Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения
информационных технологий

С. С. Куликов, Е. Е. Фадеева

РАБОТА С MYSQL, MS SQL SERVER И ORACLE В ПРИМЕРАХ

В двух частях

Часть 1

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальностей 1-40 01 01 «Программное обеспечение информационных технологий», 1-40 80 05 «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Минск БГУИР 2019

УДК 004.655.3(076)
ББК 32.972.134я73
К90

Р е ц е н з е н т ы:

отдел интеллектуальных информационных систем
государственного научного учреждения «Объединённый институт проблем
информатики Национальной академии наук Беларуси»
(протокол №1 от 14.05.2018);

доцент кафедры информационных систем управления
Белорусского государственного университета
кандидат физико-математических наук, доцент А. В. Кузьмина;

заведующий кафедрой электронных вычислительных машин
учреждения образования «Белорусский государственный
университет информатики и радиоэлектроники»
кандидат технических наук, доцент Д. И. Самаль

Куликов, С. С.

К90 Работа с MySQL, MS SQL Server и Oracle в примерах. В 2 ч. Ч. 1 : учеб.-метод. пособие / С. С. Куликов, Е. Е. Фадеева. – Минск : БГУИР, 2019. – 287 с.

ISBN 978-985-543-440-6 (ч. 1).

Содержит материалы к практическим занятиям, включающие примеры и задания.

УДК 004.655.3(076)
ББК 32.972.134я73

ISBN 978-985-543-440-6 (ч. 1)
ISBN 978-985-543-439-0

Содержание

ПРЕДИСЛОВИЕ

Данное учебно-методическое пособие посвящено практике использования SQL для решения типичных задач.

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 ранних.

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.

Условные обозначения, используемые в учебно-методическом пособии:

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, вводятся номера задач и в соответствие им поставлены номера ожидаемого результата и решения задачи.

Ожидаемый результат. Здесь мы будем показывать, что должно получаться при правильном решении задачи. Номер ожидаемого результата будет соответствовать номеру задачи, для которой он приведён.

Решение задачи. Решение всегда будет приведено для MySQL, MS SQL Server и Oracle. Иногда решения будут похожими, иногда очень разными. Номер приведённого решения будет соответствовать номеру задачи, для которой оно приведено.

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, то сначала попробуйте самостоятельно решить уже разобранные примеры, обращаясь к готовому решению лишь для самопроверки.

1. МОДЕЛЬ, ГЕНЕРАЦИЯ И НАПОЛНЕНИЕ БАЗЫ ДАННЫХ

1.1. ОБЩЕЕ ОПИСАНИЕ МОДЕЛИ

Для решения задач и рассмотрения примеров будет использоваться три базы данных: «Библиотека», «Большая библиотека» и «Исследование». База данных «Исследование» будет состоять из множества разрозненных таблиц, необходимых для демонстрации особенностей поведения СУБД, и ее формирование происходит 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 рис. 1.а. Скрипты генерации баз данных для каждой СУБД можно найти в исходном материале, ссылка на который приведена выше.

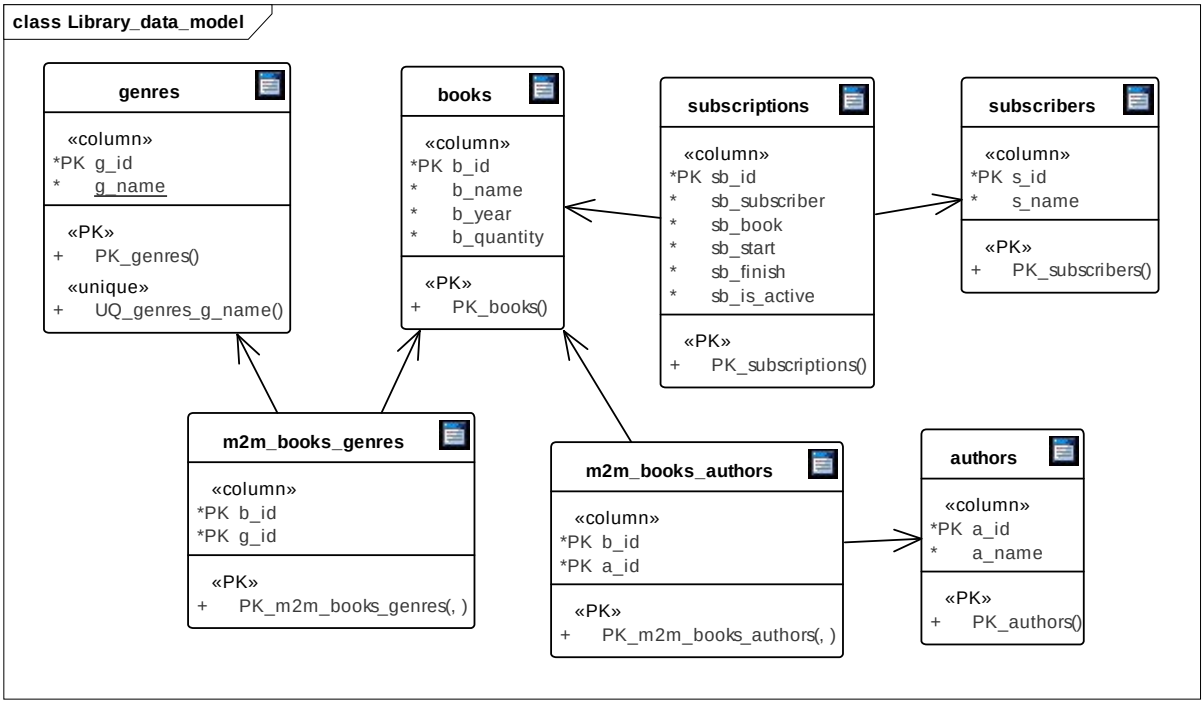


Рис. 1.a. Общая схема базы данных

1.2. МОДЕЛЬ ДЛЯ MYSQL

Модель базы данных для MySQL представлена на рис. 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 4 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, в нашем случае – **Y** и **N**).
- 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.

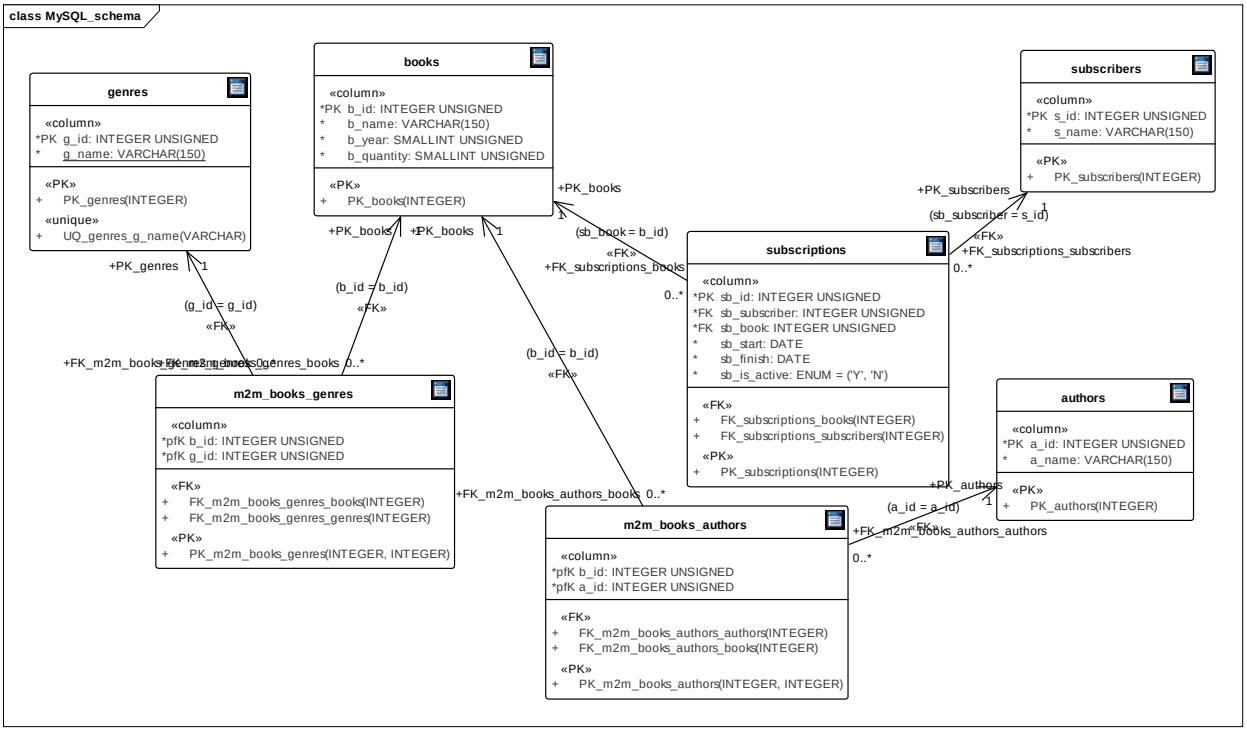


Рис. 1.b. Модель базы данных для MySQL в Sparx EA

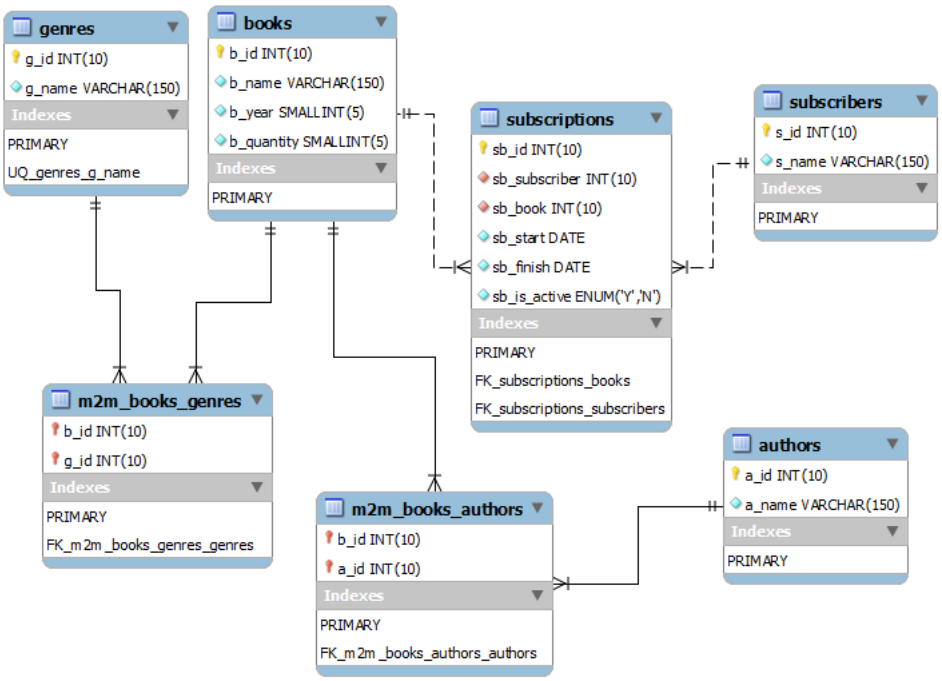


Рис. 1.c. Модель базы данных для MySQL в MySQL Workbench

1.3. МОДЕЛЬ ДЛЯ MS SQL SERVER

Модель базы данных для MS SQL Server представлена на рис. 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 2 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.

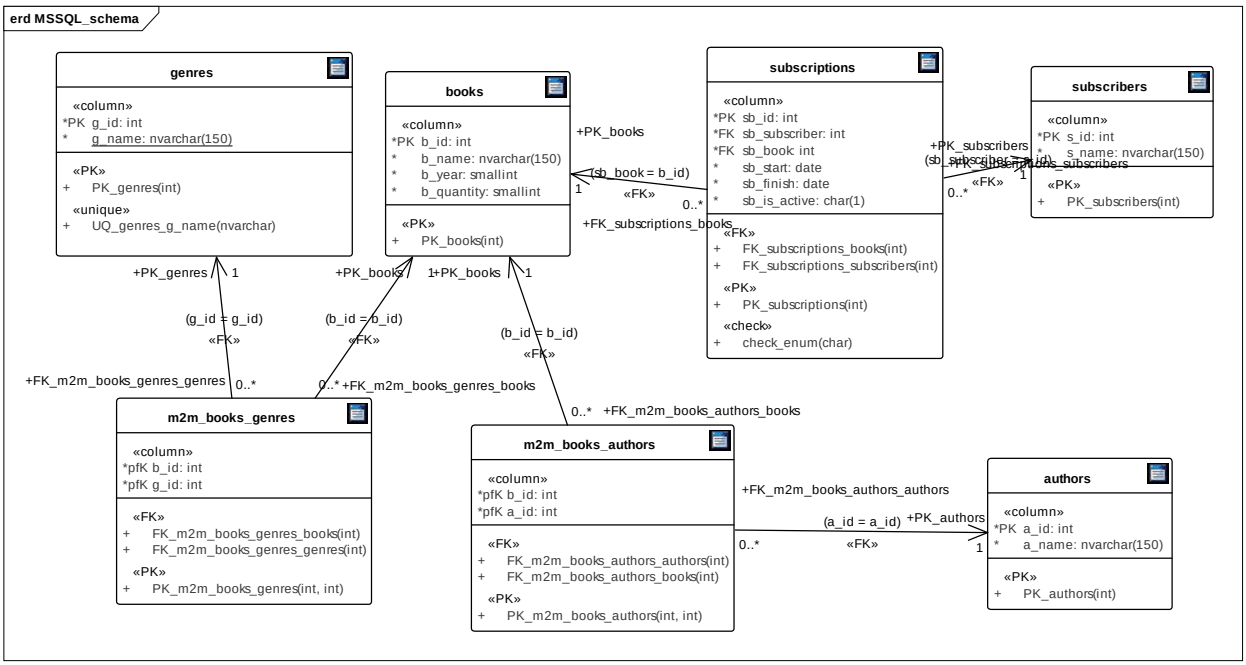


Рис. 1.d. Модель базы данных для MS SQL Server в Sparx EA

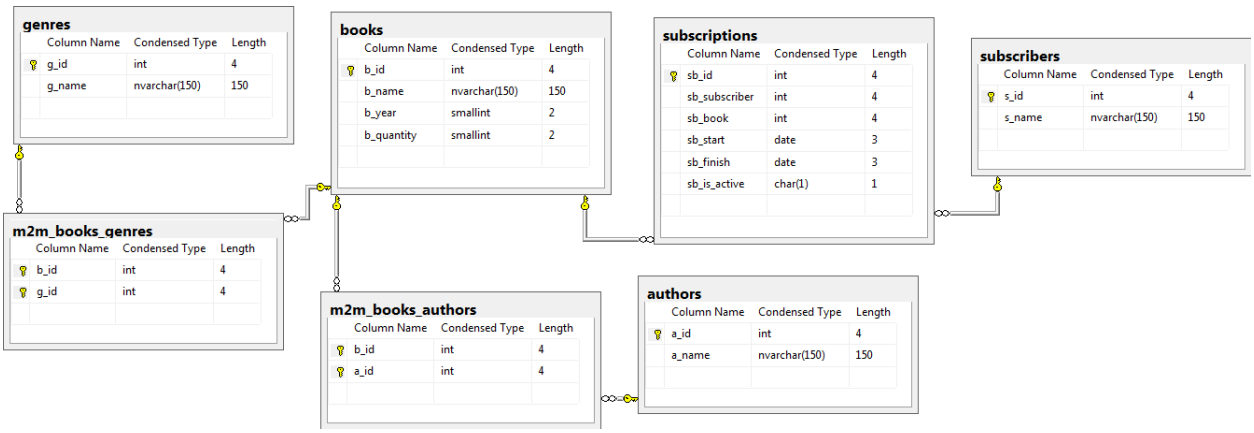


Рис. 1.e. Модель базы данных для MS SQL Server в MS SQL Server Management Studio

1.4. МОДЕЛЬ ДЛЯ ORACLE

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

- Целочисленные поля представлены типом **number** с указанием длины, что является наиболее простым способом 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 2 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.

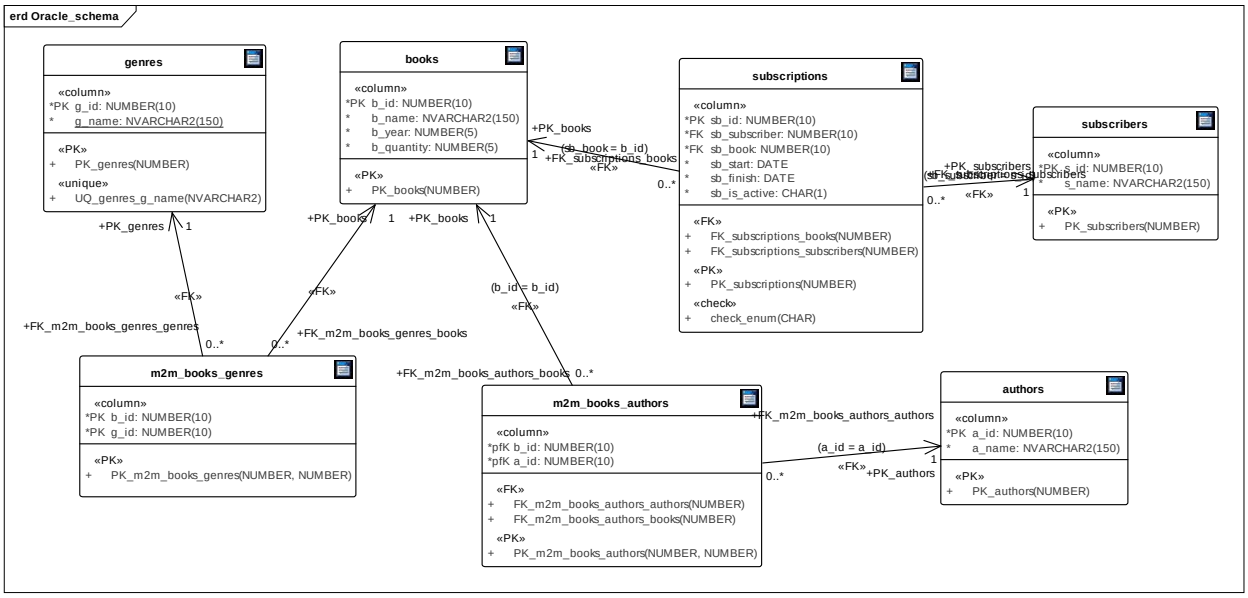


Рис. 1.f. Модель базы данных для Oracle в Sparx EA

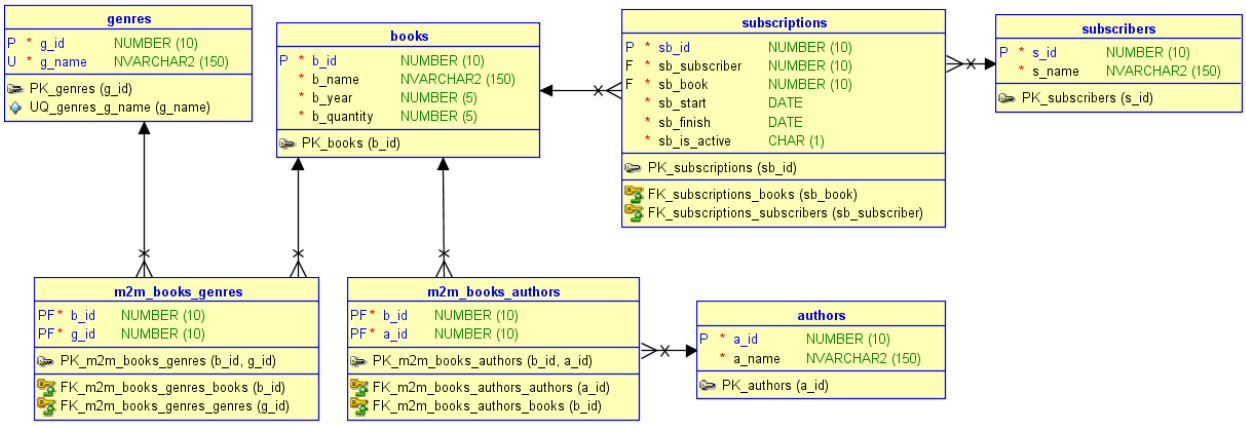


Рис. 1.g. Модель базы данных для Oracle в Oracle SQL Developer Data Modeler

1.5. ГЕНЕРАЦИЯ И НАПОЛНЕНИЕ БАЗЫ ДАННЫХ

На основе созданных в Sparx Enterprise Architect моделей получим DDL-скрипты 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.

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

Внесение двух подписчиков с именем «Сидоров С.С.» в таблице **subscribers** было сделано намеренно. Нам понадобится такой вариант полных тёзок для демонстрации нескольких типичных ошибок в запросах.

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 удобочитаемом виде для человека. Разбор этих запросов представлен в примере 1 (п. 2.1.1), а пока рассмотрим просто код.

