

Лекция №2

SQL – создание объектов, изменение данных, логические операторы

Тизер

Сегодня мы с вами будем работать с СУБД MySQL.

• MySQL – это свободно распространяемая СУБД, разработанная компанией MySQL AB (www.mysql.com). MySQL имеет клиент-серверную архитектуру: к серверу MySQL могут обращаться различные клиентские приложения, в том числе с удаленных компьютеров. Рассмотрим важнейшие особенности MySQL, благодаря которым эта программа приобрела популярность.

• MySQL – это СУБД с открытым кодом. Любой желающий может бесплатно скачать программу на сайте разработчика (<http://dev.mysql.com/downloads/>) и при необходимости доработать ее. Существует множество приложений MySQL, созданных и свободно распространяемых сторонними разработчиками. Однако для применения MySQL в коммерческом приложении необходимо приобрести коммерческую лицензированную версию программы у компании MySQL AB.

Сегодня мы изучим:

1. Типы данных, значения NULL, create table, PK, FK
2. Комментарии
3. Арифметические операций
4. Логические операторы (and, or, between, not, in)
5. Приоритет выполнения операторов, порядок выполнения запроса
6. Оператор CASE, IIF
7. Запросы изменения данных (insert, update, delete)

Термины лекции

NULL соответствует понятию «пустое поле»*null*, то есть «поле, не содержащее никакого значения».

Таблицы – это логически организованные хранилища данных (объектов), представленные в виде строк и столбцов.

Комментарии в синтаксисе SQL запросов — это необязательный текст, который описывает, что делает программа и почему код был изменен. Компилятор всегда

игнорирует комментарии.

Оглавление

Тизер	1
Термины лекции	1
Создание базы данных	3
Создание таблиц	5
Типы данных	7
Primary key (PK) и foreign key (FK)	8
Комментарии	10
Арифметические операции	11
Приоритет операций	14
Порядок операций SQL	17
Оператор CASE, IF	18
Запросы изменения данных (insert, update, delete)	20
Итоги	21

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 MySampleDB в командной строке mysql нужно ввести следующий запрос:

```
CREATE DATABASE MySampleDB;
```

Если все прошло нормально, команда сгенерирует следующий вывод:

