



# Курс базы данных и SQL. Лекция 2



На второй лекции будем рассматривать создание объектов: БД, таблиц. Разберемся с типами данных. Узнаем, как с помощью MySQL заполнить таблицу данными; пользоваться логическими операторами и операторами CASE, IF

## Что мы узнаем:

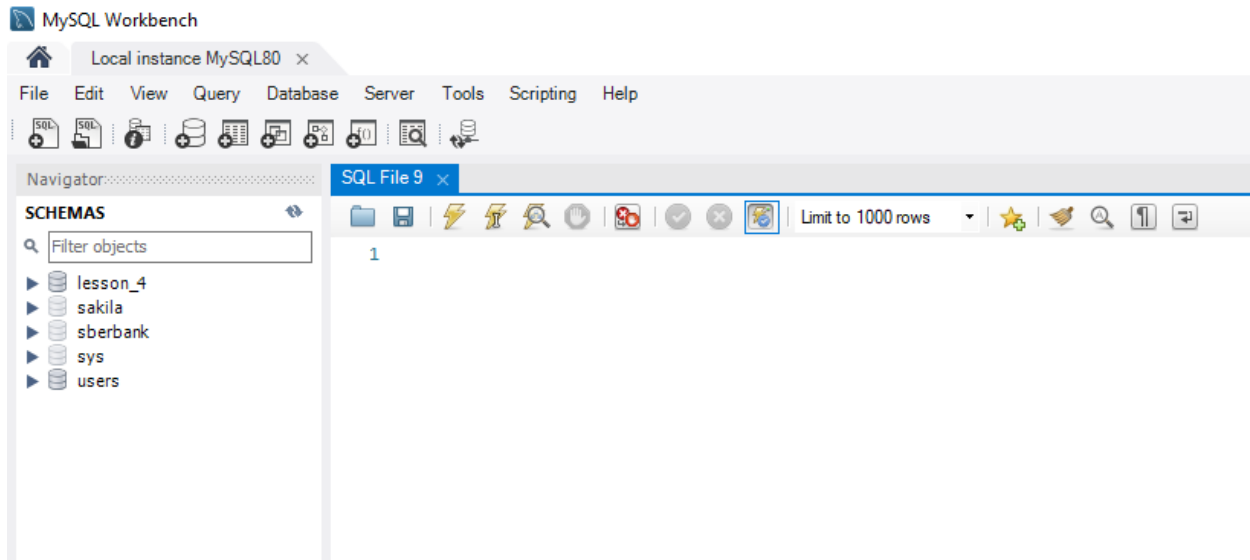
- Группа операторов: DDL (create table, PK, FK)
- Типы данных
- Комментарии
- Группа операторов: DML (insert, update, delete, select)
- Арифметические операции
- Логические операторы (and, or, between, not, in)
- Оператор CASE, IF

Доброго времени суток, уважаемые студенты! Сегодня мы продолжим изучать SQL, затронув следующие операторы DDL, DML.

Начнем с написания SQL - скриптов: создадим тестовую БД и парочку таблиц. Итак, создадим первый файл для скрипта: кнопка находится под кнопкой "File" в левой верхней части экрана.