MySQL

```
1    SELECT `b_name`,
2        `a_name`,
3        `g_name`
4    FROM   `books`
5        JOIN `m2m_books_authors` USING(`b_id`)
6        JOIN `authors` USING(`a_id`)
```

```
7           JOIN `m2m_books_genres` USING(`b_id`)
8           JOIN `genres` USING(`g_id`)
```

MS SQL

```
1   SELECT [b_name],
2          [a_name],
3          [g_name]
4   FROM   [books]
5          JOIN [m2m_books_authors]
6            ON [books].[b_id] = [m2m_books_authors].[b_id]
7          JOIN [authors]
8            ON [m2m_books_authors].[a_id] = [authors].[a_id]
9          JOIN [m2m_books_genres]
10           ON [books].[b_id] = [m2m_books_genres].[b_id]
11         JOIN [genres]
12           ON [m2m_books_genres].[g_id] = [genres].[g_id]
```

Oracle

```
1   SELECT "b_name",
2          "a_name",
3          "g_name"
4   FROM   "books"
5          JOIN "m2m_books_authors"
6            ON "books"."b_id" = "m2m_books_authors"."b_id"
7          JOIN "authors"
8            ON "m2m_books_authors"."a_id" = "authors"."a_id"
9          JOIN "m2m_books_genres"
10           ON "books"."b_id" = "m2m_books_genres"."b_id"
11         JOIN "genres"
12           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 удобочитаемую информацию для человека о том, кто и какие книги брал в библиотеке (подробнее об этом запросе сказано в примере 1 п. 2.1.1).

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 получим следующее.

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

b_name	s_id	s_name	sb_start	sb_finish
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) 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.

2. ЗАПРОСЫ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ

2.1. ВЫБОРКА ИЗ ОДНОЙ ТАБЛИЦЫ

Инструкция SELECT извлекает информацию из базы данных и возвращает её в виде таблицы результатов запроса.

В предложении SELECT указывается список столбцов, которые должны быть возвращены инструкцией SELECT. Возвращаемые столбцы могут содержать значения, извлекаемые из столбцов таблиц базы данных, или значения, вычисляемые во время выполнения запроса.

В предложении FROM указывается список таблиц, которые содержат элементы данных, извлекаемые запросом.

Предложение WHERE показывает, что в результаты запроса следует включать только некоторые строки. Для отбора строк, включаемых в результаты запроса, используется условие отбора.

Предложение GROUP BY позволяет создать итоговый запрос. Обычный запрос включает в результаты запроса по одной записи для каждой строки из таблицы. Итоговый запрос, напротив, вначале группирует строки базы данных по определённому признаку, а затем включает в результаты запроса одну итоговую строку для каждой группы.

Предложение HAVING показывает, что в результаты запроса следует включать только некоторые из групп, созданных с помощью предложения GROUP BY. В этом предложении, как и в предложении WHERE, для отбора включаемых групп используется условие отбора.

Предложение ORDER BY сортирует результаты запроса на основании данных, содержащихся в одном или нескольких столбцах. Если это предложение не указано, результаты запроса не будут отсортированы.

Предложения SELECT и FROM являются обязательными. Четыре остальных включаются в инструкцию только при необходимости.

Ниже приведены примеры использования инструкции SELECT для выборки данных из одной таблицы.

2.1.1. ПРИМЕР 1. ВЫБОРКА ВСЕХ ДАННЫХ

Задача 2.1.1.a: показать всю информацию о 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.

В данном случае достаточно 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: показать без повторений идентификаторы читателей, бравших в библиотеке книги.

Задача 2.1.2.b: 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.

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.

Типичной ошибкой является 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 (п. 2.1.10), а пока решим только что упомянутую задачу, чтобы показать разницу в работе запросов.

Решение 2.1.2.b.

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: показать, сколько всего разных книг зарегистрировано в библиотеке.

Ожидаемый результат 2.1.3.a.

total_books
7

Решение 2.1.3.a.

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:

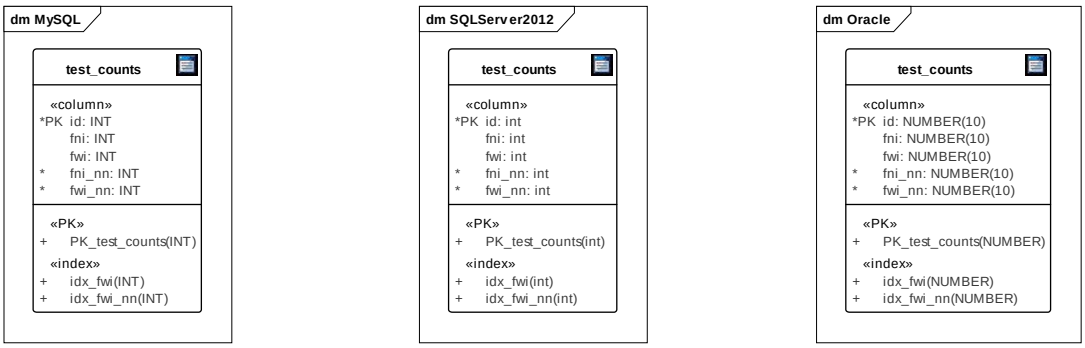
- **COUNT(*)** – классический вариант, используемый для подсчёта количества записей;
- **COUNT(1)** – альтернативная запись классического варианта;
- **COUNT(первичный_ключ)** – альтернативная запись классического варианта;
- **COUNT(поле)** – подсчёт записей, в указанном поле которых нет **NULL**-значений;
- **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 является единственным, в котором кэш СУБД не будет сбрасываться перед выполнением каждого следующего запроса.

В базе данных «Исследование» создадим таблицу **test_counts**, изображённую на рис. 2.а и 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**-значений («field, no index, no nulls») (число);
- **fwi_nn** – поле с индексом без **NULL**-значений («field, with index, no nulls») (число).



MySQL MS SQL Server Oracle

Рис. 2.а. 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]

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 рис. 2.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).

Теперь посмотрим на графики зависимости времени выполнения каждого из семи рассмотренных выше запросов 2.1.3.EXP.A от количества данных в таблице. На рис. 2.c, 2.d, 2.e приведены графики для MySQL, MS SQL Server и Oracle соответственно.

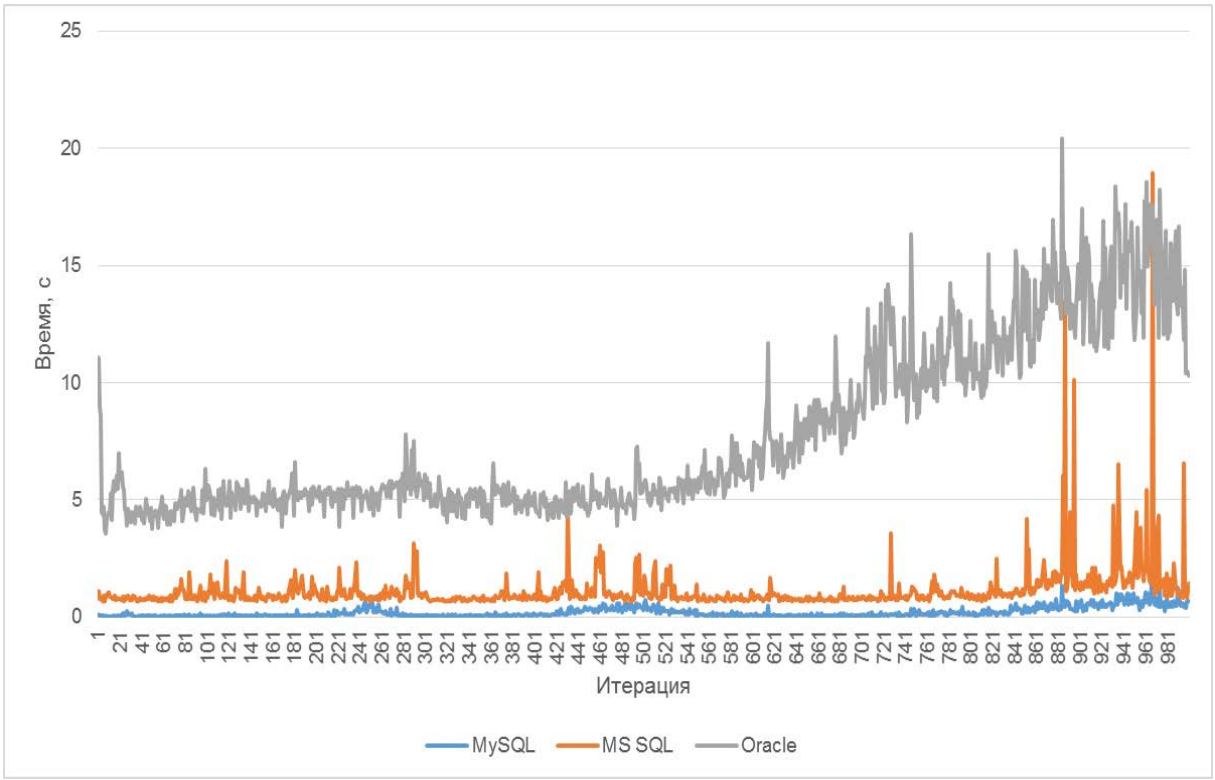


Рис. 2.b. Время, затраченное каждой СУБД на вставку тысячи записей

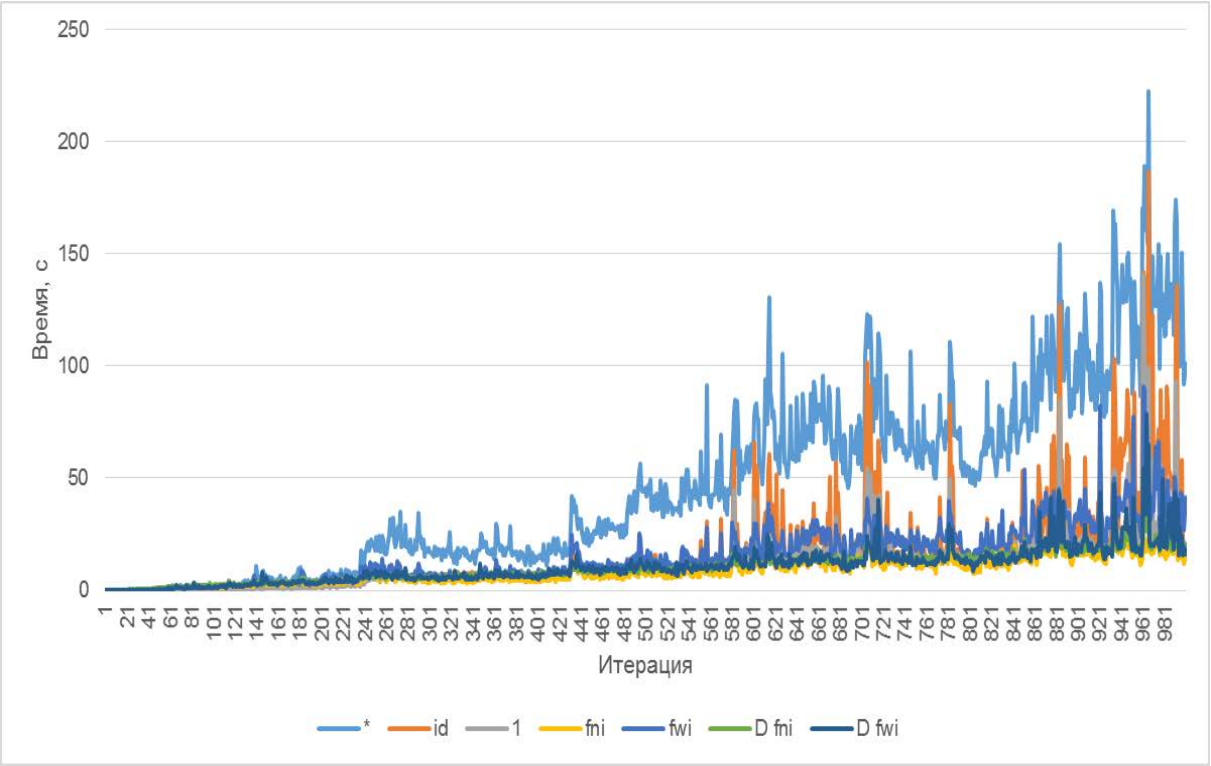


Рис. 2.c. Время, затраченное MySQL на выполнение разных COUNT

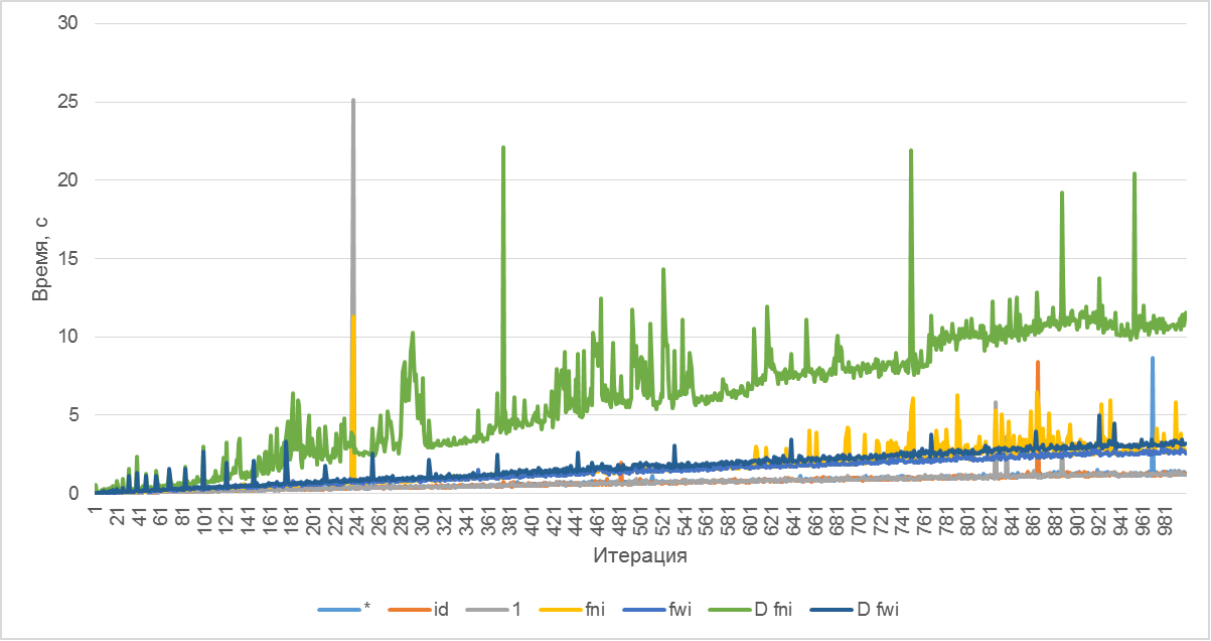


Рис. 2.d. Время, затраченное MS SQL Server на выполнение разных COUNT

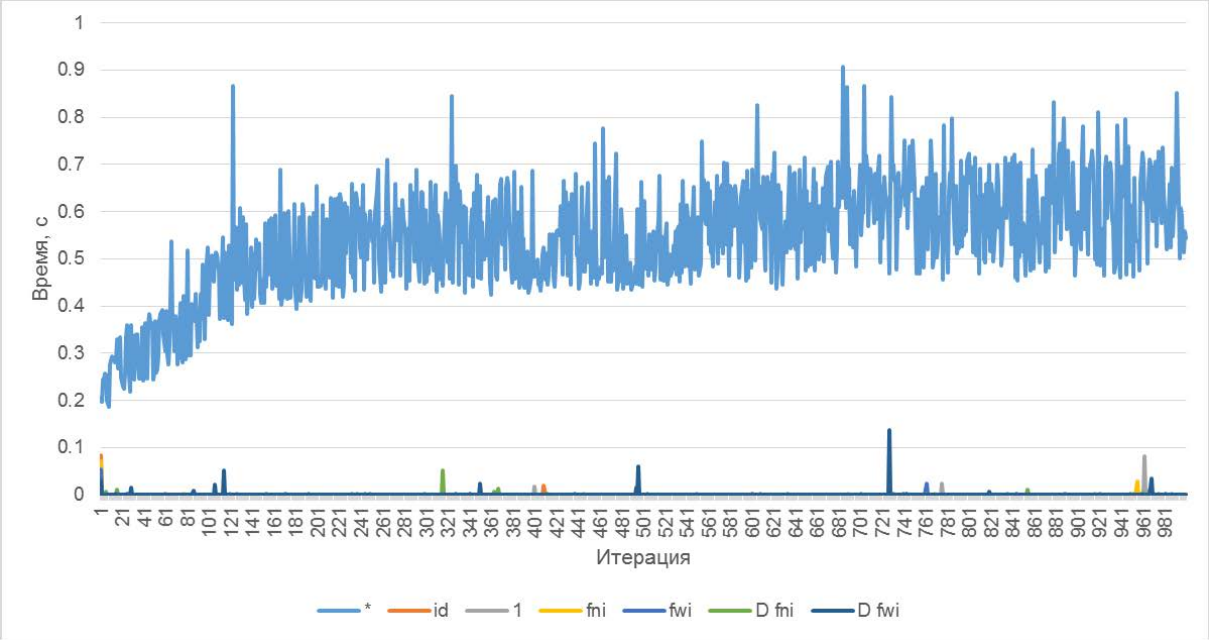


Рис. 2.е. Время, затраченное Oracle на выполнение разных COUNT

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

Запросы	**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.

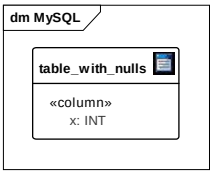
Для MySQL подтверждается бытующее мнение о том, что **COUNT(*)** работает медленнее всего, но неожиданно самым быстрым оказывается вариант с **COUNT(поле_без_индекса)**. Однако стоит отметить, что на меньшем объёме данных (до миллиона записей) ситуация оказывается совершенно иной – быстрее всего работает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

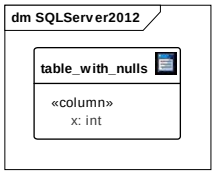
И теперь самое интересное – результаты Oracle: здесь снова подтвердились слухи о медленной работе **COUNT(*)**, но все остальные запросы показали потрясающий результат, очень высокую стабильность этого результата и полную независимость от объёма анализируемых данных (обратитесь к документации по Oracle, 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.

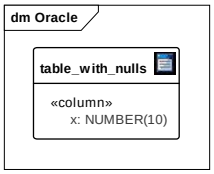
Исследование 2.1.3.EXP.B: рассмотрим, как функция **COUNT** реагирует на **NULL**-значения и на повторяющиеся значения в **DISTINCT**-режиме. Добавим в базу данных «Исследование» таблицу **table_with_nulls**. Вид этой таблицы представлен на рис. 2.f.



MySQL



MS SQL Server



Oracle

Рис. 2.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

```
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"
```

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

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

MS SQL Исследование 2.1.3.EXP.B

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

Oracle Исследование 2.1.3.EXP.B

```
1    -- Вариант 3: COUNT(поле)
2    SELECT COUNT("x") AS "total_records"
3    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

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

MS SQL Исследование 2.1.3.EXP.B

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

Oracle Исследование 2.1.3.EXP.B

```
1    -- Вариант 4: COUNT(DISTINCT поле)
2    SELECT COUNT(DISTINCT "x") AS "total_records"
3    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: показать, сколько всего экземпляров книг выдано читателям.

Задача 2.1.4.b: 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.

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 и 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 этого 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.

Здесь мы расширяем решение задачи 2.1.4.a, 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

```
1    SELECT COUNT(DISTINCT "sb_book") AS "in_use"
2    FROM  "subscriptions"
3    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: показать общее (сумму), минимальное, максимальное и среднее значения 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.

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

```
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`
```

MS SQL | 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(CAST([b_quantity] AS FLOAT)) AS [avg]
5    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 четвертую 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

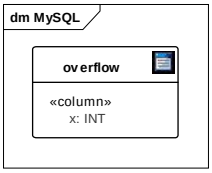
А должно быть:

sum	min	max	avg
33	1	12	4.7143

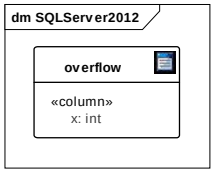
Стоит упомянуть ещё одну опасную ошибку: очень легко забыть, что при вычислении суммы и среднего значения (которое определяется как сумма, поделённая на количество значений) может произойти переполнение разрядной сетки.

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

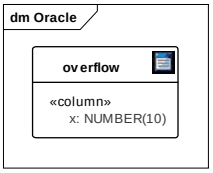
Создадим в БД «Исследование» таблицу **overflow** с одним числовым целочисленным полем (рис. 2.g).



MySQL



MS SQL Server



Oracle

Рис. 2.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**. 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

```
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`
```

MS SQL Исследование 2.1.5.EXP.B

```
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]
```

Oracle Исследование 2.1.5.EXP.B

```
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"
```

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

```
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
```

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 нуль.

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

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: показать все книги в библиотеке в порядке возрастания их года издания.

Задача 2.1.6.b: 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.

