# **Основные операции с данными**

## **Добавление данных. Команда INSERT**

Для добавления данных в БД в MySQL используется команда **INSERT**, которая имеет следующий формальный синтаксис:

|  |  |
| --- | --- |
| 1 | INSERT [INTO] имя\_таблицы [(список\_столбцов)] VALUES (значение1, значение2, ... значениеN) |

После выражения **INSERT INTO** в скобках можно указать список столбцов через запятую, в которые надо добавлять данные, и в конце после слова **VALUES** скобках перечисляют добавляемые для столбцов значения.

Например, пусть в базе данных productsdb есть следующая таблица Products:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CREATE DATABASE productsdb;  USE productsdb;  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 NOT NULL  ); |

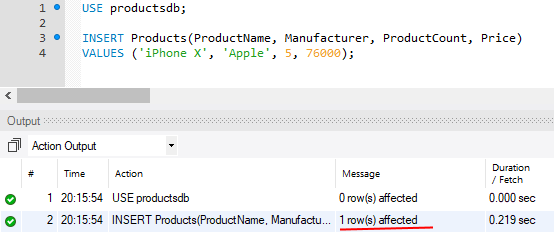
Добавим в эту таблицу одну строку с помощью следующего кода:

|  |  |
| --- | --- |
| 1  2 | INSERT Products(ProductName, Manufacturer, ProductCount, Price)  VALUES ('iPhone X', 'Apple', 5, 76000); |

В данно случае значения будут передаваться столбцам по позиции. То есть стобцу ProductName передается строка "iPhone X", столбцу Manufacturer - строка "Apple" и так далее.

Важно, чтобы между значениями и типами данных столбцов было соответствие. Так, столбец ProductName представляет тип varchar, то есть строку. Соответственно этому столбцу мы можем передать строковое значение в одинарных кавычках. А стобец ProductCount представляет тип int, то есть целое число, поэтому данному столбцу нужно передать целые числа, но никак не строки.

После удачного выполнения в MySQL Workbench в поле вывода должны появиться зеленый маркер и сообщение "1 row(s) affected":



Необязательно при добавлении данных указывать значения абсолютно для всех столбцов таблицы. Например, в примере выше не указано значение для стобца Id. Но поскольку для данного столбца определен атрибут AUTO\_INCREMENT, то его значение будет автоматически генерироваться.

Также мы можем опускать при добавлении такие столбцы, которые поддерживают значение NULL или для которых указано значение по умолчанию, то есть для них определены атрибуты NULL или DEFAULT. Так, в таблице Products столбец ProductCount имеет значение по умолчанию - число 0. Поэтому мы можем при добавлении опустить этот столбец, и ему будет передаваться число 0:

|  |  |
| --- | --- |
| 1  2 | INSERT Products(ProductName, Manufacturer, Price)  VALUES ('Galaxy S9', 'Samsung', 63000); |

С помощью ключевых слов DEFAULT и NULL можно указать, что в качестве значения будет использовать значение по умолчанию или NULL соответственно:

|  |  |
| --- | --- |
| 1  2 | INSERT Products(ProductName, Manufacturer, Price, ProductCount)  VALUES ('Nokia 9', 'HDM Global', 41000, DEFAULT); |

или

|  |  |
| --- | --- |
| 1  2 | INSERT Products(ProductName, Manufacturer, Price, ProductCount)  VALUES ('Nokia 9', 'HDM Global', 41000, NULL); |

### **Множественное добавление**

Также мы можем добавить сразу несколько строк:

|  |  |
| --- | --- |
| 1  2  3  4  5 | INSERT Products(ProductName, Manufacturer, Price, ProductCount)  VALUES  ('iPhone 8', 'Apple', 51000, 3),  ('P20 Lite', 'Huawei', 34000, 4),  ('Galaxy S8', 'Samsung', 46000, 2); |

В данном случае в таблицу будут добавлены три строки.

