Лабораторная работа №1

Основы SQL (MySQL)

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, в качестве математической базы для которых используются теоретические языки запросов, предложенные Эдгаром Коддом:

- *реляционная алгебра*;
- *реляционное исчисление.*

В первом случае (реляционная алгебра) операнды и результаты всех действий являются отношениями. Такие языки — процедурные, поскольку отношение, которое является результатом запроса к базе данных, вычисляется при последовательном выполнении операторов, применяемым к отношениям. Сами операторы состоят из операндов (отношений) и реляционных операций (их результатом тоже является отношение).

Языки реляционного исчисления — непроцедурные, их называют *описательными* или *декларативными*. Они позволяют составлять запросы с помощью предиката первого порядка, которому должны удовлетворять кортежи или домены отношений. Таким образом, запрос к базе данных на таком языке содержит только информацию о желаемом результате. К языкам этой группы относится и SQL.

SQL (Structured Query Language, "язык структурированных запросов") — универсальный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. В нем создается линейная последовательность операторов языка, которые выполняются СУБД. Операторы состоят из:

- имен операций и функций;
- имен таблиц и их столбцов;
- зарезервированных ключевых слов и специальных символов;
- логических, арифметических и строковых выражений.

Общий вид простого оператора в SQL:

```
ОПЕРАТОР параметры;
```

Если параметр не один, а несколько, то они перечисляются через запятую.

Все предложения на языке SQL оканчиваются точкой с запятой. Например:

```
USE `first_lab_data_base`;
SELECT `id`, `field1` FROM `mytable`;
```

Выражения в SQL не зависят от регистра, не требуют обязательного наличия кавычек при обозначении названий, дополнительные разделители (пробел, табуляция, переход на новую строку) игнорируются. Для обозначения названий баз данных, таблиц, атрибутов таблиц, то есть названий, связанных с объектами СУБД, могут использоваться кавычки типа «тупое ударение» («`»). Например:

```
`элемент_бд`
```

Для текстовых данных, вводимых пользователем в базу и не связанных с элементами СУБД (например, обычных строковых значений), используются обычные или двойные кавычки: 'текст', "большой текст". Чтобы с помощью двойных кавычек указать строку, которая уже содержит двойные кавычки в конце и начале выражения, нужно записывать ее так (экранирование):

```
"\"заголовок\""
```

Существует общепринятый стиль «правильного» оформления выражений. Оно заключается в том, что при написании каких-либо выражений:

- после естественных разделителей выражений (например, запятых) ставится пробел;
- дополнительные разделители (пробелы, табы) не используются, если нет необходимости записать многостроковое выражение в удобном для чтения виде;
- системные обозначения (названия операторов, функций, ключевых слов и т.п.) пишутся заглавными буквами;
- при указании названий, связанных с объектами СУБД, обязательно используются кавычки в виде «тупого ударения».

Например:

```
INSERT INTO `news` (`id`, `post_date`) VALUES (42, '2015-02-02 04:13:15');
```

Далее **будут использоваться квадратные скобки для обозначения дополнительных (но необязательных) параметров**. Выражения, с параметром в квадратных скобках и без него, являются синтаксически правильными.

Например:

```
SELECT * FROM `table` [ WHERE условие ];
```

Такое выражение можно интерпретировать не только так:

```
SELECT * FROM `table` WHERE условие;
```

Но и как аналогичное выражение без необязательных параметров, т.е.:

```
SELECT * FROM `table`;
```

Типы данных в MySQL

В MySQL представлено множество типов данных, в соответствии с которыми хранятся и обрабатываются все данные в таблицах. Перечислим некоторые из них.

Числа

Числа разделяются на целые и дробные. Целые представлены следующими типами данных:

- TINYINT — 1 байт, т.е. принимает значения от -128 до 127 (в случае использования TINYINT UNSIGNED, т.е. без учета знака перед числом — 0..255);
- SMALLINT — 2 байта, -32768..32767 (0..65535);
- MEDIUMINT — 3 байта, -8388608..8388607 (0..2^24-1);
- INT — 4 байта, -2147483648..2147483647 (0..2^32-1);
- BIGINT — 8 байт, -2^32..2^32-1 (0..2^64).

Дробные числа представлены следующими типами данных:

- FLOAT (4 байта);
- DOUBLE (8 байт) — вдвое большая точность после запятой.

Строки

- CHAR — дополняет до заданной «ширины» (например, в случае использования CHAR(6) строка из пяти символов «hello» будет храниться как «hello », т.е. с пробелом на конце);
- VARCHAR — использует необходимый минимум памяти (в случае использования VARCHAR(6) строка из пяти символов «hello» будет храниться как «hello»);
- BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB — бинарные данные разных размеров;
- TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT — текстовые данные разных размеров;
- ENUM — одно из заданных значений (например, в ячейке типа ENUM(0,1,2) может храниться ноль, единица или двойка);

- SET — ноль или более заданных значений (в ячейке типа SET(0,1,2) может храниться любая комбинация из ноля, единицы и двойки — в том числе и пустое значение).

Другие типы

- булев: BOOL, BOOLEAN;
- уникальный автоматически увеличившийся идентификатор: SERIAL (== BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE);
- дата и время: DATETIME, DATE, TIMESTAMP, TIME, YEAR.

Работа с базами данных и таблицами

1. Команды для манипулирования базами данных

1. Создание базы данных:

```
CREATE DATABASE `db_name`;
```

Параметр команды создания баз данных — имя, выдаваемое создаваемой базе данных. Например, для создания базы данных под названием "my_database" нужно ввести команду:

```
CREATE DATABASE `my_database`;
```

Несмотря на то, что современная версия MySQL позволяет создавать БД с кириллическими и специальными символами в названии, принято использовать латинские буквы, цифры и знаки подчеркивания («_»).

