' ||
GET_CT FROM DUAL;

SELECT "sb_is_active"
FROM   "subscriptions"
WHERE  "sb_id" = 2;



EXEC DBMS_LOCK.SLEEP(10);
```

Oracle Решение 6.2.2.a (код для исследования аномалии потерянного обновления) (продолжение)

```
30                                          SELECT 'Tr B UPDATE: ' ||
31                                          GET_CT FROM DUAL;
32                                          UPDATE "subscriptions"
33  EXEC DBMS_LOCK.SLEEP(10);               SET    "sb_is_active" = 'N'
34                                          WHERE  "sb_id" = 2;
35                                          SELECT 'Tr B COMMIT: ' ||
36                                          GET_CT FROM DUAL;
37                                          COMMIT;
38  SELECT 'Tr A SELECT AFTER: ' ||         SELECT 'Tr B SELECT AFTER: ' ||
39  GET_CT FROM DUAL;                       GET_CT FROM DUAL;
40  SELECT "sb_is_active"                   SELECT "sb_is_active"
41  FROM   "subscriptions"                  FROM   "subscriptions"
42  WHERE  "sb_id" = 2;                     WHERE  "sb_id" = 2;
```

Итоговые результаты взаимодействия транзакций таковы.

			Уровень изолированности транзакции B			
			READ COMMITTED		SERIALIZABLE	
			READ ONLY	READ WRITE	READ ONLY	READ WRITE
Уровень изолированности транзакции A	READ COMMITTED	READ ONLY	UPDATE в обеих транзакциях запрещён (R/O)	UPDATE в транзакции A запрещён (R/O)	UPDATE в обеих транзакциях запрещён (R/O)	UPDATE в транзакции A запрещён (R/O)
		READ WRITE	UPDATE в транзакции B запрещён (R/O)	Обновление транзакции A утеряно	UPDATE в транзакции B запрещён (R/O)	Обновление транзакции A сохранено, UPDATE в транзакции B не выполнился (см. ниже)
	SERIALIZABLE	READ ONLY	UPDATE в обеих транзакциях запрещён (R/O)	UPDATE в транзакции A запрещён (R/O)	UPDATE в обеих транзакциях запрещён (R/O)	UPDATE в транзакции A запрещён (R/O)
		READ WRITE	UPDATE в транзакции B запрещён (R/O)	Обновление транзакции A утеряно	UPDATE в транзакции B запрещён (R/O)	Обновление транзакции A сохранено, UPDATE в транзакции B не выполнился (см. ниже)

В транзакции B **UPDATE** приводит к ошибке «ORA-08177: can't serialize access for this transaction», т.к. соответствующая запись заблокирована транзакцией A.

Неповторяющееся чтение в Oracle может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

Oracle Решение 6.2.2.a (код для исследования аномалии неповторяющегося чтения)

```
1   -- Транзакция A:                     -- Транзакция B:
2   ALTER SESSION SET                    ALTER SESSION SET
3   ISOLATION_LEVEL = {УРОВЕНЬ};         ISOLATION_LEVEL = {УРОВЕНЬ};
4   SET TRANSACTION {РЕЖИМ};             SET TRANSACTION {РЕЖИМ};
5   SELECT 'Tr A: ' ||                   SELECT 'Tr B: ' ||
6   GET_IDS_AND_ISOLATION_LEVEL          GET_IDS_AND_ISOLATION_LEVEL
7   FROM DUAL;                           FROM DUAL;
8   SELECT 'Tr A START: ' ||             SELECT 'Tr B START: ' ||
9   GET_CT FROM DUAL;                    GET_CT FROM DUAL;
```

Oracle Решение 6.2.2.a (код для исследования аномалии неповторяющегося чтения) (продолжение)

```
10                                          SELECT 'Tr B SELECT-1: ' ||
11                                          GET_CT FROM DUAL;
12  EXEC DBMS_LOCK.SLEEP(5);                SELECT "sb_is_active"
13                                          FROM   "subscriptions"
14                                          WHERE  "sb_id" = 2;
15  SELECT 'Tr A UPDATE: ' ||
16  GET_CT FROM DUAL;
17  UPDATE "subscriptions"
18  SET    "sb_is_active" =
19   CASE
20    WHEN "sb_is_active" = 'Y' THEN 'N'    EXEC DBMS_LOCK.SLEEP(10);
21    WHEN "sb_is_active" = 'N' THEN 'Y'
22   END
23  WHERE  "sb_id" = 2;
24  SELECT 'Tr A COMMIT: ' ||
25  GET_CT FROM DUAL;
26  COMMIT;
27                                          SELECT 'Tr B SELECT-2: ' ||
28                                          GET_CT FROM DUAL;
29                                          SELECT "sb_is_active"
30                                          FROM   "subscriptions"
31                                          WHERE  "sb_id" = 2;
32                                          SELECT 'Tr B COMMIT: ' ||
33                                          GET_CT FROM DUAL;
34                                          COMMIT;
```

Итоговые результаты взаимодействия транзакций таковы.

			Уровень изолированности транзакции B			
			READ COMMITTED		SERIALIZABLE	
			READ ONLY	READ WRITE	READ ONLY	READ WRITE
Уровень изолированности транзакции A	READ COMMITTED	READ ONLY	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)
		READ WRITE	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные
	SERIALIZABLE	READ ONLY	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)	Первый и второй SELECT возвратили одинаковые данные, UPDATE в транзакции A запрещён (R/O)
		READ WRITE	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили разные данные	Первый и второй SELECT возвратили одинаковые данные	Первый и второй SELECT возвратили одинаковые данные

Фантомное чтение в Oracle может быть исследовано выполнением в двух отдельных сессиях следующих блоков кода:

Oracle Решение 6.2.2.a (код для исследования аномалии фантомного чтения)

```
-- Транзакция A:
ALTER SESSION SET
ISOLATION_LEVEL = {УРОВЕНЬ};
SET TRANSACTION {РЕЖИМ};
SELECT 'Tr A: ' ||
GET_IDS_AND_ISOLATION_LEVEL
FROM DUAL;
SELECT 'Tr A START: ' ||
GET_CT FROM DUAL;
```

```
-- Транзакция B:
ALTER SESSION SET
ISOLATION_LEVEL = {УРОВЕНЬ};
SET TRANSACTION {РЕЖИМ};
SELECT 'Tr B: ' ||
GET_IDS_AND_ISOLATION_LEVEL
FROM DUAL;
SELECT 'Tr B START: ' ||
GET_CT FROM DUAL;
```

```


EXEC DBMS_LOCK.SLEEP(5);


```

```
SELECT 'Tr B COUNT-1: ' ||
GET_CT FROM DUAL;
SELECT COUNT(*)
FROM   "subscriptions"
WHERE  "sb_id" > 500;
```

```
SELECT 'Tr A INSERT: ' ||
GET_CT FROM DUAL;
INSERT INTO "subscriptions"
            ("sb_id",
             "sb_subscriber",
             "sb_book",
             "sb_start",
             "sb_finish",
             "sb_is_active")
     VALUES (1000,
             1,
             1,
             TO_DATE('2025-01-12',
                     'YYYY-MM-DD'),
             TO_DATE('2026-01-12',
                     'YYYY-MM-DD'),
             'N');
```

```


EXEC DBMS_LOCK.SLEEP(10);


```

```


EXEC DBMS_LOCK.SLEEP(10);


```

```
SELECT 'Tr B COUNT-2: ' ||
GET_CT FROM DUAL;
SELECT COUNT(*)
FROM   "subscriptions"
WHERE  "sb_id" > 500;
```

```
SELECT 'Tr A ROLLBACK: ' ||
GET_CT FROM DUAL;
ROLLBACK;
```

```


EXEC DBMS_LOCK.SLEEP(15);


```

```
SELECT 'Tr B COUNT-3: ' ||
GET_CT FROM DUAL;
SELECT COUNT(*)
FROM   "subscriptions"
WHERE  "sb_id" > 500;
SELECT 'Tr B COMMIT: ' ||
GET_CT FROM DUAL;
COMMIT;
```

Перед выполнением представленных выше блоков кода необходимо отключить триггер, обеспечивающий автоинкрементацию первичного ключа в таблице **subscriptions** (**ALTER TRIGGER "TRG_subscriptions_sb_id" DISABLE**), а после проведения эксперимента — снова включить этот триггер (**ALTER TRIGGER "TRG_subscriptions_sb_id" ENABLE**).

Добавлять эти команды непосредственно перед и после **INSERT** в транзакции A нельзя, т.к. **ALTER TRIGGER** приводит к автоматическому подтверждению предыдущей транзакции и запуску новой.

Итоговые результаты взаимодействия транзакций таковы.

			Уровень изолированности транзакции B			
			READ COMMITTED		SERIALIZABLE	
			READ ONLY	READ WRITE	READ ONLY	READ WRITE
Уровень изолированности транзакции A	READ COMMITTED	READ ONLY	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)
		READ WRITE	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»
	SERIALIZABLE	READ ONLY	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)	INSERT в транзакции A запрещён (R/O)
		READ WRITE	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»	Транзакция B не получает доступа к «фантомной записи»

На этом решение данной задачи завершено.

Решение 6.2.2.b{456}.

В решении{456} задачи 6.2.2.a{456} в некоторых случаях мы получали ситуацию взаимной блокировки транзакций, но сейчас мы рассмотрим код, который гарантированно приводит к такой ситуации во всех трёх СУБД.

На низких уровнях изолированности транзакций у СУБД может появиться возможность избежать взаимной блокировки, потому мы используем уровень **SERIALIZABLE**. Исследование поведения СУБД при работе на других уровнях изолированности вам предлагается провести самостоятельно в задании 6.2.2.TSK.F{485}.

Важно отметить, что только MS SQL Server позволяет указывать приоритет транзакции, который учитывает при принятии решения о том, какая из двух взаимно заблокированных транзакций будет отменена, MySQL и Oracle принимают такое решение полностью самостоятельно.

Представленный ниже код работает по следующему алгоритму:

- транзакция A обновляет первую таблицу;
- транзакция B обновляет вторую таблицу;
- транзакция A пытается обновить вторую таблицу (ряд, заблокированный транзакцией B);
- транзакция B пытается обновить первую таблицу (ряд, заблокированный транзакцией A);
- наступает взаимная блокировка транзакций.

Рассмотрим код, реализующий этот алгоритм.

Решение для MySQL выглядит следующим образом.

MySQL Решение 6.2.2.b

```
1   -- Транзакция A:
2   SET autocommit = 0;
3   SET SESSION TRANSACTION
4   ISOLATION LEVEL SERIALIZABLE;
5   START TRANSACTION;
6   UPDATE `books`
7   SET    `b_name` =
8          CONCAT(`b_name`, '.')
9   WHERE  `b_id` = 1;
10
11  SELECT SLEEP(5);
12
13
14  UPDATE `subscribers`
15  SET    `s_name` =
16         CONCAT(`s_name`, '.')
17  WHERE  `s_id` = 1;
18  COMMIT;
19
20
21
22
```

```
-- Транзакция B:
SET autocommit = 0;
SET SESSION TRANSACTION
ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;

SELECT SLEEP(3);

UPDATE `subscribers`
SET    `s_name` =
       CONCAT(`s_name`, '.')
WHERE  `s_id` = 1;

SELECT SLEEP(3);

UPDATE `books`
SET    `b_name` =
       CONCAT(`b_name`, '.')
WHERE  `b_id` = 1;
COMMIT;
```

Решение для MS SQL Server выглядит следующим образом. Обратите внимание на строку 5, в которой для первой транзакции устанавливается повышенный, а для второй — пониженный приоритет, в силу чего СУБД всегда будет отменять вторую транзакцию, позволяя первой успешно завершиться.

MS SQL Решение 6.2.2.b

```
1   -- Транзакция A:
2   SET IMPLICIT_TRANSACTIONS ON;
3   SET TRANSACTION ISOLATION
4   LEVEL SERIALIZABLE;
5   SET DEADLOCK_PRIORITY HIGH;
6   BEGIN TRANSACTION;
6   UPDATE [books]
7   SET    [b_name] =
8          CONCAT([b_name], '.')
9   WHERE  [b_id] = 1;
10
11  WAITFOR DELAY '00:00:05';
12
13
14  UPDATE [subscribers]
15  SET    [s_name] =
16         CONCAT([s_name], '.')
17  WHERE  [s_id] = 1;
18  COMMIT;
19
20
21
22
```

```
-- Транзакция B:
SET IMPLICIT_TRANSACTIONS ON;
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
SET DEADLOCK_PRIORITY LOW;
BEGIN TRANSACTION;

WAITFOR DELAY '00:00:03';

UPDATE [subscribers]
SET    [s_name] =
       CONCAT([s_name], '.')
WHERE  [s_id] = 1;

WAITFOR DELAY '00:00:03';

UPDATE [books]
SET    [b_name] =
       CONCAT([b_name], '.')
WHERE  [b_id] = 1;
COMMIT;
```

Решение для Oracle выглядит следующим образом.

Oracle Решение 6.2.2.b

```
-- Транзакция A:
ALTER SESSION SET
ISOLATION_LEVEL = SERIALIZABLE;
SET TRANSACTION READ WRITE;
UPDATE "books"
SET    "b_name" =
       CONCAT("b_name", '.')
WHERE  "b_id" = 1;

EXEC DBMS_LOCK.SLEEP(5);


UPDATE "subscribers"
SET    "s_name" =
       CONCAT("s_name", '.')
WHERE  "s_id" = 1;
COMMIT;
```

```
-- Транзакция B:
ALTER SESSION SET
ISOLATION_LEVEL = SERIALIZABLE;
SET TRANSACTION READ WRITE;

EXEC DBMS_LOCK.SLEEP(3);


UPDATE "subscribers"
SET    "s_name" =
       CONCAT("s_name", '.')
WHERE  "s_id" = 1;

EXEC DBMS_LOCK.SLEEP(3);


UPDATE "books"
SET    "b_name" =
       CONCAT("b_name", '.')
WHERE  "b_id" = 1;
COMMIT;
```

На этом решение данной задачи завершено.

Задание 6.2.2.TSK.A: повторить исследование, представленное в решении{456} задачи 6.2.2.a{456} и лично посмотреть на поведение всех трёх СУБД во всех рассмотренных ситуациях.

Задание 6.2.2.TSK.B: повторить исследование, представленное в решении{483} задачи 6.2.2.b{456} и лично посмотреть на поведение всех трёх СУБД во всех рассмотренных ситуациях.

Задание 6.2.2.TSK.C: написать код, в котором запрос, инвертирующий значения поля `sb_is_active` таблицы `subscriptions` с Y на N и наоборот, будет иметь максимальные шансы на успешное завершение в случае возникновения ситуации взаимной блокировки с другими транзакциями.

Задание 6.2.2.TSK.D: провести исследование поведения MySQL в контексте аномалии неповторяющегося чтения, выполняя первую операцию в каждой транзакции в режимах `LOCK IN SHARE MODE` и `FOR UPDATE`. (см. решение{456} задачи 6.2.2.a{456}).

Задание 6.2.2.TSK.E: провести исследование поведения MS SQL Server в контексте аномалий потерянного обновления и неповторяющегося чтения, выполняя первую операцию в каждой транзакции с использованием «табличной подсказки[38]» `UPDLOCK` (см. решение{456} задачи 6.2.2.a{456}).

Задание 6.2.2.TSK.F: повторить решение{483} задачи 6.2.2.b{456} для всех трёх СУБД в остальных поддерживаемых ими уровнях изолированности транзакций, найти такие комбинации уровней изолированности, при которых взаимная блокировка транзакций не возникает.

38 https://msdn.microsoft.com/en-us/library/ms187373%28v=sql.110%29.aspx

6.2.3. ПРИМЕР 45: УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ В ТРИГГЕРАХ, ХРАНИМЫХ ФУНКЦИЯХ И ПРОЦЕДУРАХ

Задача 6.2.3.a{486}: создать на таблице **books** триггер, определяющий уровень изолированности транзакции, в котором сейчас проходит операция вставки, и отменяющий операцию, если уровень изолированности транзакции отличен от **SERIALIZABLE**.

Задача 6.2.3.b{490}: создать хранимую функцию, порождающую исключительную ситуацию в случае запуска в режиме автоподтверждения транзакций.

Задача 6.2.3.c{492}: создать хранимую процедуру, выполняющую подсчёт количества записей в указанной таблице таким образом, чтобы запрос выполнялся максимально быстро (вне зависимости от параллельно выполняемых запросов), даже если в итоге он вернёт не совсем корректные данные.

Ожидаемый результат 6.2.3.a.

Если операция вставки данных в таблицу **books** выполняется в транзакции с уровнем изолированности, отличным от **SERIALIZABLE**, триггер отменяет эту операцию и порождает исключительную ситуацию.

Ожидаемый результат 6.2.3.b.

Если хранимая функция оказывается вызванной в момент, когда для текущей сессии с СУБД включён режим автоподтверждения транзакций, функция должна порождать исключительную ситуацию и прекращать свою работу.

Ожидаемый результат 6.2.3.c.

Хранимая процедура должна выполнять подсчёт записей в указанной таблице в транзакции с уровнем изолированности, обеспечивающим минимальную вероятность ожидания завершения конкурирующих транзакций или отдельных операций в них.

Решение 6.2.3.a{486}.

Для простоты (отсутствия необходимости вручную выполнять вставку) и единообразия (поддержки всеми тремя СУБД) используем **AFTER**-триггеры.

Таким образом, представленные ниже решения будут отличаться только логикой определения уровня изолированности транзакций, т.к. в каждой СУБД соответствующий механизм реализован совершенно особенным, несовместимым с другими СУБД, образом.

Решение для MySQL выглядит следующим образом.

MySQL Решение 6.2.3.a (код триггера)

```
DELIMITER $$

CREATE TRIGGER `books_ins_trans`
AFTER INSERT
ON `books`
 FOR EACH ROW
  BEGIN
   DECLARE isolation_level VARCHAR(50);

   SET isolation_level =
   (
    SELECT `VARIABLE_VALUE`
    FROM   `information_schema`.
           `session_variables`
    WHERE  `VARIABLE_NAME` =
           'tx_isolation'
    );

   IF (isolation_level != 'SERIALIZABLE')
    THEN
     SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Please, switch your
     transaction to SERIALIZABLE isolation level and rerun this
     INSERT again.', MYSQL_ERRNO = 1001;
   END IF;

  END;
$$

DELIMITER ;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода: первая попытка выполнить вставку закончится исключительной ситуацией, порождённой в триггере, а вторая попытка пройдёт успешно.

MySQL Решение 6.2.3.a (код для проверки работоспособности решения)

```
SET SESSION TRANSACTION
ISOLATION LEVEL READ COMMITTED;

INSERT INTO `books`
            (`b_name`,
             `b_year`,
             `b_quantity`)
VALUES      ('И ещё одна книга',
             1985,
             2);

SET SESSION TRANSACTION
ISOLATION LEVEL SERIALIZABLE;

INSERT INTO `books`
            (`b_name`,
             `b_year`,
             `b_quantity`)
VALUES      ('И ещё одна книга',
             1985,
             2);
```

Решение для MS SQL Server выглядит следующим образом.

MS SQL Решение 6.2.3.a (код триггера)

```
CREATE TRIGGER [books_ins_trans]
ON [books]
AFTER INSERT
AS
 DECLARE @isolation_level NVARCHAR(50);

 SET @isolation_level =
 (
  SELECT [transaction_isolation_level]
  FROM [sys].[dm_exec_sessions]
  WHERE [session_id] = @@SPID
 );

 IF (@isolation_level != 4)
  BEGIN
   RAISERROR ('Please, switch your transaction to SERIALIZABLE isolation
               level and rerun this INSERT again.', 16, 1);
   ROLLBACK TRANSACTION;
   RETURN
  END;
GO
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода: первая попытка выполнить вставку закончится исключительной ситуацией, порождённой в триггере, а вторая попытка пройдёт успешно.

MS SQL Решение 6.2.3.a (код для проверки работоспособности решения)

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

INSERT INTO [books]
            ([b_name],
             [b_year],
             [b_quantity])
VALUES      ('И ещё одна книга',
             1985,
             2);

SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;

INSERT INTO [books]
            ([b_name],
             [b_year],
             [b_quantity])
VALUES      ('И ещё одна книга',
             1985,
             2);
```

Решение для Oracle выглядит следующим образом.

