

## Лекция №3

*SQL – выборка данных, сортировка, агрегатные функции*

Сегодня мы узнаем:

1. Сортировка результатов запроса, сортировка по условиям
2. Ограничение выборки (top, limit, fetch),
3. Уникальные значения - distinct
4. Группировка — group by
5. Агрегатные функции — count, sum, avg, обработка Null
6. Использование Having
7. Порядок выполнения запроса

### Термины лекции

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

**Группировка** — операция, которая создает из записей таблицы независимые группы записей, по которым проводится анализ.

**Агрегатные функции (агрегации)** — это функции, которые вычисляются от группы значений и объединяют их в одно результирующее.

### Оглавление

Тизер	1
Термины лекции	1
Сортировка результатов запроса	2
Ограничение выборки (top, limit, fetch)	4
Уникальные значения - distinct	7
Группировка GROUP BY	8
Агрегатные функции — count, sum, avg, обработка Null	9
HAVING	11
Приоритет операций	16
Итоги	21
Домашнее задание	21

## Сортировка результатов запроса

Когда мы выполняем SELECT запрос, в финальном результате строки возвращаются в неопределенном порядке. Фактический порядок строк зависит от порядка расположения данных в таблице. Для упорядочивания результатов запроса, используется конструкция ORDER BY.

## Синтаксис

```
SELECT expressions  
  
FROM tables  
  
[WHERE conditions]  
  
ORDER BY expression [ ASC | DESC ];
```

## Параметры

### expressions

Столбцы или расчеты, которые вы хотите получить

### tables

Таблицы, из которых вы хотите получить записи. В предложении FROM должна быть указана хотя бы одна таблица

### WHERE conditions

Необязательный. Условия, которые должны быть выполнены для записей, которые будут выбраны

### ASC

Необязательный. ASC сортирует результирующий набор в порядке возрастания по expressions. Это поведение по умолчанию, если модификатор не указан.

### DESC

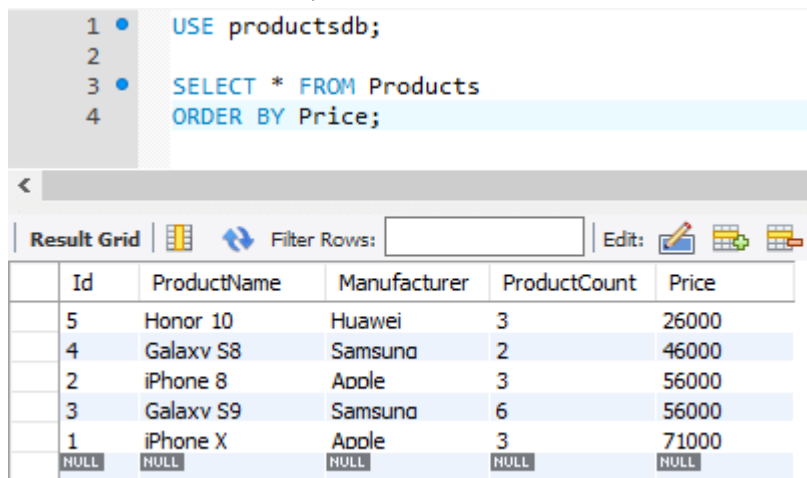
Необязательный. DESC сортирует результирующий набор в порядке убывания по expressions

В SQL часто кроме фильтрации данных, также часто требуется отсортировать их по одному из столбцов. Оператор ORDER BY сортирует значения по одному или нескольким столбцам. Например, упорядочим выборку из таблицы Products по столбцу

Price:

```
SELECT * FROM Products
```

```
ORDER BY Price;
```



	Id	ProductName	Manufacturer	ProductCount	Price
	5	Honor 10	Huawei	3	26000
	4	Galaxy S8	Samsung	2	46000
	2	iPhone 8	Apple	3	56000
	3	Galaxy S9	Samsung	6	56000
	1	iPhone X	Apple	3	71000
	NULL	NULL	NULL	NULL	NULL