## **Выборка данных. Команда SELECT**

Для выборки данных из БД в MySQL применяется команда **SELECT**. В упрощенном виде она имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | SELECT список\_столбцов FROM имя\_таблицы |

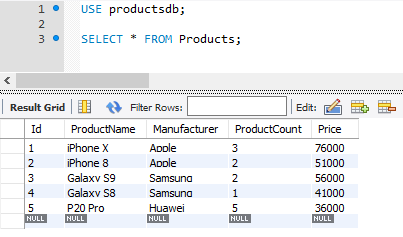
Например, пусть ранее была создана таблица Products, и в нее добавлены некоторые начальные данные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | 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); |

Получим все объекты из этой таблицы:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM Products; |

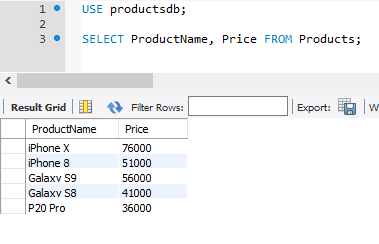
Символ звездочка \* указывает, что нам надо получить все столбцы.



Стоит отметить, что применение звездочки \* для получения данных считается не очень хорошей практикой, так как обычно необходимо получить данные по небольшому набору столбцов. Поэтому более оптимальный подход заключается в указании всех необходимых столбцов после слова SELECT. Исключение составляет тот случай, когда надо получить данные по абсолютно всем столбцам таблицы. Также использование символа \* может быть предпочтительно тогда, когда названия столбцов не известны.

Если необходимо получить данные не из всех, а из каких-то конкретных столбцов, тогда спецификации этих столбцов перечисляются через запятую после SELECT:

|  |  |
| --- | --- |
| 1 | SELECT ProductName, Price FROM Products; |



Спецификация столбца необязательно должна представлять его название. Это может быть любое выражение, например, результат арифметической операции. Так, выполним следующий запрос:

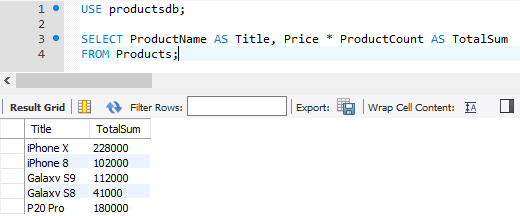
|  |  |
| --- | --- |
| 1  2 | SELECT ProductName, Price \* ProductCount  FROM Products; |

Здесь при выборке будут создаваться два столбца. Причем второй столбец представляет значение столбца Price, умноженное на значение столбца ProductCount, то есть совокупную стоимость товара.

С помощью оператора **AS** можно изменить название выходного столбца или определить его псевдоним:

|  |  |
| --- | --- |
| 1  2 | SELECT ProductName AS Title, Price \* ProductCount AS TotalSum  FROM Products; |

Здесь для первого столбца определяется псевдоним Title, хотя в реальности он будет представлять столбец ProductName. Второй столбец TotalSum хранит произведение столбцов ProductCount и Price.



## **Фильтрация данных. Оператор WHERE**

Зачастую необходимо извлекать не все данные из БД, а только те, которые соответствуют определенному условию. Для фильтрации данных в команде SELECT применяется оператор **WHERE**, после которого указывается условие:

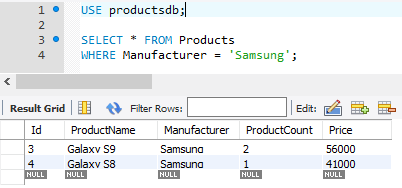
|  |  |
| --- | --- |
| 1 | WHERE условие |

Если условие истинно, то строка попадает в результирующую выборку. В качестве можно использовать операции сравнения, которые сравнивают два выражения:

* **=**: сравнение на равенство
* **!=**: сравнение на неравенство
* **<>**: сравнение на неравенство
* **<**: меньше чем
* **>**: больше чем
* **<=**: меньше чем или равно
* **>=**: больше чем или равно

К примеру, выберем всех товары, производителем которых является компания Samsung:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Manufacturer = 'Samsung'; |



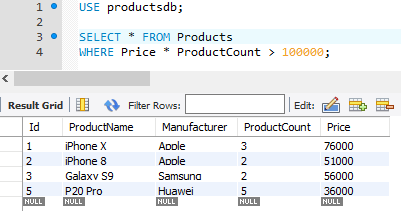
Стоит отметить, что для MySQL не важен регистр символов, и, к примеру, строка "Samsung" будет эквивалентна строке "SAMSUNG" или "sumSunG".