Oracle Решение 6.2.3.a (код триггера)

```
CREATE OR REPLACE TRIGGER "books_ins_trans"
AFTER INSERT
ON "books"
 FOR EACH ROW
  DECLARE
   isolation_level NVARCHAR2(150);
   trans_id VARCHAR(100);
  BEGIN
   trans_id := DBMS_TRANSACTION.LOCAL_TRANSACTION_ID(FALSE);
   SELECT CASE BITAND("transaction".flag, POWER(2, 28))
           WHEN 0 THEN 'READ COMMITTED'
           ELSE 'SERIALIZABLE'
          END AS "session_isolation_level"
   INTO   isolation_level
   FROM   v$transaction "transaction"
          JOIN v$session "session"
            ON "transaction".addr = "session".taddr
               AND "session".sid = SYS_CONTEXT('USERENV', 'SID');

   IF (isolation_level != 'SERIALIZABLE')
    THEN
     RAISE_APPLICATION_ERROR(-20001, 'Please, switch your transaction
     to SERIALIZABLE isolation level and rerun this INSERT again.');
    END IF;

  END;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода: первая попытка выполнить вставку закончится исключительной ситуацией, порождённой в триггере, а вторая попытка пройдёт успешно.

Oracle Решение 6.2.3.a (код для проверки работоспособности решения)

```
ALTER SESSION SET
ISOLATION_LEVEL = READ COMMITTED;

INSERT INTO "books"
            ("b_name",
             "b_year",
             "b_quantity")
VALUES      ('И ещё одна книга',
             1985,
             2);

ALTER SESSION SET
ISOLATION_LEVEL = SERIALIZABLE;

INSERT INTO "books"
            ("b_name",
             "b_year",
             "b_quantity")
VALUES      ('И ещё одна книга',
             1985,
             2);
```

На этом решение данной задачи завершено.

Решение 6.2.3.b{486}.

Поскольку в условии задачи не сказано, что именно должна делать функция, мы ограничимся проверкой режима автоподтверждения транзакций и порождения исключительной ситуации в случае, если он включён.

Решение для MySQL выглядит следующим образом.

MySQL Решение 6.2.3.b (код функции)

```
1  DELIMITER $$
2  CREATE FUNCTION NO_AUTOCOMMIT()
3  RETURNS INT DETERMINISTIC
4  BEGIN
5   IF ((SELECT @@autocommit) = 1)
6   THEN
7    SIGNAL SQLSTATE '45001'
8     SET MESSAGE_TEXT = 'Please, turn the autocommit off.',
9     MYSQL_ERRNO = 1001;
10   RETURN -1;
11  END IF;
12
13  -- Тут может быть какой-то полезный код :).
14
15  RETURN 0;
16 END$$
17
18 DELIMITER ;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода: первый вызов функции закончится исключительной ситуацией, а второй пройдёт успешно.

MySQL Решение 6.2.3.b (код для проверки работоспособности решения)

```
1  SET autocommit = 1;
2  SELECT NO_AUTOCOMMIT();
3
4  SET autocommit = 0;
5  SELECT NO_AUTOCOMMIT();
```

Решение для MS SQL Server выглядит следующим образом.

Обратите внимание на следующие важные моменты, характерные для MS SQL Server:

- состояние автоподтверждения транзакций можно определить лишь косвенно (строки 8-23 кода);
- явно породить исключительную ситуацию в коде хранимой функции невозможно, приходится использовать обходное решение (строки 27-31 кода);
- отменить транзакцию в коде хранимой функции невозможно, но в силу порождения исключительной ситуации транзакция будет остановлена.

MS SQL Решение 6.2.3.b (код функции)

```
CREATE FUNCTION NO_AUTOCOMMITT()
RETURNS INT
WITH SCHEMABINDING
AS
BEGIN
 DECLARE @autocommit INT;

 IF (@@TRANCOUNT = 0 AND (@@OPTIONS & 2 = 0))
  BEGIN
   SET @autocommit = 1;
  END
 ELSE IF (@@TRANCOUNT = 0 AND (@@OPTIONS & 2 = 2))
  BEGIN
   SET @autocommit = 0;
  END
 ELSE IF (@@OPTIONS & 2 = 0)
  BEGIN
   SET @autocommit = 1;
  END
 ELSE
  BEGIN
   SET @autocommit = 0;
  END;

 IF (@autocommit = 1)
  BEGIN
   -- В функциях MS SQL Server нельзя использовать RAISEERROR!
   -- RAISERROR ('Please, turn the autocommit off.', 16, 1);

   -- Обходной путь по порождению исключения:
   RETURN CAST('Please, turn the autocommit off.' AS INT);

   -- Отменить транзакцию из функции в MS SQL Server тоже нельзя.
   -- ROLLBACK TRANSACTION;
  END;

 -- Тут может быть какой-то полезный код :).

 RETURN 0;
END;
GO
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода: первый вызов функции закончится исключительной ситуацией, а второй пройдёт успешно.

MS SQL Решение 6.2.3.b (код для проверки работоспособности решения)

```
1  SET IMPLICIT_TRANSACTIONS OFF;
2  SELECT dbo.NO_AUTOCOMMITT();
3
4  SET IMPLICIT_TRANSACTIONS ON;
5  SELECT dbo.NO_AUTOCOMMITT();
```

Решение для Oracle выглядит следующим образом. Да, именно так и выглядит, т.к. в Oracle нет такого явления, как автоподтверждение транзакций — этот эффект может быть реализован некоторыми средствами работы с СУБД, но сама СУБД всегда ждёт явного **COMMITT** или **ROLLBACK**.

Oracle Решение 6.2.3.b (код функции)

```
1  CREATE FUNCTION NO_AUTOCOMMITT
2   RETURN INT
3   DETERMINISTIC
4   IS
5  BEGIN
6   DBMS_OUTPUT.PUT_LINE('Have a nice day :)');
7   RETURN 1;
8  END;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода.

Oracle Решение 6.2.3.b (код для проверки работоспособности решения)

```
1  SET SERVEROUTPUT ON;
2  SELECT NO_AUTOCOMMITT FROM DUAL;
```

На этом решение данной задачи завершено.

Решение 6.2.3.c{486}.

Идея решения данной задачи состоит в том, чтобы использовать такой уровень изолированности транзакций, который меньше всего подвержен влиянию со стороны блокировок, порождённых другими транзакциями.

В MySQL таким уровнем является **READ UNCOMMITTED**, в MS SQL Server — тоже **READ UNCOMMITTED** или **SNAPSHOT** (но **SNAPSHOT** может приводить к дополнительным расходам ресурсов), в Oracle чтение данных всегда происходит в независимом режиме, потому в этой СУБД можно использовать **READ COMMITTED** (тем более, что **READ UNCOMMITTED** в Oracle нет).

Решение для MySQL выглядит следующим образом.

MySQL Решение 6.2.3.c (код процедуры)

```
DELIMITER $$
CREATE PROCEDURE COUNT_ROWS(IN table_name VARCHAR(150),
                            OUT rows_in_table INT)
BEGIN
 SET SESSION TRANSACTION
 ISOLATION LEVEL READ UNCOMMITTED;

 SET @count_query =
 CONCAT('SELECT COUNT(1) INTO @rows_found
         FROM ', table_name);

 PREPARE count_stmt FROM @count_query;
 EXECUTE count_stmt;
 DEALLOCATE PREPARE count_stmt;

 SET rows_in_table := @rows_found;
END;
$$
DELIMITER ;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода.

MySQL Решение 6.2.3.c (код для проверки работоспособности решения)

```
CALL COUNT_ROWS('subscriptions', @rows_in_table);
SELECT @rows_in_table;
```

Решение для MS SQL Server выглядит следующим образом.

MS SQL Решение 6.2.3.c (код процедуры)

```
CREATE PROCEDURE COUNT_ROWS
                 @table_name NVARCHAR(150),
                 @rows_in_table INT OUTPUT
AS
 DECLARE @count_query NVARCHAR(1000) = '';

 SET TRANSACTION ISOLATION
 LEVEL READ UNCOMMITTED;

 SET @count_query =
 CONCAT('SET @rows_f = (SELECT COUNT(1) FROM [', @table_name, '])');
 EXECUTE sp_executesql @count_query,
                       N'@rows_f INT OUT',
                       @rows_in_table OUTPUT;
GO
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода.

MS SQL Решение 6.2.3.c (код для проверки работоспособности решения)

```
DECLARE @res INT;
EXECUTE COUNT_ROWS 'subscriptions', @res OUTPUT;
SELECT @res;
```

Решение для Oracle выглядит следующим образом.

Oracle Решение 6.2.3.c (код процедуры)

```
CREATE PROCEDURE COUNT_ROWS (table_name IN VARCHAR,
                             rows_in_table OUT NUMBER) AS
 count_query VARCHAR(1000) := '';
BEGIN

 EXECUTE IMMEDIATE 'ALTER SESSION SET
 ISOLATION_LEVEL = READ COMMITTED';

 count_query :=
 'SELECT COUNT(1) FROM "' || table_name || '"';
 EXECUTE IMMEDIATE count_query INTO rows_in_table;
END;
/
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода.

Oracle Решение 6.2.3.c (код для проверки работоспособности решения)

```
DECLARE
 res NUMBER;
BEGIN
 COUNT_ROWS('subscriptions', res);
 DBMS_OUTPUT.PUT_LINE('Rows: ' || res);
END;
```

На этом решение данной задачи завершено.

Задание 6.2.3.TSK.A: создать на таблице **subscriptions** триггер, определяющий уровень изолированности транзакции, в котором сейчас проходит операция обновления, и отменяющий операцию, если уровень изолированности транзакции отличен от **REPEATABLE READ**.

Задание 6.2.3.TSK.B: создать хранимую функцию, порождающую исключительную ситуацию в случае, если выполняются оба условия:

- режим автоподтверждения транзакций выключен;
- функция запущена из вложенной транзакции.

Подсказка: эта задача имеет решение только для MS SQL Server.

Задание 6.2.3.TSK.C: создать хранимую процедуру, выполняющую подсчёт количества записей в указанной таблице таким образом, чтобы она возвращала максимально корректные данные, даже если для достижения этого результата придётся пожертвовать производительностью.

РЕШЕНИЕ ТИПИЧНЫХ ЗАДАЧ И ВЫПОЛНЕНИЕ ТИПИЧНЫХ ОПЕРАЦИЙ

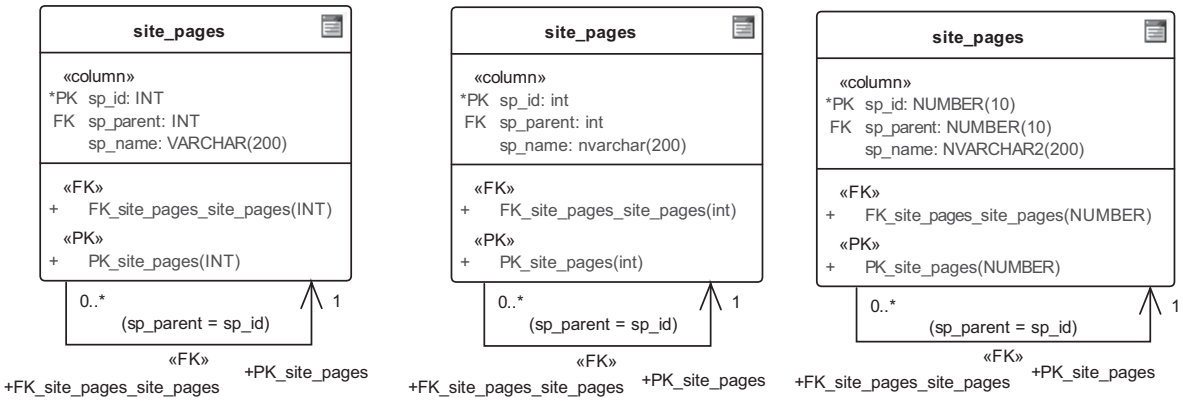
7.1. РАБОТА С ИЕРАРХИЧЕСКИМИ И СВЯЗАННЫМИ СТРУКТУРАМИ

7.1.1. ПРИМЕР 46: ФОРМИРОВАНИЕ И АНАЛИЗ ИЕРАРХИЧЕСКИХ СТРУКТУР

Для решения задач из данного примера нам понадобится новая таблица, которую мы создадим в базе данных «Исследование». Эта таблица будет хранить дерево элементов (допустим, что это будет древовидная структура сайта нашей гипотетической библиотеки).

Существует множество способов хранения древовидных структур в реляционных базах данных[39], но мы используем рекурсивные внешние ключи как одно из самых распространённых и универсальных решений. Соответствующие фрагменты схемы БД для всех трёх СУБД представлены на рисунке 7.а.

[39] http://www.amazon.com/dp/1558609202/



MySQL MS SQL Server Oracle

*Рисунок 7.a — **Таблица `site_pages` во всех трёх СУБД***

Сохраним в таблице **`site_pages`** следующий набор данных, визуально представленный на рисунке 7.b.

sp_id	sp_parent	sp_name
1	NULL	Главная
2	1	Читателям
3	1	Спонсорам
4	1	Рекламодателям
5	2	Новости
6	2	Статистика
7	3	Предложения
8	3	Истории успеха
9	4	Акции
10	1	Контакты
11	3	Документы
12	6	Текущая
13	6	Архивная
14	6	Неофициальная

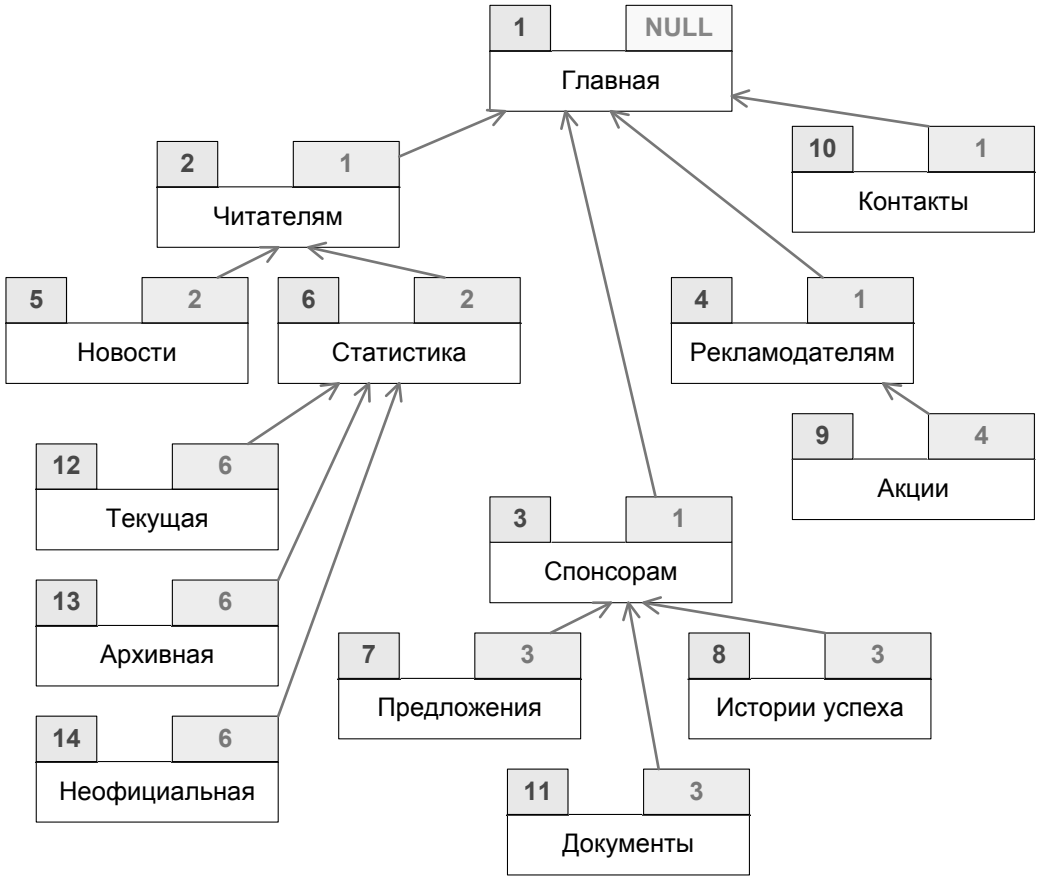


*Рисунок 7.b — **Визуальное представление карты сайта***

Теперь, когда все данные подготовлены, мы можем переходить к задачам.

Задача 7.1.1.a{498}: создать функцию, возвращающую список идентификаторов всех дочерних вершин заданной вершины (например, идентификаторов всех подстраниц страницы «Читателям»).

Задача 7.1.1.b{504}: написать запрос для показа всего поддерева заданной вершины дерева, включая саму родительскую вершину (например, всех подстраниц страницы «Читателям», включая саму эту страницу).

Задача 7.1.1.c{507}: написать функцию, возвращающую список идентификаторов вершин на пути от заданной вершины к корню дерева (например, идентификаторов всех вершин на пути от страницы «Архивная» к странице «Главная»).

Ожидаемый результат 7.1.1.a.

Для вершины с идентификатором 2 (страница «Читателям») функция должна возвратить следующие данные:

shildren_of_2
5,6,12,13,14

Ожидаемый результат 7.1.1.b.

Для вершины с идентификатором 2 (страница «Читателям») запрос должен возвратить следующие данные:

sp_id	sp_name
2	Читателям
5	Новости
6	Статистика
12	Текущая
13	Архивная
14	Неофициальная

Ожидаемый результат 7.1.1.c.

Для вершины с идентификатором 13 (страница «Архивная») запрос должен возвратить следующие данные:

path
13
6
2
1

Также допускается вариант:

path
13,6,2,1

Решение 7.1.1.a[{475}].

Решение данной задачи для MS SQL Server и Oracle можно очень легко построить на основе рекурсивных общих табличных выражений. В MySQL же общие табличные выражения не поддерживаются, равно как нет и иной возможности сделать «рекурсивный `JOIN`», потому для этой СУБД решение будет достаточно нетривиальным.

Сначала приведём готовый код функции и пример её использования.

MySQL Решение 7.1.1.a (код функции)

```
DELIMITER $$
CREATE FUNCTION GET_ALL_CHILDREN(start_node INT)
 RETURNS TEXT
 BEGIN
  DECLARE result TEXT;
  SELECT GROUP_CONCAT(`children_ids` SEPARATOR ',') INTO result
  FROM   (
              SELECT `sp_id`, @parent_values :=
                     (
                            SELECT GROUP_CONCAT(`sp_id` SEPARATOR ',')
                            FROM   `site_pages`
                            WHERE  FIND_IN_SET(`sp_parent`,
                                               @parent_values) > 0
                     ) AS `children_ids`
              FROM   `site_pages`
              JOIN   (SELECT @parent_values := start_node)
                                                    AS `initialisation`
              WHERE  `sp_id` IN (@parent_values)
        ) AS `data`;
  RETURN result;
 END$$
DELIMITER ;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода.

MySQL Решение 7.1.1.a (пример использования функции)

```
-- Проверка работоспособности функции:
SELECT GET_ALL_CHILDREN(2) AS `shildren_of_2`;

-- Использование функции:
SELECT `sp_id`, `sp_name`, GET_ALL_CHILDREN(`sp_id`) AS `children`
FROM `site_pages`;
```

Второй запрос возвратит следующие данные (список всех страниц сайта библиотеки с указанием идентификаторов всех их подстраниц).

sp_id	sp_name	children
1	Главная	2,3,4,10,5,6,7,8,9,11,12,13,14
2	Читателям	5,6,12,13,14
3	Спонсорам	7,8,11
4	Рекламодателям	9
5	Новости	NULL
6	Статистика	12,13,14
7	Предложения	NULL
8	Истории успеха	NULL
9	Акции	NULL
10	Контакты	NULL
11	Документы	NULL
12	Текущая	NULL
13	Архивная	NULL
14	Неофициальная	NULL

Теперь рассмотрим, как работает это решение.

Очевидно, главной частью представленной функции является запрос в строках 6-19. Перепишем его без функции.

MySQL Решение 7.1.1.a (запрос, на котором основана функция)

```
1   SELECT GROUP_CONCAT(`children_ids` SEPARATOR ',') AS `children_ids`
2   FROM   (
3                  SELECT `sp_id`, @parent_values :=
4                         (
5                                SELECT GROUP_CONCAT(`sp_id` SEPARATOR ',')
6                                FROM   `site_pages`
7                                WHERE  FIND_IN_SET(`sp_parent`,
8                                                   @parent_values) > 0
9                         ) AS `children_ids`
10                 FROM   `site_pages`
11                 JOIN   (SELECT @parent_values := {родительская_вершина})
12                                                    AS `initialisation`
13                 WHERE  `sp_id` IN (@parent_values)
14         ) AS `prepared_data`
```

В таком виде этот запрос вернёт данные, представленные в ожидаемом результате задачи (если значение **{родительская_вершина}** равно 2).