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

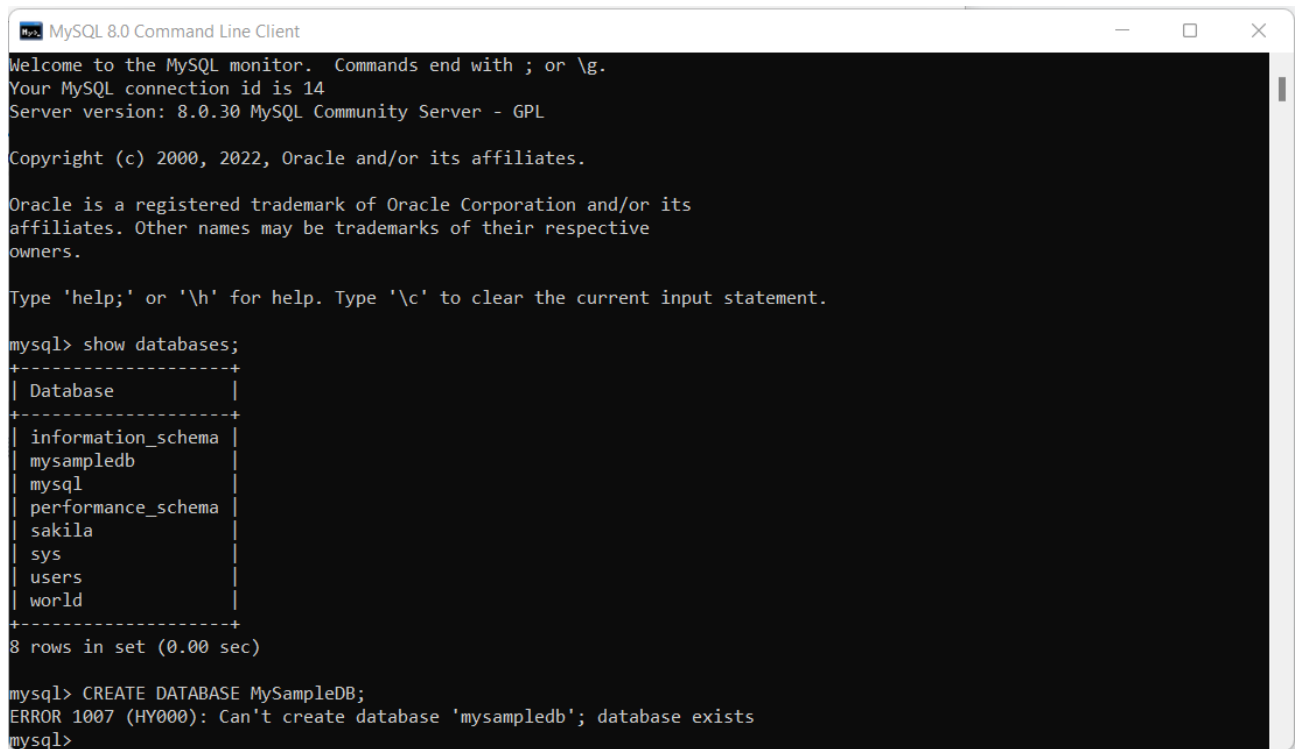
Если указанное имя базы данных конфликтует с существующей базой данных MySQL, будет выведено сообщение об ошибке:

```
ERROR 1007 (HY000): Can't create database 'MySampleDB';  
database exists
```

Проверить, что база появилась можно командой:

```
show databases;
```

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



```
MySQL 8.0 Command Line Client
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.30 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

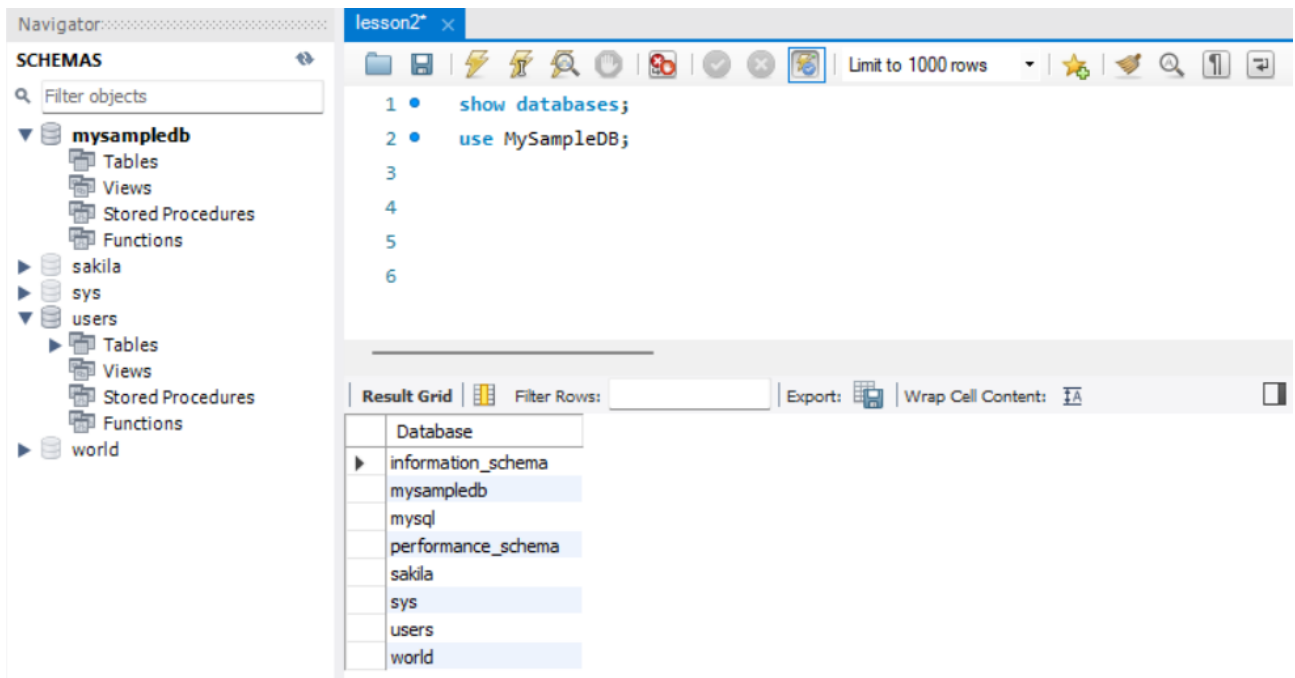
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysampled |
| mysql |
| performance_schema |
| sakila |
| sys |
| users |
| world |
+-----+
8 rows in set (0.00 sec)

mysql> CREATE DATABASE MySampleDB;
ERROR 1007 (HY000): Can't create database 'mysampled'; database exists
mysql>
```