2. При одновременной работе в нескольких базах данных в командах нужно уточнять, с данными какой БД вы работаете. Для этого используется разделитель точка — «.». Так, чтобы обратиться к атрибуту "attribute" таблицы "table", находящейся в базе данных "database1", нужно использовать запись:

```
`database1`.`table`.`attribute`
```

Если же вам понадобится обратиться к аналогичному атрибуту такой же таблицы, находящейся в БД "database2", запись станет такой:

```
`database2`.`table`.`attribute`
```

Для того, чтобы вводимые команды применялись к конкретной базе данных по умолчанию, можно воспользоваться командой USE и ввести название базы данных, с которой мы будем в дальнейшем работать:

```
USE `my_database`;
```

После выполнения команды USE следующие записи будут эквивалентны:

```
`my_database`.`table1`
`table1`
```

3. Удаление существующей базы данных выполняется командой DROP DATABASE, которая в качестве единственного аргумента принимает название удаляемой базы данных. Например, чтобы удалить созданную вами в начале работы БД "my_database", нужно выполнить:

```
DROP DATABASE `my_database`;
```

После успешного удаления вы можете заново создать ее. **Учтите, что если бы в вашей базе данных были таблицы и данные, вся эта информация была бы утеряна навсегда.**

4. Для просмотра информации о базах данных, их таблиц, а также привилегий текущего пользователя, используется команда SHOW.

1. Увидеть список всех доступных пользователю баз данных можно с помощью команды:

```
SHOW DATABASES;
```

2. Увидеть список всех таблиц в используемой базе данных можно с помощью команды:

```
SHOW TABLES;
```

3. Для вывода информации о таблице "table_name" используйте команду:

```
SHOW CREATE TABLE `table_name`;
```

Обратите внимание, что результатом выполнения команды будет команда создания таблицы с учетом всех изменений, произведенных над таблицей в процессе работы с базой данных.

4. Увидеть список всех прав текущего пользователя СУБД можно с помощью команды:

```
SHOW GRANTS;
```

2. Команды для работы с таблицами БД

1. Для **создания таблицы** используется команда CREATE TABLE. В качестве аргумента ей передается название новой таблиц и перечисление всех атрибутов таблицы и их описаний (типы данных, значения по умолчанию, ограничения на применяемые значения и т.п.) — все перечисление берется в круглые скобки, а разделителем между атрибутами служит запятая. В общем виде команда выглядит так:

```
CREATE TABLE `table_name` (
 `название_первого_поля` его_тип [параметры],
 `название_второго_поля` его_тип [параметры],
 ...
 `название_последнего_поля` его_тип [параметры]
);
```

Обратите внимание, что после последнего атрибута запятая не требуется. Пример создания таблицы "news" с тремя атрибутами (идентификатор новости, время публикации, текст новости):

```
CREATE TABLE `news` (
 `id` MEDIUMINT(8) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
 `posted` TIMESTAMP NOT NULL,
 `content` TEXT
);
```

2. Для **удаления таблицы** используется команда DROP TABLE с названием таблицы в качестве единственного аргумента. Например, для удаления созданной таблицы "news" команда будет выглядеть так:

```
DROP TABLE `news`;
```

3. Для **изменения таблиц** используется команда ALTER TABLE. Вид производимого изменения определятся последующими дополнительными командами:

3.1. **Переименование** таблицы осуществляется с помощью подкоманды RENAME. Например, чтобы переименовать таблицу "news" в "news_new", нужно выполнить следующую команду:

```
ALTER TABLE `news` RENAME TO `news_new`;
```

3.2. Для **добавления нового атрибута в таблицу** потребуется подкоманда ADD COLUMN, для которой нужно ввести название нового атрибута и указать его тип. Например, добавление к таблице "news" нового атрибута "author" (имя автора) будет выглядеть следующим образом:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42);
```

Кроме того, можно задать положение добавляемого поля. Для этого в конец команды добавляется инструкция, указывающая, после какого столбца будет добавлено новое поле. Например, команда добавления атрибута "author" в таблицу "news" с тем, чтобы "author" стал вторым полем, выглядит следующим образом:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42) AFTER `id`;
```

Чтобы поле стало первым в таблице, нужно заменить конструкцию с "AFTER ..." на ключевое слово "FIRST". Команда добавления атрибута "author" в таблицу "news" в качестве первого поля будет выглядеть так:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42) FIRST;
```

3.3. Для **изменения типа атрибута таблицы** служит подкоманда MODIFY, для которой нужно указать таблицу, название атрибута и заново перечислить все требуемые для него параметры. Например, чтобы изменить тип атрибута "author" таблицы "news" на CHAR(42), нужно выполнить следующую команду:

```
ALTER TABLE `news` MODIFY COLUMN `author` CHAR(42);
```

3.4. Для **удаления атрибута из таблицы** служит подкоманда DROP COLUMN. Например, команда удаления атрибута "author" из таблицы "news" выглядит так:

```
ALTER TABLE `news` DROP COLUMN `author`;
```

3. Команды для работы с данными таблиц баз данных

Все команды из двух предыдущих подразделов («Команды для манипулирования базами данных» и «Команды для работы с таблицами БД») относятся к языку DDL (Data Definition Language), являющемуся частью языка SQL, которая обеспечивает работу со схемой БД. Другая, не менее важная, часть SQL — язык DML (Data Manipulation Language), позволяющий манипулировать самими данными в таблицах баз данных. К нему относятся команды, речь о которых пойдет уже в этом подразделе.

1. Для **добавления строк в таблицу** используется команда INSERT. В качестве аргументов ей передается название таблицы и набор всех значений для одной из строк или набор из названий атрибутов и соответствующих им значений, подразумевая, что значения остальных атрибутов будут заполнены автоматически.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием всех значений:

```
INSERT INTO `news` VALUES (1, NOW(), "Текст новости", "Редактор");
```

Использованное в запросе выражение "NOW()" — это обращение к встроенной в MySQL функции, которая возвращает текущее время, Таким образом, в качестве значения "posted" для добавляемой строки запишется текущее время.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием только некоторых значений:

```
INSERT INTO `news` (`posted`, `content`) VALUES (NOW(), "Текст новости");
```

В данном случае для добавляемой строки мы задали только значения атрибутов "posted" и "content". В поле "id" автоматически запишется число, которое будет на единицу больше последнего значения "id" в таблице (или 1, если добавляемая строка — первая), а в поле "author" для этой строки запишется значение NULL — отметка об отсутствии значения (**не путайте ее с нулевым значением**).

2. Для **изменения данных в строке таблицы** используется команда UPDATE. В качестве аргументов ей передаются название таблицы, названия атрибутов, значения которых будут изменены, и новые значения этих атрибутов.

Например, для того, чтобы изменить значение поля "author" во всех строках таблицы "news" на "Пользователь", а значение поля "content" — на "Пустой текст", нужно выполнить следующую команду:

```
UPDATE `news` SET `author` = "Пользователь", `content` = "Пустой текст";
```

В данном случае будут изменены все строки таблицы, но зачастую нужно изменять значения только в определенных строках. Для этого предусмотрена возможность установления условия отбора тех строк (WHERE), для которых будут проведены изменения.

Например, чтобы теперь изменить значение поля "content" на "Новый текст" только для строки с "id", равным 1, в таблице "news", нужно выполнить следующую команду:

```
UPDATE `news` SET `content` = "Новый текст" WHERE id = 1;
```

Правила формирования условий запросов подробно описаны в следующем подразделе — «Команда выборки».

3. Для **удаления данных из таблицы** используется команда DELETE. В качестве аргумента ей передается только название таблицы. Кроме того, можно ограничить список удаляемых строк, задав условие с помощью WHERE (по аналогии с тем, как это делалось в команде UPDATE) — тогда СУБД выберет строки, соответствующие условию, и удалит из таблицы только их. Без указания условия будут удалены все строки из таблицы.

Например, чтобы удалить из таблицы "news" строку с "id", равным 1, нужно выполнить следующую команду:

```
DELETE FROM `news` WHERE id = 1;
```

Команда выборки

1. О выборке

Для получения значений атрибутов таблиц БД используется команда SELECT. Общий вид команды представляется следующим образом:

```
SELECT `my_field1`, `my_field2`, ..., `my_fieldN`
FROM `my_table`
WHERE условие;
```

Здесь:

- my_field1, my_field2 и т.д. — это перечисление названий атрибутов, значения которых мы "выбираем", т.е. в результирующей таблице будут выведены только значения указанных атрибутов. В случае, если требуется вывод значений всех атрибутов результирующей таблицы, для упрощения записи запроса используется символ звездочки («*»).
- my_table — это название таблицы, из которой будет сделана выборка.
- условие WHERE может иметь сложную структуру или отсутствовать.

Например, для выборки всех значений всех атрибутов из таблицы "news" достаточно ввести следующую команду:

```
SELECT * FROM `news`;
```

При выполнении запроса в оперативной памяти создается виртуальная таблица, состоящая из всех данных, удовлетворяющих условию отбора, а исходная таблица при этом никак не изменяется.

2. Условия выборки

Для решения некоторых задач условие отбора (WHERE) может иметь сложную структуру. Для построения таких условий используют логические операторы AND, OR, NOT и некоторые другие возможности языка SQL. Для группировки условий используются разделительные скобки («(» и «)»).

1. Для сравнения выражений предусмотрены: равенство («=»); больше или равно («>=»), меньше или равно («<=»), не равно («<>» или «!=»). Например, выборка значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 42, осуществляется следующим образом:

```
SELECT `content` FROM `news` WHERE `id` < 42;
```

2. Логическое умножение (И) записывается как AND и используется, когда требуется одновременное выполнение двух и более условий. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

```
SELECT `content` FROM `news` WHERE `id` > 21 AND `id` < 42;
```

3. Логическое сложение (ИЛИ) записывается как OR и используется, когда требуется, чтобы выполнялось хотя бы одно из нескольких условий. Пример

выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 или меньше 10:

```
SELECT `content` FROM `news` WHERE `id` > 21 OR `id` < 10;
```

Логическое сложение имеет меньший приоритет чем логическое умножение, поэтому для корректной записи логических формул может потребоваться использование разделяющих скобок. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" либо больше 21 и меньше 42, либо больше 84:

```
SELECT `content` FROM `news` WHERE ( `id` > 21 AND `id` < 42 ) OR `id` > 84;
```

4. Логическое отрицание (НЕ) записывается как NOT и используется для инвертирования последующего условия. Например, выборку значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 21 или больше 42, можно осуществить так:

```
SELECT `content` FROM `news` WHERE NOT ( `id` >= 21 AND `id` <= 42 );
```

5. Если у атрибута отсутствует значение, то оно записывается как NULL *(NULL — символ отсутствия значения, что не путать с пустым значением: пустое значение существует, а NULL указывает на его отсутствие)*. Для составления условий на выборку подобных отсутствующих значений предусмотрено специальное выражение IS NULL. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" отсутствует:

```
SELECT `content` FROM `news` WHERE `author` IS NULL;
```

6. Для указания принадлежности значения атрибута какому-либо интервалу предусмотрено выражение BEETWEEN .. AND. Выражение "BETWEEN a AND b". Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

```
SELECT `content` FROM `news` WHERE `id` BETWEEN 21 AND 42;
```

7. Для указания принадлежности значения атрибута какому-либо множеству предусмотрено выражение IN. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" принимает значения 21, 42, 84 или 168:

```
SELECT `content` FROM `news` WHERE `id` IN (21, 42, 84, 168);
```

8. Для поиска строковых значений, содержащих заданную строку по шаблону, предусмотрена выражение LIKE. В качестве аргумента оператору LIKE передается шаблон в виде строки, в которой помимо текста могут содержаться метасимволы «_» (обозначает любой одиночный символ) и «%» (набор любых символов любой длины). Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" соответствует шаблону «_user%» (т.е. строка, которая строится как любой символ + user + любая последовательность символов):

```
SELECT `content` FROM `news` WHERE `author` LIKE "_user%";
```

Такому шаблону будут удовлетворять значения атрибута вроде «1user», «1user222», «xuser42» и т.д.

3. Сортировка и ограничение результатов

Для **ограничения количества результирующих строк**, удовлетворяющих условию отбора, используется выражение LIMIT, которое дописывается в конец запроса с максимальным числом строк для вывода в качестве простейшего аргумента.

Пример для выборки информации о 5 первых новостях из таблицы "news" со значениями атрибута "id" больше 5:

```
SELECT * FROM `news` WHERE `id` > 5 LIMIT 5;
```

Для **сортировки результата отбора** предусмотрен оператор ORDER BY. Он позволяет сортировать строки в результирующей таблице в соответствии со значениями выбранных атрибутов (если атрибутов несколько, то приоритет сортировки убывает при перечислении атрибутов слева направо). Поддерживается два типа сортировки: по возрастанию (ASC, этот режим используется по умолчанию) и по убыванию (DESC).

Пример для выборки информации о 10 первых новостях из таблицы "news" со значениями атрибута "id" больше 5, отсортированной сначала по авторам ("author") в алфавитном порядке, а затем — по идентификаторам ("id") в обратном порядке:

```
SELECT * FROM `news` WHERE `id` > 5 ORDER BY `author` ASC, `id` DESC LIMIT 10;
```

4. Числовые операции (агрегирующие функции)

1. Для получения среднего арифметического значения атрибута по всем результирующим строкам предусмотрена функция AVG(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения среднего значения идентификатора ("id") новости:

```
SELECT AVG(`id`) FROM `news`;
```

2. Для поиска минимального и максимального значений атрибута среди всех результирующих строк есть функции MIN() и MAX() соответственно. В качестве единственного аргумента им передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения минимального и максимального значений идентификатора ("id") новости:

```
SELECT MIN(`id`), MAX(`id`) FROM `news`;
```

3. Для подсчета суммы числовых значений атрибута среди всех результирующих строк предусмотрена функция SUM(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения суммы значений идентификаторов ("id") всех новостей:

```
SELECT SUM(`id`) FROM `news`;
```

4. Для подсчета количества строк выборки предусмотрена функция COUNT(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Кроме того, поскольку обычно не важно, по какому атрибуту считать число результирующих строк, вместо названия атрибута в качестве аргумента может использоваться метасимвол звездочки («*»). Пример выборки для нахождения количества строк в таблице "news":

```
SELECT COUNT(*) FROM `news`;
```

Дополнительный параметр DISTINCT, указанный перед названием атрибута, позволяет вычесть из результата все вхождения по повторяющимся значениям выбранного поля. Пример выборки для нахождения количества новостей, написанных разными авторами:

```
SELECT COUNT(DISTINCT `author`) FROM `news`;
```

5. Группировка результатов

Для разбиения результатов выборки по группам используется оператор GROUP BY. Разбиение строк по группам бывает необходимо для проведения каких-либо операций не над всеми строками по отдельности, а над группами, отобранными по какому-либо атрибуту и условию.

Группировка производится по указываемому атрибуту (или набору атрибутов), и строки распределяются по группам в соответствии со значением атрибута в этой строке (каждую группу образует набор строк с одним значением атрибута, по которому производится группировка).

Пример выборки значений всех атрибутов таблицы "news" с группировкой по атрибуту "author":

```
SELECT * FROM `news` GROUP BY `author`;
```

С помощью группировки можно, например, посчитать количество новостей, добавленных каждым автором:

```
SELECT `author`, COUNT(*) FROM `news` GROUP BY `author`;
```

Для добавления условий на результат группировки в конструкцию GROUP BY добавляют выражение HAVING, работающее по аналогии с WHERE, но с группами строк.

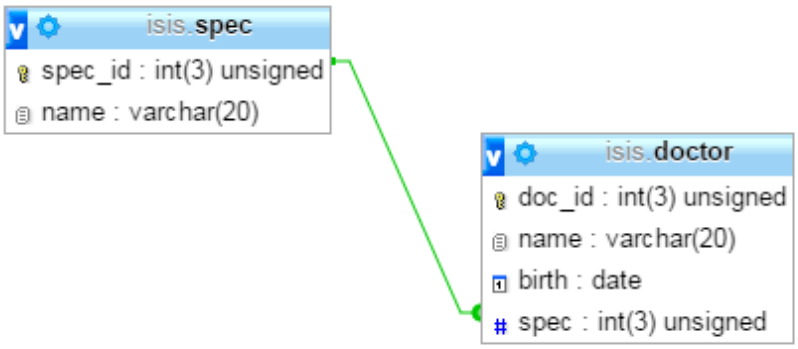
Например, к прошлому запросу с помощью HAVING можно добавить условие, по которому будут выводиться только те авторы, которые добавили более 3 новостей:

```
SELECT `author`, COUNT(*) AS `cnt` FROM `news` GROUP BY `author` HAVING `cnt` > 3;
```

Ключи

Рассмотрим базу данных из двух отношений.

- ***Специализация врача*** – ключ специализации *(первичный ключ)* и ее название;
- ***Доктор*** – ключевое поле, имя врача, дата рождения и специализация *(внешний ключ)*.



При создании таблицы ***spec*** укажем, что поле ***spec_id*** является *первичным ключом:*

```
CREATE TABLE `spec`(
`spec_id` INT(3) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
`name` VARCHAR(20)
) ENGINE=InnoDB;
```

Обратите внимание, что таблица должна иметь тип InnoDB. Это связано с тем, что данный тип единственный в MySQL, поддерживающий механизмы транзакций и внешних ключей.

При создании таблицы ***doctor*** необходимо указать, что поле ***spec*** является внешним ключом.

В общем виде синтаксис создания внешнего ключа:

```
CONSTRAINT [symbol]] FOREIGN KEY
   [index_name] (index_col_name, ...)
   REFERENCES tbl_name (index_col_name,...)
   [ON DELETE reference_option]
   [ON UPDATE reference_option]
```

Где:

- **symbol** - имя ключа
- **index_name** - имя поля в таблице, которое мы хотим сделать ключом
- **tbl_name** - таблица, с которой осуществляем связывание
- **index_col_name** - имя поля, с которым связываем наш ключ
- **reference_option** – ограничения внешнего ключа

Ограничения reference_option:

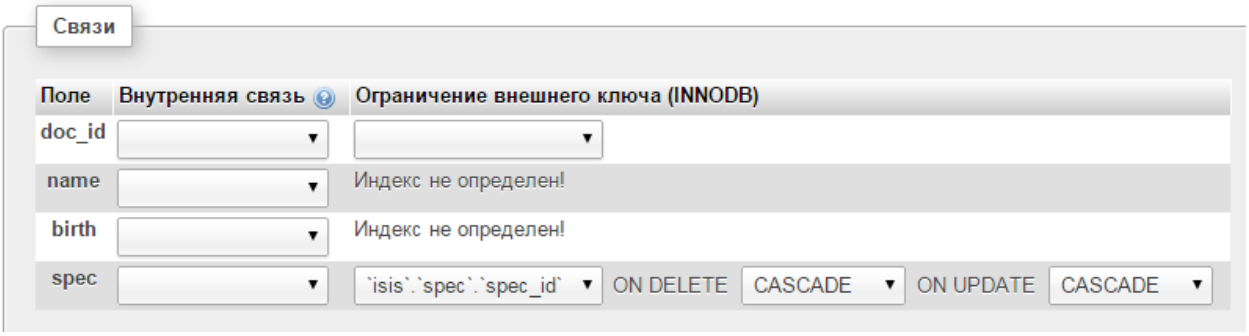
RESTRICT | CASCADE | SET NULL | NO ACTION

- **CASCADE**: если связанная запись родительской таблицы обновлена или удалена, и мы хотим чтобы соответствующие записи в таблицах-потомках также были обновлены или удалены. Что происходит с записью в родительской таблице, тоже самое произойдет с записью в дочерних таблицах.
- **SET NULL**:если запись в родительской таблице обновлена или удалена, а мы хотим чтобы в дочерней таблице некоторым занчениям было присвоено NULL (конечно если поле таблицы это позволяет)
- **NO ACTION**: смотри RESTRICT
- **RESTRICT**:если связанные записи родительской таблицы обновляются или удаляются со значениями которые уже/еще содержатся в соответствующих записях дочерней таблицы, то база данных не позволит изменять записи в родительской таблице. Обе команды **NO ACTION** и **RESTRICT** эквивалентны отсутствию подвыражений **ON UPDATE** or **ON DELETE** для внешних ключей.

Итак, создадим таблицу ***doctor*** также типа InnoDB и укажем, что поле ***spec***-внешний ключ, а также зададим ограничения целостности ON DELETE CASCADE, ON UPDATE CASCADE.