Другой пример - найдем все товары, количество которых меньше 3:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE ProductCount < 3; |

Критерий фильтрации может представлять и более сложное составное выражение. Например, найдем все товары, у которых совокупная стоимость больше 100 000:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price \* ProductCount > 100000; |



### **Логические операторы**

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

* **AND**: операция логического И. Она объединяет два выражения:

|  |  |
| --- | --- |
| 1 | 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:

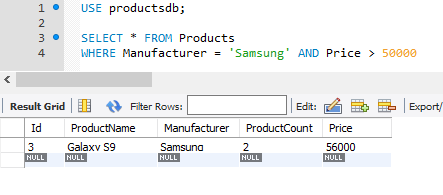
|  |  |
| --- | --- |
| 1 | выражение1 OR выражение2 |

* 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.

|  |  |
| --- | --- |
| 1 | NOT выражение |

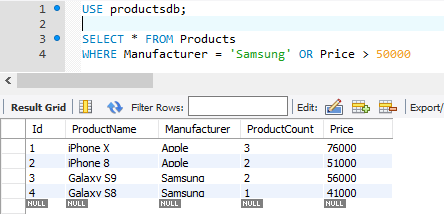
Например, выберем все товары, у которых производитель Samsung и одновременно цена больше 50000:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Manufacturer = 'Samsung' AND Price > 50000 |



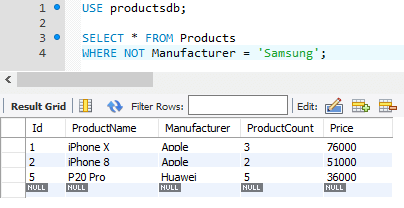
Теперь изменим оператор на **OR**. То есть выберем все товары, у которых либо производитель Samsung, либо цена больше 50000:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Manufacturer = 'Samsung' OR Price > 50000 |



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

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE NOT Manufacturer = 'Samsung'; |



### **Приоритет операций**

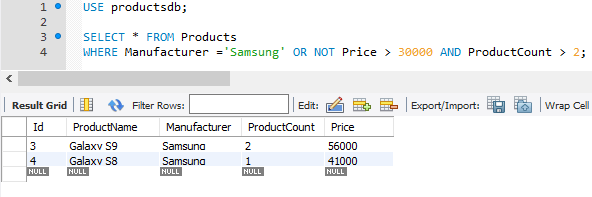
В одном условии при необходимости мы можем объединять несколько логических операций. Однако следует учитывать, что самой приоритетной операцией, которая выполняется в первую очередь, является NOT, менее приоритетная - AND и операция с наименьшим приоритетом - OR. Например:

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

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

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.

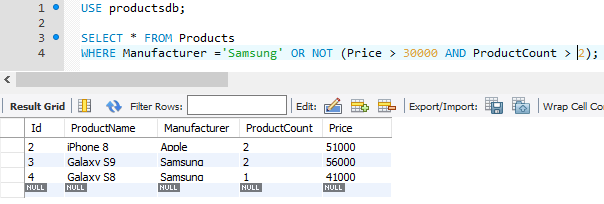
В конце вычисляется оператор OR - либо цена должна быть меньше или равна 30000 и одновременно количество товаров должно быть больше 2, либо производителем должен быть Samsung.



С помощью скобок можно переопределить приоритет операций:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  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.