Подключиться к базе можно командой:

```
use MySampleDB;
```

Данная команда подключается к базе данных с именем MySampleDB 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, и каждая таблица содержит набор столбцов. У столбца есть название и тип данных. Команда создания таблицы должна содержать все вышеупомянутое:

```
CREATE TABLE table_name  
(  
    column_name_1 column_type_1,  
    column_name_2 column_type_2,  
    ...,  
    column_name_N column_type_N,  
);
```

table_name — имя таблицы;

column_name — имя столбца;

column_type — тип данных столбца.

Типы данных

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.

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 КБ 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.

Primary key (PK) и foreign key (FK)

Первичные ключи PRIMARY KEY

Атрибут PRIMARY KEY задает первичный ключ таблицы. Первичный ключ уникально идентифицирует строку в таблице. В качестве первичного ключа необязательно должны выступать столбцы с типом int, они могут представлять любой другой тип. Установка первичного ключа на уровне таблицы:

```
CREATE TABLE Customers
```

```
(
```

```
Id INT PRIMARY KEY AUTO_INCREMENT,  
Age INT,  
FirstName VARCHAR(20),  
LastName 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.

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:

```
[CONSTRAINT имя_ограничения] FOREIGN KEY (столбец1, столбец2, ...  
столбецN) REFERENCES          главная_таблица          (столбец_главной_таблицы1,  
столбец_главной_таблицы2, ... столбец_главной_таблицыN)
```

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. Например, определим две таблицы и свяжем их посредством внешнего ключа:


```

CREATE TABLE Customers
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    Age INT,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    Phone VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE Orders
(
    Id INT PRIMARY KEY AUTO_INCREMENT,
    CustomerId INT,
    CreatedAt Date,
    FOREIGN KEY (CustomerId) REFERENCES Customers (Id)
);

```

В данном случае определены таблицы Customers и Orders. Customers является главной и представляет клиента. Orders является зависимой и представляет заказ, сделанный клиентом. Таблица Orders через столбец CustomerId связана с таблицей Customers и ее столбцом Id. То есть столбец CustomerId является внешним ключом, который указывает на столбец Id из таблицы Customers.

Комментарии

Существует три синтаксиса, которые можно использовать для создания комментария в SQL-операторах в MySQL.

1. Синтаксис с использованием символа

здесь комментарий

В MySQL комментарий, начинающийся с символа #, должен быть в конце строки SQL-выражения с разрывом строки после него. Этот метод комментирования может охватывать только одну строку внутри вашего SQL и должен находиться в конце строки.

2. Использование синтаксиса символов -- (два минуса)

Синтаксис для создания комментария SQL в MySQL с использованием

символов --:

-- здесь комментарий

В MySQL комментарий, начинающийся с символа --, похож на комментарий, начинающийся с символа #. При использовании символа -- комментарий должен быть в конце строки в вашем SQL-операторе с разрывом строки после него. Этот метод комментирования может охватывать только одну строку внутри вашего SQL и должен находиться в конце строки.

3. Синтаксис использования символов /* и */

Синтаксис для создания комментария SQL в MySQL с использованием символов /* и */:

/* здесь комментарий */

В MySQL комментарий, который начинается с символа /* и заканчивается */ может быть где угодно в вашем SQL-операторе. Этот метод комментирования может охватывать несколько строк в вашем SQL.

Арифметические операции

В MySQL можно применять обычные арифметические операторы. Если один из аргументов - беззнаковое целое число, а второй аргумент - также целое число, то результат будет беззнаковым целым числом

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

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. То есть если и первое условие истинно, и второе.

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. То есть если или первое условие истинно, или второе.

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

NOT выражение

Сравнения значений приведение типов, регистров символов, обработка Null вынесу в задачи к семинару. Очень много информации ><

БД для работы, заполненная данными:

```
CREATE TABLE Products
```

```
(  
    Id INT AUTO_INCREMENT PRIMARY KEY,  
    ProductName VARCHAR(30) NOT NULL,  
    Manufacturer VARCHAR(20) NOT NULL,  
    ProductCount INT DEFAULT 0,  
    Price DECIMAL
```

);

```
1 • USE productsdb;
2
3 • SELECT * FROM Products;
```

<					
Result Grid Filter Rows: Edit:					
	Id	ProductName	Manufacturer	ProductCount	Price
	1	iPhone X	Apple	3	76000
	2	iPhone 8	Apple	2	51000
	3	Galaxy S9	Samsung	2	56000
	4	Galaxy S8	Samsung	1	41000
	5	P20 Pro	Huawei	5	36000
	NULL	NULL	NULL	NULL	NULL

SELECT * FROM Products;

Символ звездочка * указывает, что нам надо получить все столбцы.

Выберем все товары, у которых производитель Samsung и одновременно цена больше 50000:

SELECT * FROM Products

WHERE Manufacturer = 'Samsung' AND Price > 50000

```
1 • USE productsdb;
2
3 • SELECT * FROM Products
4   WHERE Manufacturer = 'Samsung' AND Price > 50000
```

<					
Result Grid Filter Rows: Edit: Export/					
	Id	ProductName	Manufacturer	ProductCount	Price
	3	Galaxy S9	Samsung	2	56000
	NULL	NULL	NULL	NULL	NULL

Теперь изменим оператор на OR. То есть выберем все товары, у которых либо производитель Samsung, либо цена больше 50000:

```
SELECT * FROM Products
```

```
WHERE Manufacturer = 'Samsung' OR Price > 50000
```

```
1 • USE productsdb;
2 |
3 • SELECT * FROM Products
4 WHERE Manufacturer = 'Samsung' OR Price > 50000
```

Result Grid

Filter Rows:

Edit:

Export/

	Id	ProductName	Manufacturer	ProductCount	Price
	1	iPhone X	Apple	3	76000
	2	iPhone 8	Apple	2	51000
	3	Galaxy S9	Samsung	2	56000
	4	Galaxy S8	Samsung	1	41000
	NULL	NULL	NULL	NULL	NULL

Применение оператора NOT - выберем все товары, у которых производитель не Samsung:

```
SELECT * FROM Products
```

```
WHERE NOT Manufacturer = 'Samsung';
```

```
1 • USE productsdb;
2
3 • SELECT * FROM Products
4 WHERE NOT Manufacturer = 'Samsung';
```

Result Grid

Filter Rows:

Edit:

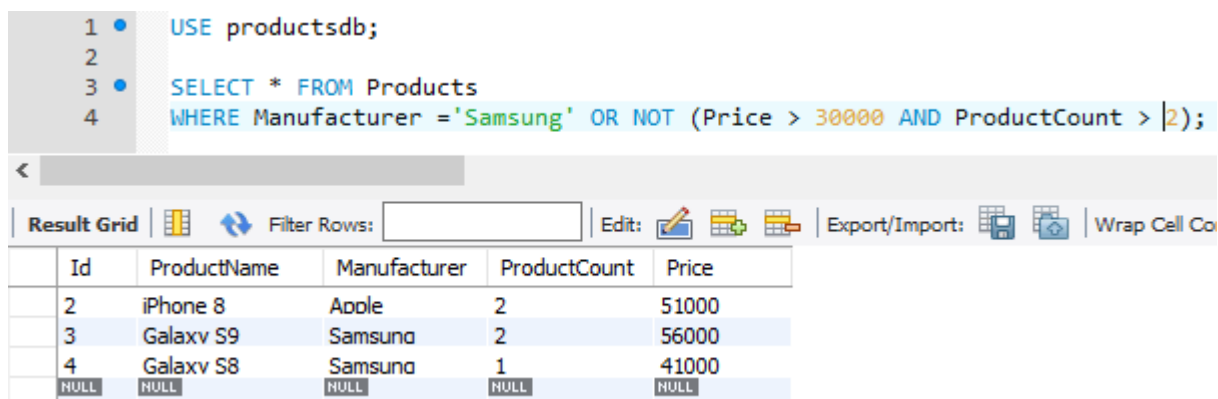
	Id	ProductName	Manufacturer	ProductCount	Price
	1	iPhone X	Apple	3	76000
	2	iPhone 8	Apple	2	51000
	5	P20 Pro	Huawei	5	36000
	NULL	NULL	NULL	NULL	NULL

Приоритет операций

В одном условии при необходимости мы можем объединять несколько логических операций. Однако следует учитывать, что самой приоритетной операцией, которая выполняется в первую очередь, является NOT, менее приоритетная - AND и операция с наименьшим приоритетом - OR. Например:

- 1 SELECT * FROM Products
- 2 WHERE Manufacturer ='Samsung' OR NOT Price > 30000 AND ProductCount > 2;

1. В



The screenshot shows a SQL query editor with the following query:

```
1 USE productsdb;  
2  
3 SELECT * FROM Products  
4 WHERE Manufacturer ='Samsung' OR NOT (Price > 30000 AND ProductCount > 2);
```

Below the query editor is a table with the following data:

Id	ProductName	Manufacturer	ProductCount	Price
2	iPhone 8	Apple	2	51000
3	Galaxy S9	Samsung	2	56000
4	Galaxy S8	Samsung	1	41000
NULL	NULL	NULL	NULL	NULL

данном случае сначала вычисляется выражение NOT Price > 30000, то есть цена должна быть меньше или равна 30000.

2. Затем вычисляется выражение NOT Price > 30000 AND ProductCount > 2, то есть цена должна быть меньше или равна 30000 и одновременно количество товаров должно быть больше 2.

3. В конце вычисляется оператор OR - либо цена должна быть меньше или равна 30000 и одновременно количество товаров должно быть больше 2, либо производителем должен быть Samsung.

1	•	USE productsdb;
2		
3	•	SELECT * FROM Products
4		WHERE Manufacturer ='Samsung' OR NOT Price > 30000 AND ProductCount > 2;

Result Grid						Filter Rows:	Edit:	Export/Import:	Wrap Cell
	Id	ProductName	Manufacturer	ProductCount	Price				
	3	Galaxy S9	Samsung	2	56000				
	4	Galaxy S8	Samsung	1	41000				
	NULL	NULL	NULL	NULL	NULL				

С помощью скобок можно переопределить приоритет операций:

- 1 SELECT * FROM Products
- 2 WHERE Manufacturer ='Samsung' OR NOT (Price > 30000 AND ProductCount > 2);

В данном случае находим товары, у которых либо производитель Samsung, либо одновременно цена товара меньше или равна 30000 и количество товаров меньше 3.

Знание порядка битов и байтов операций SQL-запроса может быть очень полезным, поскольку оно может упростить процесс написания новых запросов, а также очень полезно при попытке оптимизировать SQL-запрос. Если вы ищете короткую версию, это логический порядок операций, также известный как порядок выполнения, для SQL-запроса:

1. FROM, включая JOINS

2. WHERE

3. GROUP BY

4. HAVING

5. Функции WINDOW

6. SELECT

7. DISTINCT

8. UNION

9. ORDER BY

10. LIMIT и OFFSET

Но реальность не так проста и не прямолинейна. Как мы уже говорили, стандарт SQL определяет порядок выполнения для различных предложений SQL-запросов. Сказано, что современные базы данных уже проверяют этот порядок по умолчанию, применяя некоторые приемы оптимизации, которые могут изменить фактический порядок выполнения, хотя в конечном итоге они должны возвращать тот же результат, как если бы они выполняли запрос в порядке выполнения по умолчанию.

Оператор CASE, IF

CASE

Функция CASE проверяет истинность набора условий и в зависимости от результата проверки может возвращать тот или иной результат. Эта функция принимает следующую форму:

CASE