После нажатия в верхнем поле

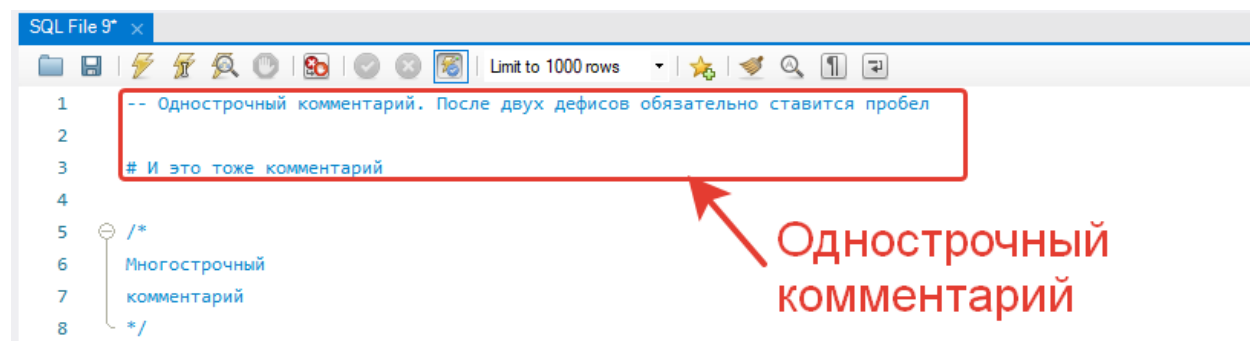


## Комментарии

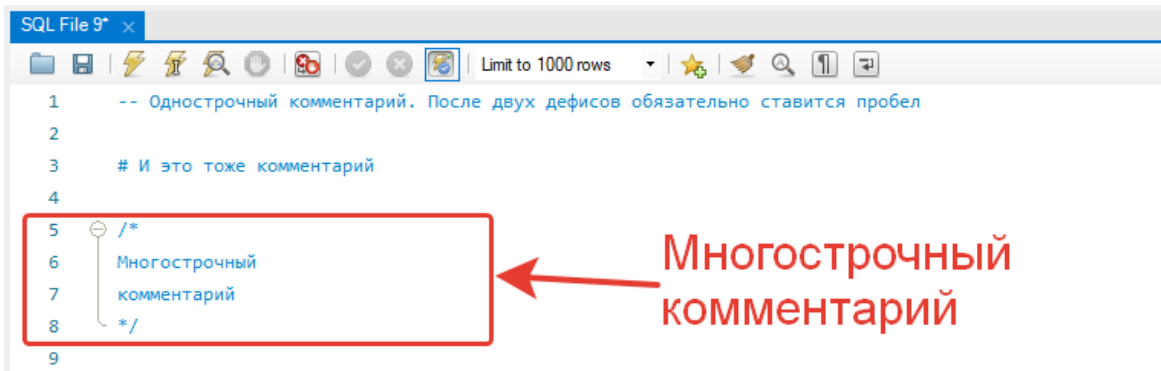
Прежде чем написать наши SQL - запросы, рассмотрим, как же написать комментарии. Комментарии позволяют лучше понимать логику SQL - запроса. Служебные строки бывают двух видов:

- Однострочные комментарии, обозначаемые двумя дефисами и знаком "решетки"
- Многострочные комментарии начинаются с сочетания символов /\* и заканчиваются символами \*/.

Пример однострочного комментария представлен на первой и второй строчке.



После двух дефисов обязательно нужно ставить "пробел".



## Создание базы данных

Для создания базы используется SQL-запрос CREATE DATABASE. Рассмотрим подробнее его использование. Новая база данных создается с помощью оператора SQL: CREATE DATABASE, за которым следует имя создаваемой базы данных. Для этой цели также используется оператор CREATE SCHEMA. Например, для создания новой базы данных под названием **lesson\_2** в командной строке mysql нужно ввести следующий запрос:

```
CREATE DATABASE lesson_2; -- В конце ставится точка с запятой - завершение оператора
```

Если все прошло нормально, команда сгенерирует следующий вывод:

**Query OK, 1 row affected (0.00 sec)**

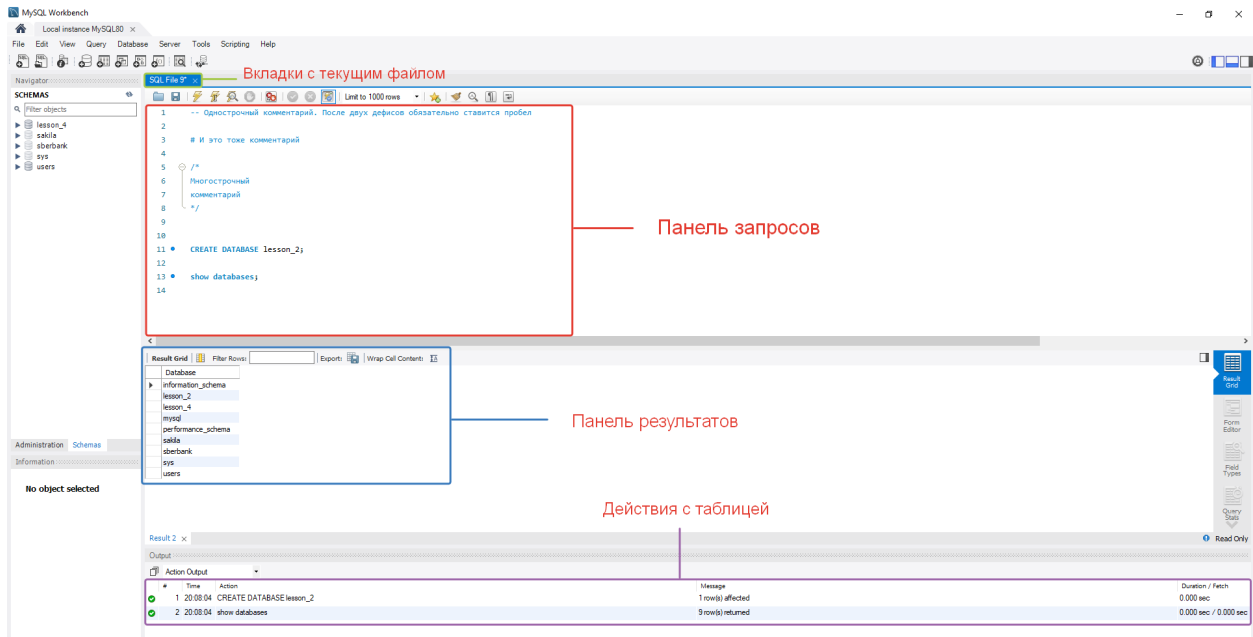
Если указанное имя базы данных конфликтует с существующей базой данных MySQL, будет выведено сообщение об ошибке:

**ERROR 1007 (HY000): Can't create database 'lesson\_2'; database exists**

Проверить, что база появилась можно командой:

```
SHOW DATABASES;
```

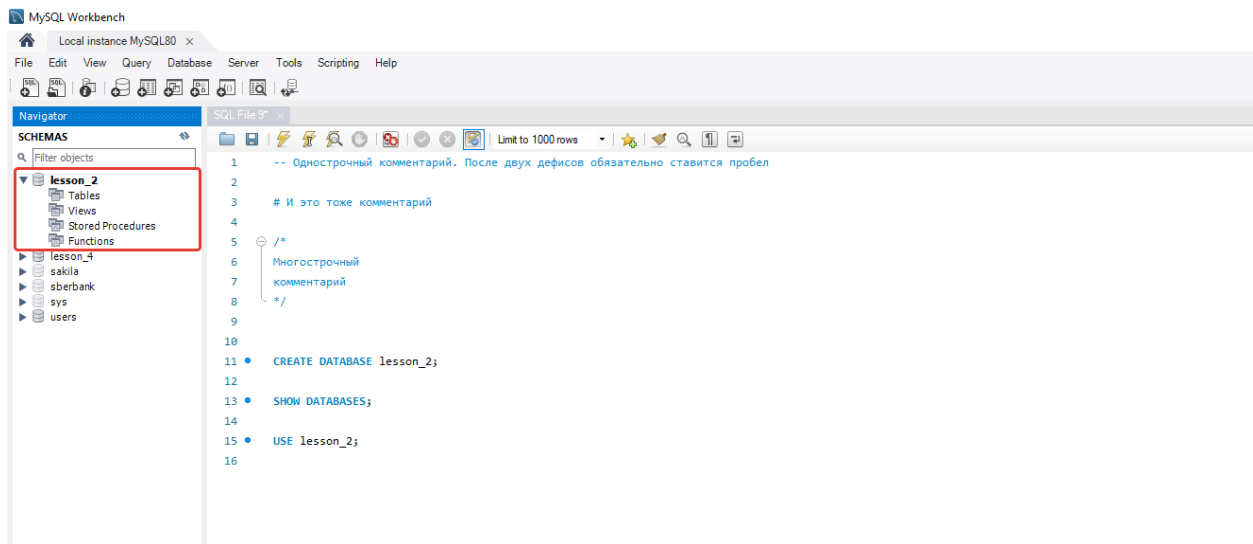
- данная команда выводит в консоль список баз, созданных в СУБД.



Подключиться к базе можно командой:

```
USE lesson_2;
```

Данная команда подключается к базе данных с именем **lesson\_2** из списка созданных баз. Если база данных была успешно подключена, то в левой части экрана черным цветом подсвечивается активная БД.



## Создание таблиц:

Реляционные базы данных хранят данные в таблицах, внутри которой содержатся некоторые столбцы. Каждый столбец имеет имя и тип данных. У столбца есть название и тип данных. Для создания таблиц используется команда **CREATE TABLE**:

```
CREATE TABLE table_name
(
    column_name_1 column_type_1,

    column_name_2 column_type_2,

    column_name_N column_type_N,

);
```


Имя столбца и таблицы придумать мы можем, но какие же типы данных бывают? Давайте разбираться, какие возможные типы данных можно применять для создания таблиц. После типов данных закрепим наши знания, заполнив БД, как минимум, 2 таблицами. Пример будет немного ниже.

## Типы данных

Более детально получить информацию о типах данных можно узнать по ссылке:

MySQL :: MySQL 8.0 Reference Manual :: 11 Data Types

MySQL supports SQL data types in several categories: numeric types, date and time types, string (character and byte) types, spatial types, and the data type. This chapter provides an overview and more detailed description of the properties of the types in each category, and a summary of the data type storage requirements.

 <https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

Для каждого столбца таблицы будет определен тип данных. Неправильное использование типов данных увеличивает как объем занимаемой памяти, так и время выполнения запросов к таблице. Это может быть незаметно на таблицах в несколько строк, но очень существенно, если количество строк будет измеряться десятками и сотнями тысяч, и это далеко не предел для рабочей базы данных. Проведем краткий обзор наиболее часто используемых типов. В MySQL все типы данных делятся на несколько классов: числовые типы, символьные, дата/время и так далее. В каждом классе есть несколько типов данных, которые внешне могут быть похожи, но их поведение или принципы хранения отличаются. Важно выбрать правильный тип сразу при создании таблицы, потому что потом готовую структуру и приложения будет сложнее переделать.

### Числовые типы

**INT** — целочисленные значения от -2147483648 до 2147483647, 4 байта.

**DECIMAL** — хранит числа с заданной точностью. Использует два параметра — максимальное количество цифр всего числа (precision) и количество цифр дробной части (scale). Рекомендуемый тип данных для работы с валютами и координатами. Можно использовать синонимы NUMERIC, DEC, FIXED.

**TINYINT** — целые числа от -127 до 128, занимает 1 байт хранимой памяти.

**BOOL** — 0 или 1. Однозначный ответ на однозначный вопрос — false или true. Название столбцов типа boolean часто начинается с is, has, can, allow. По факту это даже не отдельный тип данных, а псевдоним для типа TINYINT (1). Тип настолько востребован на практике, что для него в MySQL создали встроенные константы FALSE (0) или TRUE (1). Можно использовать синоним BOOLEAN.