Для упорядочивания 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
```

Oracle Решение 2.1.6.a

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

Решение 2.1.6.b.

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

MySQL Решение 2.1.6.b

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

MS SQL Решение 2.1.6.b

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

Oracle Решение 2.1.6.b

```
1    SELECT "b_name",
2         "b_year"
3    FROM   "books"
4    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)

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

MS SQL Решение 2.1.6.b (альтернативный вариант)

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

Oracle Решение 2.1.6.b (альтернативный вариант)

```
1    SELECT "b_name",
2         "b_year"
3    FROM   "books"
4    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**. 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

```
1    SELECT `x`
2    FROM   `table_with_nulls`
3    ORDER  BY `x` DESC
```

MS SQL Исследование 2.1.6.EXP.A

```
1    SELECT [x]
2    FROM   [table_with_nulls]
3    ORDER  BY [x] DESC
```

Oracle Исследование 2.1.6.EXP.A

```
1    SELECT "x"
2    FROM   "table_with_nulls"
3    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

```
1    SELECT `x`
2    FROM   `table_with_nulls`
3    ORDER  BY `x` IS NULL DESC,
4              `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**-значением?», 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: показать книги, изданные в период 1990–2000 гг., представленные в библиотеке в количестве трёх и более экземпляров.

Задача 2.1.7.b: показать идентификаторы и даты выдачи книг за лето 2012 г.

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.

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**, границы 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

```
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
```

Решение 2.1.7.b.

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

Правильный вариант

MySQL Решение 2.1.7.b

```
1    SELECT `sb_id`,
2         `sb_start`
3    FROM   `subscriptions`
4    WHERE  `sb_start` >= '2012-06-01'
5         AND `sb_start` < '2012-09-01'
```

MS SQL Решение 2.1.7.b

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

Oracle Решение 2.1.7.b

```
1    SELECT "sb_id",
2         "sb_start"
3    FROM   "subscriptions"
4    WHERE  "sb_start" >= TO_DATE('2012-06-01', 'yyyy-mm-dd')
5         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 не 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 фаталь-

ным 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)

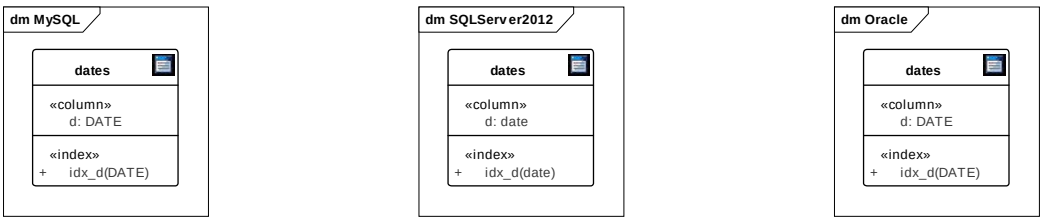
```
1    SELECT [sb_id],
2           [sb_start]
3    FROM   [subscriptions]
4    WHERE  YEAR([sb_start]) = 2012
5           AND MONTH([sb_start]) BETWEEN 6 AND 8
```

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 (рис. 2.h).



MySQL MS SQL Server Oracle

Рис. 2.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:

Вариант решения	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). 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: 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: показать просто одну любую книгу, количество экземпляров которой максимально (равно максимуму по всем книгам).

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

Задача 2.1.8.d: 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 книги «Евгений Онегин» (чтобы их тоже стало 12, как и у книги «Курс теоретической физики»), как результат становится другим (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 нуль рядов}	

Решение 2.1.8.b.

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 нуль 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.

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.

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: 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.

MS SQL Решение 2.1.8.c (фрагмент запроса)

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

Oracle Решение 2.1.8.c (фрагмент запроса)

```
1    SELECT "b_name",
2           "b_quantity",
3           RANK() OVER (ORDER BY "b_quantity" DESC) AS "rn"
4    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** и проверим.

Медианные значения времени после ста выполнений запросов таковы:

Имя функции	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 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.

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

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

MS SQL Решение 2.1.8.d

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

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
```

Oracle Решение 2.1.8.d

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

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 для того, 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:

Способ решения	MS SQL 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],
```

```
6         [sb_finish],
7         [sb_is_active]
8     FROM   [subscriptions]
9         JOIN (SELECT [sb_id],
10                    ROW_NUMBER()
11                     OVER (
12                      PARTITION BY [sb_subscriber]
13                      ORDER BY [sb_start] ASC) AS [visit]
14              FROM   [subscriptions]) AS [prepared]
15          ON [subscriptions].[sb_id] = [prepared].[sb_id]
16    WHERE  [visit] = 1
```

Oracle | Исследование 2.1.8.EXP.D

```
1     -- Вариант 1: решение на основе коррелирующих запросов
2     SELECT "sb_id",
3         "sb_subscriber",
4         "sb_book",
5         "sb_start",
6         "sb_finish",
7         "sb_is_active"
8     FROM   "subscriptions" "outer"
9     WHERE  "sb_id" = (SELECT DISTINCT FIRST_VALUE("inner"."sb_id")
10                            OVER (
11                              ORDER BY "inner"."sb_start" ASC)
12                FROM   "subscriptions" "inner"
13                WHERE  "outer"."sb_subscriber" = "inner"."sb_subscriber")

1     -- Вариант 2: решение на основе агрегирующих функций
2     SELECT "sb_id",
3         "subscriptions"."sb_subscriber",
4         "sb_book",
5         "sb_start",
6         "sb_finish",
7         "sb_is_active"
8     FROM   "subscriptions"
9     WHERE  "sb_id" IN (SELECT MIN("sb_id")
10                FROM   "subscriptions"
11                    JOIN (SELECT "sb_subscriber",
12                                 MIN("sb_start") AS "min_date"
13                          FROM   "subscriptions"
14                          GROUP  BY "sb_subscriber") "prepared"
15                      ON "subscriptions"."sb_subscriber" =
16                        "prepared"."sb_subscriber"
17                        AND "subscriptions"."sb_start" =
18                            "prepared"."min_date"
19                GROUP  BY "prepared"."sb_subscriber",
20                          "prepared"."min_date")

1     -- Вариант 3: решение на основе ранжирования
2     SELECT "subscriptions"."sb_id",
3         "sb_subscriber",
```

```
4         "sb_book",
5         "sb_start",
6         "sb_finish",
7         "sb_is_active"
8     FROM   "subscriptions"
9         JOIN (SELECT "sb_id",
10                    ROW_NUMBER()
11                      OVER (
12                       partition BY "sb_subscriber"
13                       ORDER BY "sb_start" ASC) AS "visit"
14              FROM   "subscriptions") "prepared"
15          ON "subscriptions"."sb_id" = "prepared"."sb_id"
16    WHERE  "visit" = 1
```

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

Способ решения	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 с 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: показать, сколько в среднем экземпляров книг сейчас на руках у каждого читателя.

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

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

Задача 2.1.9.d: 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.

См. пояснение ниже после решения 2.1.9.b.

MySQL 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`) 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.

MySQL Решение 2.1.9.b

```
1       SELECT AVG(`books_per_subscriber`) AS `avg_books`
2       FROM   (SELECT COUNT(DISTINCT `sb_book`) AS
3       `books_per_subscriber`
4            FROM   `subscriptions`
5            WHERE  `sb_is_active` = 'Y'
6            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
3    "books_per_subscriber"
4            FROM   "subscriptions"
5            WHERE  "sb_is_active" = 'Y'
6            GROUP  BY "sb_subscriber")
```

Суть решений 2.1.9.a и 2.1.9.b 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 и 2.1.9.b 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 отсутствии необходимости именовать подзапрос в Oracle. В остальном решения 2.1.9.a и 2.1.9.b 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.

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 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.

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 2–12 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

```
1   SELECT AVG(CAST([diff] AS FLOAT)) AS [avg_days]
2   FROM  (
3       SELECT DATEDIFF(day, [sb_start], [sb_finish]) AS [diff]
4       FROM   [subscriptions]
5       WHERE    ( [sb_finish] <= CONVERT(date, GETDATE())
6               AND [sb_is_active] = 'N' )
7           OR ( [sb_finish] > CONVERT(date, GETDATE())
8               AND [sb_is_active] = 'Y' )
9       UNION ALL
10      SELECT DATEDIFF(day, [sb_start], CONVERT(date, GETDATE())) AS
11  [diff]
12      FROM   [subscriptions]
13      WHERE    ( [sb_finish] <= CONVERT(date, GETDATE())
14              AND [sb_is_active] = 'Y' )
15          OR ( [sb_finish] > CONVERT(date, GETDATE())
16              AND [sb_is_active] = 'N' )
17      ) AS [diffs]
```

Oracle | Решение 2.1.9.d

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

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.

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-
2    01') / 12)
3    FROM dual;

1    SELECT FLOOR(MONTHS_BETWEEN(DATE '2012-12-31', DATE '2011-01-
2    01') / 12)
3    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 времени в среднем 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: показать по каждому году, сколько раз в этот год читатели брали книги.

Задача 2.1.10.b: показать по каждому году, сколько читателей в год воспользовались услугами библиотеки.

Задача 2.1.10.c: 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.

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, 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.

MySQL Решение 2.1.10.b

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

MS SQL Решение 2.1.10.b

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

Oracle Решение 2.1.10.b

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

Решение 2.1.10.b очень похоже на решение 2.1.10.a: 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.

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 удобочитаемые для человека 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

```
1     SELECT `sb_subscriber`,
2          COUNT(`sb_id`) AS `books_taken`
3     FROM   `subscriptions`
4     WHERE  `sb_is_active` = 'Y'
5     GROUP BY `sb_subscriber`
```

MS SQL | Исследование 2.1.10.EXP.A

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

Oracle | Исследование 2.1.10.EXP.A

```
1     SELECT "sb_subscriber",
2          COUNT("sb_id") AS "books_taken"
```

```
3    FROM   "subscriptions"
4    WHERE  "sb_is_active" = 'Y'
5    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., то 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 УДОБОЧИТАЕМЫХ ДЛЯ ЧЕЛОВЕКА ДАННЫХ

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

Задача 2.2.1.a: показать всю удобочитаемую для человека информацию о всех книгах (т. е. название, автора, жанр).

Задача 2.2.1.b: показать всю удобочитаемую для человека информацию о 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.

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")
```

```
6       JOIN "authors" USING("a_id")
7       JOIN "m2m_books_genres" USING("b_id")
8       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.

MySQL Решение 2.2.1.b

```
1     SELECT `b_name`,
2            `s_id`,
3            `s_name`,
4            `sb_start`,
5            `sb_finish`
6     FROM   `books`
7            JOIN `subscriptions`
8              ON `b_id` = `sb_book`
9            JOIN `subscribers`
10             ON `sb_subscriber` = `s_id`
```

MS SQL Решение 2.2.1.b

```
1     SELECT [b_name],
2            [s_id],
3            [s_name],
4            [sb_start],
5            [sb_finish]
6     FROM   [books]
7            JOIN [subscriptions]
8              ON [b_id] = [sb_book]
9            JOIN [subscribers]
10             ON [sb_subscriber] = [s_id]
```

Oracle Решение 2.2.1.b

```
1     SELECT "b_name",
2            "s_id",
3            "s_name",
4            "sb_start",
5            "sb_finish"
6     FROM   "books"
7            JOIN "subscriptions"
8              ON "b_id" = "sb_book"
9            JOIN "subscribers"
10             ON "sb_subscriber" = "s_id"
```

Логика решения 2.2.1.b полностью эквивалентна логике решения 2.2.1.a (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 задачи 2.2.1.a 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: показать все книги с их авторами (дублирование названий книг не допускается).

Задача 2.2.2.b: 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

book	author(s)
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, 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

Решение 2.2.2.a.

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 функцией, 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 запятая и пробел (,)):

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	→	Е.М. Лифшиц, Л.Д. Ландау
Основание и империя	А. Азимов	→	А. Азимов
Психология программирования	Д. Карнеги Б. Страуструп	→	Б. Страуструп, Д. Карнеги
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, 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 «, » (запятая и пробел) 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 байт, 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.

MySQL | Решение 2.2.2.b

```
1       SELECT `b_name`
2           AS `book`,
3           GROUP_CONCAT(DISTINCT `a_name` ORDER BY `a_name` SEPA-
4       RATOR ', ')
5           AS `author(s)`,
6           GROUP_CONCAT(DISTINCT `g_name` ORDER BY `g_name` SEPA-
7       RATOR ', ')
8           AS `genre(s)`
9       FROM   `books`
10          JOIN `m2m_books_authors` USING(`b_id`)
11          JOIN `authors` USING(`a_id`)
12          JOIN `m2m_books_genres` USING(`b_id`)
13          JOIN `genres` USING(`g_id`)
14      GROUP  BY `b_id`
15      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

```
1       WITH [prepared_data]
2          AS (SELECT [books].[b_id],
3                     [b_name],
4                     [a_name],
5                                      [g_name]
6              FROM   [books]
7                     JOIN [m2m_books_authors]
8                       ON [books].[b_id] = [m2m_books_authors].[b_id]
9                     JOIN [authors]
10                      ON [m2m_books_authors].[a_id] = [authors].[a_id]
11                    JOIN [m2m_books_genres]
12                      ON [books].[b_id] = [m2m_books_genres].[b_id]
13                    JOIN [genres]
14                      ON [m2m_books_genres].[g_id] = [genres].[g_id]
15         )
16      SELECT [outer].[b_name]
17           AS [book],
18           STUFF ((SELECT DISTINCT ', ' + [inner].[a_name]
19                   FROM   [prepared_data] AS [inner]
20                   WHERE  [outer].[b_id] = [inner].[b_id]
21                   ORDER  BY ', ' + [inner].[a_name]
22                   FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
23                   1, 2, '')
24           AS [author(s)],
25           STUFF ((SELECT DISTINCT ', ' + [inner].[g_name]
26                   FROM   [prepared_data] AS [inner]
27                   WHERE  [outer].[b_id] = [inner].[b_id]
28                   ORDER  BY ', ' + [inner].[g_name]
29                   FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
30                   1, 2, '')
31           AS [genre(s)]
32      FROM   [prepared_data] AS [outer]
33      GROUP  BY [outer].[b_id],
34                [outer].[b_name]
```

Данное решение для MS SQL Server тоже строится на основе решения 2.2.2.a 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, 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 следующие 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 будет выглядеть следующим образом:

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

book	author(s)	genre(s)
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: показать список читателей, когда-либо бравших в библиотеке книги (использовать **JOIN**).

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

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

Задача 2.2.3.d: 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.

MySQL Решение 2.2.3.a

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

MS SQL Решение 2.2.3.a

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

Oracle Решение 2.2.3.a

```
1    SELECT DISTINCT "s_id",
2                    "s_name"
3    FROM   "subscribers"
4           JOIN "subscriptions"
5             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 следующим (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.С. с разными идентификаторами). Сейчас же эта информация утеряна.

Решение 2.2.3.b.

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 получаются следующие данные:

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.

s_id	s_name
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.

Важно помнить, что 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.

MySQL Решение 2.2.3.c

```
1   SELECT `s_id`,
2          `s_name`
3   FROM   `subscribers`
4          LEFT JOIN `subscriptions`
5                 ON `s_id` = `sb_subscriber`
6   WHERE  `sb_subscriber` IS NULL
```

MS SQL Решение 2.2.3.c

```
1   SELECT [s_id],
2          [s_name]
3   FROM   [subscribers]
4          LEFT JOIN [subscriptions]
5                 ON [s_id] = [sb_subscriber]
6   WHERE  [sb_subscriber] IS NULL
```

Oracle Решение 2.2.3.c

```
1   SELECT "s_id",
2          "s_name"
3   FROM   "subscribers"
4          LEFT JOIN "subscriptions"
5                 ON "s_id" = "sb_subscriber"
6   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 П.П. попадёт в конечную выборку.

s_id	s_name
2	Петров П.П.

Решение 2.2.3.d.

MySQL Решение 2.2.3.d

```
1    SELECT `s_id`,
2           `s_name`
3    FROM   `subscribers`
4    WHERE  `s_id` NOT IN (SELECT DISTINCT `sb_subscriber`
5                          FROM   `subscriptions`)
```

MS SQL Решение 2.2.3.d

```
1    SELECT [s_id],
2           [s_name]
3    FROM   [subscribers]
4    WHERE  [s_id] NOT IN (SELECT DISTINCT [sb_subscriber]
5                          FROM   [subscriptions])
```

Oracle Решение 2.2.3.d

```
1    SELECT "s_id",
2           "s_name"
3    FROM   "subscribers"
4    WHERE  "s_id" NOT IN (SELECT DISTINCT "sb_subscriber"
5                          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

```
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`)
```

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, будут следующие:

Подзапрос	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: показать список читателей, у которых сейчас на руках нет книг (использовать **JOIN**).

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

В задачах 2.2.3.a–2.2.3.d примера 13 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.

MySQL 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     HAVING COUNT(IF(`sb_is_active` = 'Y', `sb_is_active`, NULL)) = 0
```

MS SQL 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
```

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.а 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 следует вывод, что нам осталось превратить в **NULL** любые значения поля **sb_is_active**, не равные **Y**, т. е. получить следующий 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.а 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	Эти записи не удовлетворяют условию **WHERE** и будут пропущены.
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	

s_id	s_name	sb_is_active
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

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.

MySQL Решение 2.2.4.b

```
1   SELECT `s_id`,
2          `s_name`
3   FROM   `subscribers`
4   WHERE  `s_id` NOT IN (SELECT DISTINCT `sb_subscriber`
5                         FROM   `subscriptions`
6                         WHERE  `sb_is_active` = 'Y')
```

MS SQL Решение 2.2.4.b

```
1   SELECT [s_id],
2          [s_name]
3   FROM   [subscribers]
4   WHERE  [s_id] NOT IN (SELECT DISTINCT [sb_subscriber]
5                         FROM   [subscriptions]
6                         WHERE  [sb_is_active] = 'Y')
```

Oracle Решение 2.2.4.b

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

```
1   SELECT `s_id`,
2          `s_name`
3   FROM   `subscribers`
4   WHERE  `s_id` IN (SELECT DISTINCT `sb_subscriber`
5                     FROM   `subscriptions`
6                     WHERE  `sb_is_active` = 'N')
```

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

```
1   SELECT [s_id],
```

```
2         [s_name]
3    FROM   [subscribers]
4    WHERE  [s_id] IN (SELECT DISTINCT [sb_subscriber]
5                 FROM   [subscriptions]
6                 WHERE  [sb_is_active] = 'N')
```

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

```
1    SELECT "s_id",
2         "s_name"
3    FROM   "subscribers"
4    WHERE  "s_id" IN (SELECT DISTINCT "sb_subscriber"
5                 FROM   "subscriptions"
6                 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: показать книги из жанров «Программирование» и/или «Классика» (без использования **JOIN**; идентификаторы жанров известны).

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

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

Задача 2.2.5.d: 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 (п. 2.2.8).

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.

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

```
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
```

MS SQL | 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
```

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.

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.

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.

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

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.

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

```
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
```

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 А.С. Пушкиным и/или А. Азимовым (индивидуально или в соавторстве – не важно).

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

2.2.6. ПРИМЕР 16. ЗАПРОСЫ НА ОБЪЕДИНЕНИЕ И ФУНКЦИЯ COUNT

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

Задача 2.2.6.b: 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.

MySQL Решение 2.2.6.a

```
1     SELECT `b_id`,
2          `b_name`,
3          COUNT(`a_id`) AS `authors_count`
4     FROM   `books`
```

```
5           JOIN `m2m_books_authors` USING (`b_id`)
6       GROUP  BY `b_id`
7       HAVING `authors_count` > 1
```

MS SQL | Решение 2.2.6.a

```
1   SELECT [books].[b_id],
2          [books].[b_name],
3          COUNT([m2m_books_authors].[a_id]) AS [authors_count]
4   FROM   [books]
5          JOIN [m2m_books_authors]
6            ON [books].[b_id] = [m2m_books_authors].[b_id]
7   GROUP  BY [books].[b_id],
8             [books].[b_name]
9   HAVING COUNT([m2m_books_authors].[a_id]) > 1
```

Oracle | Решение 2.2.6.a

```
1   SELECT "b_id",
2          "b_name",
3          COUNT("a_id") AS "authors_count"
4   FROM   "books"
5          JOIN "m2m_books_authors" USING ("b_id")
6   GROUP  BY "b_id", "b_name"
7   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.

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

```
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` AS `int`
6                                         WHERE  `int`.`sb_book` = `ext`.`sb_book`
7                                                AND `int`.`sb_is_active` = 'Y')
8                                        ) AS `real_count`
9       FROM   `books`
10          LEFT OUTER JOIN `subscriptions` AS `ext`
```

```
11                  ON `books`.`b_id` = `ext`.`sb_book`
12      ORDER  BY `real_count` DESC

1       -- Вариант 2: использование подзапроса как эмуляции общего таблично-
2       го
3       -- выражения и коррелирующего подзапроса
4       SELECT `b_id`,
5           `b_name`,
6           ( `b_quantity` - IFNULL((SELECT `taken`
7                       FROM   (SELECT `sb_book`       AS `b_id`,
8                                      COUNT(`sb_book`) AS `taken`
9                               FROM   `subscriptions`
10                              WHERE  `sb_is_active` = 'Y'
11                              GROUP  BY `sb_book`
12                              ) AS `books_taken`
13                      WHERE  `books`.`b_id` =
14                          `books_taken`.`b_id`), 0
15                  ) ) AS
16          `real_count`
17      FROM   `books`
18      ORDER  BY `real_count` DESC

1       -- Вариант 3: пошаговое применение нескольких подзапросов
2       SELECT `b_id`,
3           `b_name`,
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`
17      FROM   `books`
18      ORDER  BY `real_count` DESC

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 выполнения 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 находятся 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 таблично-
    го
    -- 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 нуль, затем из значения поля **b_quantity** вычитается значение поля **taken** и таким образом получаются конечные значения поля **real_count**.