```
CREATE TABLE `doctor` (
`doc_id` INT(3) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
`name` VARCHAR(20),
`birth` DATE,
`spec` INT(3) UNSIGNED NOT NULL,
CONSTRAINT `spec` FOREIGN KEY(`spec`) REFERENCES `spec`(`spec_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoBB;
```

В утилите phpMyAdmin ограничения внешнего ключа можно задать в редакторе связей:



Заполним таблицы данными:

Таблица ***spec***:

```
INSERT INTO `isis`.`spec` (`spec_id`, `name`)
VALUES (NULL, 'Терапевт'), (NULL, 'Кардиолог'), (NULL, 'Хирург');
```

Таблица ***doctor***:

```
INSERT INTO `isis`.`doctor` (`doc_id`, `name`, `birth`, `spec`)
VALUES (NULL, 'Иванов И.И.', '1972-02-02', '1'), (NULL, 'Жулин А.К.', '1949-03-19', '1'), (NULL, 'Голубев И.А.', '1968-06-29', '1'), (NULL, 'Терезникова Н.А.', '1955-09-17', '2'), (NULL, 'Самойлова О.Т.', '1978-12-13', '2');
```

spec_id	name
1	Терапевт
2	Кардиолог
3	Хирург

doc_id	name	birth	spec
1	Иванов И.И.	1972-02-02	1
2	Жулин А.К.	1949-03-19	1
3	Голубев И.А.	1968-06-29	1
4	Терезникова Н.А.	1955-09-17	2
5	Самойлова О.Т.	1978-12-13	2

Соединения таблиц

При разработке веб-проектов с участием базы данных нам часто необходимо в запросах объединять таблицы базы, чтобы получить необходимые данные.

Основные типы объединения таблиц в MySQL:

1. CROSS JOIN, он же INNER JOIN, он же JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. NATURAL JOIN
5. Аналоги FULL OUTER JOIN для MySQL
6. Если не указать USING или ON в объединении (Декартова выборка)

В предложении **FROM** может быть указана **явная операция соединения** двух и более таблиц. Среди ряда операций соединения, описанных в стандарте языка SQL, многими серверами баз данных поддерживается только операция **соединения по предикату**. Синтаксис соединения по предикату имеет вид:

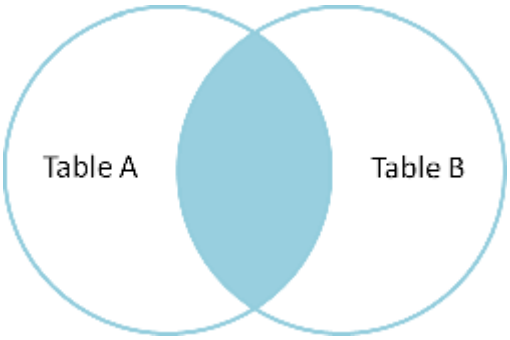
```
FROM <таблица 1>
[INNER]
{{LEFT | RIGHT | FULL } [OUTER]} JOIN <таблица 2>
[ON <предикат>]
```

Соединение может быть либо внутренним (**INNER**), либо одним из внешних (**OUTER**). Служебные слова **INNER** и **OUTER** можно опускать, поскольку внешнее соединение однозначно определяется его типом — **LEFT** (левое), **RIGHT** (правое) или **FULL** (полное), а просто **JOIN** будет означать внутреннее соединение.

Предикат определяет условие соединения строк из разных таблиц. При этом **INNER JOIN** означает, что в результирующий набор попадут только те соединения строк двух таблиц, для которых значение предиката равно **TRUE**. Как правило, предикат определяет эквисоединение по внешнему и первичному ключам соединяемых таблиц, хотя это не обязательно.

INNER JOIN

- ✓ INNER JOIN производит выборку записей, которые только существуют в TableA и TableB одновременно.
- ✓ CROSS JOIN — это эквивалент INNER JOIN.
- ✓ INNER JOIN можно заменить условием объединения в WHERE.



Пример:

```
SELECT * FROM `spec` , `doctor`
WHERE `doctor`.`spec` = `spec`.`spec_id`
```

Идентичный запрос:

Условие соединения указывается при помощи ON.

```
SELECT * FROM `doctor`
INNER JOIN `spec`
ON `spec`.`spec_id` = `doctor`.`spec`
```

Если поле имеет одинаковое название в обеих таблицах, возможно использование **USING.**