Также можно производить упорядочивание данных по псевдониму столбца, который определяется с помощью оператора AS:

```
SELECT ProductName, ProductCount * Price AS TotalSum
```

```
FROM Products
```

```
ORDER BY TotalSum;
```

В качестве критерия сортировки также можно использовать сложное выражение на основе столбцов:

```
SELECT Product Name, Price, Product Count
```

```
FROM Products
```

```
ORDER BY Product Count * Price;
```

## Ограничение выборки (top, limit, fetch)

### Оператор LIMIT

Оператор LIMIT позволяет извлечь определённый диапазон записей из одной или нескольких таблиц.

## Общая структура запроса с оператором LIMIT

```
1 SELECT поля_выборки
2 FROM список_таблиц
3 LIMIT [количество_пропущенных_записей,] количество_записей_для_вывода;
```

Оператор LIMIT реализован не во всех СУБД, например, в MSSQL для вывода записей с начала таблицы используется оператор TOP, а для тех случаев, когда необходимо сделать отступ от начала таблицы, предназначена конструкция OFFSET FETCH.

Оператор LIMIT позволяет извлечь определенное количество строк и имеет следующий синтаксис:

```
1 LIMIT [offset,] rowcount
```

Если оператору LIMIT передается один параметр, то он указывает на количество извлекаемых строк. Если передается два параметра, то первый параметр устанавливает смещение относительно начала, то есть сколько строк нужно пропустить, а второй параметр также указывает на количество извлекаемых строк. Например, выберем первые три строки:

```
1 SELECT * FROM Products
2 LIMIT 3;
```

Теперь используем второй параметр и укажем смещение, с которой должна происходить выборка

```
1 SELECT * FROM Products
2 LIMIT 2, 3
```

В данном случае пропускаются две первые строки и извлекаются следующие 3 строки:

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

< Result Grid

Filter Rows:

Edit:

	Id	ProductName	Manufacturer	ProductCount	Price
	3	Galaxy S9	Samsung	6	56000
	4	Galaxy S8	Samsung	2	46000
	5	Honor 10	Huawei	3	26000
	NULL	NULL	NULL	NULL	NULL

## **Аналоги: извлечение диапазона строк в MS SQL Server**

В MS SQL Server можно также извлекать определённый диапазон строк, но для этого существуют другие конструкции и они немного сложнее, чем в MySQL. Аналогом LIMIT с одним параметром является оператор TOP. Он может применяться только с одним параметром и служит для вывода из таблицы первых строк, число которых указано в качестве параметра. При помощи применённого ограничения диапазона будет выведена следующая таблица:

Obj_ID	Type	District	Rooms
1	flat	Центр	2
2	flat	Центр	2

## **Оператор fetch**

Синтаксис оператора FETCH в MySQL:

```
1 SELECT ColumnNames FROM TableName ORDER BY ColumnName OFFSET
   rows_to_be_skipped FETCH NEXT n ROWS ONLY;
```

## **Параметры или аргументы**

Аргумент OFFSET в MySQL определяет начальную точку строк, возвращаемых запросом. Запрос OFFSET отвечает за пропуск

количества строк перед началом выборки строк из SQL-запроса. Offset\_rows\_count может быть задан константой, любым скаляром, переменной, любым параметром, большим или равным нулю. Предложение FETCH используется для возврата количества записей после выполнения предложения OFFSET

Fetch\_rows\_count может быть задан константой, любым скаляром, переменной, любым параметром, большим или равным нулю. В запросе SQL необходимо использовать предложение OFFSET, но предложение FETCH может быть необязательным термином. Термины First и Next являются синонимами соответственно, чтобы добавлять их взаимозаменяемо, и то же самое для ключевых слов ASC и DESC для сортировки строк при выборке.