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

```
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] AS [int]
6                                     WHERE  [int].[sb_book] = [ext].[sb_book]
7                                            AND [int].[sb_is_active] = 'Y') ) AS
8                   [real_count]
9   FROM   [books]
10      LEFT OUTER JOIN [subscriptions] AS [ext]
11                   ON [books].[b_id] = [ext].[sb_book]
12  ORDER  BY [real_count] DESC
```

Теперь следует рассмотреть **вариант 2**.

MS SQL | Решение 2.2.6.b

```
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 [b_id],
10         [b_name],
11         ( [b_quantity] - ISNULL((SELECT [taken]
12                                  FROM   [books_taken]
13                                  WHERE  [books].[b_id] =
14                                         [books_taken].[b_id]), 0
15                        ) ) AS
16         [real_count]
17  FROM   [books]
18  ORDER  BY [real_count] DESC

1   -- Вариант 3: пошаговое применение общего табличного выражения и под-
2   запроса
3   WITH [books_taken]
4      AS (SELECT [sb_book]
5          FROM   [subscriptions]
6          WHERE  [sb_is_active] = 'Y'),
7      [real_taken]
```

```
8       AS (SELECT [b_id],
9             COUNT([sb_book]) AS [taken]
10          FROM   [books]
11            LEFT OUTER JOIN [books_taken]
12                    ON [b_id] = [sb_book]
13          GROUP  BY [b_id])
14    SELECT [b_id],
15         [b_name],
16         ( [b_quantity] - (SELECT [taken]
17                     FROM   [real_taken]
18                     WHERE  [books].[b_id] = [real_taken].[b_id]) ) AS
19         [real_count]
20    FROM   [books]
21    ORDER  BY [real_count] DESC

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

b_id	taken
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 нули.

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

Первое общее табличное выражение (строки 3–6) возвращает следующие данные:

sb_book
3
5
1
1
4

Второе общее табличное выражение (строки 7–13) 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 16–19 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 нули.

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

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

-- 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 под-
запроса
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")
```

```
14  SELECT "b_id",
15      "b_name",
16      ( "b_quantity" - (SELECT "taken"
17                  FROM   "real_taken"
18                  WHERE  "books"."b_id" = "real_taken"."b_id") ) AS
19      "real_count"
20  FROM   "books"
21  ORDER  BY "real_count" DESC

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" - NVL("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
```

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:

Варианты	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: показать читаемость авторов, т. е. всех авторов и то количество раз, которое книги этих авторов были взяты читателями.

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

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

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

Задача 2.2.7.e: 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
1.85714285714285714285714285714285714286

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.

MySQL Решение 2.2.7.a

```
1       SELECT `a_id`,
2           `a_name`,
3           COUNT(`sb_book`) AS `books`
4       FROM   `authors`
5           JOIN `m2m_books_authors` USING ( `a_id` )
6           LEFT OUTER JOIN `subscriptions`
7                   ON `m2m_books_authors`.`b_id` = `sb_book`
8       GROUP  BY `a_id`
9       ORDER  BY `books` DESC
```

MS SQL Решение 2.2.7.a

```
1     SELECT [authors].[a_id],
2         [authors].[a_name],
3         COUNT([sb_book]) AS [books]
4     FROM   [authors]
5         JOIN [m2m_books_authors]
```

```
6           ON [authors].[a_id] = [m2m_books_authors].[a_id]
7         LEFT OUTER JOIN [subscriptions]
8                 ON [m2m_books_authors].[b_id] = [sb_book]
9     GROUP  BY [authors].[a_id],
10          [authors].[a_name]
11    ORDER  BY COUNT([sb_book]) DESC
```

Oracle | Решение 2.2.7.a

```
1     SELECT "a_id",
2          "a_name",
3          COUNT("sb_book") AS "books"
4     FROM   "authors"
5          JOIN "m2m_books_authors" USING ( "a_id" )
6          LEFT OUTER JOIN "subscriptions"
7                 ON "m2m_books_authors"."b_id" = "sb_book"
8     GROUP  BY "a_id",
9           "a_name"
10    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.

MySQL | Решение 2.2.7.b

```
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 подсчитать, сколько раз читатели бра-

ли 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])

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

a_id	a_name	Books
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 нему 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",
```

```
                "a_name")
SELECT "a_id",
       "a_name",
       "books"
FROM   "prepared_data"
WHERE  "rank" = 1
```

Исследование 2.2.7.EXP.A: сравним скорость работы решений этой задачи, выполнив все запросы 2.2.7.a на базе данных «Большая библиотека».

Медианные значения времени после ста выполнений каждого запроса:

Возможные варианты	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, где вариант с функцией **MAX** оказался быстрее варианта с **RANK**. И это совершенно нормально, т. к. эксперименты проводились на разных наборах данных и в контексте разных запросов. То есть на скорость выполнения запроса влияет далеко не только лишь использование той или иной функции, но и множество других параметров.

Решение 2.2.7.c.

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")
```

```
5               LEFT OUTER JOIN "subscriptions"
6                    ON "m2m_books_authors"."b_id" = "sb_book"
7           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.

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.

Пусть 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

```
1   SELECT AVG(`books`) AS `med_reading`
2   FROM   (SELECT @rownum := @rownum + 1 AS `RowNumber`,
3                  `books`
4           FROM   (SELECT 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`) AS `inner_data`,
10                 (SELECT @rownum := 0) AS `rownum_initialisation`
11          ORDER  BY `books`) AS `popularity`,
12         (SELECT COUNT(*) AS `RowCount`
13          FROM   (SELECT COUNT(`sb_book`) AS `books`
14                  FROM   `authors`
15                         JOIN `m2m_books_authors` USING (`a_id`)
16                         LEFT OUTER JOIN `subscriptions`
17                         ON `m2m_books_authors`.`b_id` = `sb_book`
18                  GROUP  BY `a_id`) AS `inner_data`) AS `total_rows`
19  WHERE  `RowNumber` IN ( FLOOR(( `RowCount` + 1 ) / 2),
20                          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 единицу 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 следующим условиям:

- Условие 1: **FLOOR((`RowCount` + 1) / 2)**.
- Условие 2: **FLOOR((`RowCount` + 2) / 2)**.

В нашем конкретном случае при 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 – четвертый. Значение 2 поля books из четвертого 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, медиану которого мы ищем, сводится к подготовке множества значений 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.

Для решения этой задачи необходимо:

- Определить по каждой книге количество её экземпляров, выданных на руки читателям.
- Вычесть полученное значение из количества экземпляров книги, зарегистрированных в библиотеке.
- Проверить, существуют ли книги, для которых результат такого вычитания оказался 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 нуль 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: 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: 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.

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. четвертая и пятая 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 далее 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)

```
1   SELECT `a_id`,
2          `a_name`,
3          `b_id`,
4          `b_name`,
5          COUNT(`g_id`) AS `genres_count`
6   FROM   `authors`
7          JOIN `m2m_books_authors` USING (`a_id`)
8          JOIN `m2m_books_genres` USING (`b_id`)
9          JOIN `books` USING (`b_id`)
10  GROUP  BY `a_id`,
11            `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

```
1   SELECT [a_id],
2          [a_name],
3          MAX([genres_count]) AS [genres_count]
```

```
4     FROM   (SELECT [authors].[a_id],
5                    [authors].[a_name],
6                    COUNT([m2m_books_genres].[g_id]) AS [genres_count]
7             FROM   [authors]
8                    JOIN [m2m_books_authors]
9                      ON [authors].[a_id] = [m2m_books_authors].[a_id]
10                   JOIN [m2m_books_genres]
11                     ON [m2m_books_authors].[b_id] = [m2m_books_genres].[b_id]
12            GROUP  BY [authors].[a_id],
13                      [a_name],
14                      [m2m_books_authors].[b_id]
15            HAVING COUNT([m2m_books_genres].[g_id]) > 1) AS [prepared_data]
16    GROUP  BY [a_id],
17              [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.

Как только что было подчёркнуто в разборе решения задачи 2.2.8.a, 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]) AS
9         [prepared_data]
10        JOIN [authors]
11          ON [prepared_data].[a_id] = [authors].[a_id]
12    GROUP  BY [prepared_data].[a_id],
13              [a_name]
14    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 и 2.2.8.b 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: показать читателя, первым взявшего в библиотеке книгу.

Задача 2.2.9.b: показать читателя (или читателей, если их окажется несколько), быстрее всего прочитавшего книгу (учитывать только случаи, когда книга возвращена).

Задача 2.2.9.c: показать, какую книгу (или книги, если их несколько) каждый читатель взял в первый день своей работы с библиотекой.

Задача 2.2.9.d: 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.

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).

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

```
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 TOP 1 [sb_subscriber]
5                 FROM   [subscriptions]
6                 ORDER  BY [sb_id] ASC)
```

Oracle | 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   (SELECT "sb_subscriber",
6                           ROW_NUMBER()
7                            OVER(
8                             ORDER BY "sb_id" ASC) AS "rn"
9                      FROM   "subscriptions")
10                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:

Варианты решения	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.

Так как 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

```
1   -- Вариант 1: использование подзапроса и сортировки
2   SELECT DISTINCT [s_id],
3                   [s_name],
4                   DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
5   FROM   [subscribers]
6          JOIN [subscriptions]
```

```
7           ON [s_id] = [sb_subscriber]
8     WHERE  [sb_is_active] = 'N'
9         AND DATEDIFF(day, [sb_start], [sb_finish]) =
10        (SELECT TOP 1 DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
11         FROM       [subscriptions]
12         WHERE      [sb_is_active] = 'N'
13         ORDER  BY    [days] ASC)

1     -- Вариант 2: использование общего табличного выражения и функции
2     MIN
3     WITH [prepared_data]
4        AS (SELECT DISTINCT [s_id],
5                            [s_name],
6                            DATEDIFF(day, [sb_start], [sb_finish]) AS [days]
7           FROM   [subscribers]
8                JOIN [subscriptions]
9                  ON [s_id] = [sb_subscriber]
10          WHERE  [sb_is_active] = 'N')
11    SELECT [s_id],
12        [s_name],
13        [days]
14    FROM   [prepared_data]
15    WHERE  [days] = (SELECT MIN([days])
16                     FROM   [prepared_data])

1     -- Вариант 3: использование общего табличного выражения и ранжирова-
2     ния
3     WITH [prepared_data]
4        AS (SELECT DISTINCT [s_id],
5                            [s_name],
6                            DATEDIFF(day, [sb_start], [sb_finish]) AS [days],
7                            RANK()
8                              OVER (
9                                  ORDER BY
10                                 DATEDIFF(day, [sb_start], [sb_finish]) ASC
11                             ) AS [rank]
12          FROM   [subscribers]
13               JOIN [subscriptions]
14                 ON [s_id] = [sb_subscriber]
15          WHERE  [sb_is_active] = 'N')
16    SELECT [s_id],
17        [s_name],
18        [days]
19    FROM   [prepared_data]
20    WHERE  [rank] = 1
```

В **варианте 2** общее табличное выражение (строки 3–10) 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 15, 16 определяет минимальное значение столбца **days**, которое затем используется как условие ограничения выборки в основной части запроса (строки 11–15).

В **варианте 3** общее табличное выражение в строках 3–15 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 16–20) 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
2   MIN
3   WITH "prepared_data"
4      AS (SELECT DISTINCT "s_id",
5                          "s_name",
6                          ( "sb_finish" - "sb_start" ) AS "days"
7           FROM   "subscribers"
8                  JOIN "subscriptions"
9                    ON "s_id" = "sb_subscriber"
10          WHERE  "sb_is_active" = 'N')
11  SELECT "s_id",
12         "s_name",
```

```
13        "days"
14    FROM   "prepared_data"
15    WHERE  "days" = (SELECT Min("days")
16                    FROM   "prepared_data")

1     -- Вариант 3: использование общего табличного выражения и ранжирова-
2     ния
3     WITH "prepared_data"
4        AS (SELECT DISTINCT "s_id",
5                            "s_name",
6                            ("sb_finish" - "sb_start")            AS "days",
7                            RANK()
8                            OVER (
9                            ORDER BY ("sb_finish" - "sb_start") ASC ) AS "rank"
10          FROM   "subscribers"
11               JOIN "subscriptions"
12                 ON "s_id" = "sb_subscriber"
13          WHERE  "sb_is_active" = 'N')
14    SELECT "s_id",
15           "s_name",
16           "days"
17    FROM   "prepared_data"
18    WHERE  "rank" = 1
```

Решение 2.2.9.c.

Здесь мы не будем делать допущений, подобных сделанным в решении задачи 2.2.9.a, 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

```
1        SELECT `s_id`,
2               `s_name`,
3               GROUP_CONCAT(`b_name` ORDER BY `b_name` SEPARATOR ', ')
4               AS `books_list`
5        FROM   (SELECT `s_id`,
6                       `s_name`,
7                       `b_name`
8                FROM   `subscribers`
9                       JOIN (SELECT `subscriptions`.`sb_subscriber`,
10                                   `subscriptions`.`sb_book`
11                           FROM   `subscriptions`
12                                  JOIN (SELECT `sb_subscriber`,
13                                               MIN(`sb_start`) AS `min_date`
14                                        FROM   `subscriptions`
15                                        GROUP  BY `sb_subscriber`)
```

```
16                              AS `first_visit`
17                            ON `subscriptions`.`sb_subscriber` =
18                              `first_visit`.`sb_subscriber`
19                              AND `subscriptions`.`sb_start` =
20                                `first_visit`.`min_date`)
21                    AS `books_list`
22                  ON `s_id` = `sb_subscriber`
23                JOIN `books`
24                  ON `sb_book` = `b_id`) AS `prepared_data`
25      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])
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.

Oracle | 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_id",
18                  "b_name"
19             FROM   "subscribers"
20                    JOIN "step_2"
21                      ON "s_id" = "sb_subscriber"
22                    JOIN "books"
23                      ON "sb_book" = "b_id")
24  SELECT "s_id",
25         "s_name",
26         UTL_RAW.CAST_TO_NVARCHAR2
27         (
28          LISTAGG
29          (
30           UTL_RAW.CAST_TO_RAW("b_name"),
31           UTL_RAW.CAST_TO_RAW(N', ')
32          )
33          WITHIN GROUP (ORDER BY "b_name")
34         )
35         AS "books_list"
```

```
36    FROM   "step_3"
37    GROUP  BY "s_id",
38           "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.

Данная задача является логическим продолжением задачи 2.2.9.c, только теперь нужно определить, какая книга из тех, что читатель взял в первый свой визит в библиотеку, была выдана 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 порядковый 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 порядковый 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 читателей 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"

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

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"
```

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:

Варианты	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.

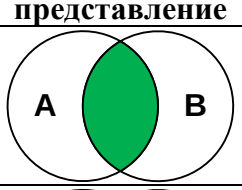
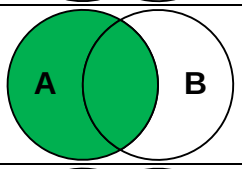
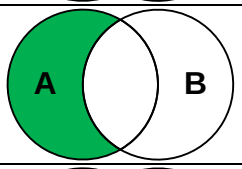
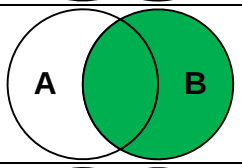
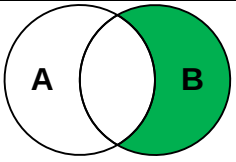
Задание 2.2.9.TSK.B: показать читателя (или читателей, если их окажется несколько), дольше всего держащего у себя книгу (учитывать только случаи, когда книга не возвращена).

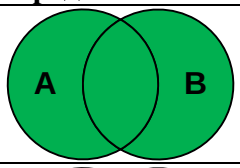
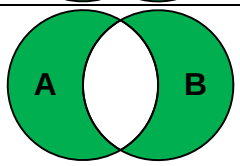
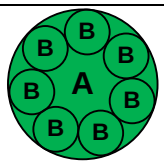
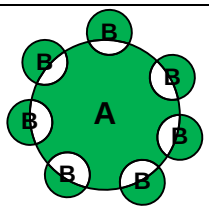
Задание 2.2.9.TSK.C: показать, какую книгу (или книги, если их несколько) каждый читатель взял в свой последний визит в библиотеку.

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

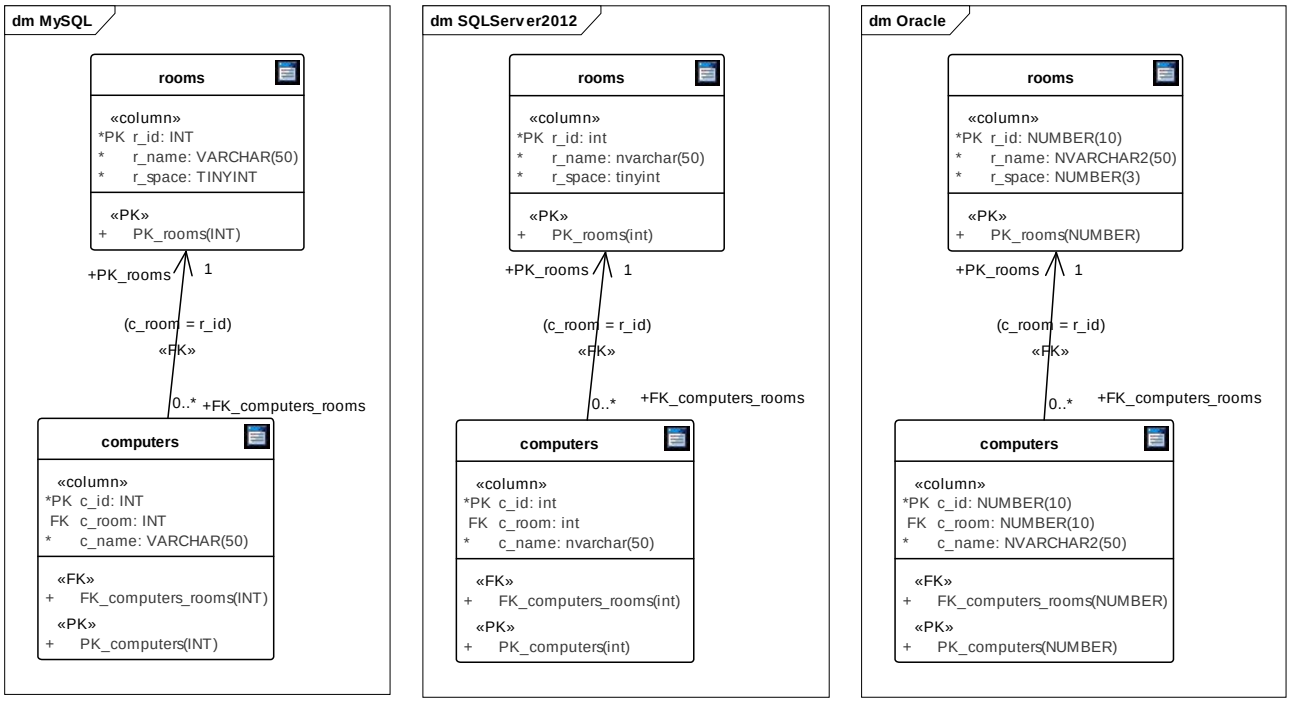
Для начала напомним, какие результаты можно получить, объединяя данные из двух таблиц с помощью классических вариантов **JOIN**.

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**
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**

Вид объединения	Графическое представление	Псевдокод запроса
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**

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 выполнение запроса) и потому являются необязательными: «просто **JOIN**» – всегда внутренний, а **LEFT JOIN**, **RIGHT JOIN** и **FULL JOIN** – всегда внешние.

Для демонстрации конкретных примеров создадим в БД «Исследование» таблицы **rooms** и **computers**, связанные связью «один ко многим» (см. рис. 2.i)



MySQL MS SQL Server Oracle

Рис. 2.i. 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

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: показать информацию о том, как компьютеры распределены по комнатам;

- 2.2.10.b: показать все комнаты с поставленными в них компьютерами;
- 2.2.10.c: показать все пустые комнаты;
- 2.2.10.d: показать все компьютеры с информацией о том, в каких комнатах они расположены;
- 2.2.10.e: показать все свободные компьютеры;
- 2.2.10.f: показать всю информацию о том, как компьютеры размещены по комнатам (включая пустые комнаты и свободные компьютеры);
- 2.2.10.g: показать информацию по всем пустым комнатам и свободным компьютерам;
- 2.2.10.h: показать возможные варианты расстановки компьютеров по комнатам (не учитывать вместимость комнат);
- 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).

Задачи на «неклассическое объединение»:

- 2.2.10.j: показать возможные варианты расстановки компьютеров по комнатам (учитывать вместимость комнат);
- 2.2.10.k: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (не учитывать вместимость комнат);
- 2.2.10.l: показать возможные варианты расстановки свободных компьютеров по пустым комнатам (учитывать вместимость комнат);
- 2.2.10.m: показать возможные варианты расстановки свободных компьютеров по комнатам (учитывать остаточную вместимость комнат);
- 2.2.10.n: показать расстановку компьютеров по непустым комнатам так, чтобы в выборку не попало больше компьютеров, чем может поместиться в комнату;
- 2.2.10.o: 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

```
1       SELECT `r_id`,
2              `r_name`,
3              `c_id`,
4              `c_room`,
5              `c_name`
6       FROM   `rooms`
7              JOIN `computers`
8                ON `r_id` = `c_room`
```

MS SQL Решение 2.2.10.a

```
1     SELECT [r_id],
2            [r_name],
3            [c_id],
4            [c_room],
5            [c_name]
6     FROM   [rooms]
7            JOIN [computers]
8              ON [r_id] = [c_room]
```

Oracle Решение 2.2.10.a

```
1     SELECT "r_id",
2            "r_name",
3            "c_id",
4            "c_room",
5            "c_name"
6     FROM   "rooms"
7            JOIN "computers"
8              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

```
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`
```

MS SQL Решение 2.2.10.b

```
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]
```

Oracle Решение 2.2.10.b

```
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"
```

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

```
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
```

MS SQL 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
```

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 комнатах они расположены.

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: 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

```
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]
```

Oracle 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"
```

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

```
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`
9   WHERE  `r_id` IS NULL
```

MS SQL Решение 2.2.10.e

```
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]
9   WHERE  [r_id] IS NULL
```

Oracle Решение 2.2.10.e

```
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"
9   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 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

c_id	c_room	c_name
7	NULL	Свободный компьютер B
8	NULL	Свободный компьютер C

MySQL Решение 2.2.10.e (упрощённый вариант)

```
1    SELECT `c_id`,
2           `c_room`,
3           `c_name`
4    FROM   `computers`
5    WHERE  `c_room` IS NULL
```

MS SQL Решение 2.2.10.e (упрощённый вариант)

```
1    SELECT [c_id],
2           [c_room],
3           [c_name]
4    FROM   [computers]
5    WHERE  [c_room] IS NULL
```

Oracle Решение 2.2.10.e (упрощённый вариант)

```
1    SELECT "c_id",
2           "c_room",
3           "c_name"
4    FROM   "computers"
5    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 и 2.2.10.d: 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:

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 решением этой задачи для MySQL является объединение решений задач 2.2.10.b и 2.2.10.d 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]
```

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

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.g: используем полное внешнее объединение с исключением. Эта задача является комбинацией задач 2.2.10.c и 2.2.10.e: 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

r_id	r_name	c_id	c_room	c_name
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
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]

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]
```

Oracle | 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"
```

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
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

```
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`
9       WHERE  `r_id` != `c_room`

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`
9       WHERE  `r_id` != `c_room`
```

MS SQL Решение 2.2.10.i

```
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]
9   WHERE  [r_id] != [c_room]

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]
9   WHERE  [r_id] != [c_room]
```

Oracle Решение 2.2.10.i

```
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"
9   WHERE  "r_id" != "c_room"

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"
9   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 мы закончим.