## **Обновление данных. Команда UPDATE**

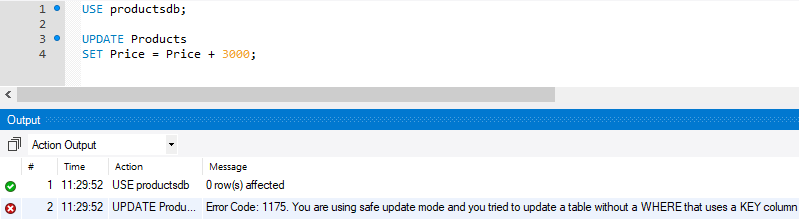
Команда **UPDATE** применяется для обновления уже имеющихся строк. Она имеет следующий формальный синтаксис:

|  |  |
| --- | --- |
| 1  2  3 | UPDATE имя\_таблицы  SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN  [WHERE условие\_обновления] |

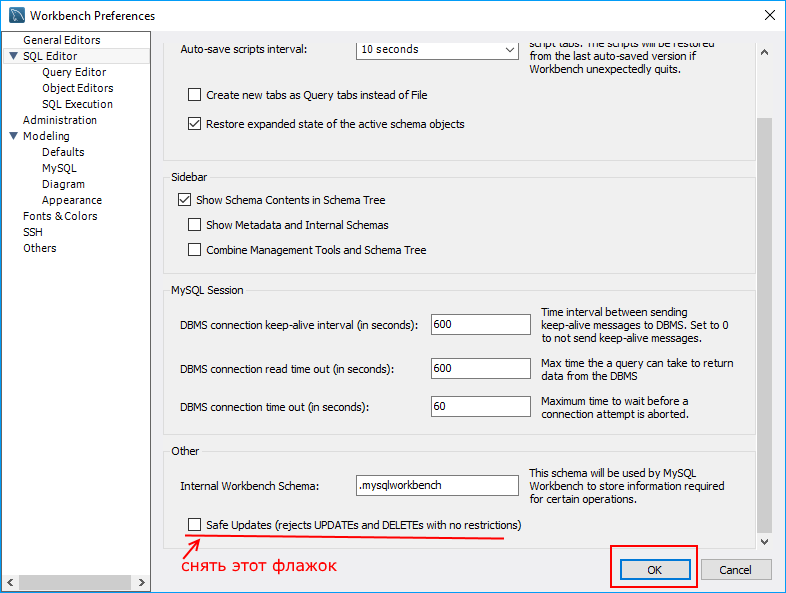
Например, увеличим у всех товаров цену на 3000:

|  |  |
| --- | --- |
| 1  2 | UPDATE Products  SET Price = Price + 3000; |

Однако при выполнении данного запроса в MySQL Workbench мы можем столкнуться с ошибкой:



Ошибка говорит о том, что мы находимся в безопасном режиме. И чтобы его отключить, в MySQL Workbench надо перейти в меню **Edit -> Preferences** и в открывшемся окне перейти к пункту **SQL Editor**:



**SET SQL\_SAFE\_UPDATES = 0;**

В открывшейся вкладке в самом низу надо снять флажок с поля "Safe Updates (reject UPDATEs and DELETEs with no restrictions)" и затем сохранить изменения, нажав на кнопку OK. После этого надо переподключиться к серверу.

Используем выражение WHERE и изменим название производителя с "Samsung" на "Samsung Inc.":

|  |  |
| --- | --- |
| 1  2  3 | UPDATE Products  SET Manufacturer = 'Samsung Inc.'  WHERE Manufacturer = 'Samsung'; |

Также можно обновлять сразу несколько столбцов:

|  |  |
| --- | --- |
| 1  2  3  4 | UPDATE Products  SET Manufacturer = 'Samsung',  ProductCount = ProductCount + 3  WHERE Manufacturer = 'Samsung Inc.'; |

При обновлении вместо конкретных значений и выражений мы можем использовать ключевые слова **DEFAULT** и **NULL** для установки соответственно значения по умолчанию или NULL:

|  |  |
| --- | --- |
| 1  2  3 | UPDATE Products  SET ProductCount= DEFAULT  WHERE Manufacturer = 'Huawei'; |

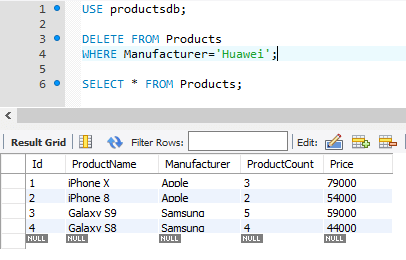
## **Удаление данных. Команда DELETE**

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

|  |  |
| --- | --- |
| 1  2 | DELETE FROM имя\_таблицы  [WHERE условие\_удаления] |