```
WHEN условие_1 THEN результат_1  
WHEN условие_2 THEN результат_2
```

```

.....
WHEN условие_N THEN условие_N
[ELSE альтернативный_результат]
END

```

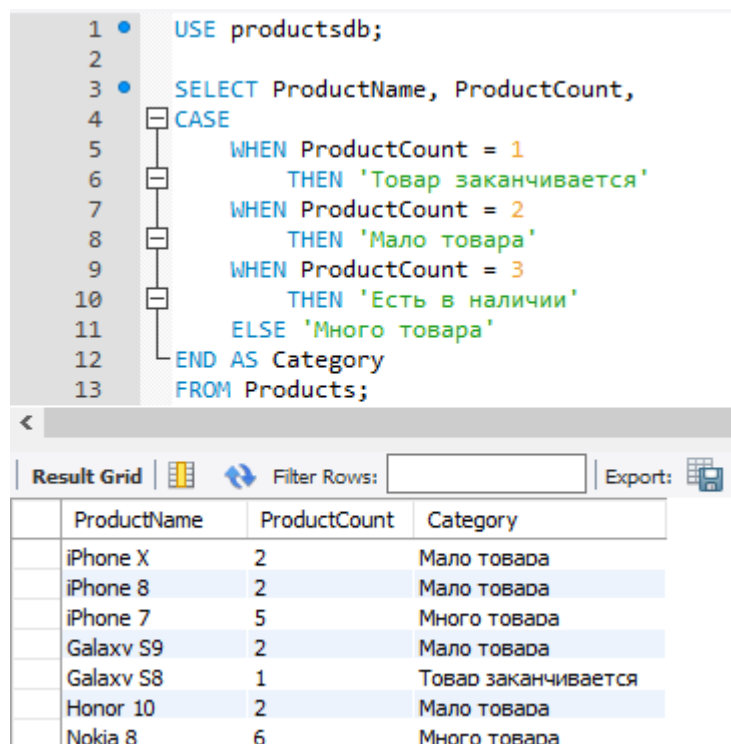
Возьмем для примера таблицу Products, созданную ранее.

Выполним запрос к этой таблице и используем функцию CASE:

```

SELECT ProductName, ProductCount,
CASE
    WHEN ProductCount = 1
        THEN 'Товар заканчивается'
    WHEN ProductCount = 2
        THEN 'Мало товара'
    WHEN ProductCount = 3
        THEN 'Есть в наличии'
    ELSE 'Много товара'
END AS Category
FROM Products;

```



The screenshot shows a SQL query in the Enterprise Manager interface. The query is as follows:

```

1 USE productsdb;
2
3 SELECT ProductName, ProductCount,
4 CASE
5     WHEN ProductCount = 1
6     THEN 'Товар заканчивается'
7     WHEN ProductCount = 2
8     THEN 'Мало товара'
9     WHEN ProductCount = 3
10    THEN 'Есть в наличии'
11    ELSE 'Много товара'
12 END AS Category
13 FROM Products;

```

Below the query, the 'Result Grid' is displayed, showing the following data:

ProductName	ProductCount	Category
iPhone X	2	Мало товара
iPhone 8	2	Мало товара
iPhone 7	5	Много товара
Galaxy S9	2	Мало товара
Galaxy S8	1	Товар заканчивается
Honor 10	2	Мало товара
Nokia 8	6	Много товара

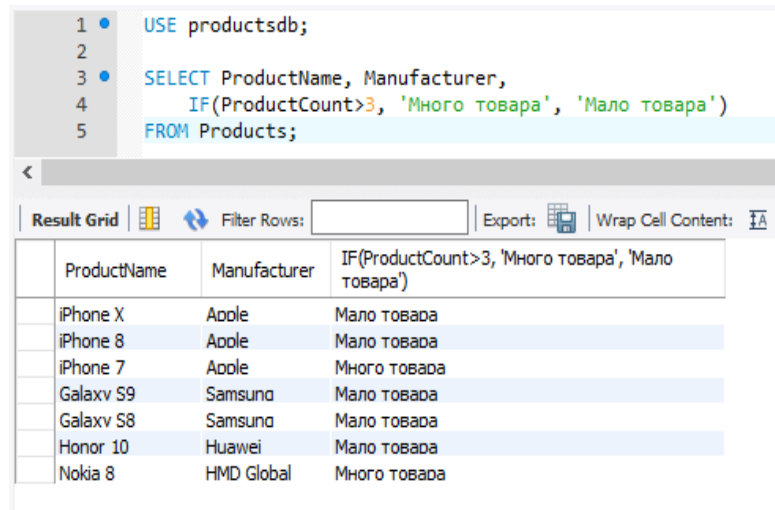
Функция IF

Функция IF в зависимости от результата условного выражения возвращает одно из двух значений. Общая форма функции выглядит следующим образом:

```
IF(условие, значение_1, значение_2)
```

Если условие, передаваемое в качестве первого параметра, верно, то возвращается первое значение, иначе возвращается второе значение. Например:

```
SELECT ProductName, Manufacturer,  
       IF(ProductCount > 3, 'Много товара', 'Мало товара')  
FROM Products;
```



ProductName	Manufacturer	IF(ProductCount>3, 'Много товара', 'Мало товара')
iPhone X	Apple	Мало товара
iPhone 8	Apple	Мало товара
iPhone 7	Apple	Много товара
Galaxv S9	Samsung	Мало товара
Galaxv S8	Samsung	Мало товара
Honor 10	Huawei	Мало товара
Nokia 8	HMD Global	Много товара

Запросы изменения данных (insert, update, delete)

INSERT – вставка новых данных

Данный оператор имеет 2 основные формы:

1. INSERT INTO таблица(перечень_полей) VALUES(перечень_значений) – вставка в таблицу новой строки значения полей которой формируются из перечисленных значений
2. INSERT INTO таблица(перечень_полей) SELECT перечень_значений FROM ... – вставка в таблицу новых строк, значения которых формируются из значений строк возвращенных запросом.

Заполним табличку Products без графического интерфейса:

```
CREATE TABLE Products  
(  
    Id INT AUTO_INCREMENT PRIMARY KEY,  
    ProductName VARCHAR(30) NOT NULL,  
    Manufacturer VARCHAR(20) NOT NULL,  
    ProductCount INT DEFAULT 0,  
    Price DECIMAL  
);  
  
INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
```

```
VALUES
('iPhone X', 'Apple', 3, 76000),
('iPhone 8', 'Apple', 2, 51000),
('Galaxy S9', 'Samsung', 2, 56000),
('Galaxy S8', 'Samsung', 1, 41000),
('P20 Pro', 'Huawei', 5, 36000);
```

Команда UPDATE - обновление данных

Она применяется для обновления уже имеющихся строк. Она имеет следующий формальный синтаксис:

```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN
[WHERE условие_обновления]
```

Например, увеличим у всех товаров цену на 3000:

```
UPDATE Products
SET Price = Price + 3000;
```

```
mysql> UPDATE Products SET Price = Price + 3000;
Query OK, 5 rows affected (0.01 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM Products;
+-----+-----+-----+-----+-----+
| Id | ProductName | Manufacturer | ProductCount | Price |
+-----+-----+-----+-----+-----+
| 1 | iPhone X | Apple | 3 | 79000 |
| 2 | iPhone 8 | Apple | 2 | 54000 |
| 3 | Galaxy S9 | Samsung | 2 | 59000 |
| 4 | Galaxy S8 | Samsung | 1 | 44000 |
| 5 | P20 Pro | Huawei | 5 | 39000 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Команда DELETE - удаление данных

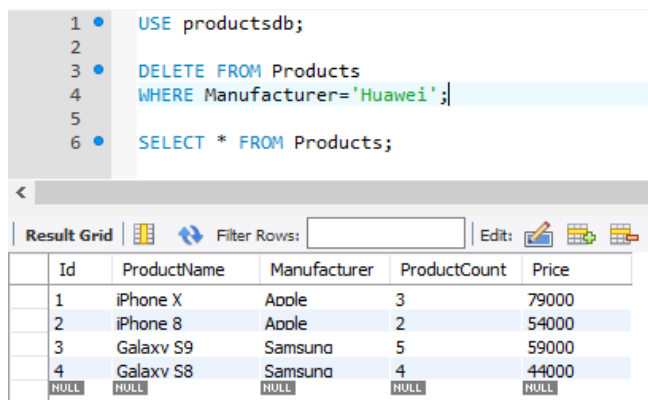
Команда DELETE удаляет данные из БД. Она имеет следующий формальный синтаксис:

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

Например, удалим строки, у которых производитель - Huawei:

```
DELETE FROM Products
```

```
WHERE Manufacturer='Huawei';
```



The screenshot shows a SQL IDE interface. At the top, a script editor contains six lines of SQL code: 1. USE productsdb; 2. (blank) 3. DELETE FROM Products 4. WHERE Manufacturer='Huawei'; 5. (blank) 6. SELECT * FROM Products;. Below the script editor is a 'Result Grid' tab. It features a toolbar with icons for grid view, refresh, filter, edit, and export. A 'Filter Rows:' text box is present. The grid displays the results of the SELECT statement, showing four rows of product data. The first two rows are for Apple products (iPhone X and iPhone 8), and the next two are for Samsung products (Galaxy S9 and Galaxy S8). Each row includes an Id, ProductName, Manufacturer, ProductCount, and Price. The last row of the grid is a summary row with NULL values for all columns.

	Id	ProductName	Manufacturer	ProductCount	Price
	1	iPhone X	Apple	3	79000
	2	iPhone 8	Apple	2	54000
	3	Galaxy S9	Samsung	5	59000
	4	Galaxy S8	Samsung	4	44000
	NULL	NULL	NULL	NULL	NULL

Итоги

Сегодня изучили основные команды для создания базы данных, вспомнили понятие первичного и внешнего ключа. Разобрались, какие типы данных существуют в MySQL, научились ставить комментарии в MySQL, прошлись по основным арифметическим операциям, логическим операциям и их приоритетам, узнали про операторы CASE, IF и с основными запросами изменения данных.

На следующей лекции мы продолжим изучение языка sql: изучим выборку и сортировку, научимся группировать данные, поработаем со встроенными функциями