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

r_id	r_name	c_id	c_room	c_name
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

```
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
```

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 и 2.2.10.k: из первой мы возьмём логику **CROSS APPLY**, из второй – логику получения списка пустых комнат.

MySQL | 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 JOIN (SELECT `c_id`,
16                            `c_room`,
17                            `c_name`,
18                            @row_num := @row_num + 1 AS `position`
19                     FROM   `computers`,
20                            (SELECT @row_num := 0) AS `x`
21                     WHERE  `c_room` IS NULL
22                     ORDER  BY `c_name` ASC)
23                     AS `cross_apply_data`
24  WHERE  `position` <= `r_space`
25  ORDER  BY `r_id`,
26            `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 2.2.10.1

```
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

```
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))
       "empty_rooms"
       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"
ORDER  BY "r_id",
          "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

r_id	r_name	r_space	r_space_left	c_id	c_name
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 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 нулю 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])
9   FROM
10                             [computers] WHERE [c_room] =  [r_id]))
11                                    [c_id],
12                          ( [r_space] - (SELECT COUNT([c_room])
13                                        FROM   [computers]
```

```
14                              WHERE  [c_room] = [r_id]) )
15                    AS [r_space_left],
16                    [c_room],
17                    [c_name]
18               FROM   [computers]
19               WHERE  [c_room] IS NULL
20               ORDER  BY [c_name] ASC) AS [cross_apply_data]
21  ORDER BY [r_id],
22        [c_id]
```

Благодаря доступу к полям анализируемой записи из левой таблицы подзапросы в строках 8–10 и 12–15 могут сразу получить всю необходимую информацию, и в результате подзапрос в строках 8–20 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

```
1   SELECT "r_id",
2          "r_name",
3          "r_space",
4          ( "r_space" - NVL("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") "computers_in_room"
12                ON "r_id" = "c_room_inner"
13         CROSS JOIN (SELECT "c_id",
14                           "c_room",
15                           "c_name",
16                           ROW_NUMBER() OVER (ORDER BY "c_name" ASC)
17                           AS "position"
18                    FROM   "computers" WHERE "c_room" IS NULL
19                    ORDER  BY "c_name" ASC) "cross_apply_data"
20  WHERE "position" <= ("r_space" - NVL("r_used", 0))
21  ORDER  BY "r_id",
22            "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) 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, 2.2.10.l и 2.210.m 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

```
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                  ( CASE
12                      WHEN "c_room" IS NULL THEN 1
13                      ELSE ROW_NUMBER()
```

```
14                          OVER (
15                            PARTITION BY "c_room"
16                            ORDER BY "c_name" ASC)
17                    END ) AS "position"
18              FROM   "computers") "cross_apply_data"
19           ON "r_id" = "c_room"
20    WHERE  "position" <= "r_space"
21    ORDER  BY "r_id",
22           "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 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

```
1   SELECT [r_id],
2          [r_name],
3          [r_space],
4          [c_id],
5          [c_room],
6          [c_name]
7   FROM   [rooms]
8          OUTER 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]
```

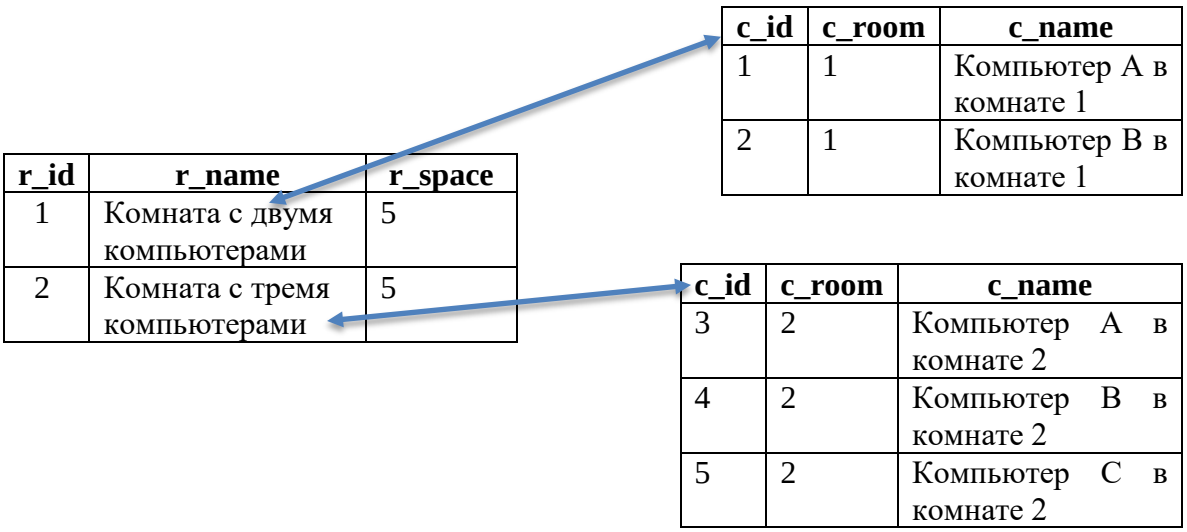
Oracle | Решение 2.2.10.n

```
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                           ( CASE
12                             WHEN "c_room" IS NULL THEN 1
13                             ELSE ROW_NUMBER()
14                                  OVER (
15                                   PARTITION BY "c_room"
16                                   ORDER BY "c_name" ASC)
17                           END ) AS "position"
18                    FROM   "computers") "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"
```

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 (**CROSS APPLY**):

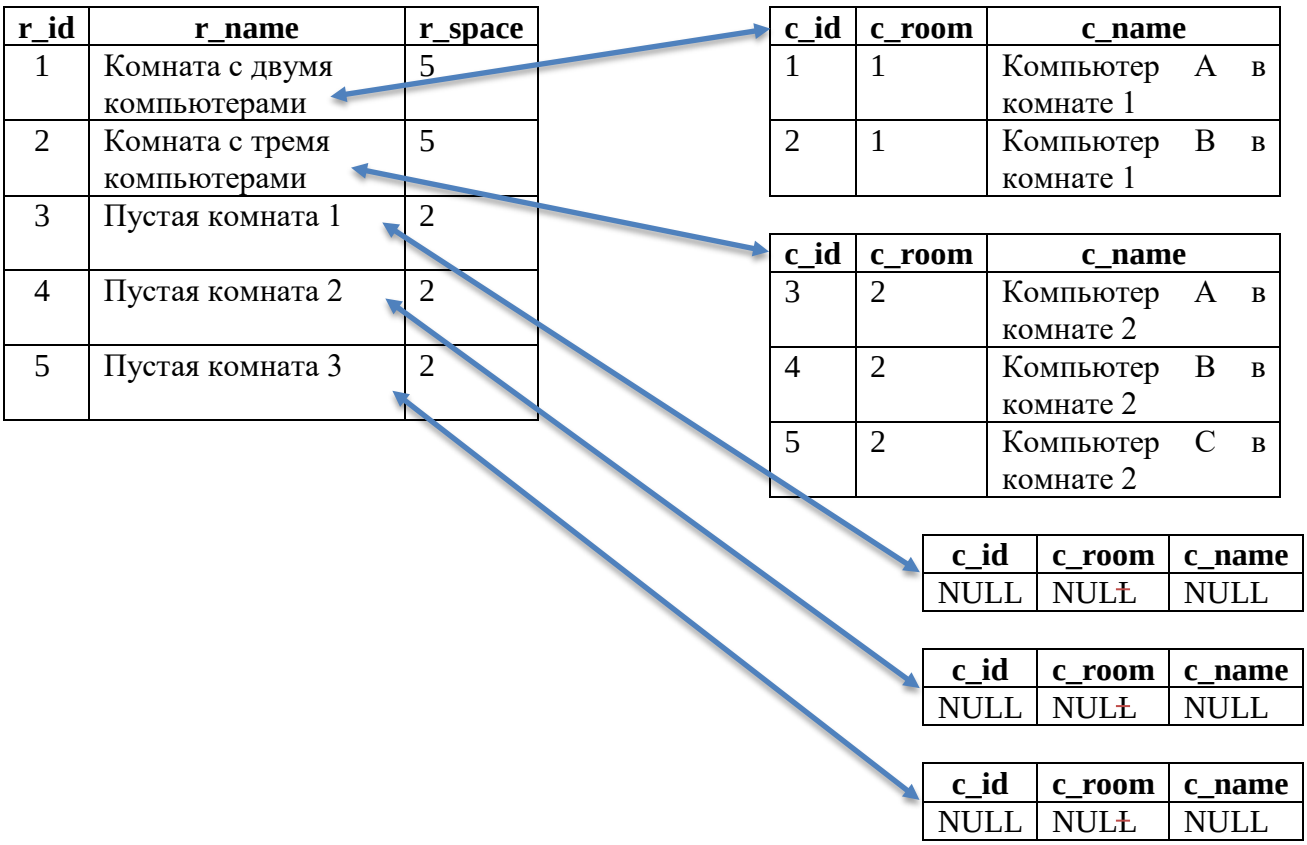


r_id	r_name	r_space
1	Комната с двумя компьютерами	5
2	Комната с тремя компьютерами	5

c_id	c_room	c_name
1	1	Компьютер A в комнате 1
2	1	Компьютер B в комнате 1

c_id	c_room	c_name
3	2	Компьютер A в комнате 2
4	2	Компьютер B в комнате 2
5	2	Компьютер C в комнате 2

В задаче 2.2.10.m (**OUTER APPLY**):



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

c_id	c_room	c_name
1	1	Компьютер A в комнате 1
2	1	Компьютер B в комнате 1

c_id	c_room	c_name
3	2	Компьютер A в комнате 2
4	2	Компьютер B в комнате 2
5	2	Компьютер C в комнате 2

c_id	c_room	c_name
NULL	NULL	NULL

c_id	c_room	c_name
NULL	NULL	NULL

c_id	c_room	c_name
NULL	NULL	NULL

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 г. книги в принципе может взять в библиотеке каждый из читателей.

Задание 2.2.10.TSK.H: показать информацию о том, какие изданные до 2010 г. 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 данном учебно-методическом пособии. При возникновении вопросов обратитесь к документации по соответствующей СУБД.

Задача 2.3.1.a: добавить в базу данных информацию о том, что читатель с идентификатором 4 взял 15 января 2016 г. в библиотеке книгу с идентификатором 3 и обещал вернуть её 30 января 2016 г.

Задача 2.3.1.b: добавить в базу данных информацию о том, что читатель с идентификатором 2 взял 25 января 2016 г. в библиотеке книги с идентификаторами 1, 3, 5 и обещал вернуть их 30 апреля 2016 г.

Ожидаемый результат 2.3.1.a: в таблице **subscriptions** должна появиться 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 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.

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, то 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.

Формально мы можем свести решение этой задачи к выполнению трёх отдельных запросов, аналогичных представленным в решении задачи 2.3.1.a, 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, работающие 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, 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.

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, неактуально для InnoDB:
2   ALTER TABLE `subscriptions` DISABLE KEYS; -- Отключение индексов.
3   -- Следующие две команды -- ОЧЕНЬ опасное решение!
4   SET foreign_key_checks = 0; -- Отключение проверки внешних ключей.
5   SET unique_checks = 0; -- Отключение уникальных индексов.
6   SET autocommit = 0; -- Отключение автоматической фиксации транзак-
7   ций.
8   -- Сам запрос на вставку:
9   INSERT INTO `subscriptions`
10          (`sb_subscriber`,
11           `sb_book`,
12           `sb_start`,
13           `sb_finish`,
14           `sb_is_active`)
15  VALUES     ( 2,
16               1,
17               '2016-01-25',
18               '2016-04-30',
19               'N' ),
20             ( 2,
21               3,
22               '2016-01-25',
23               '2016-04-30',
24               'N' ),
25             ( 2,
26               5,
27               '2016-01-25',
28               '2016-04-30',
29               'N' );
30  COMMIT; -- Фиксация транзакции.
31  SET autocommit = 1; -- Включение автоматической фиксации транзак-
32  ций.
33  SET unique_checks = 1; -- Включение уникальных индексов.
34  SET foreign_key_checks = 1; -- Включение проверки внешних ключей.
35  -- Актуально для MyISAM, не актуально для InnoDB:
36  ALTER TABLE `subscriptions` ENABLE KEYS; -- Включение индексов.
```

Ещё раз отметим, что в общем случае можно и нужно ограничиться запросом, показанным в строках 9–14. 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 соответствующими разделами документации, у этих операций могут быть крайне неожиданные и опасные по-

бочные 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 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.

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

```
1   -- ОЧЕНЬ опасное решение!
2   ALTER TABLE "subscriptions" MODIFY CONSTRAINT "PK_subscriptions"
3   DISABLE;
4   -- Сам запрос на вставку:
5   INSERT ALL
6    INTO   "subscriptions"
7          ("sb_subscriber",
8           "sb_book",
9           "sb_start",
10          "sb_finish",
11          "sb_is_active")
12   VALUES (2,
13          1,
14          TO_DATE('2016-01-25', 'YYYY-MM-DD'),
15          TO_DATE('2016-04-30', 'YYYY-MM-DD'),
16          'N')
17   INTO   "subscriptions"
18         ("sb_subscriber",
19          "sb_book",
20          "sb_start",
21          "sb_finish",
22          "sb_is_active")
23   VALUES (2,
24          3,
25          TO_DATE('2016-01-25', 'YYYY-MM-DD'),
26          TO_DATE('2016-04-30', 'YYYY-MM-DD'),
27          'N')
28   INTO   "subscriptions"
29         ("sb_subscriber",
30          "sb_book",
31          "sb_start",
32          "sb_finish",
33          "sb_is_active")
34   VALUES (2,
35          5,
36          TO_DATE('2016-01-25', 'YYYY-MM-DD'),
37          TO_DATE('2016-04-30', 'YYYY-MM-DD'),
38          'N')
39  SELECT 1 FROM "DUAL";
40  -- Включение индекса после его отключения:
41  ALTER TABLE "subscriptions" MODIFY CONSTRAINT "PK_subscriptions"
42  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 из вариантов поочерёдно.

Значения медиан времени, затраченного на вставку тысячи записей каждым из вариантов, таковы:

Варианты вставки	MySQL	MS SQL Server	Oracle
Цикл (1000 итераций) вставки одной строки	3.000	2.000	6.000
Один запрос на вставку 1000 строк	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 решений 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.

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 января 2016 г. 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: у выдачи с идентификатором 99 изменить дату возврата на текущую и отметить, что книга возвращена.

Задача 2.3.2.b: изменить ожидаемую дату возврата для всех книг, которые читатель с идентификатором 2 взял в библиотеке 25 января 2016 г., на «плюс два месяца» (т. е. читатель будет читать их на два месяца дольше, чем планировал).

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 следующий 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:

Значение текущей даты можно получить на стороне приложения и передать в СУБД, тогда решение будет предельно простым (вариант 1), но текущую дату можно получить и на стороне СУБД (вариант 2), что не сильно усложняет запрос.

MySQL Решение 2.3.2.a

```
1   -- Вариант 1: прямая подстановка текущей даты в запрос.
2   UPDATE `subscriptions`
3   SET    `sb_finish` = '2016-01-06',
4          `sb_is_active` = 'N'
5   WHERE  `sb_id` = 99

6   -- Вариант 2: получение текущей даты на стороне СУБД.
7   UPDATE `subscriptions`
8   SET    `sb_finish` = CURDATE(),
9          `sb_is_active` = 'N'
10  WHERE  `sb_id` = 99
```

MS SQL Решение 2.3.2.a

```
1   -- Вариант 1: прямая подстановка текущей даты в запрос.
2   UPDATE [subscriptions]
3   SET    [sb_finish] = CAST(N'2016-01-06' AS DATE),
4          [sb_is_active] = N'N'
5   WHERE  [sb_id] = 99

6   -- Вариант 2: получение текущей даты на стороне СУБД.
7   UPDATE [subscriptions]
8   SET    [sb_finish] = CONVERT(date, GETDATE()),
9          [sb_is_active] = N'N'
10  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     -- Вариант 2: получение текущей даты на стороне СУБД.
7     UPDATE "subscriptions"
8     SET    "sb_finish" = TRUNC(SYSDATE),
9            "sb_is_active" = 'N'
10    WHERE  "sb_id" = 99
```

Применение функции **TRUNC** в строке 9 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.

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 декабря 2011 г. надо добавить шесть месяцев: 2011-12-15 → 2011-**18**-15. Получился восемнадцатый 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

```
1       UPDATE `subscriptions`
2       SET    `sb_finish` = DATE_ADD(`sb_finish`, INTERVAL 2 MONTH)
3       WHERE  `sb_subscriber` = 2
4           AND `sb_start` = '2016-01-25';
```

MS SQL Решение 2.3.2.b

```
1    UPDATE [subscriptions]
2    SET    [sb_finish] = DATEADD(month, 2, [sb_finish])
3    WHERE  [sb_subscriber] = 2
4        AND [sb_start] = CONVERT(date, '2016-01-25');
```

Oracle Решение 2.3.2.b

```
1    UPDATE "subscriptions"
2    SET    "sb_finish" = ADD_MONTHS("sb_finish", 2)
3    WHERE  "sb_subscriber" = 2
4        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 января 2012 г., 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: удалить информацию о том, что читатель с идентификатором 4 взял 15 января 2016 г. в библиотеке книгу с идентификатором 3.

Задача 2.3.3.b: удалить информацию о 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.

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.

В решении задачи 2.1.7.b (см. п. 2.1.7.) 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 о 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 о всех выдачах книг, произведённых после 20-го числа любого месяца любого года.

2.3.4. ПРИМЕР 24. СЛИЯНИЕ ДАННЫХ

Задача 2.3.4.a: добавить в базу данных жанры «Философия», «Детектив», «Классика».

Задача 2.3.4.b: 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.

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.

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     -- Слияние данных:
4     MERGE [library].[dbo].[genres]     AS [destination]
5     USING [huge_library].[dbo].[genres] AS [source]
6     ON [destination].[g_id] = [source].[g_id]
7     WHEN MATCHED THEN
8      UPDATE SET [destination].[g_name] =
9            CONCAT([destination].[g_name], N' [OLD]')
```

```
10    WHEN NOT MATCHED THEN
11     INSERT ([g_id],
12           [g_name])
13     VALUES ([source].[g_id],
14           [source].[g_name]);
15    -- Запрет вставки явно переданных значений в IDENTITY-поле:
16    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 16).

Строки 4–6 запроса описывают таблицу-источник, таблицу-приёмник и проверяемое условие совпадения значений.

Строки 7–9 и 10–14 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
2     ключа:
3     ALTER TRIGGER "library"."TRG_genres_g_id" DISABLE;
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    -- Включение триггера, обеспечивающего автоинкремент первичного клю-
17    ча:
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 порядок проще создать пользователя с правами доступа к обеим базам данных (схемам)).

Строки 5–22 представленного ниже большого набора запросов аналогичны только что рассмотренному решению, а весь остальной код (строки 1–4, 23–32) нужен для того, чтобы обеспечить возможность одновременного доступа к данным в двух разных схемах.