**FLOAT** — дробные числа с плавающей запятой (точкой).

### Символьные

**VARCHAR(N)** — N определяет максимально возможную длину строки. Создан для хранения текстовых данных переменной длины, поэтому память хранения зависит от длины строки. Наиболее часто используемый тип строковых данных.

**TEXT** — подходит для хранения большого объема текста до 65 KB, например, целой статьи.

**CHAR** - строка фиксированной длины. Длина хранимой строки указывается в скобках, например, CHAR(10) - строка из десяти символов. И если в таблицу в данный столбец сохраняется строка из 6 символов (то есть меньше установленной длины в 10 символов), то строка дополняется 4 пробелами и в итоге все равно будет занимать 10 символов

### Дата и время

**DATE** — только дата. Диапазон от 1000-01-01 по 9999-12-31. Подходит для хранения дат рождения, исторических дат, начиная с 11 века. Память хранения — 3 байта.

**TIME** — только время — часы, минуты, секунды — «hh:mm:ss». Память хранения — 3 байта.

**DATETIME** — соединяет оба предыдущих типа — дату и время. Использует 8 байтов памяти.

**TIMESTAMP** — хранит дату и время начиная с 1970 года. Подходит для большинства бизнес-задач. Потребляет 4 байта памяти, что в два раза меньше, чем DATETIME, поскольку использует более скромный диапазон дат.

### Бинарные

Используются для хранения файлов, фото, документов, аудио и видеоконтента. Все это хранится в бинарном виде.

**BLOB** — до 65 KB бинарных данных

**LARGEBLOB** — до 4 ГБ.

Отдельно используется NULL. NULL соответствует понятию «пустое поле»null, то есть «поле, не содержащее никакого значения». Введено для того, чтобы различать в полях БД пустые (визуально не отображаемые) значения (например, строку нулевой длины) и отсутствующие значения (когда в поле не записано вообще никакого значения, даже пустого). NULL означает отсутствие, неизвестность информации.

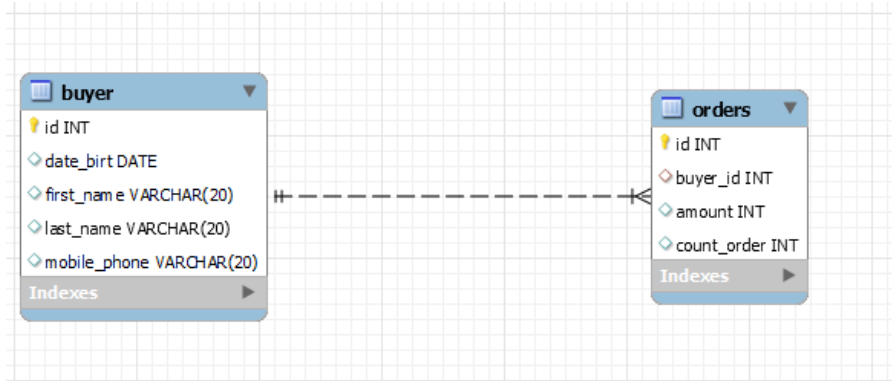
Типы данных изучены и можем приступить к созданию таблиц. Таблицы будем соединять с помощью внешнего ключа.

## Создание таблицы: первичные и внешние ключи

### Первичные ключи PRIMARY KEY

Атрибут PRIMARY KEY задает первичный ключ таблицы. Первичный ключ уникально идентифицирует строку в таблице. Первичный ключ может представлять любой тип данных. Создадим тестовую БД, заполнив ее двумя таблицами: представим, что у нас есть покупатели, оформившие заказ в Интернет-магазине. Создадим две сущности: “Покупатель”, “Заказы”.

ER - диаграмма и связи: <https://habr.com/ru/post/440556/>



Создадим таблицу “Покупатель”. Он обладает следующими атрибутами (столбцами): фамилия, имя, дата рождения, мобильный телефон.

```
CREATE TABLE buyer
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  date_birt DATE,
  first_name VARCHAR(20),
  last_name VARCHAR(20),
  mobile_phone VARCHAR(20)
);
```

Атрибут AUTO\_INCREMENT позволяет указать, что значение столбца будет автоматически увеличиваться при добавлении новой строки. Данный атрибут работает для столбцов, которые представляют целочисленный тип или числа с плавающей точкой.

Для второй таблицы мы будем использовать связь путем добавления внешнего ключа.

### Внешние ключи FOREIGN KEY

Внешние ключи позволяют установить связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы.

Общий синтаксис установки внешнего ключа на уровне таблицы:

```
FOREIGN KEY (столбец)
REFERENCES главная_таблица (столбец_главной_таблицы)
```

Для создания ограничения внешнего ключа после FOREIGN KEY указывается столбец таблицы, который будет представляет внешний ключ. А после ключевого слова REFERENCES указывается имя связанной таблицы, а затем в скобках имя связанного столбца, на который будет указывать внешний ключ.