Рассмотрим решение по частям.

1) Код в строках 11-12 инициализирует значение переменной **@parent_values** идентификатором вершины, для которой строится список идентификаторов дочерних вершин. В дальнейшем здесь будет храниться список вершин, но в начале работы здесь помещается только одно значение.
2) Код в строках 5-8 формирует набор идентификаторов дочерних элементов вершин, идентификаторы которых перечислены в переменной **@parent_values**. Полученный результат используется как новое значение переменной **@parent_values**.
3) Условие в строке 13 ограничивает выборку только теми вершинами, дочерние элементы которых ищутся на текущем шаге.
4) Алгоритм завершает работу, когда значение переменной **@parent_values** становится равным **NULL**.
5) Благодаря **GROUP_CONCAT** в строке 1 весь результат работу представляется в виде одного списка идентификаторов, в котором они перечислены через запятую. Такое представление позволяет применять функцию **FIND_IN_SET** для дальнейшей работы с полученными результатами.

Если выполнить отдельно строки 3-13 данного запроса, то будет получен следующий результат (сам результат представлен на сером фоне, чтобы не путать его с пояснениями).

Шаг	sp_id	children_ids	Новое значение @parent_values
Начальное состояние, поиск детей вершины 2.			
1	2	5,6	5,6
Теперь надо найти детей вершин 5 и 6 (строка с вершиной 6 «схлопывается» из-за **GROUP_CONCAT**, т.е. в такой выборке мы теряем все значения **sp_id**, кроме самого первого, но в **@parent_values** благодаря тому же **GROUP_CONCAT** попадают дети всех **sp_id**, даже тех, чьи значения мы «потеряли»).			
2	5	12,13,14	12,13,14
Теперь надо найти детей вершин 12, 13 и 14 (строки с вершинами 13 и 14 «схлопываются» из-за **GROUP_CONCAT**, т.е. в такой выборке мы теряем все значения **sp_id**, кроме самого первого, но в **@parent_values** благодаря тому же **GROUP_CONCAT** попадают дети всех **sp_id**, даже тех, чьи значения мы «потеряли»).			
3	12	NULL	NULL
Теперь надо найти детей вершин… NULL, т.е. никаких: конец алгоритма.			

Переходим к MS SQL Server. Здесь решение будет совершенно иным, т.к., во-первых, данная СУБД не поддерживает часть синтаксиса, необходимого для эмуляции решения MySQL, а во-вторых, с использованием возможностей MS SQL Server эта задача решается проще (несмотря на то, что кода будет больше).

MS SQL Решение 7.1.1.a (код функции)

```
CREATE FUNCTION GET_ALL_CHILDREN(@parent INT, @mode VARCHAR(50))
RETURNS @all_children TABLE
    (
     id VARCHAR(max)
    )
AS
BEGIN
 IF (@mode = 'TABLE')
  BEGIN
   WITH [tree] ([sp_id], [sp_parent])
   AS
   (
    SELECT [sp_id],
           [sp_parent]
    FROM   [site_pages]
    WHERE  [sp_id] =  @parent
    UNION ALL
    SELECT [inner].[sp_id],
           [inner].[sp_parent]
    FROM   [site_pages] AS [inner]
           JOIN [tree]
             ON [inner].[sp_parent] = [tree].[sp_id]
   )
  INSERT @all_children
  SELECT CAST([sp_id] AS VARCHAR)
  FROM   [tree]
  WHERE  [sp_id] != @parent
 END
  ELSE
 BEGIN
  WITH [tree] ([sp_id], [sp_parent])
  AS
  (
   SELECT [sp_id],
          [sp_parent]
   FROM   [site_pages]
   WHERE  [sp_id] = 2
   UNION ALL
   SELECT [inner].[sp_id],
          [inner].[sp_parent]
   FROM   [site_pages] AS [inner]
          JOIN [tree]
            ON [inner].[sp_parent] = [tree].[sp_id]
  )
```

MS SQL Решение 7.1.1.a (код функции) (продолжение)

```
45    INSERT @all_children
46    SELECT STUFF((SELECT ',' + CAST([sp_id] AS VARCHAR)
47                  FROM   [tree]
48                      WHERE  [sp_id] != 2
49                FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
50                   1, 1, '');
51   END;
52   RETURN
53  END;
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода. Первый запрос возвратит результат в том виде, в котором это требуется по условию задачи (список идентификаторов, разделённых запятыми), второй запрос возвратит таблицу из одной колонки, где каждый идентификатор будет расположен в отдельной строке.

MS SQL Решение 7.1.1.a (пример использования функции)

```
1   SELECT * FROM GET_ALL_CHILDREN(2, 'STRING');
2   SELECT * FROM GET_ALL_CHILDREN(2, 'TABLE');
```

Итак, рассмотрим решение. Мы создали именно табличную функцию затем, чтобы иметь возможность возвращать данные не только в виде списка идентификаторов, перечисленных через запятую, но и в виде таблицы из одной колонки, строками которой являются идентификаторы.

В случае, когда нам всё же нужен список идентификаторов, перечисленных через запятую, мы по-прежнему возвращаем таблицу из одной колонки, но в ней будет ровно одна строка, содержащая весь наш список.

В основе решения лежит т.н. рекурсивное общее табличное выражение[40]. Рассмотрим его отдельно.

MS SQL Решение 7.1.1.a (рекурсивное общее табличное выражение)

```
1   WITH [tree] ([sp_id], [sp_parent])
2   AS
3   (
4    SELECT [sp_id],
5           [sp_parent]
6    FROM   [site_pages]
7    WHERE  [sp_id] = {родительская_вершина}
8    UNION ALL
9    SELECT [inner].[sp_id],
10          [inner].[sp_parent]
11   FROM   [site_pages] AS [inner]
12          JOIN [tree]
13            ON [inner].[sp_parent] = [tree].[sp_id]
14  )
15  SELECT [sp_id]
16  FROM   [tree]
17  WHERE  [sp_id] != {родительская_вершина}
```

Рекурсивное общее табличное выражение должно содержать две части:

- в первой части (строки 4-7) происходит выборка родительской записи (для которой мы будем искать дочерние);

[40] https://msdn.microsoft.com/en-us/library/ms175972%28v=sql.110%29.aspx

- во второй части (строки 8-13) происходит рекурсивное обращение к результату выполнения общего табличного выражения, что и позволяет нам получить всё поддерево заданной вершины.

Поскольку первая часть (до оператора **UNION**) является обязательной, а по условию задачи идентификатор самой родительской вершины не должен попадать в выборку, мы исключаем попадание соответствующей записи в итоговую выборку условием в строке 17.

Полученное рекурсивно общее табличное выражение мы разместили в строках 10-27 кода функции, обеспечив вставку его результатов (строка 24) в результирующую таблицу, которую возвращает функция.

В строках 31-50 кода функции находится то же самое рекурсивное общее табличное выражение, но при выборке результатов его выполнения мы применяем приём, подробно описанный в решении{74} задачи 2.2.2.a{73} для эмуляции функции **GROUP_CONCAT** MySQL.

Переходим к Oracle. Здесь решение будет практически идентичным решению для MS SQL Server. Единственное отличие — в способе возврата таблицы из функции (мы рассматривали этот вопрос ранее, см. решение{372} задачи 5.1.1.b{369}). И здесь мы также вынуждены эмулировать поведение функции MySQL **GROUP_CONCAT** через использование функции Oracle **LISTAGG** (см. решение{74} задачи 2.2.2.a{73}).

Oracle Решение 7.1.1.a (код функции)

```
1  CREATE TYPE "tree_node" AS OBJECT("id" VARCHAR(32767));
2  /
3
4  CREATE TYPE "nodes_collection" AS TABLE OF "tree_node";
5  /
6
7  CREATE OR REPLACE FUNCTION GET_ALL_CHILDREN(parent_id NUMBER,
8                                              function_mode VARCHAR)
9  RETURN "nodes_collection"
10 AS
11  result_collection "nodes_collection";
12 BEGIN
13  IF (function_mode = 'TABLE')
14   THEN
15    WITH "tree" ("sp_id", "sp_parent")
16    AS
17    (
18     SELECT "sp_id",
19            "sp_parent"
20     FROM   "site_pages"
21     WHERE  "sp_id" = parent_id
22     UNION ALL
23     SELECT "inner"."sp_id",
24            "inner"."sp_parent"
25     FROM   "site_pages" "inner"
26            JOIN "tree"
27              ON "inner"."sp_parent" = "tree"."sp_id"
28    )
29    SELECT "tree_node"(TO_CHAR("sp_id"))
30    BULK COLLECT INTO result_collection
31    FROM   "tree"
32    WHERE  "sp_id" != parent_id;
```

Oracle Решение 7.1.1.a (код функции) (продолжение)

```
33    ELSE
34     WITH "tree" ("sp_id", "sp_parent")
35     AS
36     (
37      SELECT "sp_id",
38             "sp_parent"
39      FROM   "site_pages"
40      WHERE  "sp_id" = parent_id
41      UNION ALL
42      SELECT "inner"."sp_id",
43             "inner"."sp_parent"
44      FROM   "site_pages" "inner"
45             JOIN "tree"
46               ON "inner"."sp_parent" = "tree"."sp_id"
47     )
48     SELECT "tree_node"(LISTAGG(TO_CHAR("sp_id"), ',' )
49                        WITHIN GROUP (ORDER BY "sp_id"))
50     BULK COLLECT INTO result_collection
51     FROM   "tree"
52     WHERE  "sp_id" != parent_id;
53    END IF;
54   RETURN result_collection;
55  END;
56  /
```

Проверить работоспособность и корректность представленного решения можно выполнением следующего кода. Первый запрос возвратит результат в том виде, в котором это требуется по условию задачи (список идентификаторов, разделённых запятыми), второй запрос возвратит таблицу из одной колонки, где каждый идентификатор будет расположен в отдельной строке.

Oracle Решение 7.1.1.a (пример использования функции)

```
1   SELECT *
2   FROM TABLE(CAST(GET_ALL_CHILDREN(2, 'STRING') AS "nodes_collection"));
3
4   SELECT *
5   FROM TABLE(CAST(GET_ALL_CHILDREN(2, 'TABLE') AS "nodes_collection"));
```

На этом решение данной задачи завершено.

Решение 7.1.1.b[497].

Легко заметить, что решение этой задачи основано на рассуждениях, представленных в решении[498] задачи 7.1.1.a[497]. Если допустить, что соответствующие функции у нас уже есть, код для всех трёх СУБД будет выглядеть так.

MySQL Решение 7.1.1.b (использованием функции GET_ALL_CHILDREN из решения 7.1.1.a)

```
1   SELECT `sp_id`,
2          `sp_name`
3   FROM   `site_pages`
4   WHERE  `sp_id` = {родительская_вершина}
5         OR FIND_IN_SET(`sp_id`, GET_ALL_CHILDREN({родительская_вершина}))
```

MS SQL Решение 7.1.1.b (использованием функции GET_ALL_CHILDREN из решения 7.1.1.a)

```
SELECT [sp_id],
       [sp_name]
FROM   [site_pages]
WHERE  [sp_id] = {родительская_вершина}
       OR [sp_id] IN
                  (SELECT [id]
                   FROM GET_ALL_CHILDREN({родительская_вершина}, 'TABLE'))
```

Oracle Решение 7.1.1.b (использованием функции GET_ALL_CHILDREN из решения 7.1.1.a)

```
SELECT "sp_id",
       "sp_name"
FROM   "site_pages"
WHERE  "sp_id" = {родительская_вершина}
       OR "sp_id" IN
                  (SELECT "id"
                   FROM TABLE(CAST(
                              GET_ALL_CHILDREN({родительская_вершина},
                                               'TABLE')
                              AS "nodes_collection")))
```

Если же предположить, что функции **GET_ALL_CHILDREN** у нас нет, решение можно построить следующим образом.

В MySQL мы берём запрос, вокруг которого построена функция в решении{498} задачи 7.1.1.a{497}, и используем его напрямую как источник списка идентификаторов дочерних страниц.

MySQL Решение 7.1.1.b

```
SELECT `sp_id`,
       `sp_name`
FROM   `site_pages`
WHERE  `sp_id` = {родительская_вершина}
       OR FIND_IN_SET
       (`sp_id`,
        (SELECT GROUP_CONCAT(`children_ids`)
         FROM   (SELECT `sp_id`,
                 @parent_values :=
                  (
                   SELECT GROUP_CONCAT(`sp_id` SEPARATOR ',')
                   FROM   `site_pages`
                   WHERE  FIND_IN_SET(`sp_parent`, @parent_values) > 0
                  ) AS `children_ids`
                 FROM   `site_pages`
                 JOIN   (SELECT @parent_values := {родительская_вершина})
                                                    AS `initialisation`
                 WHERE  `sp_id` IN (@parent_values)
         ) AS `prepared_data`)
       )
```

В MS SQL Server мы берём рекурсивные общие табличные выражения, вокруг которых построены функции в решении{498} задачи 7.1.1.a{497}, и, добавив в выборку имя страницы (поле **sp_name**), используем полученный результат как источник требуемых данных. Здесь даже не придётся исключать из выборки саму родительскую вершину, т.к. по условию данной задачи она должна попадать в конечный набор данных.

Обратите внимание на тот факт, что MS SQL Server допускает создание рекурсивных общих табличных выражений без явного указания списка колонок, в то время как в Oracle этот список обязателен.

MS SQL Решение 7.1.1.b

```
WITH [tree]
AS
(
 SELECT [sp_id],
        [sp_parent],
           [sp_name]
 FROM   [site_pages]
 WHERE  [sp_id] = {родительская_вершина}
 UNION ALL
 SELECT [inner].[sp_id],
        [inner].[sp_parent],
           [inner].[sp_name]
 FROM   [site_pages] AS [inner]
        JOIN [tree]
          ON [inner].[sp_parent] = [tree].[sp_id]
)
SELECT [sp_id],
       [sp_name]
FROM   [tree]
```

Oracle Решение 7.1.1.b

```
WITH "tree" ("sp_id", "sp_parent", "sp_name")
AS
(
 SELECT "sp_id",
        "sp_parent",
        "sp_name"
 FROM   "site_pages"
 WHERE  "sp_id" = {родительская_вершина}
 UNION ALL
 SELECT "inner"."sp_id",
        "inner"."sp_parent",
        "inner"."sp_name"
 FROM   "site_pages" "inner"
        JOIN "tree"
          ON "inner"."sp_parent" = "tree"."sp_id"
)
SELECT "sp_id",
       "sp_name"
FROM   "tree"
```

Если же отойти от традиции, в рамках которой мы рассматриваем во всех трёх СУБД максимально похоже решения, то в Oracle данную задачу можно решить с помощью конструкции **`CONNECT BY`**.

Для наглядности выберем всю информацию о страницах, и даже добавим указание уровня, на котором каждая из страниц находится относительно исходной родительской, список подстраниц которой мы ищем.

Oracle Решение 7.1.1.b (альтернативный вариант)

```
SELECT "sp_id",
       "sp_parent",
       "sp_name",
       LEVEL
FROM   "site_pages"
START WITH "sp_id" = {родительская_вершина}
CONNECT BY PRIOR "sp_id" = "sp_parent";
```

Для страницы с идентификатором 2, этот запрос вернёт следующие данные.

sp_id	sp_parent	sp_name	LEVEL
2	1	Читателям	1
5	2	Новости	2
6	2	Статистика	2
12	6	Текущая	3
13	6	Архивная	3
14	6	Неофициальная	3

Но и это ещё не всё: функция **SYS_CONNECT_BY_PATH** позволяет для каждой вершины формировать полный путь, связывающий её с родительской (для которой строится список дочерних элементов).

Oracle Решение 7.1.1.b (альтернативный вариант)

```
SELECT LPAD(' ', 2 * LEVEL, ' ') || "sp_name" "debug",
       "sp_id",
       "sp_parent",
       "sp_name",
       SYS_CONNECT_BY_PATH("sp_name", '/') "path_with_names",
       SYS_CONNECT_BY_PATH("sp_id", ',')   "path_with_ids"
FROM   "site_pages"
START WITH "sp_id" = {родительская_вершина}
CONNECT BY PRIOR "sp_id" = "sp_parent"
ORDER SIBLINGS BY "sp_name"
```

Для страницы с идентификатором 2, этот запрос вернёт следующие данные.

debug	sp_id	sp_parent	sp_name	path_with_names	path_with_ids
Читателям	2	1	Читателям	/Читателям	,2
Новости	5	2	Новости	/Читателям/Новости	,2,5
Статистика	6	2	Статистика	/Читателям/Статистика	,2,6
Архивная	13	6	Архивная	/Читателям/Статистика/Архивная	,2,6,13
Неофициальная	14	6	Неофициальная	/Читателям/Статистика/Неофициальная	,2,6,14
Текущая	12	6	Текущая	/Читателям/Статистика/Текущая	,2,6,12

На этом решение данной задачи завершено.

Решение 7.1.1.c[497].

В MySQL нет рекурсивных запросов, нет возможности вернуть из функции таблицу, нет общих табличных выражений — остаётся использовать алгоритм с циклом, в котором на каждом шаге мы поднимаемся на один уровень иерархии вверх, пока не достигнем корневой вершины (ссылка на родителя равна **NULL**).

Здесь мы последовательно подменяем (строки 12-15) значение идентификатора текущего узла и накапливаем (строки 16-19) все полученные значения в строковой переменной, которая и является результатом работы функции.

Обратите внимание, что MySQL допускает указание **RETURN** прямо в объявлении обработчика ситуации «запрос вернул пустой результат» (строка 7), и таким образом нет необходимости явно делать **RETURN** далее в коде функции.

MySQL Решение 7.1.1.c (код функции)

```
1  DELIMITER $$
2  CREATE FUNCTION GET_PATH_TO_ROOT(start_node INT) RETURNS TEXT
3  NOT DETERMINISTIC
4   BEGIN
5    DECLARE path_to_root TEXT;
6    DECLARE current_node INT;
7    DECLARE EXIT HANDLER FOR NOT FOUND RETURN path_to_root;
8
9    SET current_node = start_node;
10   SET path_to_root = start_node;
11   LOOP
12    SELECT  `sp_parent`
13    INTO    current_node
14    FROM    `site_pages`
15    WHERE   `sp_id` = current_node;
16    IF (current_node IS NOT NULL)
17     THEN
18      SET path_to_root = CONCAT(path_to_root, ',', current_node);
19    END IF;
20   END LOOP;
21 END$$
22 DELIMITER ;
```

Использовать полученную функцию можно так.

MySQL Решение 7.1.1.c (пример использования функции)

```
1  SELECT GET_PATH_TO_ROOT(14)
```

На этом решение для MySQL завершено.

Переходим к MS SQL Server. Поскольку в данной СУБД есть возможность возвратить из функции результат в виде таблицы, мы реализуем оба варианта поведения — и возврат таблицы, и возврат строки.

В строках 9-17 мы реализуем такой же алгоритм, как и в решении для MySQL, но все рассмотренные значения идентификаторов вершин не накапливаем в строку, а помещаем в результирующую таблицу.

В строках 19-28 мы проверяем необходимость вернуть результат в виде набора идентификаторов, разделённых запятыми и, если такая необходимость есть, собираем все данные из результирующей таблицы в строку (строки 21-24 кода), очищаем результирующую таблицу (строка 25 кода) и помещаем в неё полученную строку (строки 26-27 кода).

MS SQL Решение 7.1.1.c (код функции)

```
1  CREATE FUNCTION GET_PATH_TO_ROOT(@current_node INT, @mode VARCHAR(50))
2  RETURNS @path TABLE
3   (
4    id VARCHAR(max)
5   )
6  AS
```

MS SQL Решение 7.1.1.c (код функции) (продолжение)