Строки 2–4 отвечают за создание и открытие соединения со схемой-источником.

В строке 24 команда **COMMIT** нужна потому, что без неё при закрытии соединения мы рискуем потерять часть данных.

В строках 26 и 28–30 представлены два варианта закрытия соединения. Как правило, срабатывает первый вариант, но если не сработал – есть второй.

В строке 32 ранее созданное соединение уничтожается.

Oracle | Решение 2.3.4.b (работа от имени двух пользователей)

```
1   -- Создание соединения со схемой-источником:
2   CREATE DATABASE LINK "huge"
3   CONNECT TO "логин" IDENTIFIED BY "пароль"
4   USING 'localhost:1521/xe';
5   -- Отключение триггера, обеспечивающего автоинкремент первичного
6   ключа:
7   ALTER TRIGGER "TRG_genres_g_id" DISABLE;
8   -- Слияние данных:
9   MERGE INTO "genres" "destination"
10  USING "genres"@"huge" "source"
11  ON ("destination"."g_id" = "source"."g_id")
12  WHEN MATCHED THEN
13   UPDATE SET "destination"."g_name" =
14          CONCAT("destination"."g_name", N' [OLD]')
15  WHEN NOT MATCHED THEN
16   INSERT ("g_id",
17         "g_name")
18   VALUES ("source"."g_id",
19         "source"."g_name");
20  -- Включение триггера, обеспечивающего автоинкремент первичного клю-
21  ча:
22  ALTER TRIGGER "TRG_genres_g_id" ENABLE;
23  -- Явное подтверждение сохранения всех изменений:
24  COMMIT;
25  -- Закрытие соединения со схемой-источником:
26  ALTER SESSION CLOSE DATABASE LINK "huge";
27  -- Если не помогло ALTER SESSION CLOSE ... :
28  BEGIN
29  DBMS_SESSION.CLOSE_DATABASE_LINK('huge');
30  END;
31  -- Удаление соединения со схемой-источником:
32  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: добавить в базу данных информацию о том, что читатель с идентификатором 4 взял 1 февраля 2015 г. в библиотеке книги с идентификаторами 2 и 3 и планировал вернуть их не позднее 20 июля 2015 г.; если текущая дата меньше 20 июля 2015 г., отметить выдачи как невозвращённые, если больше – как возвращённые.

Задача 2.3.5.b: 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: 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.

MySQL Решение 2.3.5.a

```
1       INSERT INTO `subscriptions`
2               (
3                `sb_id`,
4                `sb_subscriber`,
5                `sb_book`,
6                `sb_start`,
7                `sb_finish`,
8                `sb_is_active`
9               )
10              VALUES
11              (
12               NULL,
13               4,
14               2,
15               '2015-02-01',
16               '2015-07-20',
17               CASE
18                WHEN CURDATE() < '2015-07-20'
19                THEN (SELECT 'N')
20                ELSE (SELECT 'Y')
21               END
22              ),
23              (
24               NULL,
25               4,
26               3,
27               '2015-02-01',
28               '2015-07-20',
29               CASE
30                WHEN CURDATE() < '2015-07-20'
31                THEN (SELECT 'N')
32                ELSE (SELECT 'Y')
33               END
34              )
```

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-
17  20')
18            THEN (SELECT N'N')
19            ELSE (SELECT N'Y')
20           END
21          ),
22          (
23           4,
24           3,
25           CONVERT(date, '2015-02-01'),
26           CONVERT(date, '2015-07-20'),
27           CASE
28            WHEN CONVERT(date, GETDATE()) < CONVERT(date, '2015-07-
29  20')
30            THEN (SELECT N'N')
31            ELSE (SELECT N'Y')
32           END
33          )
```

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 значением константы 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

```
1   INSERT ALL
2       INTO "subscriptions"
3           (
4            "sb_subscriber",
5            "sb_book",
6            "sb_start",
7            "sb_finish",
8            "sb_is_active"
9           )
10          VALUES
11          (
12           4,
13           2,
14           TO_DATE('2015-02-01', 'YYYY-MM-DD'),
15           TO_DATE('2015-07-20', 'YYYY-MM-DD'),
16           CASE
17            WHEN TRUNC(SYSDATE) < TO_DATE('2015-07-20', 'YYYY-MM-
18  DD')
19            THEN (SELECT 'N' FROM "DUAL")
20            ELSE (SELECT 'Y' FROM "DUAL")
21           END
22          )
23      INTO "subscriptions"
24          (
25           "sb_subscriber",
26           "sb_book",
27           "sb_start",
28           "sb_finish",
29           "sb_is_active"
30          )
31          VALUES
32          (
33           4,
34           3,
35           TO_DATE('2015-02-01', 'YYYY-MM-DD'),
36           TO_DATE('2015-07-20', 'YYYY-MM-DD'),
37           CASE
38            WHEN TRUNC(SYSDATE) < TO_DATE('2015-07-20', 'YYYY-MM-
39  DD')
40            THEN (SELECT 'N' FROM "DUAL")
41            ELSE (SELECT 'Y' FROM "DUAL")
42           END
```

```
43          )
44  SELECT 1 FROM "DUAL";
```

Решение 2.3.5.b.

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 нуль 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.

В отличие от предыдущих задач данного примера, здесь для каждой СУБД решения будут 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, т. к. 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)

```
1       UPDATE `subscribers`
2       SET    `s_name` =
3        (SELECT
4          CASE
5           WHEN `x` > 5 THEN
6            (SELECT CONCAT(`s_name`, ' [', `x`, '] [RED]'))
7           WHEN `x` >= 3 AND `x` <= 5 THEN
8            (SELECT CONCAT(`s_name`, ' [', `x`, '] [YELLOW]'))
9           ELSE (SELECT CONCAT(`s_name`, ' [', `x`, '] [GREEN]'))
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') AS `active_only`
18                        ON `s_id` = `sb_subscriber`
19                 GROUP  BY `s_id`) AS `prepared_data`
20         WHERE  `subscribers`.`s_id` = `prepared_data`.`s_id`)
```

MS SQL Решение 2.3.5.c (получено на основе решения для Oracle)

```
1       UPDATE [subscribers]
2       SET    [s_name] =
3        (SELECT
4          CASE
5           WHEN [x] > 5
6            THEN (SELECT CONCAT([s_name], ' [', [x], '] [RED]'))
7           WHEN [x] >= 3 AND [x] <= 5
8          THEN (SELECT CONCAT([s_name], ' [', [x], '] [YELLOW]'))
9           ELSE (SELECT CONCAT([s_name], ' [', [x], '] [GREEN]'))
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') AS [active_only]
18                       ON [s_id] = [sb_subscriber]
19                GROUP  BY [s_id]) AS [prepared_data]
20        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.

3. ИСПОЛЬЗОВАНИЕ ПРЕДСТАВЛЕНИЙ

3.1. 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.

Задача 3.1.1.a: упростить использование решения задачи 2.2.9.d (см. п. 2.2.9) так, чтобы для получения нужных данных не приходилось использовать представленные в решении объёмные запросы.

Задача 3.1.1.b: 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, 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.

Построим решение этой задачи для MySQL на основе следующего уже написанного ранее (см. решение задачи 2.2.9.d в п. 2.2.9) запроса:

MySQL — Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
1   SELECT `s_id`,
2          `s_name`,
3          `b_name`
4   FROM   (SELECT `subscriptions`.`sb_subscriber`,
5                  `sb_book`
6           FROM   `subscriptions`
7                  JOIN (SELECT `subscriptions`.`sb_subscriber`,
8                               MIN(`sb_id`) AS `min_sb_id`
9                        FROM   `subscriptions`
10                              JOIN (SELECT `sb_subscriber`,
11                                           MIN(`sb_start`) AS `min_sb_start`
12                                    FROM   `subscriptions`
13                                    GROUP  BY `sb_subscriber`)
14                                    AS `step_1`
15                                 ON `subscriptions`.`sb_subscriber` =
16                                    `step_1`.`sb_subscriber`
17                                    AND `subscriptions`.`sb_start` =
18                                        `step_1`.`min_sb_start`
19                       GROUP  BY `subscriptions`.`sb_subscriber`,
20                                 `min_sb_start`)
21                       AS `step_2`
22                    ON `subscriptions`.`sb_id` = `step_2`.`min_sb_id`)
23          AS `step_3`
24          JOIN `subscribers`
25           ON `sb_subscriber` = `s_id`
26          JOIN `books`
27           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 подзапросов 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

```
1   -- Замена первого подзапроса представлением:
2   CREATE OR REPLACE VIEW `first_book_step_1`
3   AS
4    SELECT `sb_subscriber`,
```

```
5           MIN(`sb_start`) AS `min_sb_start`
6       FROM   `subscriptions`
7       GROUP  BY `sb_subscriber`
8     -- Замена второго подзапроса представлением:
9     CREATE OR REPLACE VIEW `first_book_step_2`
10    AS
11      SELECT `subscriptions`.`sb_subscriber`,
12          MIN(`sb_id`) AS `min_sb_id`
13      FROM   `subscriptions`
14          JOIN `first_book_step_1`
15            ON `subscriptions`.`sb_subscriber` =
16               `first_book_step_1`.`sb_subscriber`
17               AND `subscriptions`.`sb_start` =
18                   `first_book_step_1`.`min_sb_start`
19      GROUP  BY `subscriptions`.`sb_subscriber`,
20            `min_sb_start`
22    -- Замена третьего подзапроса представлением:
23    CREATE OR REPLACE VIEW `first_book_step_3`
24    AS
25      SELECT `subscriptions`.`sb_subscriber`,
26          `sb_book`
27      FROM   `subscriptions`
28          JOIN `first_book_step_2`
29            ON `subscriptions`.`sb_id` = `first_book_step_2`.`min_sb_id`
30    -- Создание основного представления:
31    CREATE OR REPLACE VIEW `first_book`
32    AS
33      SELECT `s_id`,
34          `s_name`,
35          `b_name`
36      FROM   `subscribers`
37          JOIN `first_book_step_3`
38            ON `sb_subscriber` = `s_id`
39          JOIN `books`
40            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 задачи 2.2.9.d в п. 2.2.9).

MS SQL	Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
1     WITH [step_1]
2       AS (SELECT [sb_subscriber],
3               MIN([sb_start]) AS [min_sb_start]
4          FROM   [subscriptions]
5          GROUP  BY [sb_subscriber]),
6       [step_2]
```

```
7         AS (SELECT [subscriptions].[sb_subscriber],
8                   MIN([sb_id]) AS [min_sb_id]
9              FROM   [subscriptions]
10                  JOIN [step_1]
11                    ON [subscriptions].[sb_subscriber] =
12                       [step_1].[sb_subscriber]
13                   AND [subscriptions].[sb_start] =
14                       [step_1].[min_sb_start]
15             GROUP  BY [subscriptions].[sb_subscriber],
16                       [min_sb_start]),
17        [step_3]
18        AS (SELECT [subscriptions].[sb_subscriber],
19                   [sb_book]
20              FROM   [subscriptions]
21                  JOIN [step_2]
22                    ON [subscriptions].[sb_id] = [step_2].[min_sb_id])
23    SELECT [s_id],
24           [s_name],
25           [b_name]
26    FROM   [step_3]
27           JOIN [subscribers]
28             ON [sb_subscriber] = [s_id]
29           JOIN [books]
30             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

```
1     CREATE VIEW [first_book] AS
2     {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 задачи 2.2.9.d в п. 2.2.9):

Oracle | Решение 3.1.1.a (исходный запрос, который надо «спрятать» в представлении)

```
1     WITH "step_1"
2        AS (SELECT "sb_subscriber",
3                   MIN("sb_start") AS "min_sb_start"
4              FROM   "subscriptions"
5              GROUP  BY "sb_subscriber"),
6         "step_2"
7         AS (SELECT "subscriptions"."sb_subscriber",
8                   MIN("sb_id") AS "min_sb_id"
9              FROM   "subscriptions"
10                  JOIN "step_1"
11                    ON "subscriptions"."sb_subscriber" =
```

```
12                      "step_1"."sb_subscriber"
13                  AND "subscriptions"."sb_start" =
14                      "step_1"."min_sb_start"
15          GROUP  BY "subscriptions"."sb_subscriber",
16                    "min_sb_start"),
17       "step_3"
18       AS (SELECT "subscriptions"."sb_subscriber",
19                  "sb_book"
20           FROM   "subscriptions"
21                  JOIN "step_2"
22                    ON "subscriptions"."sb_id" = "step_2"."min_sb_id")
23   SELECT "s_id",
24          "s_name",
25          "b_name"
26   FROM   "step_3"
27          JOIN "subscribers"
28            ON "sb_subscriber" = "s_id"
29          JOIN "books"
30            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

```
1    CREATE OR REPLACE VIEW "first_book" AS
2    {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.

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

```
1    CREATE OR REPLACE VIEW `authors_with_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`
9      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 (см. п. 2.2.8) так, чтобы для получения нужных данных не приходилось использовать представленные в решении 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 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.

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: создать представление, ускоряющее получение информации о количестве экземпляров книг, имеющихся в библиотеке, выданных на руки, оставшихся в библиотеке.

Задача 3.1.2.b: создать представление, ускоряющее получение всей информации из таблицы **subscriptions** в удобочитаемом для человека 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.

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 будет изложено во второй части учебно-методического пособия). 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)

```
1   CREATE TABLE `books_statistics`
2   (
3       `total` INTEGER UNSIGNED NOT NULL,
4       `given` INTEGER UNSIGNED NOT NULL,
5       `rest`  INTEGER UNSIGNED NOT NULL
6   )
```

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 нуль 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)

```
1   -- Очистка таблицы:
2   TRUNCATE TABLE `books_statistics`;
3   -- Инициализация данных:
4   INSERT INTO `books_statistics`
5       (`total`,
6        `given`,
7        `rest`)
8   SELECT IFNULL(`total`, 0),
9       IFNULL(`given`, 0),
10      IFNULL(`total` - `given`, 0) AS `rest`
11  FROM  (SELECT (SELECT SUM(`b_quantity`)
12          FROM  `books`)        AS `total`,
13         (SELECT COUNT(`sb_book`)
```

```
14              FROM   `subscriptions`
15              WHERE  `sb_is_active` = 'Y') AS `given`)
16          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 вам предлагается доработать данное решение, устранив эту проблему.

MySQL | Решение 3.1.2.a (триггеры для таблицы books)

```
1       -- Удаление старых версий триггеров
2       -- (удобно в процессе разработки и отладки):
3       DROP TRIGGER `upd_bks_sts_on_books_ins`;
4       DROP TRIGGER `upd_bks_sts_on_books_del`;
5       DROP TRIGGER `upd_bks_sts_on_books_upd`;
6       -- Переключение разделителя завершения запроса,
7       -- т. к. сейчас запросом будет создание триггера,
8       -- внутри которого есть свои, классические запросы:
9       DELIMITER $$
10      -- Создание триггера, реагирующего на добавление книг:
11      CREATE TRIGGER `upd_bks_sts_on_books_ins`
12      BEFORE INSERT
13      ON `books`
14       FOR EACH ROW
15        BEGIN
16         UPDATE `books_statistics` SET
17              `total` = `total` + NEW.`b_quantity`,
18              `rest` = `total` - `given`;
19        END;
20      $$
21      -- Создание триггера, реагирующего на удаление книг:
22      CREATE TRIGGER `upd_bks_sts_on_books_del`
23      BEFORE DELETE
24      ON `books`
25       FOR EACH ROW
26        BEGIN
27         UPDATE `books_statistics` SET
28              `total` = `total` - OLD.`b_quantity`,
29              `given` = `given` - (SELECT COUNT(`sb_book`)
30                          FROM   `subscriptions`
31                          WHERE  `sb_book`=OLD.`b_id`
32                            AND `sb_is_active` = 'Y'),
33              `rest` = `total` - `given`;
```

```
34      END;
35      $$
36      -- Создание триггера, реагирующего на
37      -- изменение количества книг:
38      CREATE TRIGGER `upd_bks_sts_on_books_upd`
39      BEFORE UPDATE
40      ON `books`
41       FOR EACH ROW
42       BEGIN
43        UPDATE `books_statistics` SET
44            `total` = `total` - OLD.`b_quantity` + NEW.`b_quantity`,
45            `rest` = `total` - `given`;
46       END;
47      $$
48      -- Восстановление разделителя завершения запросов:
49      DELIMITER ;
```

Переключение разделителя (признака) завершения запроса (строки 6–9) 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 49).

Выражение **FOR EACH ROW** (строки 14, 25, 41) означает, 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 модификацию данных таблицы **books** сами запросы (строки 16–18, 27–33, 43–45) 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       -- 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 добавление выдачи книг:
```

```
11      CREATE TRIGGER `upd_bks_sts_on_subscriptions_ins`
12      BEFORE INSERT
13      ON `subscriptions`
14       FOR EACH ROW
15        BEGIN
16         SET @delta = 0;
17         IF (NEW.`sb_is_active` = 'Y') THEN
18          SET @delta = 1;
19         END IF;
20         UPDATE `books_statistics` SET
21               `rest` = `rest` - @delta,
22               `given` = `given` + @delta;
23        END;
24      $$
25      -- Создание триггера, реагирующего на удаление выдачи книг:
26      CREATE TRIGGER `upd_bks_sts_on_subscriptions_del`
27      BEFORE DELETE
28      ON `subscriptions`
29       FOR EACH ROW
30        BEGIN
31         SET @delta = 0;
32         IF (OLD.`sb_is_active` = 'Y') THEN
33          SET @delta = 1;
34         END IF;
35         UPDATE `books_statistics` SET
36               `rest` = `rest` + @delta,
37               `given` = `given` - @delta;
38        END;
39      $$
40      -- Создание триггера, реагирующего на обновление выдачи книг:
41      CREATE TRIGGER `upd_bks_sts_on_subscriptions_upd`
42      BEFORE UPDATE
43      ON `subscriptions`
44       FOR EACH ROW
45        BEGIN
46         SET @delta = 0;
47         IF ((NEW.`sb_is_active` = 'Y') AND (OLD.`sb_is_active` = 'N')) THEN
48          SET @delta = -1;
49         END IF;
50         IF ((NEW.`sb_is_active` = 'N') AND (OLD.`sb_is_active` = 'Y')) THEN
51          SET @delta = 1;
52         END IF;
53         UPDATE `books_statistics` SET
54               `rest` = `rest` + @delta,
55               `given` = `given` - @delta;
56         END;
57      $$
58      -- Восстановление разделителя завершения запросов:
59      DELIMITER ;
```

Триггеры на таблице **subscriptions** оказываются чуть более сложными, чем триггеры на таблице **books**: здесь приходится анализировать происходящее и предпринимать действия в зависимости от ситуации.

В триггере, реагирующем на добавление выдачи книг (строки 10–24), 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 16 переменную **@delta** значением «0». Если далее оказывается, что книга всё же выдана, мы изменяем значение этой переменной на «1» (строки 17–19). Таким образом, в запросе в строках 20–22 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 25–38).

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 47–49 запроса));
- книга не была на руках у читателя, но он её забрал (значение **`sb_is_active`** было равно **N**, но поменялось на **Y** (строки 50–52 запроса)).

Очевидно, что количество выданных и оставшихся в библиотеке книг изменяется только в двух последних случаях, которые и учтены в условиях, представленных в строках 47–52. Запрос в строках 53–56 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)
```

Версия	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 книг)

```
1       UPDATE `books`
2       SET    `b_quantity` = `b_quantity` + 5
3       WHERE  `b_quantity` = 10
```

Версия	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находятся 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
```

Версия	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
```

Версия	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
```

Версия	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)

```
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               2,
10              5,
11              '2016-01-10',
12              '2016-02-10',
13              'Y'),
14             (NULL,
15              2,
16              6,
17              '2016-01-10',
18              '2016-02-10',
19              'Y')
```

Версия	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)

```
1       DELETE FROM `subscriptions`
2       WHERE  `sb_id` = 42
```

Версия	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 невозвращённой книгой)

```
1       DELETE FROM `subscriptions`
2       WHERE  `sb_id` = 62
```

Версия	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)

```
1       DELETE FROM `books`
```

Версия	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, 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)

```
1    CREATE TABLE [books_statistics]
2    (
3            [total] INTEGER NOT NULL,
4            [given] INTEGER NOT NULL,
5            [rest]  INTEGER NOT NULL
6    )
```

Проинициализируем данные в созданной таблице.