Мы можем посмотреть на полный синтаксис использования MySQL FETCH со OFFSET, чтобы вернуть количество строк, исключая первые строки, и получить следующие строки из таблицы. Это базовый синтаксис запроса для исключения первых m строк.

```
SELECT ColumnNames FROM TableName ORDER BY  
ColumnNames OFFSET m ROWS FETCH NEXT p ROWS ONLY;
```

```
1 SELECT ColumnNames FROM TableName ORDER BY ColumnNames OFFSET m ROWS  
   FETCH NEXT p ROWS ONLY;
```

Теперь снова вы можете использовать следующий код, чтобы исключить m строк и выбрать

следующие  $p$  строк из таблицы. Это будет извлекать строки только от  $(m+1)$  до  $(m+1+p)$ .

## Уникальные значения - distinct

С помощью оператора DISTINCT можно выбрать уникальные данные по определенным столбцам. К примеру, разные товары могут иметь одних и тех же производителей, и, допустим, у нас следующая таблица товаров:

```
1 USE productsdb;
2
3 DROP TABLE IF EXISTS Products;
4
5 CREATE TABLE Products
6 (
7     Id INT AUTO_INCREMENT PRIMARY KEY,
8     ProductName VARCHAR(30) NOT NULL,
9     Manufacturer VARCHAR(20) NOT NULL,
10    ProductCount INT DEFAULT 0,
11    Price DECIMAL NOT NULL
12 );
13 INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
14 VALUES
15 ('iPhone X', 'Apple', 3, 71000),
16 ('iPhone 8', 'Apple', 3, 56000),
17 ('Galaxy S9', 'Samsung', 6, 56000),
18 ('Galaxy S8', 'Samsung', 2, 46000),
19 ('Honor 10', 'Huawei', 3, 26000);
```

Применим оператор DISTINCT для выборки уникальных значений - уникальных производителей:

```
1 SELECT DISTINCT Manufacturer FROM Products;
```

Также мы можем задавать выборку уникальных значений по нескольким столбцам:

```
1 SELECT DISTINCT Manufacturer, ProductCount FROM Products;
```

В данном случае для выборки используются столбцы Manufacturer и ProductCount. Из пяти строк только для двух строк эти столбцы имеют повторяющиеся значения. Поэтому в выборке будет 4 строки:

## Группировка — GROUP BY

Операторы GROUP BY и HAVING позволяют сгруппировать данные. Они употребляются в рамках команды SELECT:

```
1 SELECT столбцы
2 FROM таблица
3 [WHERE условие_фильтрации_строк]
4 [GROUP BY столбцы_для_группировки]
5 [HAVING условие_фильтрации_групп]
6 [ORDER BY столбцы_для_сортировки]
```

### GROUP BY

Оператор GROUP BY определяет, как строки будут группироваться.

Например, сгруппируем товары по производителю

***SELECT Manufacturer, COUNT(\*) AS ModelsCount***

***FROM Products***

***GROUP BY Manufacturer***

Первый столбец в выражении SELECT - Manufacturer представляет название группы, а второй столбец - ModelsCount представляет результат функции Count, которая вычисляет количество строк в группе.



```
20 • SELECT Manufacturer, COUNT(*) AS ModelsCount
21 FROM Products
22 GROUP BY Manufacturer;
```

Result Grid			Filter Rows:	<input type="text"/>	Export:		Wrap Cell Con
	Manufacturer	ModelsCount					
▶	Apple	2					
	Samsung	2					
	Huawei	1					

## Агрегатные функции — count, sum, avg, обработка Null

Агрегатные функции вычисляют некоторые скалярные значения в наборе строк. В MySQL есть следующие агрегатные функции:

AVG: вычисляет среднее значение

SUM: вычисляет сумму значений

MIN: вычисляет наименьшее значение

MAX: вычисляет наибольшее значение

COUNT: вычисляет количество строк в запросе

### AVG

Функция Avg возвращает среднее значение на диапазоне значений столбца таблицы. Например, пусть есть следующая таблица товаров Products:

```

1 CREATE TABLE Products
2 (
3     Id INT AUTO_INCREMENT PRIMARY KEY,
4     ProductName VARCHAR(30) NOT NULL,
5     Manufacturer VARCHAR(20) NOT NULL,
6     ProductCount INT DEFAULT 0,
7     Price DECIMAL NOT NULL
8 );
9
10 INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)
11 VALUES
12 ('iPhone X', 'Apple', 3, 76000),
13 ('iPhone 8', 'Apple', 2, 51000),
14 ('iPhone 7', 'Apple', 5, 32000),
15 ('Galaxy S9', 'Samsung', 2, 56000),
16 ('Galaxy S8', 'Samsung', 1, 46000),
17 ('Honor 10', 'Huawei', 5, 28000),
18 ('Nokia 8', 'HMD Global', 6, 38000)

```

Средняя цена товаров из БД:

Для поиска среднего значения в качестве выражения в функцию передается столбец Price. Для получаемого значения устанавливается псевдоним Average\_Price, хотя в принципе устанавливать псевдоним необязательно.

```

1 SELECT AVG(Price) AS Average_Price FROM Products

```

```

1 • USE productsdb;
2
3 • SELECT AVG(Price) AS Average_Price FROM Products

```

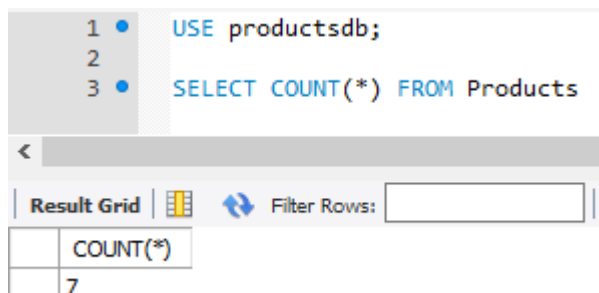
Average_Price
46714.2857

## Count

Функция Count вычисляет количество строк в выборке. Есть две формы этой функции. Первая форма COUNT(\*) подсчитывает

число строк в выборке:

```
SELECT COUNT(*) FROM Products
```



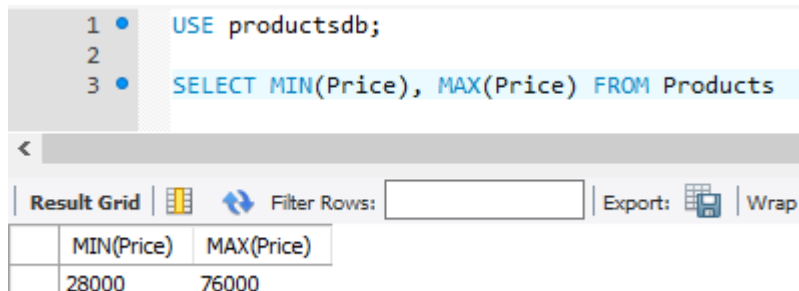
The screenshot shows a SQL IDE with three lines of code: 1. USE productsdb; 2. 3. SELECT COUNT(\*) FROM Products. Below the code, there is a 'Result Grid' tab and a 'Filter Rows' input field. The result grid contains two columns: 'COUNT(\*)' and a single row with the value '7'.

COUNT(*)
7

## Min и Max

Функции Min и Max вычисляют минимальное и максимальное значение по столбцу соответственно. Например, найдем минимальную цену среди товаров:

```
SELECT MIN(Price), MAX(Price) FROM Products;
```



The screenshot shows a SQL IDE with three lines of code: 1. USE productsdb; 2. 3. SELECT MIN(Price), MAX(Price) FROM Products. Below the code, there is a 'Result Grid' tab and a 'Filter Rows' input field. The result grid contains two columns: 'MIN(Price)' and 'MAX(Price)'. The first row shows the values '28000' and '76000'.

MIN(Price)	MAX(Price)
28000	76000

Данные функции также игнорируют значения NULL и не учитывают их при подсчете.