```
…
INNER JOIN `таблица`
USING (`поле`);
```

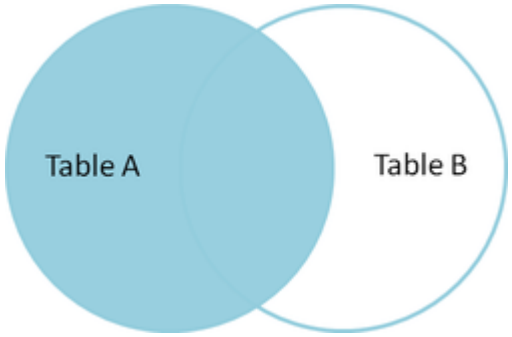
Результат:

spec_id	name	doc_id	name	birth	spec
1	Терапевт	1	Иванов И.И.	1972-02-02	1
1	Терапевт	2	Жулин А.К.	1949-03-19	1
1	Терапевт	3	Голубев И.А.	1968-06-29	1
2	Кардиолог	4	Терезникова Н.А.	1955-09-17	2
2	Кардиолог	5	Самойлова О.Т.	1978-12-13	2

Таким образом, соединяются только те строки таблицы, у которых в указанных столбцах находятся равные значения (***эквисоединение***). В таблице нет ни одного доктора со специализацией 3, т.е. хирург, поэтому **INNER JOIN** для него не сработал. Если разные таблицы имеют столбцы с одинаковыми именами, то для однозначности требуется использовать точечную нотацию, которая называется уточнением имени столбца: **<имя таблицы>.<имя столбца>.** В тех случаях, когда это не вызывает неоднозначности, использование данной нотации не является обязательным.

LEFT JOIN

LEFT OUTER JOIN (LEFT JOIN) указывает, что левая таблица управляющая (в нашем случае TableA) и производит из нее полную выборку, осуществляя поиск соответствующих записей в таблице TableB. Если таких соответствий не найдено, то база вернет пустой указатель - NULL. Указание OUTER - необязательно.



Запрос:

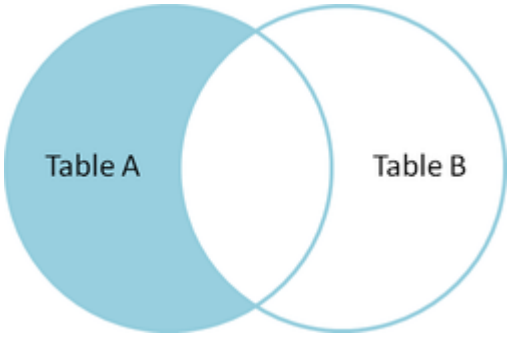
```
SELECT * FROM `spec`
LEFT JOIN `doctor`
ON `spec`.`spec_id` = `doctor`.`spec`
```

Результат:

spec_id	name	doc_id	name	birth	spec
1	Терапевт	1	Иванов И.И.	1972-02-02	1
1	Терапевт	2	Жулин А.К.	1949-03-19	1
1	Терапевт	3	Голубев И.А.	1968-06-29	1
2	Кардиолог	4	Терезникова Н.А.	1955-09-17	2
2	Кардиолог	5	Самойлова О.Т.	1978-12-13	2
3	Хирург	*NULL*	*NULL*	*NULL*	*NULL*

Для полей таблицы **spec** (терапевта и кардиолога) были найдены соответствия в таблице **doctor**, а для хирурга не было найдено ни одного соответствия, поэтому был возвращен NULL.

Чтобы произвести выборку записей из таблицы TableA, которых не существует в таблице TableB, мы выполняем LEFT JOIN, но затем из результата исключаем записи, которые не хотим видеть, путем указания, что TableB.id является нулем (указывая, что записи нет в таблице TableB).



Запрос:

```
SELECT *
FROM `spec`
LEFT JOIN `doctor` ON `spec`.`spec_id` = `doctor`.`spec`
WHERE `doctor`.`spec` IS NULL
```

Результат:

spec_id	name	doc_id	name	birth	spec
3	Хирург	*NULL*	*NULL*	*NULL*	*NULL*

RIGHT JOIN

RIGHT JOIN выполняет те же самые функции, что и LEFT JOIN, за исключением того, что правая таблица будет прочитана первой. Таким образом, если в предыдущих запросах LEFT заменить на RIGHT, то таблица результатов, грубо говоря, отразится по вертикали. То есть, в результате вместо значений TableA будут записи TableB и наоборот.

NATURAL JOIN

Суть этой конструкции в том, что база сама выбирает, по каким столбцам сравнивать и объединять таблицы. А выбор этот падает на столбцы с одинаковыми именами. С одной стороны, это весьма удобно, но с другой стороны, база может выбрать совершенно не те столбцы для объединения и запрос будет работать совершенно не так, как вы предполагали. Нет гарантии того, что столбцы с одинаковыми именами в таблицах будут именно ключевыми и предназначены для объединения. **NATURAL JOIN** ухудшает читаемость кода, так как разработчик не сможет по запросу определить, как объединяются таблицы. Поэтому, обращая внимание на такие факторы, **NATURAL JOIN использовать не рекомендуется.** Рассмотрим примеры.

Запрос:

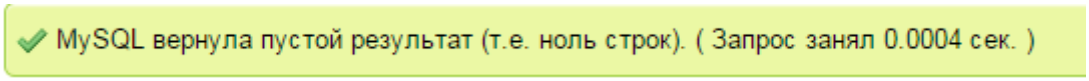
```
SELECT * FROM `doctor`
NATURAL JOIN `spec`
```

В этом случае СУБД выполняет объединение таблиц по столбцу ***name***, так как он присутствует в обеих таблицах, что идентично следующему запросу:

```
SELECT * FROM `doctor`
INNER JOIN `spec`
USING ( `name` )
```

Результат:

Поскольку не существует записей с одинаковым полем name одновременно в обеих таблицах, запрос вернет пустой результат.



Если попробовать изменить запрос следующим образом.

Запрос:

```
SELECT `spec` . * , `doctor` . *
FROM `doctor`
NATURAL RIGHT JOIN `spec`
```

Такой запрос приводится к следующему:

```
SELECT `spec` . * , `doctor` . *
FROM `doctor`
RIGHT JOIN `spec`
USING ( `name` )
```

Результат:

spec_id	name	doc_id	name	birth	spec
1	Терапевт	*NULL*	*NULL*	*NULL*	*NULL*
2	Кардиолог	*NULL*	*NULL*	*NULL*	*NULL*
3	Хирург	*NULL*	*NULL*	*NULL*	*NULL*

Происходит это так: так как правая таблица управляющая (т.е. таблица ***spec***), то она читается первой и полностью выбирается, независимо от левой (***doctor)*** таблицы; когда начинается поиск соответствующих записей в левой таблице, то СУБД не находит ни одной записи, которая была бы идентична по ***name***, поэтому возвращаются пустые указатели.

Для более подробного понимания работы NATURAL JOIN изменим имя доктора на Терапевт, а затем выполним те же самые запросы.