Например, удалим строки, у которых производитель - Huawei:

|  |  |
| --- | --- |
| 1  2 | DELETE FROM Products  WHERE Manufacturer='Huawei'; |



Или удалим все товары, производителем которых является Apple и которые имеют цену меньше 60000:

|  |  |
| --- | --- |
| 1  2 | DELETE FROM Products  WHERE Manufacturer='Apple' AND Price < 60000; |

Если необходимо вовсе удалить все строки вне зависимости от условия, то условие можно не указывать:

|  |  |
| --- | --- |
| 1 | DELETE FROM Products; |

# **Запросы**

## **Выборка уникальных значений. Оператор DISTINCT**

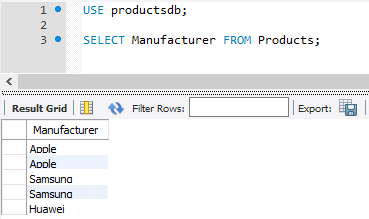
С помощью оператора **DISTINCT** можно выбрать уникальные данные по определенным столбцам.

К примеру, разные товары могут иметь одних и тех же производителей, и, допустим, у нас следующая таблица товаров:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | USE productsdb;    DROP TABLE IF EXISTS Products;    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 NOT NULL  );  INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)  VALUES  ('iPhone X', 'Apple', 3, 71000),  ('iPhone 8', 'Apple', 3, 56000),  ('Galaxy S9', 'Samsung', 6, 56000),  ('Galaxy S8', 'Samsung', 2, 46000),  ('Honor 10', 'Huawei', 3, 26000); |

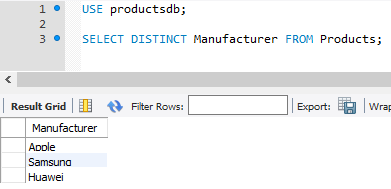
Выберем всех производителей:

|  |  |
| --- | --- |
| 1 | SELECT Manufacturer FROM Products; |



Однако при таком запросе производители повторяются. Теперь применим оператор **DISTINCT** для выборки уникальных значений:

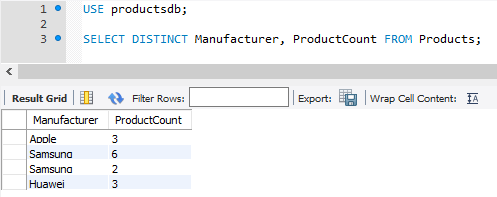
|  |  |
| --- | --- |
| 1 | SELECT DISTINCT Manufacturer FROM Products; |



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:



## **Операторы фильтрации**

### **Оператор IN**

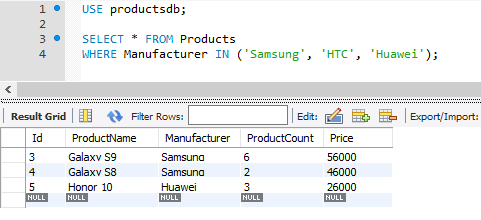
Оператор **IN** определяет набор значений, которые должны иметь столбцы:

|  |  |
| --- | --- |
| 1 | WHERE выражение [NOT] IN (выражение) |

Выражение в скобках после IN определяет набор значений. Этот набор может вычисляться динамически на основании, например, еще одного запроса, либо это могут быть константные значения.

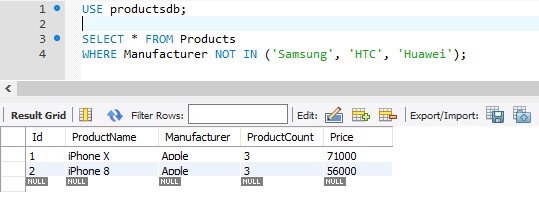
Например, выберем товары, у которых производитель либо Samsung, либо Xiaomi, либо Huawei:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Manufacturer IN ('Samsung', 'HTC', 'Huawei'); |



Оператор **NOT**, наоборот, позволяет выбрать все строки, столбцы которых не имеют определенных значений:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Manufacturer NOT IN ('Samsung', 'HTC', 'Huawei'); |