```
7   BEGIN
8    DECLARE @all_as_string VARCHAR(max) = '';
9    WHILE (@current_node IS NOT NULL)
10   BEGIN
11    INSERT INTO @path
12    SELECT CAST(@current_node AS VARCHAR);
13
14    SET @current_node = (SELECT [sp_parent]
15                         FROM   [site_pages]
16                         WHERE  [sp_id] = @current_node);
17   END;
18
19   IF (@mode = 'STRING')
20    BEGIN
21     SET @all_as_string  = (SELECT STUFF((SELECT ',' + CAST([id] AS VARCHAR)
22                   FROM   @path
23                   FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
24                   1, 1, ''));
25     DELETE FROM @path;
26     INSERT INTO @path
27     SELECT @all_as_string;
28    END;
29
30   RETURN
31  END;
```

Проверить работоспособность и корректность полученного решения можно следующими запросами, первый из которых вернёт колонку идентификаторов, а второй — таблицу из одной ячейки, в которой будет строка с идентификаторами, перечисленными через запятую.

MS SQL Решение 7.1.1.c (пример использования функции)

```
1   SELECT * FROM GET_PATH_TO_ROOT(14, 'TABLE');
2   SELECT * FROM GET_PATH_TO_ROOT(14, 'STRING');
```

Поскольку MS SQL Server поддерживает рекурсивные общие табличные выражения, эту задачу можно решить также с их помощью. Для краткости мы не будем создавать отдельную функцию, но приведём запрос, который даже сам по себе возвращает именно такой результат, какой требуется по условию задачи.

JOIN в строке 11 как раз и обеспечивает необходимое нам рекурсивное поведение.

MS SQL Решение 7.1.1.c (альтернативный вариант)

```
1   WITH [path_to_root] AS
2   (
3    SELECT [sp_id],
4           [sp_parent]
5    FROM   [site_pages]
6    WHERE  [sp_id] = {исходная_вершина}
7    UNION ALL
8    SELECT [inner].[sp_id],
9           [inner].[sp_parent]
10   FROM   [site_pages] AS [inner]
11     JOIN [path_to_root] ON [path_to_root].[sp_parent] = [inner].[sp_id]
12  )
13  SELECT [sp_id] FROM [path_to_root]
```

На этом решение для MS SQL Server завершено.

Переходим к Oracle и реализуем решение по аналогии с MS SQL Server.

Oracle Решение 7.1.1.c (код функции)

```
1  CREATE TYPE "ptr_tree_node" AS OBJECT("id" VARCHAR(32767));
2  /
3
4  CREATE TYPE "ptr_nodes_collection" AS TABLE OF "ptr_tree_node";
5  /
6
7  CREATE OR REPLACE FUNCTION GET_PATH_TO_ROOT(start_id NUMBER,
8                                              function_mode VARCHAR)
9  RETURN "ptr_nodes_collection"
10 AS
11  result_collection  "ptr_nodes_collection" := "ptr_nodes_collection"();
12  all_as_string VARCHAR(32767);
13  temp_int_value NUMBER(10);
14 BEGIN
15  temp_int_value := start_id;
16
17  WHILE (temp_int_value IS NOT NULL)
18   LOOP
19    IF (function_mode = 'TABLE')
20     THEN
21      result_collection.extend;
22      result_collection(result_collection.last) :=
23                                "ptr_tree_node"(TO_CHAR(temp_int_value));
24     ELSE
25      all_as_string := all_as_string || ',' || temp_int_value;
26    END IF;
27    SELECT "sp_parent" INTO temp_int_value
28    FROM "site_pages"
29    WHERE "sp_id" = temp_int_value;
30   END LOOP;
31
32   all_as_string := SUBSTR(all_as_string, 2);
33
34   IF (function_mode != 'TABLE')
35   THEN
36    SELECT "ptr_tree_node"(TO_CHAR(all_as_string))
37    BULK COLLECT INTO result_collection
38    FROM DUAL;
39   END IF;
40
41  RETURN result_collection;
42 END;
```

Поскольку в Oracle нельзя присваивать новое значение входному параметру функции, мы используем временную переменную (строки 13 и 15) для хранения значений идентификаторов вершин и использования в цикле.

В отличие от MS SQL Server, где внутри функции у нас есть полноценная таблица для хранения данных, в Oracle мы используем коллекцию, потому для упрощения кода мы сразу в теле цикла проверяем, придётся ли нам возвратить колонку идентификаторов (тогда мы добавляем их в коллекцию) или строку с их перечислением (тогда мы накапливаем их в строковой переменной).

В строках 34-39 мы снова проверяем режим работы функции и помещаем строку с перечислением идентификаторов (из которой в строке 32 убираем первую лишнюю запятую) в коллекцию (которая сейчас пуста, если функция вызвана в режиме возврата перечисления идентификаторов).

Проверить работоспособность и корректность полученного решения можно следующими запросами, первый из которых вернёт колонку идентификаторов, а второй — таблицу из одной ячейки, в которой будет строка с идентификаторами, перечисленными через запятую.

Oracle Решение 7.1.1.c (пример использования функции)

```
1   SELECT *
2   FROM TABLE(CAST(GET_PATH_TO_ROOT(14, 'TABLE') AS "ptr_nodes_collection"));
3
4   SELECT *
5   FROM TABLE(CAST(GET_PATH_TO_ROOT(14, 'STRING') AS "ptr_nodes_collection"));
```

Поскольку Oracle (как и MS SQL Server) поддерживает рекурсивные общие табличные выражения, эту задачу можно решить также с их помощью. Для краткости мы не будем создавать отдельную функцию, но приведём запрос, который даже сам по себе возвращает именно такой результат, какой требуется по условию задачи.

Oracle Решение 7.1.1.c (альтернативный вариант)

```
1   WITH "path_to_root" ("sp_id", "sp_parent") AS
2   (
3    SELECT "sp_id",
4           "sp_parent"
5    FROM   "site_pages"
6    WHERE  "sp_id" = {исходная_вершина}
7    UNION ALL
8    SELECT "inner"."sp_id",
9           "inner"."sp_parent"
10   FROM   "site_pages" "inner"
11     JOIN "path_to_root" ON "path_to_root"."sp_parent" = "inner"."sp_id"
12  )
13  SELECT "sp_id" FROM "path_to_root"
```

Как и в решении задачи 7.1.1.b, мы рассмотрим ещё один вариант, доступный только в Oracle. На основе выражения **CONNECT BY** и функции **SYS_CONNECT_BY_PATH** очень легко получается решение с представлением результата в виде строки идентификаторов.

Oracle Решение 7.1.1.c (альтернативный вариант)

```
1   SELECT SUBSTR(SYS_CONNECT_BY_PATH("sp_id", ','), 2) "path_with_ids"
2   FROM   "site_pages"
3   WHERE  "sp_id" = {исходная_вершина}
4   START WITH "sp_id" = (SELECT "sp_id"
5                         FROM   "site_pages"
6                         WHERE  "sp_parent" IS NULL)
7   CONNECT BY PRIOR "sp_id" = "sp_parent"
8   ORDER SIBLINGS BY "sp_name"
```

Единственная особенность заключается в последовательности размещения идентификаторов: во всех ранее рассмотренных вариантах корневая вершина находилась справа (например, для вершины 14 последовательность была 14,6,2,1), а здесь корневая вершина будет находиться слева (т.е. получится 1,2,6,14).

На этом решение данной задачи завершено.

Задание 7.1.1.TSK.A: создать функцию, возвращающую список идентификаторов всех дочерних вершин заданной вершины (например, идентификаторов всех подстраниц страницы «Читателям») на глубину, не более заданной.

Задание 7.1.1.TSK.B: написать запрос для показа всего поддерева заданной вершины дерева, включая саму родительскую вершину (например, всех подстраниц страницы «Читателям», включая саму эту страницу), в котором с каждого уровня иерархии в выборку попадает не более одной вершины.

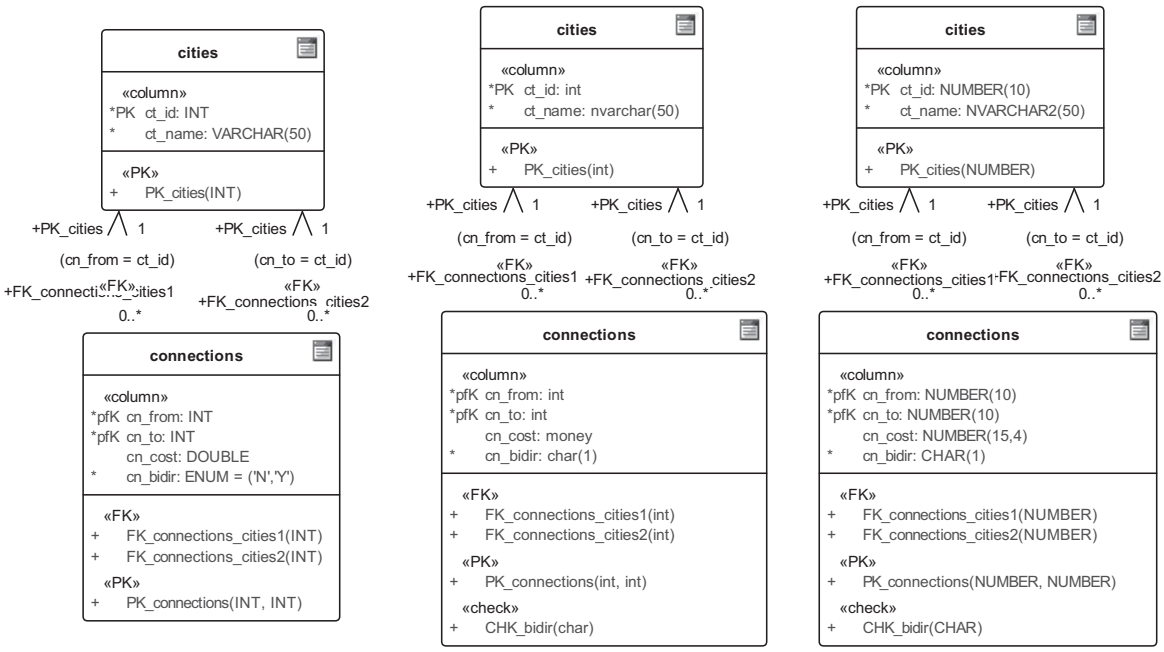
Задание 7.1.1.TSK.C: написать функцию, возвращающую список идентификаторов вершин на пути от корня дерева к заданной вершине (например, идентификаторов всех вершин на пути от страницы «Главная» к странице «Архивная»).

7.1.2. ПРИМЕР 47: ФОРМИРОВАНИЕ И АНАЛИЗ СВЯЗАННЫХ СТРУКТУР

Для решения задач из данного примера нам понадобится новая таблица, которую мы создадим в базе данных «Исследование». Эта таблица будет хранить граф (допустим, что это будет информация о стоимости доставки книг из одного города в другой для организации сотрудничества с другими библиотеками).

Существует множество способов хранения иерархических структур в реляционных базах данных[41], но мы используем таблицу связей как одно из самых распространённых и универсальных решений. Соответствующие фрагменты схемы БД для всех трёх СУБД представлены на рисунке 7.c.

Поле **`cn_bidir`** в таблице connections является признаком того, что стоимость доставки одинакова как при отправке книги из города **`cn_from`** в город **`cn_to`**, так и из **`cn_to`** в **`cn_from`**.



MySQL MS SQL Server Oracle

*Рисунок 7.c — **Таблицы `cities` и `connections` во всех трёх СУБД***

41 http://www.amazon.com/dp/1558609202/

Сохраним в таблице `cities` следующий набор данных.

ct_id	ct_name
1	Лондон
2	Париж
3	Мадрид
4	Токио
5	Москва
6	Киев
7	Минск
8	Рига
9	Варшава
10	Берлин

Сохраним в таблице `connections` следующий набор данных.

cn_from	cn_to	cn_cost	cn_bidir
1	5	10	Y
1	7	20	N
7	1	25	N
7	2	15	Y
2	6	50	N
6	8	40	Y
8	4	30	N
4	8	35	N
8	9	15	Y
9	1	20	N
7	3	5	N
3	6	5	N

Теперь, когда все данные подготовлены, мы можем переходить к задачам.

Задача 7.1.2.a[514]: доработать модель базы данных таким образом, чтобы для прямых маршрутов (без пересадок), цена перемещения по которым «туда» и «обратно» одинакова, в запросе на поиск такого маршрута можно было произвольно менять местами точки отправки и назначения.

Задача 7.1.2.b[516]: написать хранимую процедуру, проверяющую существование маршрута (с возможными пересадками) между двумя указанными городами, и вычисляющую стоимость отправки книги по такому маршруту (при его наличии).

Ожидаемый результат 7.1.2.a.

Запрос вида

```
SELECT *
FROM   {источник_данных}
WHERE  {откуда} = 5
   AND {куда} = 1
```

должен возвращать такой результат (обратите внимание: в представленных выше данные нет маршрута из города 5 в город 1, есть только из 1 в 5, но этот маршрут — двунаправленный):

cn_from	cn_to	cn_cost	cn_bidir
5	1	10	Y

Ожидаемый результат 7.1.2.b.

Например, для городов с идентификаторами 1 и 6 хранимая процедура должна возвратить такие данные.

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	6	31	N	3	1,7,3,6
1	6	85	N	3	1,7,2,6

Решение 7.1.2.a[{513}].

Для решения этой задачи нам необходимо обеспечить такое поведение СУБД, чтобы для двунаправленных маршрутов при указании условия поиска как `cn_from=A ADN cn_to=B` в выборку попадали также и маршруты, для которых выполняется условие `cn_from=B AND cn_to=A`. Проще всего такого эффекта можно добиться с использованием представлений.

В строке 8 представленного ниже кода для MySQL можно было не писать ключевое слово `DISTINCT` (т.к. по умолчанию (без ключевого слова `ALL`) оператор `UNION` работает в `DISTINCT`-режиме), но оно там есть для наглядности, чтобы подчеркнуть необходимость устранения дублирующихся записей.

MySQL Решение 7.1.2.a

```
1  CREATE OR REPLACE VIEW `connections_bidir`
2  AS
3    SELECT `cn_from`,
4           `cn_to`,
5           `cn_cost`,
6           `cn_bidir`
7    FROM   `connections`
8    UNION DISTINCT
9    SELECT `cn_to`,
10          `cn_from`,
11          `cn_cost`,
12          `cn_bidir`
13   FROM   `connections`
14   WHERE  `cn_bidir` = 'Y'
```

На примере решения для MySQL рассмотрим подробно, как работает такое представление. Если выбрать из него все данные, получится следующая картина. Серым фоном отмечены строки, появившиеся в результате выполнения `UNION`-части запроса: для всех двунаправленных маршрутов добавились записи с инвертированными пунктами отправки и назначения.

cn_from	cn_to	cn_cost	cn_bidir
1	**5**	**10**	**Y**
1	7	20	N
2	6	50	N
3	6	6	N
4	8	35	N
6	8	40	Y
7	1	25	N
7	2	15	Y
7	3	5	N
8	4	30	N
8	9	15	Y
9	1	20	N
5	**1**	**10**	**Y**
8	6	40	Y
2	7	15	Y
9	8	15	Y

Теперь выполнение запроса

MySQL Решение 7.1.2.a (проверка работоспособности)

```
1  SELECT *
2  FROM   `connections_bidir`
3  WHERE  `cn_from` = 5
4         AND `cn_to` = 1
```

вернёт корректный ожидаемый результат.

cn_from	cn_to	cn_cost	cn_bidir
5	1	10	Y

В MS SQL Server и Oracle синтаксис оператора **UNOIN** не допускает явного указания слова **DISTINCT** (строка 8 двух показанных ниже запросов), но это — не проблема, т.к. по умолчанию (без ключевого слова **ALL**) оператор **UNION** работает в **DISTINCT**-режиме.

MS SQL Решение 7.1.2.a

```
1  CREATE VIEW [connections_bidir]
2  AS
3    SELECT [cn_from],
4           [cn_to],
5           [cn_cost],
6           [cn_bidir]
7    FROM   [connections]
8    UNION
9    SELECT [cn_to],
10          [cn_from],
11          [cn_cost],
12          [cn_bidir]
13   FROM   [connections]
14   WHERE  [cn_bidir] = 'Y'
```

Oracle Решение 7.1.2.a

```
1   CREATE OR REPLACE VIEW "connections_bidir"
2   AS
3     SELECT "cn_from",
4            "cn_to",
5            "cn_cost",
6            "cn_bidir"
7     FROM   "connections"
8     UNION
9     SELECT "cn_to",
10           "cn_from",
11           "cn_cost",
12           "cn_bidir"
13    FROM   "connections"
14    WHERE  "cn_bidir" = 'Y'
```

На этом решение данной задачи завершено.

Решение 7.1.2.b{513}.

Решение данной задачи стоит начать с подчёркивания того факта, что реляционные СУБД не оптимизированы для хранения графовых структур и выполнения над ними подобных операций. Потому представленные ниже решения могут показаться излишне громоздкими (с использованием классических языков программирования можно создать гораздо более компактный и оптимальный код).

Традиционно начнём с MySQL и рассмотрим два варианта решения, первый из которых максимально использует возможности СУБД, а второй эмулирует классическое алгоритмическое решение.

В первом варианте реализуется следующий подход[42]:

- в оперативной памяти (`ENGINE = HEAP`) создаётся временная таблица для хранения найденных путей (строки 10-18);
- в созданную таблицу переносятся все данные из таблицы `connections` с учётом двунаправленности некоторых связей (строки 21-40; аналогичный подзапрос, учитывающий двунаправленные связи, используется в строках 68-80 — фактически, он представляет собой ничто иное, как тело представления из решения{514} задачи 7.1.2.a{513});
- выполняется цикл поиска производных маршрутов (строки 45-92), в котором:
  - условием выхода является отсутствие новых маршрутов (функция MySQL `ROW_COUNT` возвращает количество записей, затронутых последней операцией модификации данных);
  - идея поиска новых маршрутов строится на том, чтобы к уже найденным маршрутам добавлять следующие шаги (конечная точка найденного маршрута совпадает с отправной точкой связи между городами, что проверяется в условии объединения в строке 81; условие в строках 82-83 исключает порождение циклических маршрутов; условие в строках 88-89 исключает бесконечное повторное дублирующихся маршрутов между двумя любыми городами).
- после того, как все возможные производные маршруты построены, в качестве результата работы хранимой процедуры возвращаются только те маршруты, точки отправки и назначения которых совпадают с переданными в хранимую процедуру параметрами (строки 94-98).

42 https://www.artfulsoftware.com/mysqlbook/sampler/mysqled1ch20.html

У этого решения есть два недостатка:

- предварительное построение всех возможных маршрутов избыточно и приводит к бессмысленным затратам памяти и потере производительности;
- альтернативные маршруты могут быть обнаружены только в том случае, если у них одинаковая длина (т.е. они найдены на одном шаге цикла).

MySQL Решение 7.1.2.b (код процедуры, первый вариант)

```
DELIMITER $$
CREATE PROCEDURE FIND_PATH(IN start_node INT,
                           IN finish_node INT)
BEGIN
 DECLARE rows_inserted INT DEFAULT 0;

 -- Пересоздание временной таблицы для хранения маршрутов
 -- (именно DROP/CREATE на случай, если такая таблица была):
 DROP TABLE IF EXISTS `connections_temp`;
 CREATE TABLE IF NOT EXISTS `connections_temp`
 (
  `cn_from` INT,
  `cn_to` INT,
  `cn_cost` DOUBLE,
  `cn_bidir` CHAR(1),
  `cn_steps` SMALLINT,
  `cn_route` VARCHAR(1000)
 ) ENGINE = MEMORY;
 -- Первичное наполнение временной таблицы
 -- существующими маршрутами:
 INSERT INTO `connections_temp`
 SELECT `cn_from`,
        `cn_to`,
        `cn_cost`,
        `cn_bidir`,
        1,
        CONCAT(`cn_from`, ',', `cn_to`)
 FROM   (SELECT `cn_from`,
                `cn_to`,
                `cn_cost`,
                `cn_bidir`
         FROM   `connections`
         UNION DISTINCT
         SELECT `cn_to`,
                `cn_from`,
                `cn_cost`,
                `cn_bidir`
         FROM   `connections`
         WHERE  `cn_bidir` = 'Y'
        ) AS `connections_bidir`;

  -- Наполнение временной таблицы производными
  -- маршрутами:
  SET rows_inserted = ROW_COUNT();
  WHILE (rows_inserted > 0)
  DO
    INSERT INTO `connections_temp`
```

MySQL Решение 7.1.2.b (код процедуры, первый вариант) (продолжение)