MS SQL Решение 3.1.2.a (очистка таблицы и инициализация данных)

```
1    -- Очистка таблицы:
2    TRUNCATE TABLE [books_statistics];
3    -- Инициализация данных:
4    INSERT INTO [books_statistics]
5            ([total],
6             [given],
7             [rest])
8    SELECT ISNULL([total], 0) AS [total],
9        ISNULL([given], 0) AS [given],
10       ISNULL([total] - [given], 0) AS [rest]
11   FROM   (SELECT (SELECT SUM([b_quantity])
12                   FROM   [books])          AS [total],
13                  (SELECT COUNT([sb_book])
14                   FROM   [subscriptions]
15                   WHERE  [sb_is_active] = 'Y') AS [given])
16       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)

```
1    -- Удаление старых версий триггеров
2    -- (удобно в процессе разработки и отладки):
3    DROP TRIGGER [upd_bks_sts_on_books_ins];
4    DROP TRIGGER [upd_bks_sts_on_books_del];
```

```
5   DROP TRIGGER [upd_bks_sts_on_books_upd];
6   GO
7   -- Создание триггера, реагирующего на добавление книг:
8   CREATE TRIGGER [upd_bks_sts_on_books_ins]
9   ON [books]
10  AFTER INSERT
11  AS
12   UPDATE [books_statistics] SET
13        [total] = [total]  + (SELECT SUM([b_quantity])
14                    FROM   [inserted]);
15   UPDATE [books_statistics] SET
16        [rest] = [total] - [given];
17  GO
18  -- Создание триггера, реагирующего на удаление книг:
19  CREATE TRIGGER [upd_bks_sts_on_books_del]
20  ON [books]
21  AFTER DELETE
22  AS
23   UPDATE [books_statistics] SET
24        [total] = [total] - (SELECT SUM([b_quantity])
25                    FROM   [deleted]),
26        [given] = [given] - (SELECT COUNT([sb_book])
27                    FROM   [subscriptions]
28                    WHERE  [sb_book] IN (SELECT [b_id]
29                                FROM   [deleted])
30                      AND [sb_is_active] = 'Y');
31   UPDATE [books_statistics] SET
32        [rest] = [total] - [given];
33  GO
34  -- Создание триггера, реагирующего на
35  -- изменение количества книг:
36  CREATE TRIGGER [upd_bks_sts_on_books_upd]
37  ON [books]
38  AFTER UPDATE
39  AS
40   UPDATE [books_statistics] SET
42        [total] = [total] - (SELECT SUM([b_quantity])
42                    FROM   [deleted]) + (SELECT SUM([b_quantity])
43                                FROM   [inserted]);
44   UPDATE [books_statistics] SET
45        [rest] = [total] - [given];
46  GO
```

Основных отличий 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 модификацию данных в таблице **books** идентично поведению соответствующих триггеров в MySQL. А в триггерах на модификацию данных таблицы **subscriptions** есть существенные отличия.

MS SQL | 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   GO
7   -- Создание триггера, реагирующего на добавление выдачи книг:
8   CREATE TRIGGER [upd_bks_sts_on_subscriptions_ins]
9   ON [subscriptions]
10  AFTER INSERT
11  AS
12   DECLARE @delta INT = (SELECT COUNT(*)
13                         FROM   [inserted]
14                         WHERE  [sb_is_active] = 'Y');
15   UPDATE [books_statistics] SET
16        [rest] = [rest] - @delta,
17        [given] = [given] + @delta;
18  GO
19  -- Создание триггера, реагирующего на удаление выдачи книг:
20  CREATE TRIGGER [upd_bks_sts_on_subscriptions_del]
21  ON [subscriptions]
22  AFTER DELETE
23  AS
24   DECLARE @delta INT = (SELECT COUNT(*)
25                         FROM   [deleted]
26                         WHERE  [sb_is_active] = 'Y');
27   UPDATE [books_statistics] SET
28        [rest] = [rest] + @delta,
29        [given] = [given] - @delta;
30  GO
31  -- Создание триггера, реагирующего на обновление выдачи книг:
32  CREATE TRIGGER [upd_bks_sts_on_subscriptions_upd]
33  ON [subscriptions]
34  AFTER UPDATE
35  AS
36   DECLARE @taken INT = (
37    SELECT COUNT(*)
38    FROM   [inserted]
39         JOIN [deleted]
40           ON [inserted].[sb_id] = [deleted].[sb_id]
41    WHERE  [inserted].[sb_is_active] = 'Y'
42      AND [deleted].[sb_is_active] = 'N');
```

```
43    DECLARE @returned INT = (
44     SELECT COUNT(*)
45     FROM   [inserted]
46           JOIN [deleted]
47             ON [inserted].[sb_id] = [deleted].[sb_id]
48     WHERE  [inserted].[sb_is_active] = 'N'
49       AND [deleted].[sb_is_active] = 'Y');
50    DECLARE @delta INT = @taken - @returned;
51    UPDATE [books_statistics] SET
52          [rest] = [rest] - @delta,
53          [given] = [given] + @delta;
54    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 12–14 и 24–26 значение переменной **@delta** определяется как количество записей, удовлетворяющих условию работы триггера.

В строках 36–50 этот подход ещё больше усложняется: мы должны определить количество выданных (строки 36–42) и возвращённых (строки 43–49) книг, а затем в строке 50 мы можем определить разность полученных чисел и использовать её значение (строки 51–53) 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)
```

Версия	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
```

Версия	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находятся 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
```

Версия	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)

```
1     UPDATE [subscriptions]
2     SET    [sb_is_active] = 'N'
3     WHERE  [sb_id] = 3
```

Версия	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)

```
1     UPDATE [subscriptions]
2     SET    [sb_is_active] = 'Y'
3     WHERE  [sb_id] = 3
```

Версия	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)

```
1     INSERT INTO [subscriptions]
2            ([sb_subscriber],
3             [sb_book],
4             [sb_start],
5             [sb_finish],
6             [sb_is_active])
7     VALUES     (2,
```

```
8               5,
9               CAST(N'2016-01-10' AS DATE),
10              CAST(N'2016-02-10' AS DATE),
11              'Y'),
12             (2,
13              6,
14              CAST(N'2016-01-10' AS DATE),
15              CAST(N'2016-02-10' AS DATE),
16              'Y')
```

Версия	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)

```
1    DELETE FROM [subscriptions]
2    WHERE  [sb_id] = 42
```

Версия	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 невозвращённой книгой)

```
1    DELETE FROM [subscriptions]
2    WHERE  [sb_id] = 62
```

Версия	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)

```
1    DELETE FROM [books]
```

Версия	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

```
1   -- Удаление старой версии материализованного представления
2   -- (удобно при разработке и отладке):
3   DROP MATERIALIZED VIEW "books_statistics";
4   -- Создание материализованного представления:
5   CREATE MATERIALIZED VIEW "books_statistics"
6   BUILD IMMEDIATE
7   REFRESH FORCE
8   START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)
9   AS
10    SELECT "total",
11         "given",
12         "total" - "given" AS "rest"
13    FROM   (SELECT SUM("b_quantity") AS "total"
14          FROM   "books")
15         JOIN (SELECT COUNT("sb_book") AS "given"
16               FROM   "subscriptions"
17               WHERE  "sb_is_active" = 'Y')
18          ON 1 = 1
```

Выражение **BUILD IMMEDIATE** в строке 6 предписывает СУБД немедленно инициализировать материализованное представление данными.

Выражения в строках 7, 8 описывают способ обновления данных в материализованном представлении (при опции **REFRESH FORCE** СУБД сама выбирает оптимальный способ из доступных – либо быстрое обновление, либо полное) и периодичность этой операции (опция **START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)** указывает начать операцию немедленно и повторять каждую 1/1440 часть суток, т. е. каждую минуту).

Запрос в строках 10–18 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 13, 14 и вычисляющий количество выданных читателям книг – строки 15–17), 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 13, 14 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 10–12.

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.
```

Версия	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 книг)

```
1   UPDATE "books"
2   SET    "b_quantity" = "b_quantity" + 5
3   WHERE  "b_quantity" = 10;
4   COMMIT; -- И подождать минуту.
```

Версия	total	given	rest
Было	48	5	43
Стало	53	5	48

Удалим книгу, оба экземпляра которой сейчас находятся 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.
```

Версия	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.
```

Версия	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.
```

Версия	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)

```
1     INSERT ALL
2      INTO "subscriptions" ("sb_subscriber",
3                            "sb_book",
4                            "sb_start",
5                            "sb_finish",
6                            "sb_is_active")
7                    VALUES (2,
8                            5,
9                            TO_DATE('2016-01-10', 'YYYY-MM-DD'),
10                           TO_DATE('2016-02-10', 'YYYY-MM-DD'),
11                           'Y')
12     INTO "subscriptions" ("sb_subscriber",
13                           "sb_book",
14                           "sb_start",
15                           "sb_finish",
16                           "sb_is_active")
```

```
17              VALUES (2,
18                      6,
19                      TO_DATE('2016-01-10', 'YYYY-MM-DD'),
20                      TO_DATE('2016-02-10', 'YYYY-MM-DD'),
21                      'Y')
22      SELECT 1 FROM "DUAL";
23      COMMIT; -- И подождать минуту.
```

Версия	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)

```
1       DELETE FROM "subscriptions"
2       WHERE "sb_id" = 42;
3       COMMIT; -- И подождать минуту.
```

Версия	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 невозвращённой книгой)

```
1       DELETE FROM "subscriptions"
2       WHERE "sb_id" = 62;
3       COMMIT; -- И подождать минуту.
```

Версия	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.
```

Версия	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.

Если вы пропустили решение задачи 3.1.2.a, настоятельно рекомендуется ознакомиться с ним перед тем, как продолжить чтение, т. к. многие неочевидные моменты, которые встретятся в данном решении, были рассмотрены ранее.

По сравнению с предыдущей задачей 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, 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)

```
1   -- Очистка таблицы:
2   TRUNCATE TABLE `subscriptions_ready`;
3   -- Инициализация данных:
4   INSERT INTO `subscriptions_ready`
5       (`sb_id`,
6        `sb_subscriber`,
7        `sb_book`,
8        `sb_start`,
9        `sb_finish`,
10       `sb_is_active`)
11  SELECT `sb_id`,
12      `s_name` AS `sb_subscriber`,
13      `b_name` AS `sb_book`,
14      `sb_start`,
15      `sb_finish`,
16      `sb_is_active`
17  FROM   `books`
18      JOIN `subscriptions`
```

```
19              ON `b_id` = `sb_book`
20            JOIN `subscribers`
21              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 триггеры на модификации данных 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, т. к. 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 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       -- Переключение разделителя завершения запроса,
6       -- т. к. сейчас запросом будет создание триггера,
7       -- внутри которого есть свои классические запросы:
8       DELIMITER $$
9       -- Создание триггера, реагирующего на удаление книг:
10      CREATE TRIGGER `upd_sbs_rdy_on_books_del`
11      AFTER DELETE
12      ON `books`
13       FOR EACH ROW
14        BEGIN
15         DELETE FROM `subscriptions_ready`;
```

```
16        INSERT INTO `subscriptions_ready`
17                (`sb_id`,
18                 `sb_subscriber`,
19                 `sb_book`,
20                 `sb_start`,
21                 `sb_finish`,
22                 `sb_is_active`)
23             SELECT `sb_id`,
24                 `s_name`,
25                 `b_name`,
26                 `sb_start`,
27                 `sb_finish`,
28                 `sb_is_active`
29             FROM   `books`
30                 JOIN `subscriptions`
31                  ON `b_id` = `sb_book`
32                 JOIN `subscribers`
33                  ON `sb_subscriber` = `s_id`;
34       END;
35      $$
38      -- Создание триггера, реагирующего на
39      -- изменение данных о книгах:
40      CREATE TRIGGER `upd_sbs_rdy_on_books_upd`
41      AFTER UPDATE
42      ON `books`
43       FOR EACH ROW
44        BEGIN
45        IF (OLD.`b_name` != NEW.`b_name`)
46         THEN
47          DELETE FROM `subscriptions_ready`;
48          INSERT INTO `subscriptions_ready`
49                  (`sb_id`,
50                   `sb_subscriber`,
51                   `sb_book`,
52                   `sb_start`,
53                   `sb_finish`,
54                   `sb_is_active`)
55              SELECT `sb_id`,
56                   `s_name`,
57                   `b_name`,
58                   `sb_start`,
59                   `sb_finish`,
60                   `sb_is_active`
61              FROM   `books`
62                   JOIN `subscriptions`
63                    ON `b_id` = `sb_book`
64                   JOIN `subscribers`
65                    ON `sb_subscriber` = `s_id`;
66        END IF;
67       END;
68      $$
```

```
69      -- Восстановление разделителя завершения запросов:
70      DELIMITER ;
```

Вся логика проверки работы таких триггеров подробно показана в решении задачи 3.1.2.a. 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       -- Переключение разделителя завершения запроса,
6       -- т. к. сейчас запросом будет создание триггера,
7       -- внутри которого есть свои классические запросы:
8       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      -- Создание триггера, реагирующего на
38      -- изменение данных о книгах:
```

```
39      CREATE TRIGGER `upd_sbs_rdy_on_subscribers_upd`
40      AFTER UPDATE
41      ON `subscribers`
42       FOR EACH ROW
43        BEGIN
44        IF (OLD.`s_name` != NEW.`s_name`)
45         THEN
46          DELETE FROM `subscriptions_ready`;
47          INSERT INTO `subscriptions_ready`
48                 (`sb_id`,
49                  `sb_subscriber`,
50                  `sb_book`,
51                  `sb_start`,
52                  `sb_finish`,
53                  `sb_is_active`)
54              SELECT `sb_id`,
55                  `s_name`,
56                  `b_name`,
57                  `sb_start`,
58                  `sb_finish`,
59                  `sb_is_active`
60              FROM   `books`
61                   JOIN `subscriptions`
62                    ON `b_id` = `sb_book`
63                   JOIN `subscribers`
64                    ON `sb_subscriber` = `s_id`;
65        END IF;
66       END;
67      $$
68      -- Восстановление разделителя завершения запросов:
69      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)

```
1       -- Удаление старых версий триггеров
2       -- (удобно в процессе разработки и отладки):
3       DROP TRIGGER `upd_sbs_rdy_on_subscriptions_ins`;
4       DROP TRIGGER `upd_sbs_rdy_on_subscriptions_del`;
5       DROP TRIGGER `upd_sbs_rdy_on_subscriptions_upd`;
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 добавление выдачи книг:
11      CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_ins`
12      AFTER INSERT
13      ON `subscriptions`
14       FOR EACH ROW
```

```
15      BEGIN
16       INSERT INTO `subscriptions_ready`
17              (`sb_id`,
18               `sb_subscriber`,
19               `sb_book`,
20               `sb_start`,
21               `sb_finish`,
22               `sb_is_active`)
23           SELECT `sb_id`,
24               `s_name`,
25               `b_name`,
26               `sb_start`,
27               `sb_finish`,
28               `sb_is_active`
29           FROM   `books`
30               JOIN `subscriptions`
31                ON `b_id` = `sb_book`
32               JOIN `subscribers`
33                ON `sb_subscriber` = `s_id`
34           WHERE `s_id` = NEW.`sb_subscriber`
35             AND `b_id` = NEW.`sb_book`;
36       END;
37      $$
38      -- Создание триггера, реагирующего на удаление выдачи книг:
39      CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_del`
40      AFTER DELETE
41      ON `subscriptions`
42       FOR EACH ROW
43        BEGIN
44         DELETE FROM `subscriptions_ready`
45         WHERE `subscriptions_ready`.`sb_id` = OLD.`sb_id`;
46        END;
47      $$
48      -- Создание триггера, реагирующего на обновление выдачи книг:
49      CREATE TRIGGER `upd_sbs_rdy_on_subscriptions_upd`
50      AFTER UPDATE
51      ON `subscriptions
52      FOR EACH ROW
53       BEGIN
54        UPDATE `subscriptions_ready`
55            JOIN (SELECT `sb_id`,
56                     `s_name`,
57                     `b_name`
58                FROM   `books`
59                    JOIN `subscriptions`
60                     ON `b_id` = `sb_book`
61                    JOIN `subscribers`
62                     ON `sb_subscriber` = `s_id`
63                WHERE  `s_id` = NEW.`sb_subscriber`
64                  AND `b_id` = NEW.`sb_book`
65                  AND `sb_id` = NEW.`sb_id`) AS `new_data`
```

```
66        SET   `subscriptions_ready`.`sb_id` = NEW.`sb_id`,
67              `subscriptions_ready`.`sb_subscriber` = `new_data`.`s_name`,
68              `subscriptions_ready`.`sb_book` = `new_data`.`b_name`,
69              `subscriptions_ready`.`sb_start` = NEW.`sb_start`,
70              `subscriptions_ready`.`sb_finish` = NEW.`sb_finish`,
71              `subscriptions_ready`.`sb_is_active` = NEW.`sb_is_active`
72        WHERE  `subscriptions_ready`.`sb_id` = OLD.`sb_id`;
73       END;
74      $$
75      -- Восстановление разделителя завершения запросов:
76      DELIMITER ;
```

Код **INSERT**-триггера (строки 10–36) 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 38–46) получается 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 48–73) – самый нетривиальный. Сам по себе синтаксис обновления на основе выборки выглядит непривычно (мы вынуждены объединять результаты выборки из обновляемой таблицы и выборки-источника – строки 54–65).

Условия в строках 63, 64 позволяют сократить количество выбираемых рядов, а условие в строке 65 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 66–71 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 72 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 66 это значение тоже обновится, если оно изменилось).

Вся логика проверки работы таких триггеров подробно показана в решении задачи 3.1.2.a. 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     -- Создание представления:
```

```
5   CREATE VIEW [subscriptions_ready]
6   WITH SCHEMABINDING
7   AS
8   SELECT [sb_id],
9        [s_name] AS [sb_subscriber],
10       [b_name] AS [sb_book],
11       [sb_start],
12       [sb_finish],
13       [sb_is_active]
14  FROM   [dbo].[books]
15       JOIN [dbo].[subscriptions]
16         ON [b_id] = [sb_book]
17       JOIN [dbo].[subscribers]
18         ON [sb_subscriber] = [s_id];
19  -- Создание уникального кластерного индекса на представлении.
20  -- Именно эта операция "включает" автоматическое обновление
21  -- представления при изменении данных в таблицах,
22  -- на которых оно построено:
23  CREATE UNIQUE CLUSTERED INDEX [idx_subscriptions_ready]
24    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 23, 24).

Необходимым условием создания такого индекса на представлении является привязка представления к схеме базы данных (строка 6), 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 один раз в минуту (логика такого поведения и остальные особенности создания материализованных представлений в Oracle была рассмотрена в решении задачи 3.1.2.a).

Oracle | Решение 3.1.2.b

```
1   -- Удаление старой версии материализованного представления
2   -- (удобно при разработке и отладке):
3   DROP MATERIALIZED VIEW "subscriptions_ready";
4   -- Создание материализованного представления:
5   CREATE MATERIALIZED VIEW "subscriptions_ready"
6   BUILD IMMEDIATE
7   REFRESH FORCE
8   START WITH (SYSDATE) NEXT (SYSDATE + 1/1440)
9   AS
10    SELECT "sb_id",
11         "s_name" AS "sb_subscriber",
12         "b_name" AS "sb_book",
13         "sb_start",
14         "sb_finish",
```

```
15          "sb_is_active"
16      FROM   "books"
17          JOIN "subscriptions"
18            ON "b_id" = "sb_book"
19          JOIN "subscribers"
20            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.

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 задачи 3.1.2.a 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: создать представление, через которое невозможно получить информацию о том, какой конкретно читатель взял ту или иную книгу.

Задача 3.1.3.b: 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

sb_id	sb_book	sb_start	sb_finish	sb_is_active
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.

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.

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 г. до указанной даты.

Пояснения относительно опции **WITH SCHEMABINDING** см. в решении задачи 3.1.3.a.

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 г. 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 10 800 с (т. е. 180 мин, т. е. 3 ч). 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 функцией **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: создать представление, извлекающее информацию о читателях, переводя весь текст в верхний регистр и при этом допускающее модификацию списка читателей.

Задача 3.2.1.b: 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.

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 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.

MySQL Решение 3.2.1.a (проверка работы обновления и удаления)

```
1       UPDATE `subscribers_upper_case_trick`
2       SET    `s_name` = 'Сидоров А.А.'
3       WHERE  `s_id` = 4;
4       UPDATE `subscribers_upper_case_trick`
5       SET    `s_id` = 10
6       WHERE  `s_id` = 4;
7       DELETE FROM `subscribers_upper_case`
8       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, 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)

```
1     CREATE VIEW [subscribers_upper_case]
2     WITH SCHEMABINDING
3     AS
4      SELECT [s_id],
5           UPPER([s_name]) AS [s_name]
6      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)

```
1     CREATE TRIGGER [subscribers_upper_case_ins]
2     ON [subscribers_upper_case]
3     INSTEAD OF INSERT
4     AS
5       SET IDENTITY_INSERT [subscribers] ON;
6       INSERT INTO [subscribers]
7              ([s_id],
8               [s_name])
9       SELECT ( CASE
10             WHEN [s_id] IS NULL
```

```
11                  OR [s_id] = 0 THEN IDENT_CURRENT('subscribers')
12                              + IDENT_INCR('subscribers')
13                              + ROW_NUMBER() OVER (ORDER BY
14                                      (SELECT 1))
15                              - 1
16              ELSE [s_id]
17            END ) AS [s_id],
18          [s_name]
19      FROM   [inserted];
20      SET IDENTITY_INSERT [subscribers] OFF;
21  GO
```

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   INSERT INTO [subscribers_upper_case]
5           ([s_name])
6   VALUES     (N'Соколов С.С.'),
7           (N'Беркутов Б.Б.');
8   INSERT INTO [subscribers_upper_case]
9           ([s_id],
10          [s_name])
11  VALUES     (30,
12          N'Ястребов Я.Я.');
13  INSERT INTO [subscribers_upper_case]
14          ([s_id],
15          [s_name])
16  VALUES     (31,
17          N'Синицын С.С.'),
18          (32,
19          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   UPDATE [subscribers_upper_case]
5   SET    [s_name] = N'И ещё одно имя'
6   WHERE  [s_id] >= 31;
7   DELETE FROM [subscribers_upper_case]
8   WHERE  [s_id] = 30;
9   DELETE FROM [subscribers_upper_case]
10  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)

```
1   INSERT ALL
2    INTO "subscribers" ("s_id", "s_name") VALUES (1, N'Соколов С.С.')
3    INTO "subscribers" ("s_id", "s_name") VALUES (2, N'Беркутов Б.Б.')
4    INTO "subscribers" ("s_id", "s_name") VALUES (3, N'Филинов Ф.Ф.')
5   SELECT 1 FROM "DUAL";
6   UPDATE "subscribers_upper_case"
7   SET    "s_name" = N'Синицын З.З.'
8   WHERE  "s_id" = 6;
9   -- Такой запрос НЕ НАЙДЁТ искомое, т. к. имя должно быть в верхнем ре-
10  гистре:
11  UPDATE "subscribers_upper_case"
12  SET    "s_name" = N'Синцын С.С.'
13  WHERE  "s_name" = N'Синицын З.З.';
14  -- А такой запрос найдёт искомое:
15  UPDATE "subscribers_upper_case"
16  SET    "s_name" = N'Синцын С.С.'
17  WHERE  "s_name" = N'СИНИЦЫН З.З.';
18  DELETE FROM "subscribers_upper_case"
19  WHERE  "s_id" = 6;
20  -- Такой запрос НЕ НАЙДЁТ искомое, т. к. имя должно быть в верхнем ре-
21  гистре:
22  DELETE FROM "subscribers_upper_case"
23  WHERE  "s_name" = N'Филинов Ф.Ф.';
24  -- А такой запрос найдёт искомое:
25  DELETE FROM "subscribers_upper_case"
26  WHERE  "s_name" = N'ФИЛИНОВ Ф.Ф.';
27  DELETE FROM "subscribers_upper_case"
28  WHERE  "s_id" > 4;
```

Решение 3.2.1.b.