## **HAVING**

MySQL оператор HAVING используется в сочетании с оператором GROUP BY, чтобы ограничить группы возвращаемых строк только тех, чье условие TRUE.

Синтаксис оператора HAVING в MySQL:

SELECT expression1, expression2, ... expression\_n,

aggregate\_function (expression)

FROM tables

[WHERE conditions]

GROUP BY expression1, expression2, ... expression\_n

HAVING condition;

### **Параметры или аргументы**

`aggregate_function` - функция, такая как функции `SUM`, `COUNT`, `MIN`, `MAX` или `AVG`.

`expression1`, `expression2`, ... `expression_n` - выражения, которые не заключены в агрегированную функцию и должны быть включены в предложение GROUP BY.

`WHERE conditions` - необязательный. Это условия для выбора записей.

`HAVING condition` - Это дополнительное условие применяется только к агрегированным результатам для ограничения групп возвращаемых строк. В результирующий набор будут включены только те группы, состояние которых соответствует TRUE.

## Фильтрация групп. HAVING

Оператор HAVING позволяет выполнить фильтрацию групп, то есть определяет, какие группы будут включены в выходной результат. Использование HAVING во многом аналогично применению WHERE. Только если WHERE применяется для фильтрации строк, то HAVING - для фильтрации групп. Например, найдем все группы товаров по производителям, для которых определено более 1 модели:

```
1 SELECT Manufacturer, COUNT(*) AS ModelsCount
2 FROM Products
3 GROUP BY Manufacturer
4 HAVING COUNT(*) > 1
```

```
1 SELECT Manufacturer, COUNT(*) AS ModelsCount
2 FROM Products
3 WHERE Price * ProductCount > 80000
4 GROUP BY Manufacturer
5 HAVING COUNT(*) > 1;
```

То есть в данном случае сначала фильтруются строки: выбираются те товары, общая стоимость которых больше 80000. Затем выбранные товары группируются по производителям. И далее фильтруются сами группы - выбираются те группы, которые содержат больше 1 модели. Если при этом необходимо провести сортировку, то выражение ORDER BY идет после выражения HAVING:

```
SELECT Manufacturer, COUNT(*) AS Models, SUM(Product Count) AS Units
```

FROM Products

WHERE Price \* ProductCount > 80000

GROUP BY Manufacturer




HAVING SUM(ProductCount) > 2

ORDER BY Units DESC;

Здесь группировка идет по производителям, и также выбирается количество моделей для каждого производителя (Models) и общее количество всех товаров по всем этим моделям (Units). В конце группы сортируются по количеству товаров по убыванию.

```
1 • USE productsdb;
2
3 • SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units
4 FROM Products
5 WHERE Price * ProductCount > 80000
6 GROUP BY Manufacturer
7 HAVING SUM(ProductCount) > 2
8 ORDER BY Units DESC;
```

<

Result Grid |  Filter Rows:  | Export:  | Wrap Cell Content: 

	Manufacturer	Models	Units
	Apple	3	10
	HMD Global	1	6
	Huawei	1	5

```
21 • SELECT Manufacturer, COUNT(*) AS Models, ProductCount
22 FROM main
23 WHERE Price > 40000
24 GROUP BY Manufacturer;
```

Result Grid



Filter Rows:

Export:



Wrap Cell Content:



	Manufacturer	Models	ProductCount
▶	Apple	2	3
	Samsung	2	2

Доп задание: 5 минутка

Группировка происходит по моделям, подсчитывается количество моделей в исходной таблице и количество товара той или фирмы. Так как 2 поля “Apple” повторяются, выводится наибольшее значение: поле с ценой в 76000 (выполняется условие, при котором цена должна быть больше 40000)

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:

## Приоритет операций

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.



## **Итоги**

Сегодня мы изучили способы сортировки по полю, агрегатные функции. Разобрались, как задавать только уникальные значения, узнали про приоритет операций и использовали HAVING в запросах с GROUP BY.

На следующей лекции мы изучим применение операторов join, union, подзапросов