```
48      SELECT `connections_next`.`cn_from`,
49             `connections_next`.`cn_to`,
50             `connections_next`.`cn_cost`,
51             `connections_next`.`cn_bidir`,
52             `connections_next`.`cn_steps`,
53             `connections_next`.`cn_route`
54      FROM (SELECT `connections_temp`.`cn_from` AS `cn_from`,
55                   `connections`.`cn_to` AS `cn_to`,
56                  (`connections_temp`.`cn_cost` +
57                   `connections`.`cn_cost`) AS `cn_cost`,
58                  CASE
59                   WHEN (`connections_temp`.`cn_bidir` = 'Y')
60                    AND (`connections`.`cn_bidir` = 'Y')
61                   THEN 'Y'
62                   ELSE 'N'
63                  END AS  `cn_bidir`,
64                  (`connections_temp`.`cn_steps` + 1) AS `cn_steps`,
65                  CONCAT(`connections_temp`.`cn_route`, ',',
66                         `connections`.`cn_to`) AS `cn_route`
67            FROM `connections_temp`
68            JOIN (SELECT `cn_from`,
69                         `cn_to`,
70                         `cn_cost`,
71                         `cn_bidir`
72                  FROM   `connections`
73                         UNION DISTINCT
74                  SELECT `cn_to`,
75                         `cn_from`,
76                         `cn_cost`,
77                         `cn_bidir`
78                  FROM   `connections`
79                  WHERE  `cn_bidir` = 'Y'
80                 ) AS `connections`
81              ON `connections_temp`.`cn_to` = `connections`.`cn_from`
82                 AND FIND_IN_SET(`connections`.`cn_to`,
83                                 `connections_temp`.`cn_route`) = 0
84          ) AS `connections_next`
85      LEFT JOIN `connections_temp`
86             ON `connections_next`.`cn_from` = `connections_temp`.`cn_from`
87                AND `connections_next`.`cn_to` = `connections_temp`.`cn_to`
88      WHERE `connections_temp`.`cn_from` IS NULL
89        AND `connections_temp`.`cn_to` IS NULL;
90
91      SET rows_inserted = ROW_COUNT();
92    END WHILE;
93    -- Извлечение маршрутов, соответствующих условию поиска:
94    SELECT *
95    FROM `connections_temp`
96    WHERE `cn_from` = start_node
97      AND `cn_to` = finish_node
98    ORDER BY `cn_cost` ASC;
99    DROP TABLE IF EXISTS `connections_temp`;
100 END;
101 $$
102 DELIMITER ;
```

Альтернативное решение, построенное на основе классического алгоритма «поиска вглубь», требует предварительной подготовки: создания в оперативной памяти (`ENGINE = MEMORY`) двух таблиц и установки максимального уровня вложенности рекурсивных вызовов.

MySQL Решение 7.1.2.b (подготовка ко второму варианту решения)

```
1   -- Создание таблицы для хранения текущего пути:
2   CREATE TABLE IF NOT EXISTS `current_path`
3    (
4     `cp_id` INT PRIMARY KEY AUTO_INCREMENT,
5     `cp_from` INT,
6     `cp_to` INT,
7     `cp_cost` DOUBLE,
8     `cp_bidir` CHAR(1)
9    ) ENGINE = MEMORY;
10
11  -- Создание таблицы для хранения готовых путей:
12  CREATE TABLE IF NOT EXISTS `final_paths`
13   (
14    `fp_id` DOUBLE,
15    `fp_from` INT,
16    `fp_to` INT,
17    `fp_cost` DOUBLE,
18    `fp_bidir` CHAR(1)
19   ) ENGINE = MEMORY;
20
21  -- Установка максимального уровня вложенности рекурсивных вызовов:
22  SET @@SESSION.max_sp_recursion_depth = 255;
```

Теперь можно реализовывать алгоритм:

- если текущий путь пуст, отправной точкой является точка старта, иначе отправной точкой является точка прибытия последней связи в пути (строки 40-54);
- открыть курсор для выбора всех связей между городами (строки 18-32, 56);
- для всех связей повторять цикл, в котором:
  - проверить, совпадает ли отправная точка рассматриваемой связи с текущей отправной точкой (строки 69-72) и, если нет, перейти к следующей итерации цикла;
  - проверить, не присутствует ли уже рассматриваемая связь в пути (строки 74-80) и не приводит ли переход по этой связи к циклическому маршруту (строки 83-88) — в случае выполнения любого из этих условий перейти к следующей итерации цикла;
  - проверить (строка 91), не совпала ли конечная точка связи с точкой финиша:
    - если совпала — мы нашли путь, для которого генерируем уникальный идентификатор (строка 93) и переносим в таблицу для хранения найденных путей (строки 95-118), не забыв добавить в конец саму связь, которую мы только что рассматривали (строки 108-118);
    - если не совпала — путь ещё не найден, а потому: добавляем рассматриваемую связь к текущему пути (строки 121-131), выполняем рекурсивный вызов (строка 134), после которого убираем из текущего пути последнюю связь (строки 137-140: MySQL не позволяет одновременно читать и удалять данные из таблицы, потому идентификатор последней записи мы помещаем в переменную в строках 137-138, а затем используем в условии в строке 140).

По завершении работы в таблице `final_paths` будут находиться все найденные пути между двумя указанными городами.

MySQL Решение 7.1.2.b (код процедуры, второй вариант)

```
DELIMITER $$
CREATE PROCEDURE FIND_PATH (IN start_node INT,
                            IN finish_node INT)
BEGIN
 -- Признак выхода из цикла курсора:
 DECLARE done INT DEFAULT 0;

 -- Переменные для извлечения данных из курсора:
 DECLARE cn_from_value INT DEFAULT 0;
 DECLARE cn_to_value INT DEFAULT 0;
 DECLARE cn_cost_value DOUBLE DEFAULT 0;
 DECLARE cn_bidir_value CHAR(1) DEFAULT 0;

 -- Текущая "отправная точка"
 -- ВАЖНО! Эту переменную нельзя делать @глобальной !
 DECLARE from_node INT DEFAULT 0;
-- Курсор
 DECLARE nodes_cursor CURSOR FOR
 SELECT *
 FROM (SELECT `cn_from`,
              `cn_to`,
              `cn_cost`,
              `cn_bidir`
       FROM   `connections`
       UNION DISTINCT
       SELECT `cn_to`,
              `cn_from`,
              `cn_cost`,
              `cn_bidir`
       FROM   `connections`
       WHERE  `cn_bidir` = 'Y')
       AS `connections_bidir`;
       -- здесь можно дописать
       -- WHERE `cn_from` = from_node
       -- и убрать далее
       -- IF (cn_from_value != from_node)

 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

 IF ((SELECT COUNT(1)
      FROM `current_path`) = 0)
  THEN
   -- Если текущий путь пуст, отправной точкой
   -- является точка старта
   SET from_node = start_node;
  ELSE
```

MySQL Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
47      -- Если текущий путь НЕ пуст, отправной точкой
48      -- является точка прибытия последней связи в пути
49      SET from_node = (SELECT `cp_to`
50                       FROM `current_path`
51                       WHERE `cp_id` = (SELECT MAX(`cp_id`)
52                                        FROM `current_path`)
53                      );
54    END IF;
55
56    OPEN nodes_cursor;
57
58    nodes_loop: LOOP
59    FETCH nodes_cursor INTO cn_from_value,
60                            cn_to_value,
61                            cn_cost_value,
62                            cn_bidir_value;
63     IF done THEN
64      LEAVE nodes_loop;
65     END IF;
66
67     -- Отправная точка связи не совпадает с текущей
68     -- отправной точкой, пропускаем
69     IF (cn_from_value != from_node)
70      THEN
71       ITERATE nodes_loop;
72      END IF;
73     -- Такая связь уже есть в текущем пути, пропускаем
74     IF EXISTS (SELECT 1
75                FROM `current_path`
76                WHERE `cp_from` = cn_from_value
77                  AND `cp_to` = cn_to_value)
78      THEN
79       ITERATE nodes_loop;
80      END IF;
81
82     -- Такая связь приводит к циклу, пропускаем
83     IF EXISTS (SELECT 1
84                FROM `current_path`
85                WHERE `cp_from` = cn_to_value)
86      THEN
87       ITERATE nodes_loop;
88      END IF;
89
90     -- Конечная точка связи совпала с точкой финиша, путь найден
91     IF (cn_to_value = finish_node)
92      THEN
93       SET @rand_value = RAND();
94
```

MySQL Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
95      INSERT INTO `final_paths`
96                  (`fp_id`,
97                   `fp_from`,
98                   `fp_to`,
99                   `fp_cost`,
100                  `fp_bidir`)
101     SELECT @rand_value,
102            `cp_from`,
103            `cp_to`,
104            `cp_cost`,
105            `cp_bidir`
106     FROM   `current_path`;
107
108     INSERT INTO `final_paths`
109                 (`fp_id`,
110                  `fp_from`,
111                  `fp_to`,
112                  `fp_cost`,
113                  `fp_bidir`)
114     VALUES (@rand_value,
115             cn_from_value,
116             cn_to_value,
117             cn_cost_value,
118             cn_bidir_value);
119    ELSE
120     -- Добавляем связь в текущий путь
121     INSERT INTO `current_path`
122                 (`cp_id`,
123                  `cp_from`,
124                  `cp_to`,
125                  `cp_cost`,
126                  `cp_bidir`)
127     VALUES (NULL,
128             cn_from_value,
129             cn_to_value,
130             cn_cost_value,
131             cn_bidir_value);
132
133     -- Продолжаем рекурсивно искать следующие связи
134     CALL FIND_PATH (start_node, finish_node);
135
136     -- Удаляем последнюю связь из текущего пути
137     SET @max_cp_id = (SELECT MAX(`cp_id`)
138                       FROM `current_path`);
139     DELETE FROM  `current_path`
140            WHERE `cp_id` = @max_cp_id;
141   END IF;
142  END LOOP nodes_loop;
143  CLOSE nodes_cursor;
144 END;
145 $$
146 DELIMITER ;
```

Проверим, как работают полученные решения, выполнив такой код.

MySQL Решение 7.1.2.b (код для проверки работоспособности)

```
-- Для первого варианта решения:
CALL FIND_PATH({начальная_точка}, {конечная_точка});

-- Для второго варианта решения:
TRUNCATE TABLE `current_path`;
TRUNCATE TABLE `final_paths`;
CALL FIND_PATH({начальная_точка}, {конечная_точка});
SELECT * FROM `final_paths`;
```

На представленном в начале данного примера наборе данных для поиска пути из города 1 в город 6 оба решения возвращают одинаковые (хоть и по-разному представленные) результаты.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	6	30	N	3	1,7,3,6
1	6	85	N	3	1,7,2,6

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
0.42358866466543516	1	7	20	N
0.42358866466543516	7	2	15	Y
0.42358866466543516	2	6	50	N
0.34713028031134074	1	7	20	N
0.34713028031134074	7	3	5	N
0.34713028031134074	3	6	5	N

Но если в таблицу `connections` поместить следующие данные

cn_from	cn_to	cn_cost	cn_bidir
1	3	100	N
1	5	100	N
3	5	20	N
5	3	200	N

и поискать путь между городами 1 и 5, результаты будут разными.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	5	100	N	1	1,5

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
0.6203666074391143	1	3	100	N
0.6203666074391143	3	5	20	N
0.2569376912498266	1	5	100	N

Как и было сказано выше, первый вариант решения не находит альтернативные пути разной длины, в то время как второй вариант справляется с этим.

На этом решение для MySQL завершено.

Переходим к MS SQL Server и реализуем один-в-один решение, представленное выше для MySQL.

Алгоритм первого варианта решения:

- удаляется (если существует) и создаётся временная таблица для хранения найденных путей (строки 7-19);
- в созданную таблицу переносятся все данные из таблицы **connections** с учётом двунаправленности некоторых связей (строки 23-42; аналогичный подзапрос, учитывающий двунаправленные связи, используется в строках 69-81 — фактически, он представляет собой ничто иное, как тело представления из решения{514} задачи 7.1.2.a{513});
- выполняется цикл поиска производных маршрутов (строки 46-93), в котором:
  - условием выхода является отсутствие новых маршрутов (переменная MS SQL Server **@@ROWCOUNT** содержит количество записей, затронутых последней операцией модификации данных);
  - идея поиска новых маршрутов строится на том, чтобы к уже найденным маршрутам добавлять следующие шаги (конечная точка найденного маршрута совпадает с отправной точкой связи между городами, что проверяется в условии объединения в строке 82; условие в строках 83-84 исключает порождение циклических маршрутов; условие в строках 89-90 исключает бесконечное повторное дублирующихся маршрутов между двумя любыми городами).
- после того, как все возможные производные маршруты построены, в качестве результата работы хранимой процедуры возвращаются только те маршруты, точки отправки и назначения которых совпадают с переданными в хранимую процедуру параметрами (строки 96-100).

Обратите внимание на то, как формируется и анализируется маршрут, размещаемый в поле **cn_route** таблицы **connections_temp**: в MS SQL Server нет прямого аналога функции MySQL **FIND_IN_SET**, а при поиске вхождения подстроки в строку есть шанс, например, «найти» число 12 в числе 123 и т.д.

Потому каждое значение идентификатора берётся в квадратные скобки (маршрут примет вид наподобие «[1][7][3][6]»), и поиск тоже производится с предварительным заключением искомого идентификатора в квадратные скобки, что гарантирует отсутствие ложных срабатываний.

MS SQL Решение 7.1.2.b (код процедуры, первый вариант)

```
1   CREATE PROCEDURE FIND_PATH
2                    @start_node INT,
3                    @finish_node INT
4   AS
5    DECLARE @rows_inserted INT = 0;
6
7    -- Создание временной таблицы для хранения маршрутов:
8    IF OBJECT_ID('tempdb.dbo.#connections_temp', 'U') IS NOT NULL
9     DROP TABLE #connections_temp;
10
11   CREATE TABLE #connections_temp
12   (
13    [cn_from] INT,
14    [cn_to] INT,
15    [cn_cost] DOUBLE PRECISION,
16    [cn_bidir] CHAR(1),
17    [cn_steps] SMALLINT,
18    [cn_route] VARCHAR(1000)
19   );
20
```

MS SQL Решение 7.1.2.b (код процедуры, первый вариант) (продолжение)

```
21    -- Первичное наполнение временной таблицы
22    -- существующими маршрутами:
23    INSERT INTO #connections_temp
24    SELECT [cn_from],
25           [cn_to],
26           [cn_cost],
27           [cn_bidir],
28           1,
29           CONCAT('[', [cn_from], '][', [cn_to], ']')
30    FROM   (SELECT [cn_from],
31                   [cn_to],
32                   [cn_cost],
33                   [cn_bidir]
34            FROM   [connections]
35            UNION
36            SELECT [cn_to],
37                   [cn_from],
38                   [cn_cost],
39                   [cn_bidir]
40            FROM   [connections]
41            WHERE  [cn_bidir] = 'Y'
42           ) AS [connections_bidir];
43     -- Наполнение временной таблицы производными
44     -- маршрутами:
45     SET @rows_inserted = @@ROWCOUNT;
46     WHILE (@rows_inserted > 0)
47     BEGIN
48       INSERT INTO #connections_temp
49       SELECT [connections_next].[cn_from],
50              [connections_next].[cn_to],
51              [connections_next].[cn_cost],
52              [connections_next].[cn_bidir],
53              [connections_next].[cn_steps],
54              [connections_next].[cn_route]
55       FROM (SELECT #connections_temp.[cn_from] AS [cn_from],
56                    [connections].[cn_to] AS [cn_to],
57                   (#connections_temp.[cn_cost] +
58                    [connections].[cn_cost]) AS [cn_cost],
59                   CASE
60                    WHEN (#connections_temp.[cn_bidir] = 'Y')
61                     AND ([connections].[cn_bidir] = 'Y')
62                    THEN 'Y'
63                    ELSE 'N'
64                   END AS  [cn_bidir],
65                   (#connections_temp.[cn_steps] + 1) AS [cn_steps],
66                   CONCAT(#connections_temp.[cn_route], '[',
67                          [connections].[cn_to], ']') AS [cn_route]
68             FROM #connections_temp
69             JOIN (SELECT [cn_from],
70                          [cn_to],
71                          [cn_cost],
72                          [cn_bidir]
73                   FROM   [connections]
74                          UNION
75                   SELECT [cn_to],
76                          [cn_from],
77                          [cn_cost],
78                          [cn_bidir]
```

MS SQL Решение 7.1.2.b (код процедуры, первый вариант) (продолжение)

```
                  FROM   [connections]
                  WHERE  [cn_bidir] = 'Y'
                ) AS [connections]
              ON #connections_temp.[cn_to] = [connections].[cn_from]
                 AND CHARINDEX(CONCAT('[', [connections].[cn_to], ']'),
                                      #connections_temp.[cn_route]) = 0
          ) AS [connections_next]
      LEFT JOIN #connections_temp
             ON [connections_next].[cn_from] = #connections_temp.[cn_from]
                AND [connections_next].[cn_to] = #connections_temp.[cn_to]
      WHERE #connections_temp.[cn_from] IS NULL
        AND #connections_temp.[cn_to] IS NULL;

      SET @rows_inserted = @@ROWCOUNT;
    END; -- WHILE

    -- Извлечение маршрутов, соответствующих условию поиска:
    SELECT *
    FROM #connections_temp
    WHERE [cn_from] = @start_node
      AND [cn_to] = @finish_node
    ORDER BY [cn_cost] ASC;
GO
```

Для второго варианта решения, как и в случае с MySQL, нам понадобятся вспомогательные таблицы для хранения текущего и финальных путей.

MS SQL Решение 7.1.2.b (подготовка ко второму варианту решения)

```
-- Создание таблицы для хранения текущего пути:
IF OBJECT_ID('tempdb.dbo.#current_path', 'U') IS NOT NULL
  DROP TABLE #current_path;
CREATE TABLE #current_path
 (
  [cp_id] INT NOT NULL IDENTITY (1, 1),
  [cp_from] INT,
  [cp_to] INT,
  [cp_cost] DOUBLE PRECISION,
  [cp_bidir] CHAR(1)
 );

-- Создание таблицы для хранения готовых путей:
IF OBJECT_ID('tempdb.dbo.#final_paths', 'U') IS NOT NULL
  DROP TABLE #final_paths;
CREATE TABLE #final_paths
 (
  [fp_id] DOUBLE PRECISION,
  [fp_from] INT,
  [fp_to] INT,
  [fp_cost] DOUBLE PRECISION,
  [fp_bidir] CHAR(1)
 );
```

Алгоритм второго варианта решения:

- если текущий путь пуст, отправной точкой является точка старта, иначе отправной точкой является точка прибытия последней связи в пути (строки 34-45);
- открыть курсор для выбора всех связей между городами (строки 18-32, 47);
- для всех связей повторять цикл, в котором:
  - проверить, совпадает ли отправная точка рассматриваемой связи с текущей отправной точкой (строки 60-61 и, если нет, перейти к следующей итерации цикла;
  - проверить, не присутствует ли уже рассматриваемая связь в пути (строки 63-67) и не приводит ли переход по этой связи к циклическому маршруту (строки 70-73) — в случае выполнения любого из этих условий перейти к следующей итерации цикла;
  - проверить (строка 76), не совпала ли конечная точка связи с точкой финиша:
    - если совпала — мы нашли путь, для которого генерируем уникальный идентификатор (строка 78) и переносим в таблицу для хранения найденных путей (строки 79-90), не забыв добавить в конец саму связь, которую мы только что рассматривали (строки 91-101);
    - если не совпала — путь ещё не найден, а потому: добавляем рассматриваемую связь к текущему пути (строки 106-114), выполняем рекурсивный вызов (строка 117), после которого убираем из текущего пути последнюю связь (строки 120-121).

По завершении работы в таблице **`final_paths`** будут находиться все найденные пути между двумя указанными городами.

MS SQL Решение 7.1.2.b (код процедуры, второй вариант)