### Пример схемы базы данных (DDL):

```
CREATE TABLE buyer
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  date_birt DATE,
  first_name VARCHAR(20),
  last_name VARCHAR(20),
  mobile_phone VARCHAR(20)
```

```
);
CREATE TABLE orders
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  buyer_id INT,
  amount INT,
  count_order INT,
  manufacturer VARCHAR(45),
  FOREIGN KEY (buyer_id)
  REFERENCES Buyer(id)
);
```

В нашем случае созданы две таблицы: “Buyer” и “Orders”. Таблица “Заказы” связана с таблицей “Покупатели” через внешний ключ: “buyer\_id” ссылается на таблицу “Buyer” поле “id”.

Если вдруг наша БД не актуальна, то можно ее удалить с помощью команды:

```
DROP объект имя_объекта;
```

```
DROP DATABASE Test; -- Удалить БД с именем Test
```

Удаление таблицы:

```
DROP TABLE Test; -- Удалить таблицу с именем Test
```

Переименовать таблицу можно с помощью команды RENAME:

```
RENAME TABLE old_name TO new_name;
```

```
RENAME TABLE buyer TO customer;
-- Изменить имя таблицы "buyer" на "customer"
```

Небольшая база данных создана, давайте наполним ее первыми данными - следующая часть группы операторов - DML (insert, update, delete).

## DML (insert, update, delete, select)

Первая команда, которую мы изучим - заполнение таблицы данными.

- **INSERT – вставка новых данных**

Данный оператор имеет 2 основные формы:

```
-- Пусть имеются 2 столбца в таблице Table: column1, column2
-- 1. Заполняется только 1 столбец в таблице Table
INSERT Table (column1)
VALUES (value1);
-- 2. Заполняются все столбцы в таблице Table.
INSERT Table
VALUES (value1, value2);
```



Заполним нашу таблицу “Покупатели” данными:

```
-- DATE - format YYYY-MM-DD
-- DATETIME - format: YYYY-MM-DD HH:MI:SS
-- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
-- YEAR - format YYYY or YY
-- Сносок №1
INSERT buyer (date_birt, first_name,last_name,mobile_phone)
VALUES
  ("2023-01-01", "Михаил", "Меркушов", "+7-999-888-77-66"), -- id = 1
  ("2022-12-31", "Сергей", "Сергеев", "60-70-80"), -- id = 2
  ("2022-12-30", "Том", "Круз", "80-70-80"), -- id = 3
  ("2022-01-02", "Филл", "Поляков", "+7-999-888-77-55"); -- id = 4
```

```
-- DATE - format YYYY-MM-DD
-- DATETIME - format: YYYY-MM-DD HH:MI:SS
-- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
-- YEAR - format YYYY or YY
-- Сносок №2
INSERT buyer
VALUES
  (1, "2023-01-01", "Михаил", "Меркушов", "+7-999-888-77-66"),
  (2, "2022-12-31", "Сергей", "Сергеев", "60-70-80"),
  (3, "2022-12-30", "Том", "Круз", "80-70-80"),
  (4, "2022-01-02", "Филл", "Поляков", "+7-999-888-77-55");
```

Покупатели сделали заказы в нашем магазине. Чтобы увидеть их, создадим таблицу с заказами клиентов:

```
INSERT orders (buyer_id, amount,count_order, manufacter)
VALUES
  (1, 1000, 3, "Ягодки"),-- Первый заказ из "Покупатели" по id = 1 (Меркушов Михал)
  (1, 400 , 2, "Амазон"),-- Второй заказ из "Покупатели" по id = 2 (Меркушов Михал)
  (2, 1200 , 5, "Амазон"),
  (3, 2000 , 1, "Ягодки"),
  (4, 5000 , 4, "Ягодки");

-- Получается, что Меркушов Михаил сделал 2 заказа в 1 день
```

Попробуем увидеть эту связь среди таблиц:

```
SELECT buyer.first_name, buyer.id, orders.buyer_id, orders.amount
FROM orders, Buyer
WHERE orders.buyer_id = buyer.id;
```

	first_name	id	buyer_id	amount
►	Михаил	1	1	1000
	Михаил	1	1	400
	Сергей	2	2	1200
	Том	3	3	2000
	Филл	4	4	5000

Получается, что в операторе SELECT мы указали, какие данные и из какой таблицы мы получаем. Запись “Buyer.id” позволяет получить столбец “id” из таблицы “Buyer”, “Orders.buyer\_id” позволяет получить столбец “buyer\_id” из таблицы “Orders”. Чтобы соединить две таблички, мы использовали равенство ключей: если внешний ключ равняется первичному, значит, наш клиент имеет как минимум 1 заказ. В SQL для

оптимизации больших запросов применяются “псевдонимы”. Они упрощают читаемость кода и немного сокращают его:

- Псевдонимы - временное имя.
- Псевдонимы делает имена столбцов более удобочитаемыми.
- Псевдоним существует только на время выполнения запроса.