Решение этой задачи для MySQL подпадает под все ограничения, характерные для решения задачи 3.2.1.a: 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)

```
1   CREATE VIEW `subscriptions_wcd`
2   AS
3    SELECT `sb_id`,
4         `sb_subscriber`,
5         `sb_book`,
6         CONCAT(`sb_start`, ' - ', `sb_finish`) AS `sb_dates`,
7         `sb_is_active`
8    FROM   `subscriptions`
```

MySQL — Решение 3.2.1.b (представление, допускающее удаление и обновление данных)

```
1   CREATE VIEW `subscriptions_wcd_trick`
2   AS
3    SELECT `sb_id`,
4         `sb_subscriber`,
5         `sb_book`,
6         CONCAT(`sb_start`, ' - ', `sb_finish`) AS `sb_dates`,
7         `sb_start`,
8         `sb_finish`,
9         `sb_is_active`
10   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 задачи 3.2.1.a 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)

```
1   CREATE VIEW [subscriptions_wcd]
2   WITH SCHEMABINDING
3   AS
4    SELECT [sb_id],
5         [sb_subscriber],
6         [sb_book],
7         CONCAT([sb_start], ' - ', [sb_finish]) AS [sb_dates],
8         [sb_is_active]
9    FROM   [dbo].[subscriptions]
```

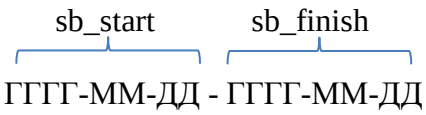
Такое представление уже позволяет извлекать данные в указанном в условии задачи формате, а также выполнять удаление данных. Для того чтобы через это представление можно было выполнять вставку и обновление данных, нужно создать на нём два триггера – на операциях вставки и обновления.

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 задачи 3.2.1.a.

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. То есть единственный способ 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.

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)

```
1     CREATE OR REPLACE TRIGGER "subscriptions_wcd_ins"
2     INSTEAD OF UPDATE ON "subscriptions_wcd"
3     FOR EACH ROW
4      BEGIN
5       UPDATE "subscriptions"
6       SET    "sb_id" = :new."sb_id",
7           "sb_subscriber" = :new."sb_subscriber",
8           "sb_book" = :new."sb_book",
9           "sb_start" = TO_DATE(SUBSTR(:new."sb_dates", 1,
10                      (INSTR(:new."sb_dates", ' ') - 1)),
11                      'YYYY-MM-DD'),
12          "sb_finish" = TO_DATE(SUBSTR(:new."sb_dates",
13                      (INSTR(:new."sb_dates", ' ') + 3 )),
14                      'YYYY-MM-DD'),
15          "sb_is_active" = :new."sb_is_active"
16       WHERE  "sb_id" = :old."sb_id";
17     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)

```
1     INSERT INTO "subscriptions_wcd"
2               ("sb_subscriber",
3                "sb_book",
4                "sb_dates",
5                "sb_is_active")
6     VALUES     (1,
7               3,
8               '2019-01-12 - 2019-02-12',
```

```
9             'N');
10    UPDATE "subscriptions_wcd"
11    SET    "sb_dates" = '2019-01-12 - 2019-02-12'
12    WHERE  "sb_id" = 100;
13    DELETE FROM "subscriptions_wcd"
14    WHERE  "sb_id" = 100;
15    DELETE FROM "subscriptions_wcd"
16    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 и при этом допускающее модификацию списка книг, переводя весь текст в верхний регистр.

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: создать представление, извлекающее из таблицы **subscriptions** удобочитаемую для человека (с именами читателей и названиями книг вместо идентификаторов) информацию и при этом позволяющее модифицировать данные в таблице **subscriptions**.

Задача 3.2.2.b: 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.

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 задачи 3.2.1.a. 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])
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      -- Запрос 2 (вставка выполняется):
22      INSERT INTO [subscriptions_with_text]
23              ([sb_subscriber],
24               [sb_book],
25               [sb_start],
26               [sb_finish],
27               [sb_is_active])
28      VALUES      (1,
29               3,
30               '2015-01-12',
31               '2015-02-12',
32               'N'),
33              (1,
34               1,
35               '2015-01-12',
36               '2015-02-12',
37               'N');
38      -- Запрос 3 (вставка НЕ выполняется):
39      INSERT INTO [subscriptions_with_text]
40              ([sb_subscriber],
41               [sb_book],
42               [sb_start],
43               [sb_finish],
44               [sb_is_active])
45      VALUES      (N'Иванов И.И.',
46               3,
47               '2015-01-12',
48               '2015-02-12',
49               'N');
50      -- Запрос 4 (вставка НЕ выполняется):
51      INSERT INTO [subscriptions_with_text]
52              ([sb_subscriber],
53               [sb_book],
54               [sb_start],
55               [sb_finish],
56               [sb_is_active])
57      VALUES      (1,
58               N'Какая-то книга',
59               '2015-01-12',
60               '2015-02-12',
61               'N');
65      -- Запрос 5 (вставка НЕ выполняется):
66      INSERT INTO [subscriptions_with_text]
67              ([sb_subscriber],
68               [sb_book],
```

```
69              [sb_start],
70              [sb_finish],
71              [sb_is_active])
72      VALUES     (N'Какой-то читатель',
73              N'Какая-то книга',
74              '2015-01-12',
75              '2015-02-12',
76              '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 удобочитаемые для человека имена читателя и название книги (т. к. псевдотаблицы **inserted** и **deleted** наполняются данными из представления).

Эту ситуацию мы рассматриваем и исправляем в строках 28–40: 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 задачи 3.2.1.a, мы не можем корректно определить взаимоотношение записей в псевдотаблицах **inserted** и **deleted**, если было изменено значение первичного ключа, поэтому мы запрещаем эту операцию (строки 18–24).

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
38                 WHEN (PATINDEX('%[^0-9]%',
39                     [inserted].[sb_book]) = 0)
40                 THEN [inserted].[sb_book]
41                 ELSE [subscriptions].[sb_book]
42                END,
43                [subscriptions].[sb_start] = [inserted].[sb_start],
44                [subscriptions].[sb_finish] = [inserted].[sb_finish],
45                [subscriptions].[sb_is_active] =
46                            [inserted].[sb_is_active]
47            FROM   [subscriptions]
48              JOIN [inserted]
49                ON [subscriptions].[sb_id] = [inserted].[sb_id];
50           END
51          END
52    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)

```
1     -- Запрос 1 (обновление выполняется):
2     UPDATE [subscriptions_with_text]
3     SET    [sb_start] = '2021-01-12'
4     WHERE  [sb_id] = 5000;
5     -- Запрос 2 (обновление выполняется):
6     UPDATE [subscriptions_with_text]
7     SET    [sb_subscriber] = 3
8     WHERE  [sb_id] = 5000;
9     -- Запрос 3 (обновление выполняется):
10    UPDATE [subscriptions_with_text]
11    SET    [sb_book] = 4
12    WHERE  [sb_id] = 5000;
13    -- Запрос 4 (обновление НЕ выполняется):
14    UPDATE [subscriptions_with_text]
15    SET    [sb_subscriber] = N'Читатель'
16    WHERE  [sb_id] = 5000;
```

```
17    -- Запрос 5 (обновление НЕ выполняется):
18    UPDATE [subscriptions_with_text]
19    SET    [sb_book] = N'Книга'
20    WHERE  [sb_id] = 5000;
21    -- Запрос 6 (обновление НЕ выполняется):
22    UPDATE [subscriptions_with_text]
23    SET    [sb_id] = 5001
24    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)

```
1     CREATE TRIGGER [subscriptions_with_text_del]
2     ON [subscriptions_with_text]
3     INSTEAD OF DELETE
4     AS
5      DELETE FROM [subscriptions]
6      WHERE [sb_id] IN (SELECT [sb_id]
7                        FROM   [deleted]);
8     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)

```
1     CREATE TRIGGER [subscriptions_with_text_del]
2     ON [subscriptions_with_text]
3     INSTEAD OF DELETE
4     AS
5      -- Попытка определить, переданы ли в DELETE-запрос
6      -- поля sb_subscriber и/или sb_book:
7      SET NOCOUNT ON;
```

```
8     DECLARE @ExecStr VARCHAR(50), @Qry NVARCHAR(255);
9     CREATE TABLE #inputbuffer
10    (
11     [EventType] NVARCHAR(30),
12     [Parameters] INT,
13     [EventInfo] NVARCHAR(255)
14    );
15     SET @ExecStr = 'DBCC INPUTBUFFER(' + STR(@@SPID) + ')';
16    INSERT INTO #inputbuffer EXEC (@ExecStr);
17    SET @Qry = LOWER((SELECT [EventInfo] FROM #inputbuffer));
18     -- Для отладки можно раскомментировать следующую строку
19    -- и убедиться, что в ней расположен запрос, вызвавший
20    -- срабатывание триггера:
21    -- PRINT(@Qry);
22
23    IF    ((CHARINDEX ('sb_subscriber', @Qry) > 0)
24      OR  (CHARINDEX ('sb_book', @Qry) > 0))
25        BEGIN
26         RAISERROR ('Deletion from [subscriptions_with_text] view
27                 using [sb_subscriber] and/or [sb_book]
28                 is prohibited.', 16, 1);
29         ROLLBACK;
30        END
31    SET NOCOUNT OFF;
32    -- Здесь выполняется само удаление:
33    DELETE FROM [subscriptions]
34    WHERE [sb_id] IN (SELECT [sb_id]
35                 FROM   [deleted]);
36   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)

```
1     -- Запрос 1 (удаление работает):
2     DELETE FROM [subscriptions_with_text]
3     WHERE  [sb_id] < 10;
4     -- Запрос 2 (удаление работает):
5     DELETE FROM [subscriptions_with_text]
6     WHERE  [sb_start] = '2012-01-12';
7     -- Запрос 3 (удаление НЕ работает
8     -- при обеих версиях триггера):
9     DELETE FROM [subscriptions_with_text]
10    WHERE  [sb_book] = 2;
11    -- Запрос 4 (нулевое количество совпадений
12    -- в первой версии триггера и отмена транзакции
13    -- во второй версии триггера):
14    DELETE FROM [subscriptions_with_text]
15    WHERE  [sb_book] = '2';
```

```
16  -- Запрос 5 (возможно обнаружение совпадений
17  -- в первой версии триггера; во второй версии
18  -- триггера всегда будет отмена транзакции):
19  DELETE FROM [subscriptions_with_text]
20  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    -- Запрос 2 (вставка выполняется):
16    INSERT INTO "subscriptions_with_text"
17            ("sb_subscriber",
18             "sb_book",
19             "sb_start",
20             "sb_finish",
21             "sb_is_active")
22    VALUES     (1,
23             3,
24             TO_DATE('2015-01-12', 'YYYY-MM-DD'),
25             TO_DATE('2015-02-12', 'YYYY-MM-DD'),
26             'N');
27    -- Запрос 3 (вставка НЕ выполняется):
28    INSERT INTO "subscriptions_with_text"
29            ("sb_subscriber",
30             "sb_book",
```

```
31              "sb_start",
32              "sb_finish",
33              "sb_is_active")
34      VALUES      (N'Иванов И.И.',
35              3,
36              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
37              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
38              'N');
39      -- Запрос 4 (вставка НЕ выполняется):
40      INSERT INTO "subscriptions_with_text"
41             ("sb_subscriber",
42              "sb_book",
43              "sb_start",
44              "sb_finish",
45              "sb_is_active")
46      VALUES      (1,
47              N'Какая-то книга',
48              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
49              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
50              'N');
51      -- Запрос 5 (вставка НЕ выполняется):
52      INSERT INTO "subscriptions_with_text"
53             ("sb_subscriber",
54              "sb_book",
55              "sb_start",
56              "sb_finish",
57              "sb_is_active")
58      VALUES      (N'Какой-то читатель',
59              N'Какая-то книга',
60              TO_DATE('2015-01-12', 'YYYY-MM-DD'),
61              TO_DATE('2015-02-12', 'YYYY-MM-DD'),
62              '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 17–30 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 22 и 28 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"
```

```
7     SET    "sb_subscriber" = 3
8     WHERE  "sb_id" = 101;
9     -- Запрос 3 (обновление выполняется):
10    UPDATE "subscriptions_with_text"
11    SET    "sb_book" = 4
12    WHERE  "sb_id" = 101;
13    -- Запрос 4 (обновление НЕ выполняется):
14    UPDATE "subscriptions_with_text"
15    SET    "sb_subscriber" = N'Читатель'
16    WHERE  "sb_id" = 101;
17    -- Запрос 5 (обновление НЕ выполняется):
18    UPDATE "subscriptions_with_text"
19    SET    "sb_book" = N'Книга'
20    WHERE  "sb_id" = 101;
21    -- Запрос 6 (обновление выполняется):
22    UPDATE "subscriptions_with_text"
23    SET    "sb_id" = 1001
24    WHERE  "sb_id" = 101;
```

Создадим триггер, позволяющий реализовать операцию удаления данных. Его код отличается от кода аналогичного триггера для MS SQL Server только логикой определения идентификаторов удаляемых записей.

Oracle | Решение 3.2.2.a (создание триггера для реализации операции удаления)

```
1     CREATE OR REPLACE TRIGGER "subscriptions_with_text_del"
2      INSTEAD OF DELETE ON "subscriptions_with_text"
3      FOR EACH ROW
4       BEGIN
5        DELETE FROM "subscriptions"
6        WHERE "sb_id" = :old."sb_id";
7       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)

```
-- 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.

Логика выборки данных в этом представлении является упрощённым вариантом решения задачи 2.2.2.b (см. п. 2.2.2), а сами триггеры в сравнении с предыдущим примером – в разы более простыми.

Так как 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 задачи 2.2.2.b, п. 2.2.2).

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],
```

```
6                                   [g_name]
7           FROM   [books]
8                JOIN [m2m_books_genres]
9                  ON [books].[b_id] = [m2m_books_genres].[b_id]
10               JOIN [genres]
11                 ON [m2m_books_genres].[g_id] = [genres].[g_id]
12        )
13  SELECT [outer].[b_id],
14       [outer].[b_name],
15       STUFF ((SELECT DISTINCT ',' + [inner].[g_name]
16               FROM   [prepared_data] AS [inner]
17               WHERE  [outer].[b_id] = [inner].[b_id]
18               ORDER  BY ',' + [inner].[g_name]
19               FOR XML PATH(''), TYPE).value('.', 'nvarchar(max)'),
20               1, 1, '')
21       AS [genres]
22  FROM   [prepared_data] AS [outer]
23  GROUP  BY [outer].[b_id],
24          [outer].[b_name]
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

MS SQL | Решение 3.2.2.b (создание триггера для реализации операции вставки)

```
1   CREATE TRIGGER [books_with_genres_ins]
2   ON [books_with_genres]
3   INSTEAD OF INSERT
4   AS
5    INSERT INTO [genres]
6           ([g_name])
7    SELECT [genres]
8    FROM   [inserted];
9   GO
```

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 задачи 2.2.2.b, п. 2.2.2):

Oracle | Решение 3.2.2.b (создание представления)

```
1   CREATE VIEW "books_with_genres"
2   AS
3   SELECT "b_id", "b_name",
4        UTL_RAW.CAST_TO_NVARCHAR2
5        (
6          LISTAGG
7          (
8            UTL_RAW.CAST_TO_RAW("g_name"),
9            UTL_RAW.CAST_TO_RAW(N',')
10         )
```

```
11          WITHIN GROUP (ORDER BY "g_name")
12        )
13        AS "genres"
14        FROM "books"
15        JOIN "m2m_books_genres" USING ("b_id")
16        JOIN "genres" USING ("g_id")
17        GROUP  BY "b_id",
18              "b_name"
```

Создадим триггер, позволяющий реализовать операцию вставки данных.

Oracle | Решение 3.2.2.b (создание триггера для реализации операции вставки)

```
1   CREATE OR REPLACE TRIGGER "books_with_genres_ins"
2   INSTEAD OF INSERT ON "books_with_genres"
3   FOR EACH ROW
4    BEGIN
5     INSERT INTO "genres"
6           ("g_name")
7     VALUES    (:new."genres");
8    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** удобочитаемую для человека (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.

4. КРАТКОЕ СРАВНЕНИЕ MYSQL, MS SQL SERVER, ORACLE

Данный раздел является своего рода шпаргалкой по основным различиям трёх СУБД, рассмотренных в данном учебно-методическом пособии. Ссылки на соответствующие примеры и задачи позволяют ознакомиться с подробностями нужной части материала.

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

Суть	MySQL	MS SQL Server	Oracle	Ссылка
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
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
Приведение строкового представления даты к датавременному типу	Автоматическое	Автоматическое	Требуется явная конвертация	2.1.7.b
Показ первых нескольких рядов выборки	**LIMIT x**	**TOP x** или **OFFSET y ROWS FETCH NEXT x ROWS ONLY** Есть также поддержка **TOP ... WITH TIES**	До версии 12c нужен вложенный запрос с **ROW_NUMBER()**, с версии 12c поддерживается **OFFSET y ROWS FETCH NEXT x ROWS ONLY**	2.1.8.b, 2.2.7.b
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, 2.2.7.d, 2.2.9.d
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, 2.1.8.c, 2.1.8.d
Общие табличные выражения	Не поддерживаются, но иногда можно эмулировать	Поддерживаются	Поддерживаются	2.1.8.d, 2.2.6.b
Определение разницы в днях между двумя датами	**DATEDIFF (большая, меньшая)**	**DATEDIFF (day, меньшая, большая)**	**(большая, меньшая)**	2.1.9.d

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Получение текущей даты	**CURDATE()**	**CONVERT (date, GETDATE())**	**TRUNC(SYSDATE)**	2.1.9.d
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, 2.2.2.a
Групповая конкатенация	**GROUP_CONCAT()**	**STUFF (FOR XML…)**	**LISTAGG()**	2.2.2.a, 2.2.2.b
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
Защита от **NULL**	**IFNULL (поле, подстановочное значение)**	**ISNULL (поле, подстановочное значение)**	**NVL (поле, подстановочное значение)**	2.2.6.b
TRUE / **FALSE**	Есть, трактуются как 1 и 0	Явным образом отсутствуют	Явным образом отсутствуют	2.2.7.e
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
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, 2.3.5.b

Суть	MySQL	MS SQL Server	Oracle	Ссылка
Конкатенация строк	**CONCAT (сколько угодно параметров)**	**CONCAT (сколько угодно параметров)**	**CONCAT (ровно два параметра)** или использовать **||**	2.3.5.c
Вставка значения в автоинкрементируемый первичный ключ	Просто передать значение и всё	Включить для таблицы **IDENTITY_INSERT**	Отключить триггер, реализующий автоинкрементацию	2.3.1.a
Вставка или обновление в зависимости от совпадения значений первичного ключа	**REPLACE** и **ON DUPLICATE KEY UPDATE**	**MERGE**	**MERGE**	2.3.4.a, 2.3.4.b

Св. план 2018, поз. 40

Учебное издание

Куликов Святослав Святославович
Фадеева Елена Евгеньевна

РАБОТА С MYSQL, MS SQL SERVER И ORACLE В ПРИМЕРАХ

В двух частях

Часть 1

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. С. Юрец*
Корректор *Е. И. Костина*
Компьютерная правка, оригинал-макет *М. В. Касабуцкий*

Подписано в печать 12.03.2019. Формат 60×84 1/8. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 33,71. Уч.-изд. л. 24,0. Тираж 100 экз. Заказ 274.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6