```
UPDATE `isis`.`doctor` SET `name` = 'Терапевт' WHERE `doctor`.`doc_id` =1;
```

Запрос:

```
SELECT * FROM `doctor`
NATURAL JOIN `spec`
```

Результат:

name	doc_id	birth	spec	spec_id
Терапевт	1	1972-02-02	1	1

Выводится одна запись, поскольку теперь в обеих таблицах существует запись с одинаковым значением поля ***name.***

Запрос:

```
SELECT `spec` . * , `doctor` . *
FROM `doctor`
NATURAL RIGHT JOIN `spec`
```

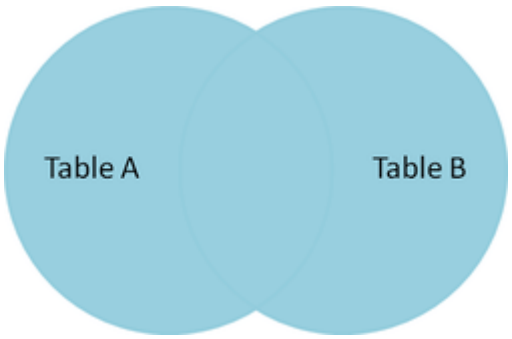
Возвращает результат:

spec_id	name	doc_id	name	birth	spec
1	Терапевт	1	Терапевт	1972-02-02	1
2	Кардиолог	*NULL*	*NULL*	*NULL*	*NULL*
3	Хирург	*NULL*	*NULL*	*NULL*	*NULL*

FULL OUTER JOIN

***Недоступно в MySQL**

FULL OUTER JOIN производит выборку всех записей из TableA и TableB, вне зависимости есть ли соответствующая запись в соседней таблице. Если таковой нет, то недостающая сторона будет содержать пустой указатель и результатом будет выводится NULL.



Декартова выборка

Если при объединении таблиц не указать условие объединения через ON или USING, то база произведет так называемую Декартову выборку, когда значению одной таблицы приравнивается каждое значение другой. Таким образом, СУБД, в нашем случае, возвращает 3x5 = 15 строк.

Запрос:

```
SELECT *
FROM `doctor`
JOIN `spec`
```

Результат:

doc_id	name	birth	spec	spec_id	name
1	Иванов И.И.	1972-02-02	1	1	Терапевт
1	Иванов И.И.	1972-02-02	1	2	Кардиолог
1	Иванов И.И.	1972-02-02	1	3	Хирург
2	Жулин А.К.	1949-03-19	1	1	Терапевт
2	Жулин А.К.	1949-03-19	1	2	Кардиолог
2	Жулин А.К.	1949-03-19	1	3	Хирург
3	Голубев И.А.	1968-06-29	1	1	Терапевт
3	Голубев И.А.	1968-06-29	1	2	Кардиолог
3	Голубев И.А.	1968-06-29	1	3	Хирург
4	Терезникова Н.А.	1955-09-17	2	1	Терапевт
4	Терезникова Н.А.	1955-09-17	2	2	Кардиолог
4	Терезникова Н.А.	1955-09-17	2	3	Хирург
5	Самойлова О.Т.	1978-12-13	2	1	Терапевт
5	Самойлова О.Т.	1978-12-13	2	2	Кардиолог
5	Самойлова О.Т.	1978-12-13	2	3	Хирург

Объединения запросов

UNION

Объединением двух множеств называется множество всех элементов, принадлежащих какому-либо одному или обоим исходным множествам.

Результатом **объединения** будет множество, состоящее из всех строк, входящих в какое-либо одно или в оба первоначальных отношения. Однако, если этот результат сам по себе должен быть другим отношением, а не просто разнородной смесью строк, то два исходных отношения должны быть совместимыми по объединению, т. е. строки в обоих отношениях должны быть одной и той же формы. Что касается **SQL**, то две таблицы совместимы по объединению (и к ним может быть применен оператор объединения **UNION**) тогда и только тогда, когда:

- Они имеют одинаковое число полей (например, m);
- Для всех i (i = 1, 2, …, m) i-е поле первой таблицы и i-е поле второй таблицы имеют в точности одинаковый тип данных.

Пример:

```
(
SELECT * FROM `doctor`
WHERE `name` LIKE '%ов%'
)
UNION
(
SELECT * FROM `doctor`
WHERE `name` LIKE 'Ж%'
)
```

В данном абстрактном примере **UNION** используется для объединения результатов работы нескольких команд SELECT в один набор результатов. Круглые скобки используются для визуального разделения запросов и обязательны при использовании операторов ORDER BY и др.

Результат:

doc_id	name	birth	spec
1	Иванов И.И.	1972-02-02	1
4	Терезникова Н.А.	1955-09-17	2
5	Самойлова О.Т.	1978-12-13	2
2	Жулин А.К.	1949-03-19	1

UNION ALL и UNION DISTINCT

Если не используется ключевое слово ALL для UNION, все возвращенные строки будут уникальными, так как по умолчанию подразумевается DISTINCT для всего результирующего набора данных. Если указать ключевое слово ALL, то результат будет содержать все найденные строки из всех примененных команд SELECT.

Пример:

```
(
SELECT *
FROM `doctor`
WHERE `name` LIKE '%ов%'
)
UNION ALL (
SELECT *
FROM `doctor`
WHERE `name` LIKE 'И%'
)
```

Результат:

doc_id	name	birth	spec
1	Иванов И.И.	1972-02-02	1
4	Терезникова Н.А.	1955-09-17	2
5	Самойлова О.Т.	1978-12-13	2
1	Иванов И.И.	1972-02-02	1

Подзапросы

При работе с базой данных может возникнуть потребность в запросе, который зависел бы от результата другого запроса. Подзапрос - это запрос, результат которого используется в условии другого запроса. Основные преимущества подзапросов:

- Они позволяют писать *структурированные* запросы таким образом, что можно изолировать части оператора.
- Они представляют альтернативный способ выполнения операций, которые требуют применения сложных соединений и слияний (JOIN и UNION).
- По мнению многих, они более читабельны.

Обратите внимание, что такого рода подзапросы не могут использоваться с SELECT *, поскольку они выбирают лишь одиночный столбец.

Пример:

```
SELECT * FROM `doctor`
WHERE `spec` = ( SELECT `spec_id` FROM `spec` WHERE `name` LIKE 'К%' )
```

Результат:

	doc_id	name	birth	spec
Изменить Копировать Удалить	4	Терезникова Н.А.	1955-09-17	2
Изменить Копировать Удалить	5	Самойлова О.Т.	1978-12-13	2

Таким образом, выполняется выборка всех врачей со специализацией, начинающейся на букву К. Подзапрос используется для поиска spec_id, т.е. идентификатора специализации

на букву К. Затем найденный идентификатор служит условием поиска врачей с такой специализацией. На подзапросы можно налагать дополнительные ограничения, такие как:

- EXISTS
- SOME/ANY
- ALL

EXISTS

Используется, чтобы указать предикату, производит ли подзапрос вывод. Возвращает булево значение.

Пример:

```
SELECT * FROM `doctor`
WHERE EXISTS (
SELECT *
FROM `spec`
WHERE `name` LIKE 'Хирург'
)
```

Результат:

	doc_id	name	birth	spec
Изменить Копировать Удалить	1	Иванов И.И.	1972-02-02	1
Изменить Копировать Удалить	2	Жулин А.К.	1949-03-19	1
Изменить Копировать Удалить	3	Голубев И.А.	1968-06-29	1
Изменить Копировать Удалить	4	Терезникова Н.А.	1955-09-17	2
Изменить Копировать Удалить	5	Самойлова О.Т.	1978-12-13	2

Оператор EXISTS во внешнем предикате отмечает, что извлекать данные о врачах следует только в случае, если существует запись Хирург. В противном случае будет возвращен пустой результат.

SOME/ANY

Операторы SOME/ANY (взаимозаменяемые — различие в терминологии состоит в том, чтобы позволить людям использовать тот термин, который наиболее однозначен).

Оператор ANY берет все значения выведенные подзапросом, причем такого же типа, как и те, которые сравниваются в основном предикате. В этом его отличие от EXISTS, который просто определяет, производит ли подзапрос результаты, и фактически не использует эти результаты.

Пример:

```
SELECT * FROM `doctor`
WHERE `spec` = ANY(
```

```
SELECT `spec_id`
FROM `spec`
WHERE `spec_id` <3
)
```

Результат:

	doc_id	name	birth	spec
Изменить Копировать Удалить	1	Иванов И.И.	1972-02-02	1
Изменить Копировать Удалить	2	Жулин А.К.	1949-03-19	1
Изменить Копировать Удалить	3	Голубев И.А.	1968-06-29	1
Изменить Копировать Удалить	4	Терезникова Н.А.	1955-09-17	2
Изменить Копировать Удалить	5	Самойлова О.Т.	1978-12-13	2

В данном случае оператор ANY берет все значения, выведенные подзапросом (все записи с идентификатором < 3) и оценивает их как верные, если любой из них равняется идентификатору текущей строки внешнего запроса.

ALL

Предикат является верным, если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса.

Пример:

```
SELECT * FROM `doctor`
WHERE `spec` = ALL (
SELECT `spec_id`
FROM `spec`
WHERE `name` LIKE '%or'
)
```

Результат:

	doc_id	name	birth	spec
Изменить Копировать Удалить	4	Терезникова Н.А.	1955-09-17	2
Изменить Копировать Удалить	5	Самойлова О.Т.	1978-12-13	2

Данный запрос позволяет найти всех врачей, у которых название специализации заканчивается на «ог».

Представления

Представление — запрос на выборку, сохраненный в базе данных под каким-то названием, виртуальная таблица. Данные представления динамически рассчитываются по

запросу из данных реальных таблиц, но структура (поля) результата запроса не меняются при изменении исходных таблиц.

Плюсы использования виртуальных таблиц:

1. Повышение скорости работы. Когда прикладной программе требуется таблица с определённым набором данных, она делает простейший запрос из подготовленного представления. Поскольку SQL-запрос, выбирающий данные представления, зафиксирован на момент его создания, СУБД получает возможность применить к этому запросу оптимизацию или предварительную компиляцию, что положительно сказывается на скорости обращения к представлению по сравнению с прямым выполнением того же запроса из прикладной программы.
2. Независимая модификация прикладной программы и схемы хранения данных.
3. Повышение безопасности. За счет предоставления пользователю только тех данных, на которые он имеет права — от него скрыта реальная структура таблиц базы данных.

Может возникнуть такая ситуация, что в исходных таблицах есть обязательные и не пустые (NOT NULL) поля, а в представлении их нет. Тогда операция добавления строки данных не может быть выполнена за одно действие. Во избежание потерь данных подобных действий следует избегать.

Работа с представлениями:

Создание:

```
CREATE VIEW `name` [ FIELDS ] AS { запрос };
```

Удаление:

```
DROP VIEW `name`;
```

Изменение:

```
ALTER VIEW `name` [ FIELDS ] AS { новый запрос }
```

Задания к лабораторной работе

Внимание! Все запросы необходимо сохранять в БД и делать скрины запросов и результатов их работы.

1. Создайте базу данных, отражающую предметную область электронной библиотеки. Необходимо реализовать хранение информации об авторах (ФИО, дата рождения, дата смерти и т.д.), книгах (название, описание, год написания и т.д), жанрах (название жанра и пр.).

Для упрощения задачи будем считать, что каждая книга написана только в одном жанре (например, только детектив, или только исторический роман), и каждая книга принадлежит перу только одного автора (т.е. соавторство не учитывается).

2. Реализуйте связи между таблицами с использованием PRIMARY и FOREIGN KEYS.
3. Заполните таблицы данными.
4. Выполните следующие запросы:

   В запросах указан общий шаблон. То есть, если у вас нет автора Льва Толстого, можете использовать любого другого автора и т.д.

   1) Найти книги, написанные в последнее десятилетие XX века. Вывести название книги, ее жанр и год написания.
   2) Определить количество книг каждого автора. Вывести имя автора и количество книг.
   3) Найти все книги, написанные Агатой Кристи и отсортировать их по годам написания, начиная с самого последнего. Вывести название книги и год написания.
   4) Добавьте 2 новые записи в таблицу жанр, не имеющих связей в других таблицах. Выполните запрос на поиск книг в каждом жанре. Добавленные жанры также должны присутствовать в таблице, несмотря на то, что для них соответствий не найдется. Вывести название жанра и название книги.
   5) Выведите таблицу вида: название книги, ее автор и жанр, в котором она написана.
   6) Найдите всех авторов, написавших книги в жанре фэнтези. Вывести только имена авторов.
   7) Найдите автора книги Русалочка.
   8) Вывести все жанры, в которых нет ни одной книги.
   9) Найти книги, авторы которых родились в 20 веке. Вывести названия книг и их описание.
   10) Определите, сколько лет прошло между написанием первой и последней книги Льва Толстого.
   11) Объедините два запроса: 1 – Поиск авторов, чье имя начинается на букву А и 2 – Поиск авторов, которые родились в 1972 году.
   12) Создайте представление по одному из запросов и затем выведите из него информацию по произвольному критерию.
   13) Измените год написания книги Война и мир.
   14) Удалите все книги, написанные в жанре детектив