Как же добавить псевдонимы? Рассмотрим простые примеры:

## 1. Псевдоним столбца

### 1.1. Задается с помощью команды “AS”:

```
-- Посчитаем чек по заказу. Для этого умножаю количество на цену:  
SELECT amount * count_order AS result -- Псевдоним - result  
FROM orders;
```

	id	buyer_id	amount	count_order
▶	1	1	1000	3
	2	1	400	2
	3	2	1200	5
	4	3	2000	1
	5	4	5000	4
*	NULL	NULL	NULL	NULL

	result
▶	3000
	800
	6000
	2000
	20000

Если псевдонима не будет, то столбец с результатом назывался бы “amount \* count\_order”

	amount * count_order
▶	3000
	800
	6000
	2000
	20000

### 1.2. Задается с помощью пробела после имени столбца:

```
SELECT amount * count_order result -- Псевдоним - result  
FROM Orders;
```

После задания столбца можно указать его псевдоним.

## 2. Псевдоним таблицы - задается аналогично: с помощью пробела и слова AS:

```
-- 1  
SELECT B.first_name, B.id, O.buyer_id, O.amount  
FROM Orders O, Buyer B  
WHERE O.buyer_id = B.id;
```

```
-- 2  
SELECT B.first_name, B.id, O.buyer_id, O.amount
```

```
FROM Orders AS O, Buyer AS B
WHERE O.buyer_id = B.id;
```

- **UPDATE - обновление данных**

Команда применяется для обновления уже имеющихся строк:

```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2
[WHERE условие_обновления];
```

Представим, что в нашем магазине произошла акция, благодаря которой мы снижаем цены на 25%. Чтобы это сделать, нужно разобраться, какие операторы бывают в SQL. Краткую справку представлю вам прямо сейчас:

**Сложение “+”:**

```
mysql> SELECT 3+5;
-> 8
```

**Вычитание “-”:**

```
mysql> SELECT 3-5;
-> -2
```

**Умножение “\*”:**

```
mysql> SELECT 3*5;
-> 15

mysql> SELECT 18014398509481984*18014398509481984;
-> 0
```

В последнем выражении мы получим неверный результат, так как произведение умножения целых чисел выходит за границы 64-битового диапазона для вычислений с точностью BIGINT.

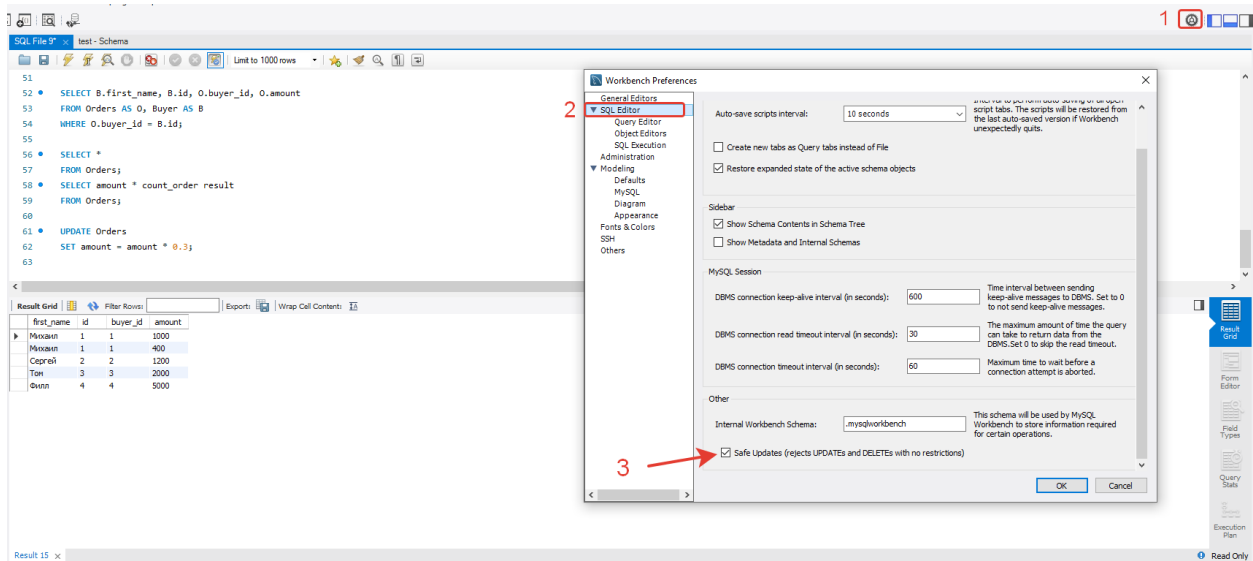
**Деление “/”:**

```
mysql> SELECT 3/5;
-> 0.60
```

**Деление на ноль приводит к результату NULL:**

```
mysql> SELECT 102/0;
-> NULL
```

Приступим к обновлению. Для начала отключим мод, который сохраняет ваши данные в исходном виде: **галочку, отмеченную в 3 пункте, необходимо убрать и перезапустить MySQL.**



Готово! Пора скидок активирована:

```
UPDATE Orders
SET amount = amount * 0.75;
-- 100% цены = 1, 25% скидка = 0.25
-- Товар после уценки: 1.00 - 0.25 = 0.75
SELECT amount new_amount
FROM Orders;
```

	id	buyer_id	amount	count_order
▶	1	1	1000	3
	2	1	400	2
	3	2	1200	5
	4	3	2000	1
	5	4	5000	4
*	NULL	NULL	NULL	NULL

	new_amount
▶	750
	300
	900
	1500
	3750

Новая цена стала ниже на 25% :) Проверим, так ли это: 1000 рублей - 100%.