```
1   CREATE PROCEDURE FIND_PATH
2                    @start_node INT,
3                    @finish_node INT
4   AS
5    -- Переменные для извлечения данных из курсора:
6    DECLARE @cn_from_value INT = 0;
7    DECLARE @cn_to_value INT = 0;
8    DECLARE @cn_cost_value DOUBLE PRECISION = 0;
9    DECLARE @cn_bidir_value CHAR(1) = '0';
10
11   -- Текущая "отправная точка"
12   DECLARE @from_node INT = 0;
13
14   -- Идентификатор найденного пути
15   DECLARE @rand_value DOUBLE PRECISION = 0;
16
17   -- Курсор для прохода по связям между городами
18   DECLARE nodes_cursor CURSOR LOCAL FAST_FORWARD FOR
19    SELECT *
20     FROM (SELECT [cn_from],
21                  [cn_to],
22                  [cn_cost],
23                  [cn_bidir]
24           FROM   [connections]
25           UNION
26           SELECT [cn_to],
27                  [cn_from],
28                  [cn_cost],
29                  [cn_bidir]
30           FROM   [connections]
31           WHERE  [cn_bidir] = 'Y'
32          ) AS [connections_bidir];
33
```

MS SQL Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
34    IF ((SELECT COUNT(1)
35          FROM #current_path) = 0)
36      -- Если текущий путь пуст, отправной точкой является точка старта
37      SET @from_node = @start_node;
38     ELSE
39      -- Если текущий путь НЕ пуст, отправной точкой
40      -- является точка прибытия последней связи в пути
41      SET @from_node = (SELECT [cp_to]
42                        FROM   #current_path
43                        WHERE  [cp_id] = (SELECT MAX([cp_id])
44                                          FROM #current_path)
45                       );
46
47    OPEN nodes_cursor;
48
49    WHILE (1 = 1)
50     BEGIN
51      FETCH NEXT FROM nodes_cursor INTO @cn_from_value,
52                                        @cn_to_value,
53                                        @cn_cost_value,
54                                        @cn_bidir_value;
55      IF (@@FETCH_STATUS != 0)
56       BREAK;
57
58      -- Отправная точка связи не совпадает с текущей
59      -- отправной точкой, пропускаем
60      IF (@cn_from_value != @from_node)
61       CONTINUE;
62      -- Такая связь уже есть в текущем пути, пропускаем
63      IF EXISTS (SELECT 1
64                 FROM #current_path
65                 WHERE [cp_from] = @cn_from_value
66                   AND [cp_to] = @cn_to_value)
67       CONTINUE;
68
69      -- Такая связь приводит к циклу, пропускаем
70      IF EXISTS (SELECT 1
71                 FROM #current_path
72                 WHERE [cp_from] = @cn_to_value)
73       CONTINUE;
74
75     -- Конечная точка связи совпала с точкой финиша, путь найден
76      IF (@cn_to_value = @finish_node)
77       BEGIN
78        SET @rand_value = RAND();
79        INSERT INTO #final_paths
80                    ([fp_id],
81                     [fp_from],
82                     [fp_to],
83                     [fp_cost],
84                     [fp_bidir])
85        SELECT @rand_value,
86               [cp_from],
87               [cp_to],
88               [cp_cost],
89               [cp_bidir]
```

MS SQL Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
90        FROM   #current_path;
91        INSERT INTO #final_paths
92                    ([fp_id],
93                     [fp_from],
94                     [fp_to],
95                     [fp_cost],
96                     [fp_bidir])
97        VALUES (@rand_value,
98                @cn_from_value,
99                @cn_to_value,
100               @cn_cost_value,
101               @cn_bidir_value);
102     END
103    ELSE
104     BEGIN
105      -- Добавляем связь в текущий путь
106      INSERT INTO #current_path
107                   ([cp_from],
108                    [cp_to],
109                    [cp_cost],
110                    [cp_bidir])
111      VALUES (@cn_from_value,
112               @cn_to_value,
113               @cn_cost_value,
114               @cn_bidir_value);
115
116     -- Продолжаем рекурсивно искать следующие связи
117     EXEC FIND_PATH @start_node, @finish_node;
118
119     -- Удаляем последнюю связь из текущего пути
120     DELETE FROM  #current_path
121           WHERE [cp_id] = (SELECT MAX([cp_id]) FROM #current_path);
122    END;
123  END;
124  CLOSE nodes_cursor;
125  DEALLOCATE nodes_cursor;
126 GO
```

Проверим, как работают полученные решения, выполнив такой код.

MS SQL Решение 7.1.2.b (код для проверки работоспособности)

```
1  -- Для первого варианта решения:
2  EXEC FIND_PATH {начальная_точка}, {конечная_точка};
3
4  -- Для второго варианта решения:
5  TRUNCATE TABLE #current_path;
6  TRUNCATE TABLE #final_paths;
7  EXEC FIND_PATH {начальная_точка}, {конечная_точка};
8  SELECT * FROM #final_paths;
```

Поведение MS SQL Server оказывается полностью эквивалентным поведению MySQL.

На представленном в начале данного примера наборе данных для поиска пути из города 1 в город 6 оба решения возвращают одинаковые (хоть и по-разному представленные) результаты.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	6	30	N	3	[1][7][3][6]
1	6	85	N	3	[1][7][2][6]

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
0.159113517642648	1	7	20	N
0.159113517642648	7	2	15	Y
0.159113517642648	2	6	50	N
0.716279602109358	1	7	20	N
0.716279602109358	7	3	5	N
0.716279602109358	3	6	5	N

Но если в таблицу `connections` поместить следующие данные

cn_from	cn_to	cn_cost	cn_bidir
1	3	100	N
1	5	100	N
3	5	20	N
5	3	200	N

и поискать путь между городами 1 и 5, результаты будут разными.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	5	100	N	1	[1][5]

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
0.948062188221448	1	3	100	N
0.948062188221448	3	5	20	N
0.562740117282937	1	5	100	N

Таким образом, как и было сказано выше, в MS SQL Server первый вариант решения тоже не находит альтернативные пути разной длины, в то время как второй вариант справляется с этим.

На этом решение для MS SQL Server завершено.

Переходим к Oracle и реализуем решение, представленное выше для MySQL и MS SQL Server.

В отличие от двух других СУБД, Oracle не позволяет скомпилировать хранимую процедуру, внутри которой есть обращение к несуществующим объектам — даже если объекты создаются в этой же процедуре несколькими строками выше. Это ограничение можно обходить через EXECUTE IMMEDIATE и иными изощрёнными способами, но для простоты кода мы вынесем создание временной таблицы, с которой работает хранимая процедура, в отдельный код.

Oracle Решение 7.1.2.b (подготовка к первому варианту решения)

```
1   CREATE GLOBAL TEMPORARY TABLE "connections_temp"
2    (
3     "cn_from" NUMBER(10),
4     "cn_to" NUMBER(10),
5     "cn_cost" DOUBLE PRECISION,
6     "cn_bidir" CHAR(1),
7     "cn_steps" NUMBER(5),
8     "cn_route" VARCHAR(1000)
9    )
10   ON COMMIT PRESERVE ROWS;
```

Алгоритм первого варианта решения:

- в созданную до компиляции хранимой процедуры временную таблицу переносятся все данные из таблицы **connections** с учётом двунаправленности некоторых связей (строки 10-28; аналогичный подзапрос, учитывающий двунаправленные связи, используется в строках 55-67 — фактически, он представляет собой ничто иное, как тело представления из решения{514} задачи 7.1.2.a{513});
- выполняется цикл поиска производных маршрутов (строки 32-79), в котором:
  - условием выхода является отсутствие новых маршрутов (переменная Oracle **SQL%ROWCOUNT** содержит количество записей, затронутых последней операцией модификации данных);
  - идея поиска новых маршрутов строится на том, чтобы к уже найденным маршрутам добавлять следующие шаги (конечная точка найденного маршрута совпадает с отправной точкой связи между городами, что проверяется в условии объединения в строке 68; условие в строках 69-70 исключает порождение циклических маршрутов; условие в строках 75-76 исключает бесконечное повторное дублирующихся маршрутов между двумя любыми городами).
- после того, как все возможные производные маршруты построены, в качестве результата работы хранимой процедуры возвращаются только те маршруты, точки отправки и назначения которых совпадают с переданными в хранимую процедуру параметрами (строки 82-87).

Обратите внимание на то, как формируется и анализируется маршрут, размещаемый в поле **cn_route** таблицы **connections_temp**: в Oracle (как и в MS SQL Server) нет прямого аналога функции MySQL **FIND_IN_SET**, а при поиске вхождения подстроки в строку есть шанс, например, «найти» число 12 в числе 123 и т.д.

Потому каждое значение идентификатора берётся в квадратные скобки (маршрут примет вид наподобие «[1][7][3][6]»), и поиск тоже производится с предварительным заключением искомого идентификатора в квадратные скобки, что гарантирует отсутствие ложных срабатываний.

Oracle Решение 7.1.2.b (код процедуры, первый вариант)

```
1   CREATE OR REPLACE PROCEDURE FIND_PATH (start_node IN NUMBER,
2                                          finish_node IN NUMBER,
3                                          final_paths OUT SYS_REFCURSOR)
4   AS
5    rows_inserted NUMBER := 0;
6   BEGIN
7
8    -- Первичное наполнение временной таблицы существующими маршрутами:
9    INSERT INTO "connections_temp"
10   SELECT "cn_from",
11          "cn_to",
12          "cn_cost",
13          "cn_bidir",
14          1,
15          ('[' || "cn_from" || '][' || "cn_to" || ']')
16   FROM   (SELECT "cn_from",
17                  "cn_to",
18                  "cn_cost",
19                  "cn_bidir"
20           FROM   "connections"
21           UNION
22           SELECT "cn_to",
23                  "cn_from",
24                  "cn_cost",
25                  "cn_bidir"
26           FROM   "connections"
27           WHERE  "cn_bidir" = 'Y'
28          ) "connections_bidir";
29
```

Oracle Решение 7.1.2.b (код процедуры, первый вариант) (продолжение)

```
30    -- Наполнение временной таблицы производными маршрутами:
31    rows_inserted := SQL%ROWCOUNT;
32    WHILE (rows_inserted > 0)
33     LOOP
34      INSERT INTO "connections_temp"
35      SELECT "connections_next"."cn_from",
36             "connections_next"."cn_to",
37             "connections_next"."cn_cost",
38             "connections_next"."cn_bidir",
39             "connections_next"."cn_steps",
40             "connections_next"."cn_route"
41      FROM (SELECT "connections_temp"."cn_from" AS "cn_from",
42                   "connections"."cn_to" AS "cn_to",
43                  ("connections_temp"."cn_cost" +
44                   "connections"."cn_cost") AS "cn_cost",
45                  CASE
46                   WHEN ("connections_temp"."cn_bidir" = 'Y')
47                    AND ("connections"."cn_bidir" = 'Y')
48                   THEN 'Y'
49                   ELSE 'N'
50                  END AS  "cn_bidir",
51                  ("connections_temp"."cn_steps" + 1) AS "cn_steps",
52                  ("connections_temp"."cn_route" || '[' ||
53                   "connections"."cn_to" || ']') AS "cn_route"
54            FROM "connections_temp"
55            JOIN (SELECT "cn_from",
56                         "cn_to",
57                         "cn_cost",
58                         "cn_bidir"
59                  FROM   "connections"
60                         UNION
61                  SELECT "cn_to",
62                         "cn_from",
63                         "cn_cost",
64                         "cn_bidir"
65                  FROM   "connections"
66                  WHERE  "cn_bidir" = 'Y'
67                 ) "connections"
68              ON "connections_temp"."cn_to" = "connections"."cn_from"
69                 AND INSTR("connections_temp"."cn_route",
70                           '[' || "connections"."cn_to" || ']') = 0
71          ) "connections_next"
72      LEFT JOIN "connections_temp"
73             ON "connections_next"."cn_from" = "connections_temp"."cn_from"
74                AND "connections_next"."cn_to" = "connections_temp"."cn_to"
75      WHERE "connections_temp"."cn_from" IS NULL
76        AND "connections_temp"."cn_to" IS NULL;
77
78      rows_inserted := SQL%ROWCOUNT;
79     END LOOP;
80
81    -- Извлечение маршрутов, соответствующих условию поиска:
82    OPEN final_paths FOR
83    SELECT *
84    FROM "connections_temp"
85    WHERE "cn_from" = start_node
86      AND "cn_to" = finish_node
87    ORDER BY "cn_cost" ASC;
88  END;
```

Для второго варианта решения, как и в случае с MySQL и MS SQL Server, нам понадобятся вспомогательные таблицы для хранения текущего и финальных путей.

Oracle Решение 7.1.2.b (подготовка ко второму варианту решения)

```
1    -- Создание таблицы для хранения текущего пути:
2   CREATE GLOBAL TEMPORARY TABLE "current_path"
3    (
4     "cp_id" NUMBER(10),
5     "cp_from" NUMBER(10),
6     "cp_to" NUMBER(10),
7     "cp_cost" NUMBER(15,4),
8     "cp_bidir" CHAR(1)
9    );
10
11  -- Создание таблицы для хранения готовых путей:
12  CREATE GLOBAL TEMPORARY TABLE "final_paths"
13   (
14    "fp_id" NUMBER(15,4),
15    "fp_from" NUMBER(15,4),
16    "fp_to" NUMBER(15,4),
17    "fp_cost" NUMBER(15,4),
18    "fp_bidir" CHAR(1)
19   );
```

Алгоритм второго варианта решения:

- если текущий путь пуст, отправной точкой является точка старта, иначе отправной точкой является точка прибытия последней связи в пути (строки 26-40; обратите внимание на то, как в Oracle проверяется на пустоту результат выполнения запроса — классическое выражение IF (NOT) EXISTS здесь не работает);
- открыть курсор для выбора всех связей между городами (строки 8-24, 42);
- для всех связей повторять цикл, в котором:
  - проверить, совпадает ли отправная точка рассматриваемой связи с текущей отправной точкой (строки 46-49 и, если нет, перейти к следующей итерации цикла;
  - проверить, не присутствует ли уже рассматриваемая связь в пути (строки 52-60) и не приводит ли переход по этой связи к циклическому маршруту (строки 63-70) — в случае выполнения любого из этих условий перейти к следующей итерации цикла;
  - проверить (строка 73), не совпала ли конечная точка связи с точкой финиша:
    - если совпала — мы нашли путь, для которого генерируем уникальный идентификатор (строка 75) и переносим в таблицу для хранения найденных путей (строки 77-88), не забыв добавить в конец саму связь, которую мы только что рассматривали (строки 89-99);
    - если не совпала — путь ещё не найден, а потому: добавляем рассматриваемую связь к текущему пути (строки 102-113), выполняем рекурсивный вызов (строка 116), после которого убираем из текущего пути последнюю связь (строки 119-121).

По завершении работы в таблице `final_paths` будут находиться все найденные пути между двумя указанными городами.

Обратите внимание, как в строках 108-109 реализована эмуляция автоинкрементируемого первичного ключа без использования триггера.

Oracle Решение 7.1.2.b (код процедуры, второй вариант)

```
CREATE OR REPLACE PROCEDURE FIND_PATH (start_node IN NUMBER,
                                       finish_node IN NUMBER)
AS
 from_node NUMBER(10) := 0;
 rows_count NUMBER(10) := 0;
 rand_value NUMBER(15,4) := 0;

 CURSOR nodes_cursor IS
  SELECT "cn_from",
         "cn_to",
         "cn_cost",
         "cn_bidir"
  FROM (SELECT "cn_from",
               "cn_to",
               "cn_cost",
               "cn_bidir"
        FROM   "connections"
        UNION
        SELECT "cn_to",
               "cn_from",
               "cn_cost",
               "cn_bidir"
        FROM   "connections"
        WHERE  "cn_bidir" = 'Y');
BEGIN
 SELECT COUNT(1) INTO rows_count
 FROM "current_path";

 IF (rows_count = 0)
  THEN
   -- Если текущий путь пуст, отправной точкой является точка старта
   from_node := start_node;
  ELSE
   -- Если текущий путь НЕ пуст, отправной точкой
   -- является точка прибытия последней связи в пути
   SELECT "cp_to" INTO from_node
   FROM "current_path"
   WHERE "cp_id" = (SELECT MAX("cp_id")
                    FROM "current_path");
 END IF;

 FOR one_link IN nodes_cursor
  LOOP
   -- Отправная точка связи не совпадает с текущей
   -- отправной точкой, пропускаем
   IF (one_link."cn_from" != from_node)
   THEN
    CONTINUE;
   END IF;

```

Oracle Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
51    -- Такая связь уже есть в текущем пути, пропускаем
52    SELECT COUNT(1) INTO rows_count
53    FROM (SELECT 1
54          FROM  "current_path"
55          WHERE "cp_from" = one_link."cn_from"
56            AND "cp_to" = one_link."cn_to");
57    IF (rows_count > 0)
58     THEN
59      CONTINUE;
60     END IF;
61
62    -- Такая связь приводит к циклу, пропускаем
63    SELECT COUNT(1) INTO rows_count
64    FROM (SELECT 1
65          FROM  "current_path"
66          WHERE "cp_from" = one_link."cn_to");
67    IF (rows_count > 0)
68     THEN
69      CONTINUE;
70     END IF;
71
72    -- Конечная точка связи совпала с точкой финиша, путь найден
73    IF (one_link."cn_to" = finish_node)
74     THEN
75      rand_value := DBMS_RANDOM.VALUE(1,10);
76
77      INSERT INTO "final_paths"
78                  ("fp_id",
79                   "fp_from",
80                   "fp_to",
81                   "fp_cost",
82                   "fp_bidir")
83      SELECT rand_value,
84             "cp_from",
85             "cp_to",
86             "cp_cost",
87             "cp_bidir"
88      FROM   "current_path";
89      INSERT INTO "final_paths"
90                  ("fp_id",
91                   "fp_from",
92                   "fp_to",
93                   "fp_cost",
94                   "fp_bidir")
95      VALUES (rand_value,
96              one_link."cn_from",
97              one_link."cn_to",
98              one_link."cn_cost",
99              one_link."cn_bidir");
100    ELSE
```

Oracle Решение 7.1.2.b (код процедуры, второй вариант) (продолжение)

```
    -- Добавляем связь в текущий путь
    INSERT INTO "current_path"
                ("cp_id",
                 "cp_from",
                 "cp_to",
                 "cp_cost",
                 "cp_bidir")
    VALUES (NVL((SELECT MAX("cp_id") + 1
                 FROM "current_path"), 1),
                one_link."cn_from",
                one_link."cn_to",
                one_link."cn_cost",
                one_link."cn_bidir");

    -- Продолжаем рекурсивно искать следующие связи
    FIND_PATH (start_node, finish_node);

    -- Удаляем последнюю связь из текущего пути
    DELETE FROM  "current_path"
           WHERE "cp_id" = (SELECT MAX("cp_id")
                            FROM "current_path");
   END IF;
  END LOOP;
END;
```

Проверим, как работают полученные решения, выполнив представленный ниже код. Поведение Oracle оказывается полностью эквивалентным поведению MySQL и MS SQL Server.

На представленном в начале данного примера наборе данных для поиска пути из города 1 в город 6 оба решения возвращают одинаковые (хоть и по-разному представленные) результаты.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	6	30	N	3	[1][7][3][6]
1	6	85	N	3	[1][7][2][6]

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
4.5847	1	7	20	N
4.5847	7	2	15	Y
4.5847	2	6	50	N
8.5731	1	7	20	N
8.5731	7	3	5	N
8.5731	3	6	5	N

Oracle Решение 7.1.2.b (код для проверки работоспособности)

```
-- Для первого варианта решения:
TRUNCATE TABLE "connections_temp";
DECLARE
  fp SYS_REFCURSOR;
  cn_from NUMBER(10);
  cn_to NUMBER(10);
  cn_cost DOUBLE PRECISION;
  cn_bidir CHAR(1);
  cn_steps NUMBER(5);
  cn_route VARCHAR(1000);
```

Oracle Решение 7.1.2.b (код для проверки работоспособности) (продолжение)

```
BEGIN
  FIND_PATH({начальная_точка}, {конечная_точка}, fp);
  LOOP
    FETCH fp INTO cn_from,
                  cn_to,
                  cn_cost,
                  cn_bidir,
                  cn_steps,
                  cn_route;
    EXIT WHEN fp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(cn_from || ' | ' ||
                         cn_to || ' | ' ||
                         cn_cost || ' | ' ||
                         cn_bidir || ' | ' ||
                         cn_steps || ' | ' ||
                         cn_route);
  END LOOP;
  CLOSE fp;
END;

-- Для второго варианта решения:
TRUNCATE TABLE "current_path";
TRUNCATE TABLE "final_paths";
BEGIN
 FIND_PATH({начальная_точка}, {конечная_точка});
END;
SELECT * FROM "final_paths";
```

Но если в таблицу **connections** поместить следующие данные

cn_from	cn_to	cn_cost	cn_bidir
1	3	100	N
1	5	100	N
3	5	20	N
5	3	200	N

и поискать путь между городами 1 и 5, результаты будут разными.