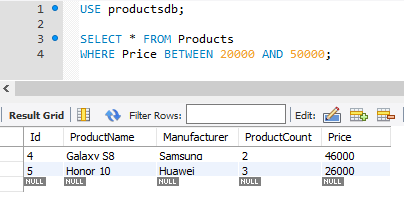
### **Оператор BETWEEN**

Оператор **BETWEEN** определяет диапазон значений с помощью начального и конечного значения, которому должно соответствовать выражение:

|  |  |
| --- | --- |
| 1 | WHERE выражение [NOT] BETWEEN начальное\_значение AND конечное\_значение |

Например, получим все товары, у которых цена от 20 000 до 50 000 (начальное и конечное значения также включаются в диапазон):

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price BETWEEN 20000 AND 50000; |



Если надо, наоборот, выбрать те строки, которые не попадают в данный диапазон, то добавляется оператор **NOT**:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price NOT BETWEEN 20000 AND 50000; |

Также можно использовать более сложные выражения. Например, получим товары по совокупной стоимости (цена \* количество):

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price \* ProductCount BETWEEN 90000 AND 150000; |

### **Операторы LIKE и REGEXP**

Оператор **LIKE** принимает шаблон строки, которому должно соответствовать выражение.

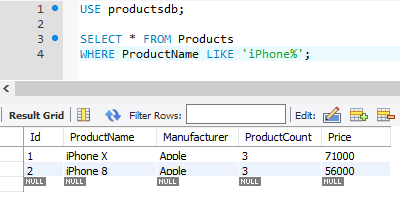
|  |  |
| --- | --- |
| 1 | WHERE выражение [NOT] LIKE шаблон\_строки |

Для определения шаблона могут применяться ряд специальных символов подстановки:

* **%**: соответствует любой подстроке, которая может иметь любое количество символов, при этом подстрока может и не содержать ни одного символа  
  Например, выражение WHERE ProductName LIKE 'Galaxy%' соответствует таким значениям как "Galaxy Ace 2" или "Galaxy S7"
* **\_**: соответствует любому одиночному символу  
  Например, выражение WHERE ProductName LIKE 'Galaxy S\_' соответствует таким значениям как "Galaxy S7" или "Galaxy S8".

Применим оператор LIKE:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE ProductName LIKE 'iPhone%'; |



**REGEXP** позволяет задать регулярное выражение, которому должно соответствовать значение столбца. В этом плане REGEXP представляет более изощренный и комплексный способ фильтрации, нежели оператор LIKE. REGEXP имеет похожий синтаксис:

|  |  |
| --- | --- |
| 1 | WHERE выражение [NOT] REGEXP регулярное выражение |

Регулярное выражение может принимать следующие специальные символы:

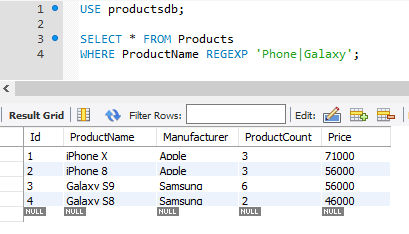
* **^**: указывает на начало строки
* **$**: указывает на конец строки
* **.**: соответствует любому одиночному символу
* **[символы]**: соответствует любому одиночному символу из скобок
* **[начальный\_символ-конечный\_символ]**: соответствует любому одиночному символу из диапазона символов
* **|**: отделяет два шаблона строки, и значение должно соответствовать одну из этих шаблонов

Примеры REGEXP:

* WHERE ProductName REGEXP 'Phone': строка должна содержать "Phone", например, iPhone X, Nokia Phone N, iPhone
* WHERE ProductName REGEXP '^Phone': строка должна начинаться с "Phone", например, Phone 34, PhoneX
* WHERE ProductName REGEXP 'Phone$': строка должна заканчиваться на "Phone", например, iPhone, Nokia Phone
* WHERE ProductName REGEXP 'iPhone [78]';: строка должна содержать либо iPhone 7, либо iPhone 8
* WHERE ProductName REGEXP 'iPhone [6-8]';: строка должна содержать либо iPhone 6, либо iPhone 7, либо iPhone 8