1000 : 100 = 10 рублей (1 %), 25% = 10 рублей \* 25% = 250 рублей (25 %).

1000 - 250 = 750 рублей.

Мы изменили данные в исходной таблице и хотим вернуть исходный результат. Для этого можно полностью очистить таблицу от данных с помощью команды TRUNCATE:

```
TRUNCATE Orders; -- Удаляет все записи из таблицы Orders

SELECT *
FROM Orders;
```

	id	buyer_id	amount	count_order	status	manuacter
*	NULL	NULL	NULL	NULL	NULL	NULL

Чтобы заполнить данными нашу табличку, можно заново выполнить операцию заполнения таблицы:

```
44 • TRUNCATE TABLE Orders;
45 • INSERT INTO Orders (id, buyer_id, amount, count_order, manufacturer)
46     VALUES
47     (1, 1000, 3, "Ягодки"),
48     (1, 400, 2, "Амазон"),
49     (2, 1200, 5, "Амазон"),
50     (3, 2000, 1, "Ягодки"),
51     (4, 5000, 4, "Ягодки");
52
```

Запускаем конкретный запрос. Теперь таблица заполнена.

```
SELECT *
FROM Orders;
```

	id	buyer_id	amount	count_order
▶	1	1	1000	3
	2	1	400	2
	3	2	1200	5
	4	3	2000	1
	5	4	5000	4
*	NULL	NULL	NULL	NULL

### Небольшое задание:

В нашем магазине действует акция: скидка 50% на заказы, в которых есть минимум 4 товара. Скидка распространяется на 2 покупателей: id = 2, amount = 1200, count = 5 и id = 4, amount = 5000, count = 4

### Решение:

```
UPDATE Orders
SET amount = amount * 0.50
WHERE count_order >= 4; -- ИЛИ WHERE count_order > 3
```

```
SELECT amount new_amount, id
FROM Orders;
```

	id	buyer_id	amount	count_order
▶	1	1	1000	3
	2	1	400	2
	3	2	1200	5
	4	3	2000	1
	5	4	5000	4
*	NULL	NULL	NULL	NULL

	id	buyer_id	amount	count_order	manufacturer
▶	1	1	1000	3	Ягодки
	2	1	400	2	Амазон
	3	2	600	5	Амазон
	4	3	2000	1	Ягодки
	5	4	2500	4	Ягодки
*	NULL	NULL	NULL	NULL	NULL

И последняя команда, которую мы с вами не затронули - удаление.

- **DELETE** - удаление данных

**Синтаксис:**

```
DELETE FROM имя_таблицы  
[WHERE условие_удаления]
```

Пусть в нашей базе хранятся тестовые данные, используемые нами только для тестирования. Необходимость в тестировании пропала и мы можем их удалить:

```
INSERT Buyer (date_birt, first_name, last_name, mobile_phone)  
VALUES  
  ("2023-01-01", "Тестовый", "Пользователь", "+7-999-888-77-66");  
-- Добавили клиента  
DELETE FROM Buyer  
WHERE first_name='Тестовый';  
-- Удалили строчку со значением
```

Если имеется необходимость объединять несколько условий, то нам потребуются логические операторы.

## Логические операторы

Логические операторы позволяют объединить несколько условий. В MySQL можно использовать следующие логические операторы:

**AND** - операция логического И.

Она объединяет два выражения

**выражение1 AND выражение2**

Только если оба этих выражения одновременно истинны, то и общее условие оператора AND также будет истинно. Должны быть истинны и первое условие , и второе.

```
-- Получим заказы от 1500 рублей из магазина "Ягодки"  
SELECT amount, count_order  
FROM Orders  
WHERE amount > 1500 AND manufacturer = "Ягодки";
```

	id	buyer_id	amount	count_order	manufacturer
▶	1	1	1000	3	Ягодки
	2	1	400	2	Амазон
	3	2	1200	5	Амазон
	4	3	2000	1	Ягодки
	5	4	5000	4	Ягодки

	amount	count_order
▶	2000	1
	5000	4

**OR**: операция логического ИЛИ.

Она также объединяет два выражения:

**выражение1 OR выражение2**

Если хотя бы одно из этих выражений истинно, то общее условие оператора OR также будет истинно. Должно быть истинно хотя бы 1 условие: или первое условие , или второе.

```
-- Хотим получить товары или из "Амазона", или товары из диапазона (3;5)
SELECT amount, count_order, manufacturer
FROM Orders
WHERE manufacturer = "Амазон" OR count_order > 2 AND count_order < 5;
-- Оператор AND имеет более высокий приоритет, чем OR
```

	id	buyer_id	amount	count_order	manufacturer
▶	1	1	1000	3	Ягодки
	2	1	400	2	Амазон
	3	2	1200	5	Амазон
	4	3	2000	1	Ягодки
	5	4	5000	4	Ягодки

	amount	count_order	manufacturer
▶	1000	3	Ягодки
	400	2	Амазон
	1200	5	Амазон
	5000	4	Ягодки