Результат первого варианта решения:

cn_from	cn_to	cn_cost	cn_bidir	cn_steps	cn_route
1	5	100	N	1	[1][5]

Результат второго варианта решения:

fp_id	fp_from	fp_to	fp_cost	fp_bidir
3.5748	1	3	100	N
3.5748	3	5	20	N
5.881	1	5	100	N

Таким образом, как и было сказано выше, в Oracle первый вариант решения тоже не находит альтернативные пути разной длины, в то время как второй вариант справляется с этим.

На этом решение данной задачи завершено.

Задание 7.1.2.TSK.A: в тех СУБД, которые поддерживают соответствующую функциональность, реализовать решение{516} задачи 7.1.2.b{513} не через хранимую процедуру, а через хранимую функцию.

Задание 7.1.2.TSK.B: сравнить производительность первого и второго вариантов решения{516} задачи 7.1.2.b{513} для MySQL.

Задание 7.1.2.TSK.C: реализовать первый вариант решения{516} задачи 7.1.2.b{513} для Oracle с использованием выражения `CONNECT BY`.

7.2. ОПЕРАЦИИ С БАЗАМИ ДАННЫХ

7.2.1. ПРИМЕР 48: РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОСТАНОВЛЕНИЕ БАЗЫ ДАННЫХ

Задача 7.2.1.a{539}: написать командный файл для автоматизации создания резервной копии базы данных.

Задача 7.2.1.b{540}: написать командный файл для автоматизации восстановления базы данных из резервной копии.

Задача 7.2.1.c{541}: написать командный файл для автоматизации создания рабочей копии базы данных.

Ожидаемый результат 7.2.1.a:

Поскольку само решение и является ожидаемым результатом, см. решение 7.2.1.a{539}.

Ожидаемый результат 7.2.1.b:

Поскольку само решение и является ожидаемым результатом, см. решение 7.2.1.b{540}.

Ожидаемый результат 7.2.1.c:

Поскольку само решение и является ожидаемым результатом, см. решение 7.2.1.c{541}.

Решение 7.2.1.a{538}.

В данном случае под «резервной копией базы данных» мы будем понимать набор SQL-команд, выполнив которые (возможно, с предварительной правкой) на произвольном сервере, мы получим полноценную рабочую копию исходной базы данных.

Здесь **не** будет идти речь о создании резервных копий конкретного сервера и/или его конкретных баз данных, о резервировании по расписанию, инкрементных резервных копиях и прочих способах автоматизировать резервное копирование и восстановление в рамках одной системы.

В MySQL для выполнения соответствующей операции существует утилита mysqldump. Удостоверьтесь, что путь к ней прописан в переменной окружения PATH, после чего следующий командный файл будет создавать полную резервную копию указанной базы данных.

MySQL Решение 7.2.1.a (код cmd-файла) (общая идея)

```
1  mysqldump -uИМЯ -pПАРОЛЬ БАЗА_ДАННЫХ > БАЗА_ДАННЫХ.sql
```

MySQL Решение 7.2.1.a (код cmd-файла) (пример)

```
1  mysqldump -uabc -pdef library > library.sql
```

В MS SQL Server получить резервную копию базы данных в виде SQL-кода можно с помощью MS SQL Server Management Studio[43] или с помощью SQL Server Management Objects[44], но столь же простого и элегантного решения, как в MySQL, в этой СУБД нет. Зато есть возможность получить бинарную резервную копию с помощью следующего командного файла (удостоверьтесь, что путь к утилите sqlcmd прописан в переменной окружения PATH).

Представленный ниже код должен быть записан **одной строкой**, и расширение файла обязательно должно быть **bak**.

MS SQL Решение 7.2.1.a (код cmd-файла) (общая идея)

```
1  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "BACKUP DATABASE
   [БАЗА_ДАННЫХ] TO DISK='ПОЛНЫЙ_ПУТЬ_К_ФАЙЛУ.bak'"
```

MS SQL Решение 7.2.1.a (код cmd-файла) (пример)

```
1  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "BACKUP DATABASE
   [library] TO DISK='C:\backup\library.bak'"
```

В Oracle (как и в MS SQL Server) нет простого пути получения из командной строки полноценного SQL-кода с полной резервной копией базы данных (но можно использовать Oracle SQL Developer[45]), зато есть утилита expdp, создающая бинарную резервную копию. Код с её использованием показан ниже (удостоверьтесь, что путь к утилите expdp прописан в переменной окружения PATH).

Также обратите внимание на то, что в командном файле первые две строки отвечают за выполнение SQL-запросов, которые производят необходимую подготовку на уровне СУБД.

Oracle Решение 7.2.1.a (код cmd-файла) (общая идея)

```
1  @echo GRANT CREATE ANY DIRECTORY TO ИМЯ; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
2  @echo CREATE DIRECTORY ЛОГИЧЕСКОЕ_ИМЯ_КАТАЛОГА AS 'ПОЛНЫЙ_ПУТЬ_К_
   КАТАЛОГУ'; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
3  expdp ИМЯ/ПАРОЛЬ@СЕРВЕР schemas=ИМЯ directory=ЛОГИЧЕСКОЕ_ИМЯ_КАТАЛОГА
   dumpfile=ИМЯ_ФАЙЛА.dmp logfile=ИМЯ_ЛОГ_ФАЙЛА.log
```

43 http://blog.sqlauthority.com/2011/05/07/sql-server-2008-2008-r2-create-script-to-copy-database-schema-and-all-the-objects-data-schema-stored-procedure-functions-triggers-tables-views-constraints-and-all-other-database-objects/

44 https://www.simple-talk.com/sql/database-administration/automated-script-generation-with-powershell-and-smo/

45 http://docs.oracle.com/cd/E17781_01/server.112/e18804/impexp.htm#ADMQS256

Oracle Решение 7.2.1.a (код cmd-файла) (пример)

```
1  @echo GRANT CREATE ANY DIRECTORY TO abc; | sqlplus abc/def@COMP
2  @echo CREATE DIRECTORY LIBRARY_BACKUP AS 'C:\backup'; | sqlplus abc/
   def@COMP
3  expdp abc/def@COMP schemas=abc directory=LIBRARY_BACKUP
   dumpfile=library.dmp logfile=library_log.log
```

Пользуясь случаем, напоминаем: обязательно делайте резервные копии своих данных. Полученные резервные копии стоит проверить на пригодность к использованию для восстановления (т.е. восстановить на тестовом сервере).

На этом решение данной задачи завершено.

Решение 7.2.1.b[{538}].

В MySQL для решения данной задачи мы используем поставляемый с данной СУБД консольный клиент.

Удаление и повторное создание (строки 1-2) являются способом быстрого получения пустой базы данных: как правило, база данных, которую нужно восстанавливать из резервной копии, уже повреждена настолько, что её можно удалять.

Внимательно следите за именами баз данных в представленном ниже коде командного файла. Есть риск удалить «не ту» базу данных.

Если база данных, которую вы собираетесь «пересоздать», содержит полезную информацию, рекомендуется в любом случае сделать её отдельную резервную копию перед выполнением восстановления.

MySQL Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  mysql -uИМЯ -pПАРОЛЬ -e "DROP SCHEMA `БАЗА_ДАННЫХ`;";
2  mysql -uИМЯ -pПАРОЛЬ -e "CREATE SCHEMA `БАЗА_ДАННЫХ` DEFAULT
   CHARACTER SET utf8 COLLATE utf8_general_ci;";
3  mysql -uИМЯ -pПАРОЛЬ БАЗА_ДАННЫХ < БАЗА_ДАННЫХ.sql
```

MySQL Решение 7.2.1.b (код cmd-файла) (пример)

```
1  mysql -uabc -pdef -e "DROP SCHEMA `library`;";
2  mysql -uabc -pdef -e "CREATE SCHEMA `library` DEFAULT CHARACTER
   SET utf8 COLLATE utf8_general_ci;";
3  mysql -uabc -pdef library < library.sql
```

В MS SQL Server есть два пути восстановления базы данных из резервной копии. Первый путь актуален, если у вас есть полный SQL-код базы данных со всеми её структурами и данными.

MS SQL Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -i БАЗА_ДАННЫХ.sql
```

MS SQL Решение 7.2.1.b (код cmd-файла) (пример)

```
1  sqlcmd -U abc -P def -S COMP\MSSQLSRV -i library.sql
```

Второй путь актуален, если восстановление нужно произвести из бинарной резервной копии. Следующий код представляет собой **одну строку**.

MS SQL Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "RESTORE DATABASE
   [БАЗА_ДАННЫХ] FROM DISK='ПОЛНЫЙ_ПУТЬ_К_ФАЙЛУ.bak'"
```

MS SQL Решение 7.2.1.b (код cmd-файла) (пример)

```
1  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "RESTORE DATABASE
   [library] FROM DISK='C:\backup\library.bak'"
```

В Oracle восстановление можно выполнить с использованием утилиты impdp (при условии, что резервное копирование было выполнено утилитой expdp). Следующий код представляет собой **одну строку**.

Oracle Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  impdp ИМЯ/ПАРОЛЬ@СЕРВЕР schemas=ИМЯ directory=ЛОГИЧЕСКОЕ_ИМЯ_
   КАТАЛОГА dumpfile=ИМЯ_ФАЙЛА.dmp logfile=ИМЯ_ЛОГ_ФАЙЛА.log
```

Oracle Решение 7.2.1.b (код cmd-файла) (пример)

```
1  impdp abc/def@COMP schemas=abc directory=LIBRARY_BACKUP
   dumpfile=library.dmp logfile=library_log.log
```

На этом решение данной задачи завершено.

Решение 7.2.1.c[538].

Самым простым, быстрым и универсальным решением данной задачи для всех трёх СУБД является комбинация решения[539] задачи 7.2.1.a[538] и решения[540] задачи 7.2.1.b[538] с небольшой доработкой — восстанавливать базу данных мы будем под новым именем.

В MySQL не требуется никаких дополнительных приготовлений и ухищрений — просто реализуем только что описанную логику.

MySQL Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  rem Экспорт
2  mysqldump -uИМЯ -pПАРОЛЬ ИСХОДНАЯ_БАЗА_ДАННЫХ > ИСХОДНАЯ_БАЗА_ДАННЫХ.sql
3  rem Импорт
4  mysql -uИМЯ -pПАРОЛЬ -e "DROP SCHEMA `НОВАЯ_БАЗА_ДАННЫХ`;";
5  mysql -uИМЯ -pПАРОЛЬ -e "CREATE SCHEMA `НОВАЯ_БАЗА_ДАННЫХ` DEFAULT
   CHARACTER SET utf8 COLLATE utf8_general_ci;";
6  mysql -uИМЯ -pПАРОЛЬ НОВАЯ_БАЗА_ДАННЫХ < ИСХОДНАЯ_БАЗА_ДАННЫХ.sql
```

MySQL Решение 7.2.1.b (код cmd-файла) (пример)

```
1  rem Экспорт
2  mysqldump -uabc -pdef library > library.sql
3  rem Импорт
4  mysql -uabc -pdef -e "DROP SCHEMA `library_copy`;";
5  mysql -uabc -pdef -e "CREATE SCHEMA `library_copy` DEFAULT CHARACTER SET
   utf8 COLLATE utf8_general_ci;";
6  mysql -uabc -pdef library_copy < library.sql
```

В MS SQL Server экспорт представляет собой обычное резервное копирование, а в процессе импорта необходимо использовать конструкцию `WITH MOVE`, чтобы указать новые имена файлов базы данных.

Строка 5 нужна для того, чтобы определить логические имена, присутствующие в резервной копии, и вписать их в строку 6. Т.е. при первом запуске строку 6 можно закомментировать,

т.к. она всё равно, скорее всего, не сработает. А при повторных запусках можно закомментировать строку 5, т.к. она уже не нужна.

MS SQL Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  rem Экспорт
2  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "BACKUP DATABASE [ИСХОДНАЯ_
   БАЗА_ДАННЫХ] TO DISK='ПОЛНЫЙ_ПУТЬ_К_ФАЙЛУ.bak'"
3  rem Импорт
4  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "DROP DATABASE [НОВАЯ_
   БАЗА_ДАННЫХ]"
5  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "RESTORE FILELISTONLY FROM
   DISK='C:\!\ 'ПОЛНЫЙ_ПУТЬ_К_ФАЙЛУ.bak'"
6  sqlcmd -U ИМЯ -P ПАРОЛЬ -S СЕРВЕР\СЕРВИС -Q "RESTORE DATABASE [НОВАЯ_
   БАЗА_ДАННЫХ] FROM DISK='ПОЛНЫЙ_ПУТЬ_К_ФАЙЛУ.bak' WITH MOVE 'ЛОГИЧЕСКОЕ_
   ИМЯ_1' TO 'НОВОЕ_ПОЛНОЕ_ИМЯ_ФАЙЛА_1', MOVE 'ЛОГИЧЕСКОЕ_ИМЯ_2' TO
   'НОВОЕ_ПОЛНОЕ_ИМЯ_ФАЙЛА_2'"
```

MS SQL Решение 7.2.1.b (код cmd-файла) (пример)

```
1  rem Экспорт
2  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "BACKUP DATABASE [library] TO
   DISK='C:\backup\library.bak'"
3  rem Импорт
4  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "DROP DATABASE [library_copy]"
5  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "RESTORE FILELISTONLY FROM
   DISK='C:\backup\library.bak'"
6  sqlcmd -U abc -P def -S COMP\MSSQLSRV -Q "RESTORE DATABASE [library_
   copy] FROM DISK='C:\backup\library.bak' WITH MOVE 'library' TO 'C:\
   new\library_copy.mdf', MOVE 'library_log' TO 'C:\new\library_copy_log.
   ldf'"
```

В Oracle решение является самым объёмным.

Oracle Решение 7.2.1.b (код cmd-файла) (общая идея)

```
1  rem Экспорт
2  del ИМЯ_ФАЙЛА
3  del ИМЯ_ЛОГ_ФАЙЛА
4  @echo GRANT CREATE ANY DIRECTORY TO ИМЯ; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
5  @echo CREATE DIRECTORY ЛОГИЧЕСКОЕ_ИМЯ_КАТАЛОГА AS 'ПОЛНЫЙ_ПУТЬ_К_
   КАТАЛОГУ'; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
6  expdp ИМЯ/ПАРОЛЬ@СЕРВЕР schemas=ИМЯ directory=ЛОГИЧЕСКОЕ_ИМЯ_КАТАЛОГА
   dumpfile=ИМЯ_ФАЙЛА.dmp logfile=ИМЯ_ЛОГ_ФАЙЛА.log
7  rem Импорт
8  @echo DROP TABLESPACE ТАБЛИЧНОЕ_ПРОСТРАНСТВО_КОПИИ INCLUDING CONTENTS AND
   DATAFILES; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
9  @echo DROP USER ИМЯ_ДЛЯ_КОПИИ CASCADE; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
10 @echo CREATE TABLESPACE ТАБЛИЧНОЕ_ПРОСТРАНСТВО_КОПИИ DATAFILE 'ФАЙЛ_
   ДАННЫХ_КОПИИ.dat' SIZE 5M REUSE AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED;
   | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
11 @echo CREATE USER ИМЯ_ДЛЯ_КОПИИ IDENTIFIED BY ПАРОЛЬ_ДЛЯ_КОПИИ DEFAULT
   TABLESPACE ТАБЛИЧНОЕ_ПРОСТРАНСТВО_КОПИИ TEMPORARY TABLESPACE temp; |
   sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
12 @echo GRANT ALL PRIVILEGES TO ИМЯ_ДЛЯ_КОПИИ; | sqlplus ИМЯ/ПАРОЛЬ@СЕРВЕР
13 impdp ИМЯ/ПАРОЛЬ@СЕРВЕР schemas=ИМЯ directory=ЛОГИЧЕСКОЕ_ИМЯ_КАТАЛОГА
   remap_schema=ИМЯ:ИМЯ_ДЛЯ_КОПИИ remap_tablespace=ТАБЛИЧНОЕ_ПРОСТРАНСТВО:
   ТАБЛИЧНОЕ_ПРОСТРАНСТВО_КОПИИ dumpfile=ИМЯ_ФАЙЛА.dmp logfile=ИМЯ_ЛОГ_ФАЙЛА.
   log
```

Oracle Решение 7.2.1.b (код cmd-файла) (пример)

```
1  rem Экспорт
2  del "C:\backup\library.dmp"
3  del "C:\backup\library_log.log"
4  @echo GRANT CREATE ANY DIRECTORY TO abc; | sqlplus abc/def@COMP
5  @echo CREATE DIRECTORY LIBRARY_BACKUP AS 'C:\backup'; | sqlplus abc/def@
   COMP
6  expdp abc/def@COMP schemas=abc directory=LIBRARY_BACKUP dumpfile=library.
   dmp logfile=library_log.log
7  rem Импорт
   @echo DROP TABLESPACE library_copy_ts INCLUDING CONTENTS AND DATAFILES; |
8  sqlplus abc/def@COMP
9  @echo DROP USER library_copy CASCADE; | sqlplus abc/def@COMP
   @echo CREATE TABLESPACE library_copy_ts DATAFILE 'library_copy_ts.dat' SIZE
10 5M REUSE AUTOEXTEND ON NEXT 1M MAXSIZE UNLIMITED; | sqlplus abc/def@COMP
   @echo CREATE USER library_copy IDENTIFIED BY library_copy_pwd DEFAULT
11 TABLESPACE library_copy_ts TEMPORARY TABLESPACE temp; | sqlplus abc/def@
   COMP
12 @echo GRANT ALL PRIVILEGES TO library_copy; | sqlplus abc/def@COMP
13 impdp abc/def@COMP schemas=abc directory=LIBRARY_BACKUP remap_schema=abc:
   library_copy remap_tablespace=library_ts:library_copy_ts dumpfile=library.
   dmp logfile=library_log.log
```

Для начала нам нужно удалить существующие резервные копии (строки 2-3) и выполнить экспорт (строки 4-6) который представляет собой обычное резервное копирование, рассмотренное в решении{540} задачи 7.2.1.b{538}.

Затем мы:

- удаляем табличное пространство копии базы данных — на случай, если оно уже было (строка 8);
- удаляем пользователя/схему копии базы данных — на случай, если он существовал (строка 9);
- создаём табличное пространство копии базы данных (строка 10);
- создаём пользователя/схему копии базы данных (строка 11);
- выдаём пользователю копии базы данных все права (строка 12) — не делайте такого на реальных серверах, но в учебном контексте это допустимо;
- производим импорт (строка 13), указав изменение схемы (`remap_schema`) и табличного пространства (`remap_tablespace`).

На этом решение данной задачи завершено.

Задание 7.2.1.TSK.A: переписать командные файлы из решений 7.2.1.a{539}, 7.2.1.b{540} и 7.2.1.c{541} для работы под Linux.

Задание 7.2.1.TSK.B: реализовать (в тех СУБД, которые поддерживают такую функциональность) решение{541} задачи 7.2.1.c{538} без создания промежуточных файлов на диске.

7.3. ОПЕРАЦИИ С СУБД

7.3.1. ПРИМЕР 49: УПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯМИ, ЗАПУСК И ОСТАНОВКА СУБД

Задача 7.3.1.a{544}: создать нового пользователя СУБД и предоставить ему полный набор прав на базу данных «Библиотека».

Задача 7.3.1.b{545}: сменить созданному в задаче 7.3.1.a пользователю пароль.

Задача 7.3.1.c{546}: создать командные файлы для запуска, остановки, перезапуска СУБД.

Ожидаемый результат 7.3.1.a.

Поскольку само решение и является ожидаемым результатом, см. решение 7.3.1.a{544}.

Ожидаемый результат 7.3.1.b.

Поскольку само решение и является ожидаемым результатом, см. решение 7.3.1.b{545}.

Ожидаемый результат 7.3.1.b.

Поскольку само решение и является ожидаемым результатом, см. решение 7.3.1.c{546}.

Решение 7.3.1.a{544}.

В MySQL решение данной задачи достигается двумя командами. В строке 1 мы создаём пользователя (конструкция `'%'` после @ означает, что ему разрешён доступ с любых хостов), в строке 2 мы выдаём ему полные права на базу данных «Библиотека».

MySQL Решение 7.3.1.a

```
1  CREATE USER 'new_user'@'%' IDENTIFIED BY 'new_password';
2  GRANT ALL PRIVILEGES ON `library`.* TO 'new_user'@'%';
```

В MS SQL Server сначала необходимо создать логин для подключения к СУБД (строки 2-3), а затем (строки 6-8) можно создать пользователя, проассоциированного с ранее созданным ло-

гином, и сделать его владельцем базы данных «Библиотека», что предоставит ему полные права на неё.