**NOT:** операция логического отрицания. Если выражение в этой операции ложно, то общее условие истинно.

```
-- Исключим товары марки "Ягодки"
SELECT amount, count_order, manufacturer
FROM Orders
WHERE manufacturer != "Ягодки";
-- ИЛИ через "!="
SELECT amount, count_order, manufacturer
FROM Orders
WHERE NOT manufacturer = "Ягодки";
```

	id	buyer_id	amount	count_order	manufacturer
▶	1	1	1000	3	Ягодки
	2	1	400	2	Амазон
	3	2	1200	5	Амазон
	4	3	2000	1	Ягодки
	5	4	5000	4	Ягодки

	amount	count_order	manufacturer
▶	400	2	Амазон
	1200	5	Амазон

Отлично, почти все. Остались операторы, проверяющие истинность значения:

- CASE
- IF

## Операторы CASE, IF

### 1. CASE

Проверяет истинность набора условий и возвращает результат в зависимости от проверки.

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
```

```
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Чтобы продемонстрировать пример на нашей таблице, немного обновим нашу таблицу. Для изменения таблицы используется оператор “ALTER TABLE

```
-- Добавить столбец "new_column" в таблицу "Table_name"
ALTER TABLE Table_name
ADD new_column VARCHAR(50);
-- Удалить столбец "new_column" из таблицы "Table_name"
ALTER TABLE Table_name
DROP COLUMN new_column;
```

Давайте добавим в исходную таблицу столбец “статус”, в котором будет два значения:

- 0 - заказ не оплачен
- 1 - заказ оплачен

Заполнение произведем с помощью функции для получения рандомного числа = **RAND()**.

```
ALTER TABLE Orders
ADD COLUMN status INT AFTER count_order;
-- RAND(): https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html#function\_rand
-- Возвращает числа от 0 до 1
UPDATE Orders
SET status = RAND();
```

Задача: в зависимости от значения поля “status” вывести сообщение о факте оплаты: “заказ оплачен”, “оплатите заказ”.

```
SELECT *
FROM Orders;

SELECT status, -- Перед "CASE" ставится запятая, после перечисления столбцов
CASE WHEN status IS TRUE THEN 'заказ оплачен'
ELSE 'оплатите заказ'
END AS message
FROM Orders;

-- ИЛИ

SELECT status, -- Перед "CASE" ставится запятая, после перечисления столбцов
CASE WHEN status = 1 THEN 'заказ оплачен'
ELSE 'оплатите заказ'
END AS message
FROM Orders;
```

	id	buyer_id	amount	count_order	status	manuфacter
▶	1	1	1000	3	1	Ягодки
	2	1	400	2	1	Амазон
	3	2	1200	5	1	Амазон
	4	3	2000	1	1	Ягодки
	5	4	5000	4	0	Ягодки



	status	message
►	1	заказ оплачен
	1	заказ оплачен
	1	заказ оплачен
	1	заказ оплачен
	0	оплатите заказ

## 2. Функция IF

Функция IF в зависимости от результата условного выражения возвращает одно из двух значений.

```
IF(условие, значение_для_истины, значение_для_лжи);
```

```
-- Представьте, что мы страхуем заказы со средним чеком от 3000 включительно.
-- Сообщим клиентам о наличии или отсутствии страховки
SELECT status, amount, count_order, manufacturer, -- Перед "IF" тоже ставится запятая
        IF(amount * count_order >= 3000, 'Страховка включена в стоимость', 'Страховка оплачивается отдельно') AS info_message
FROM Orders;
```

	id	buyer_id	amount	count_order	status	manufacturer
►	1	1	1000	3	1	Ягодки
	2	1	400	2	1	Амазон
	3	2	1200	5	1	Амазон
	4	3	2000	1	1	Ягодки
	5	4	5000	4	0	Ягодки

	status	amount	count_order	manufacturer	info_message
►	1	1000	3	Ягодки	Страховка включена в стоимость
	1	400	2	Амазон	Страховка оплачивается отдельно
	1	1200	5	Амазон	Страховка включена в стоимость
	1	2000	1	Ягодки	Страховка оплачивается отдельно
	0	5000	4	Ягодки	Страховка включена в стоимость

Полезные ссылки и рекомендации:

- <https://habr.com/ru/company/oleg-bunin/blog/348172/> - DDL, DCL на **MS SQL Server (1 часть)**.
- <https://habr.com/ru/post/255523/> - DDL, DCL на **MS SQL Server (2 часть)**.
- Руководство по стилю написания SQL: <https://www.sqlstyle.guide/ru/>
- ER - диаграмма и связи: <https://habr.com/ru/post/440556/> (связи были изучены, конкретнее изучить ER - диаграммы можно по ссылке)
- Старайтесь не использовать русские буквы при работе с СУБД

Книги:

- "Изучаем SQL", книга Бейли Л.
- Алан Бьюли "Изучаем SQL" (2007)
- Энтони Молинаро "SQL. Сборник рецептов" (2009)