MS SQL Решение 7.3.1.a

```
1  -- Создание логина (для подключения к серверу)
2  USE [master];
3  CREATE LOGIN [new_login] WITH PASSWORD = 'new_password';
4
5  -- Создание пользователя и выдача полных прав
6  USE [library];
7  CREATE USER [new_user] FOR LOGIN [new_login];
8  EXEC sp_addrolemember N'db_owner', N'new_user';
```

В Oracle нет простого способа выдать одному пользователю все права на всю схему другого пользователя. Существуют обходные пути[46], но они слишком громоздки.

Однако можно воспользоваться интересной особенностью данной СУБД (которую также поддерживает MySQL) — проксированием доступа, когда авторизация происходит от имени одного пользователя, а работа — от имени другого.

В строках 1-2 мы создаём нового пользователя, в строке 4 позволяем ему работать от имени владельца базы данных «Библиотека» и в строке 5 разрешаем ему устанавливать соединения с СУБД.

Oracle Решение 7.3.1.a

```
1  CREATE USER new_user IDENTIFIED BY new_password
2  DEFAULT TABLESPACE library_ts TEMPORARY TABLESPACE temp;
3
4  ALTER USER library GRANT CONNECT THROUGH new_user;
5  GRANT CREATE SESSION TO new_user;
```

Теперь можно проверить работоспособность этого решения, подключившись к СУБД через sqlplus с такими учётными данными: `new_user[library]/new_password@CEPBEP`. После запуска sqlplus можно удостовериться, что мы работаем от имени пользователя library, выполнив команду `SHOW USER`.

На этом решение данной задачи завершено.

Решение 7.3.1.b{544}.

Во всех трёх СУБД эта задача решается в одну строку (в MySQL необходимо использовать функцию PASSWORD для хеширования исходного значения пароля).

MySQL Решение 7.3.1.b

```
1  SET PASSWORD FOR 'new_user'@'%' = PASSWORD('some_password')
```

MS SQL Решение 7.3.1.b

```
1  ALTER LOGIN [new_login] WITH PASSWORD = 'some_password_3@'
```

Oracle Решение 7.3.1.b

```
1  ALTER USER new_user IDENTIFIED BY some_password
```

На этом решение данной задачи завершено.

46 https://community.oracle.com/thread/2386990?start=0&tstart=0

Решение 7.3.1.c{544}.

Решение данной задачи для всех трёх СУБД основано на использовании команд net start[47] и net stop[48]. Отличия будут только в именах сервисов, и для MS SQL Server (возможно) придётся управлять также его сопутствующими сервисами, набор и имена которых зависят от конкретной версии СУБД.

Представленный ниже код командных файлов производит перезапуск СУБД. Очевидно, что можно использовать отдельно части, отвечающие за остановку и запуск СУБД для достижения соответствующего результата.

MySQL Решение 7.3.1.c

```
1  rem Остановка
2  net stop MySQL56
3
4  rem Запуск
5  net start MySQL56
```

MS SQL Решение 7.3.1.c

```
1  rem Остановка
2  net stop "SQL Server (SQLEXPRESS)"
3
4  rem Запуск
5  net start "SQL Server (SQLEXPRESS)"
```

Oracle Решение 7.3.1.c

```
1  rem Остановка
2  net stop OracleServiceXE
3
4  rem Запуск
5  net start OracleServiceXE
```

На этом решение данной задачи завершено.

Задание 7.3.1.TSK.A: убрать у пользователя, созданного в решении{545} задачи 7.3.1.b{544} права доступа к базе данных «Библиотека».

Задание 7.3.1.TSK.B: удалить созданного в решении{544} задачи 7.3.1.a{544} пользователя.

Задание 7.3.1.TSK.C: написать командный файл, показывающий, запущена ли в настоящий момент каждая из трёх СУБД.

Задание 7.3.1.TSK.D: переписать решение{546} задачи 7.3.1.c{544} под управлением Linux.

47 https://technet.microsoft.com/en-us/library/bb490713.aspx

48 https://technet.microsoft.com/en-us/library/bb490715.aspx

7.3.2. ПРИМЕР 50: ОПРЕДЕЛЕНИЕ И ИЗМЕНЕНИЕ КОДИРОВОК

Задача 7.3.2.a{547}: вывести всю информацию о текущих настройках СУБД относительно кодировок, используемых по умолчанию.

Задача 7.3.2.b{549}: изменить все настройки кодировок СУБД по умолчанию на UTF8.

Ожидаемый результат 7.3.2.a.

В результате выполнения запроса (запросов) отображаются текущие настройки СУБД относительно кодировок, используемых по умолчанию.

Ожидаемый результат 7.3.2.b.

В результате выполнения запроса (запросов) кодировки СУБД по умолчанию меняются на UTF8.

Решение 7.3.2.a{547}:

В MySQL информацию о кодировках можно получить следующим запросом (обратите внимание: символ _ экранирован символом \, т.к. _ обозначает «любой символ» — в данном случае это не критично, но всё равно следует писать правильно).

MySQL Решение 7.3.2.a

```
1  SHOW GLOBAL VARIABLES
2  WHERE `variable_name` LIKE 'character\_set%'
3     OR `variable_name` LIKE 'collat%'
```

Результатом выполнения такого запроса будет таблица со следующими данными.

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
character_set_system	utf8
character_sets_dir	C:\Program Files\MySQL\MySQL Server 5.6\share\charsets\
collation_connection	utf8_general_ci
collation_database	utf8_general_ci
collation_server	utf8_general_ci

В MS SQL Server информацию о кодировках можно получить следующим запросом.

MS SQL Решение 7.3.2.a

```
SELECT [collation_name]
FROM [sys].[databases]
WHERE [name] = 'master'
```

Результатом выполнения такого запроса будет таблица со следующими данными.

collation_name
Cyrillic_General_CI_AS

MS SQL Server будет применять эту кодировку по умолчанию ко всем новым базам данных и их структурам, если в соответствующих запросах не буде указана иная кодировка.

В Oracle информацию о кодировках можно получить следующим запросом.

Oracle Решение 7.3.2.a

```
-- Упрощённый вариант:
SELECT value$
FROM   sys.props$
WHERE  name = 'NLS_CHARACTERSET';

-- Расширенный вариант:
SELECT parameter,
       value
FROM   nls_database_parameters
WHERE  parameter = 'NLS_CHARACTERSET'
    OR parameter = 'NLS_NCHAR_CHARACTERSET';
```

Первый запрос вернёт следующую информацию.

VALUE$
AL32UTF8

Второй запрос вернёт следующую информацию.

PARAMETER	VALUE
NLS_CHARACTERSET	AL32UTF8
NLS_NCHAR_CHARACTERSET	AL16UTF16

Значение параметра **NLS_CHARACTERSET** отвечает за кодировку «обычных» данных (без приставки N, т.е. **CHAR**, **VARCHAR2** и т.д.), а значение **NLS_NCHAR_CHARACTERSET** — за кодировку «национальных данных» (т.е. **NCHAR**, **NVARCHAR2** и т.д.)

На этом решение данной задачи завершено.

Решение 7.3.2.b{547}.

Изменение кодировок на уровне всего сервера может привести к временному или постоянному искажению данных в имеющихся базах данных. Обязательно сделайте полную резервную копию перед проведением соответствующих экспериментов.

В MySQL изменить настройки кодировок СУБД по умолчанию можно или в конфигурационном файле my.ini (постоянные изменения), или запросами следующего вида (изменения действуют до перезапуска СУБД):

MySQL Решение 7.3.1.b

```
1  -- Повторять для всех необходимых переменных:
2  SET character_set_database = 'utf8';
```

В MS SQL Server изменить настройки кодировок СУБД по умолчанию можно только достаточно сложным способом:

- остановить СУБД;
- выполнить из консоли следующую команду (её параметры могут отличаться в зависимости от версии СУБД):

MS SQL Решение 7.3.1.b

```
1  sqlservr -m -T4022 -T3659 -s"СЕРВИС" -q"КОДИРОВКА"
```

- запустить СУБД.

Что касается самой кодировки, то «в чистом виде» UTF8 не поддерживается в MS SQL Server, но хранения и обработки соответствующих данных можно добиться через использование других кодировок[49].

И, наконец, в Oracle изменение кодировок по умолчанию возможно только с выполнением достаточно нетривиальной процедуры[50] (описание которой явно выходит за рамки данной книги), а потому рекомендуется на стадии инсталляции СУБД указать все необходимые параметры по умолчанию, а также указывать кодировки при создании отдельных баз данных и таблиц.

Задание 7.3.2.TSK.A: провести во всех трёх СУБД эксперимент с изменением кодировки существующей базы данных, содержащей строки с кириллическими символами; проверить, как изменится отображение, поиск и упорядочение таких данных.

Задание 7.3.2.TSK.B: провести во всех трёх СУБД эксперимент с изменением кодировки соединения на отличную от кодировки базы данных, с которой предстоит работа; проверить, как изменится отображение, поиск и упорядочение строк, содержащих кириллические символы.

49 https://msdn.microsoft.com/en-us/library/ms143726%28v=sql.110%29.aspx

50 https://docs.oracle.com/cd/E11882_01/server.112/e10729/ch11charsetmig.htm#NLSPG011

КРАТКОЕ СРАВНЕНИЕ MYSQL, MS SQL SERVER, ORACLE

Данный раздел является своего радо шпаргалкой по основным различиям трёх СУБД, упомянутым в данной книге. Ссылки на соответствующие примеры и задачи позволяют быстро перейти к нужной части материала и ознакомиться с подробностями.

Особенности запросов на выборку и модификацию данных.

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Условия объединения в `JOIN`	Можно использовать **ON** или **USING(одноимённые поля в объединяемых таблицах)**	Можно использовать только `ON`	Можно использовать **ON** или **USING(одноимённые поля в объединяемых таблицах)**	2.2.1.a{69}
Типы данных для `AVG` и иных числовых операций	Результирующий тип выбирается автоматически, конвертация не нужна	Нужна конвертация, иначе будут неверные результаты (например, `AVG` от целочисленного поля тоже будет целым числом, т.е. дробная часть будет утеряна)	Результирующий тип выбирается автоматически, конвертация не нужна	2.1.5.a{32}
Расположение по умолчанию `NULL`-значений при сортировке	В конце	В конце	В начале; можно изменить через `NULLS FIRST` и `NULLS LAST`	2.1.6.EXP.A{39}
Приведение строкового представления даты к датавременному типу	Автоматическое	Автоматическое	Требуется явная конвертация	2.1.7.b{42}

Особенности запросов на выборку и модификацию данных (продолжение).

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Показ первых нескольких рядов выборки	`LIMIT x`	`TOP x` или `OFFSET y ROWS FETCH NEXT x ROWS ONLY`	До 12c нужен вложенный запрос с `ROW_NUMBER()`, с 12c поддерживается `OFFSET y ROWS FETCH NEXT x ROWS ONLY`	2.1.8.b{45}
		Есть также поддержка `TOP ... WITH TIES`		2.2.7.b{119}
Ранжирующие (оконные функции)	Не поддерживаются, но можно проэмулировать	`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `NTILE`	`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `NTILE`	2.1.8.EXP.D{52}, 2.2.7.d{123}, 2.2.9.d{144}
Именование подзапросов, являющихся источником табличных данных (в секции `FROM` и `JOIN`)	Обязательно	Обязательно	Обязательно, при этом нельзя писать `AS` перед именем подзапроса	2.1.8.EXP.D{52}, 2.1.8.c{47}, 2.1.8.d{49}
Общие табличные выражения	Не поддерживаются, но иногда можно эмулировать	Поддерживаются	Поддерживаются	2.1.8.d{49}, 2.2.6.b{106}
Определение разницы в днях между двумя датами	`DATEDIFF(большая, меньшая)`	`DATEDIFF(day, меньшая, большая)`	`(большая - меньшая)`	2.1.9.d{59}
Получение текущей даты	`CURDATE()`	`CONVERT(date, GETDATE()`	`TRUNC(SYSDATE)`	2.1.9.d{59}
Особенности `GROUP BY`	Можно в `SELECT` указывать поля, не являющиеся агрегирующими функциями и не перечисленные в `GROUP BY`. Можно в `GROUP BY` ссылаться на имя (псевдоним) выражения, используемого в `SELECT`.	Нельзя в `SELECT` указывать поля, не являющиеся агрегирующими функциями и не перечисленные в `GROUP BY`. Нельзя в `GROUP BY` ссылаться на имя (псевдоним) выражения, используемого в `SELECT`.	Нельзя в `SELECT` указывать поля, не являющиеся агрегирующими функциями и не перечисленные в `GROUP BY`. Нельзя в `GROUP BY` ссылаться на имя (псевдоним) выражения, используемого в `SELECT`.	2.1.10.a{64}, 2.2.2.a{74}
Групповая конкатенация	`GROUP_CONCAT()`	`STUFF (FOR XML...)`	`LISTAGG()`	2.2.2.a{74}, 2.2.2.b{77}
Выборка данных из нескольких таблиц и одноимённые поля	Как правило, не требуется указания имени таблицы, СУБД автоматически определяет, откуда извлекать данные	Всегда требуется явное указание имени таблицы, если в выборку попадает два и более одноимённых поля	Всегда требуется явное указание имени таблицы, если в выборку попадает два и более одноимённых поля	2.2.5.c{102}
«Защита от `NULL`»	`IFNULL(поле, подстановочное значение)`	`ISNULL(поле, подстановочное значение)`	`NVL(поле, подстановочное значение)`	2.2.6.b{106}

Особенности запросов на выборку и модификацию данных (продолжение).

Суть	MySQL	MS SQL Server	Oracle	Ссылка
`TRUE` / `FALSE`	Есть, трактуются как 1 и 0	Явным образом отсутствуют	Явным образом отсутствуют	2.2.7.e{127}
Разновидности `JOIN`	`JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `CROSS JOIN`, не поддерживается `FULL OUTER JOIN` и `CROSS/ OUTER APPLY`	`JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `CROSS JOIN`, `FULL OUTER JOIN`, `CROSS APPLY` и `OUTER APPLY`	`JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `CROSS JOIN`, `FULL OUTER JOIN`, с версии 12c поддерживается `CROSS APPLY` и `OUTER APPLY`	Пример 20{153}
Проверка того факта, что запрос вернул (не)пустой результат	`(NOT) EXISTS (запрос)`	`(NOT) EXISTS (запрос)`	`EXISTS (запрос)`, а вместо `NOT EXISTS` нужно объявить переменную, сделать `SELECT COUNT` в неё, сравнить её с нулём	2.2.7.e{127} 2.3.5.b{210} 6.1.2.a{441}
Конкатенация строк	`CONCAT(сколько угодно параметров)`	`CONCAT(сколько угодно параметров)`	`CONCAT(ровно два параметра)` или использовать `||`	2.3.5.c{212}
Вставка значения в автоинкрементируемый первичный ключ	Просто передать значение – и всё	Включить для таблицы `IDENTITY_INSERT`	Отключить триггер, реализующий автоинкрементацию	2.3.1.a{189}
Вставка или обновление в зависимости от совпадения значений первичного ключа	`REPLACE` и `ON DUPLICATE KEY UPDATE`	`MERGE`	`MERGE`	2.3.4.a{201}, 2.3.4.b{203}

Особенности представлений, триггеров, хранимых подпрограмм, транзакций.

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Построение представлений	Нельзя использовать подзапросы в секции `FROM`	Можно использовать подзапросы в секции `FROM`	Можно использовать подзапросы в секции `FROM`	3.1.1.a{218}
Кэширующие (материализованные, индексированные) представления	Не поддерживаются	Поддерживаются	Поддерживаются	Пример 27{222}

Особенности представлений, триггеров, хранимых подпрограмм, транзакций (продолжение).

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Триггеры	Не более шести штук (даже если пытаться задать разные имена): `BEFORE/AFTER INSERT/UPDATE/DELETE`	`BEFORE`-триггеров нет, но можно создавать любое количество `AFTER` и `INSTEAD OF` триггеров на `INSERT/UPDATE/DELETE`	`BEFORE`-триггеров нет, но можно создавать любое количество `AFTER` и `INSTEAD OF` триггеров на `INSERT/UPDATE/DELETE`	3.1.2.b{240}
	Работают только на уровне записи (активируются каждый раз для каждой записи обрабатываемых данных)	Работают только на уровне выражения (активируются один раз для всего выражения)	Есть как триггеры уровня записи, так и триггеры уровня выражения (с некоторыми ограничениями: нет псевдотаблиц `inserted` и `updated`)	4.1.1.a{285}, 4.1.1.b{294}
	Нельзя «навесить» одно тело триггера на несколько операций	Можно «навесить» одно тело триггера на несколько операций	Можно «навесить» одно тело триггера на несколько операций	4.1.1.b{294}
	Не активируются каскадными операциями	Активируются каскадными операциями	Активируются каскадными операциями	4.1.1.a{285}, 4.1.2.a{306}
Триггеры на представлениях	Не поддерживаются	Поддерживаются только `INSTEAD OF`	Поддерживаются только `INSTEAD OF`	Пример 30{267}
Неявные транзакции и их автоподтверждение	Есть	Есть	Нет	Пример 41{429}
Уровни изолированности транзакций	`READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE`	`READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, `SNAPSHOT`, `SERIALIZABLE`	Уровни `READ COMMITTED`, `SERIALIZABLE`; режимы `READ ONLY`, `READ WRITE`	Пример 44{456}
Уровень изолированности по умолчанию	`REPEATABLE READ`	`READ COMMITTED`	`READ COMMITTED`	6.2.1.a{450}
Вывод отладочной информации	`SELECT x`	`PRINT x`	`DBMS_OUTPUT.PUT_LINE(x)`	6.2.2.a{456}
Пауза в выполнении скрипта	`SELECT SLEEP(s)`	`WAITFOR DELAY 'hh:mm:ss'`	`EXEC DBMS_LOCK.SLEEP(s)`	6.2.1.a{450}
Модификация данных в хранимых функциях	Да	Нет	Да	5.1.1.c{385}

Задание 8.TSK.A: дополните приведённую выше таблицу списками индексов, поддерживаемых каждой из трёх СУБД.

Задание 8.TSK.B: заведите себе подобную таблицу и отмечайте особенности каждой СУБД – в первую очередь те, столкнувшись с которыми, вы вынуждены искать дополнительную информацию.

ЛИЦЕНЗИЯ И РАСПРОСТРАНЕНИЕ

Текст книги периодически обновляется и дорабатывается. Если вы хотите поделиться этой книгой, пожалуйста, делитесь ссылкой на самую актуальную версию, доступную здесь:
http://svyatoslav.biz/database_book/.

Исходные материалы (схемы, скрипты и т.д.) можно получить по этой ссылке:
http://svyatoslav.biz/database_book_download/src.zip

В случае возникновения вопросов или обнаружения ошибок, опечаток и иных недочётов в книге, пишите: dbb@svyatoslav.biz.

[51] «Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International». [https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode]

Производственно-практическое издание

Святослав Святославович **Куликов**

РАБОТА С MYSQL, MS SQL SERVER И ORACLE В ПРИМЕРАХ

Практическое пособие

Публикуется в авторской редакции

Дизайн обложки *Р. В. Щуцкий*
Компьютерная вёрстка *А. М. Сергеев*

Подписано в печать 08.08.2016.
Формат 60×84/8. Бумага мелованная. Печать офсетная.
Усл. печ. л. 64,63. Уч.-изд. л. 42,71.
Тираж 1000 экз. Заказ 2096.

УП «БОФФ».
Ул. Кнорина, 50, корп. 4, к. 102а, 220103, г. Минск.
Свидетельство о государственной регистрации
издателя, изготовителя и распространителя печатных изданий
№ 1/88 от 18.11.2013

Отпечатано в типографии ООО «Поликрафт».
Ул. Кнорина, 50, корп. 4, к. 401а, 220103, г. Минск.
Лицензия № 02330/466 от 21.04.2014.

Об авторе:

Святослав Куликов

Специалист по подготовке персонала EPAM Systems, кандидат технических наук, доцент Белорусского государственного университета информатики и радиоэлектроники.

Автор и ведущий тренингов “Основы функционального тестирования”, “Автоматизация тестирования”, “Веб-разработка с использованием PHP”.

Более двадцати лет в IT, более десяти лет опыта подготовки тестировщиков и веб-разработчиков.

Блог автора: http://svyatoslav.biz

epam | EPAM training center