Например, найдем товары, названия которых содержат либо "Phone", либо "Galaxy":

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE ProductName REGEXP 'Phone|Galaxy'; |



### **IS NULL**

Оператор **IS NULL** позволяет выбрать все строки, столбцы которых имеют значение NULL:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE ProductCount IS NULL; |

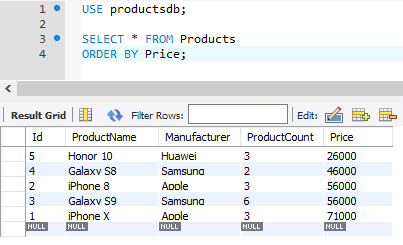
С помощью добавления оператора **NOT** можно, наоброт, выбрать строки, столбцы которых не имеют значения NULL:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE ProductCount IS NOT NULL; |

## **Сортировка. ORDER BY**

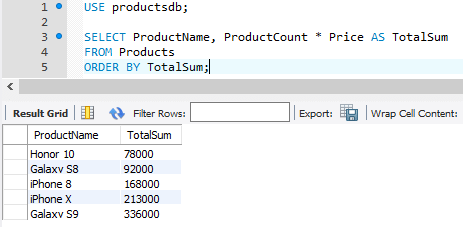
Оператор **ORDER BY** сортируют значения по одному или нескольких столбцам. Например, упорядочим выборку из таблицы Products по столбцу Price:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  ORDER BY Price; |



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

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, ProductCount \* Price AS TotalSum  FROM Products  ORDER BY TotalSum; |



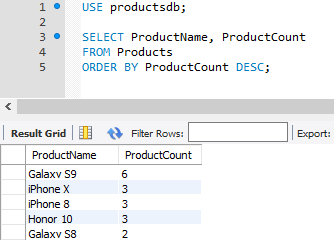
В качестве критерия сортировки также можно использовать сложно выражение на основе столбцов:

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, Price, ProductCount  FROM Products  ORDER BY ProductCount \* Price; |

### **Сортировка по убыванию**

По умолчанию данные сортируются по возрастанию, однако с помощью оператора **DESC** можно задать сортировку по убыванию.

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, ProductCount  FROM Products  ORDER BY ProductCount DESC; |



По умолчанию вместо DESC используется оператор **ASC**, который сортирует по возрастанию:

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, ProductCount  FROM Products  ORDER BY ProductCount ASC; |

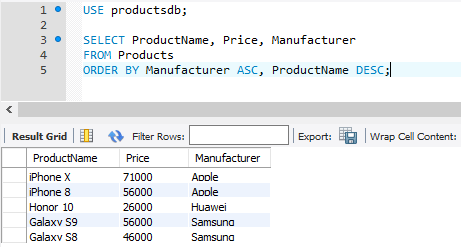
### **Сотировка по нескольким столбцам**

При сортировке сразу по нескольким столбцам все эти столбцы указываются через запятую после оператора **ORDER BY**:

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, Price, Manufacturer  FROM Products  ORDER BY Manufacturer, ProductName; |

Здесь строки сначала сортируются по столбцу Manufacturer по возрастанию. Затем если есть две строки, в которых столбец Manufacturer имеет одинаковое значение, то они сортируются по столбцу ProductName также по возрастанию. Но опять же с помощью ASC и DESC можно отдельно для разных столбцов определить сортировку по возрастанию и убыванию:

|  |  |
| --- | --- |
| 1  2  3 | SELECT ProductName, Price, Manufacturer  FROM Products  ORDER BY Manufacturer ASC, ProductName DESC; |



## **Получение диапазона строк. Оператор LIMIT**

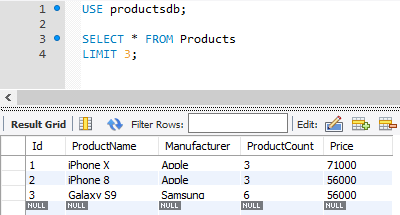
Оператор **LIMIT** позволяет извлечь определенное количество строк и имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | LIMIT [offset,] rowcount |

Если оператору LIMIT передается один параметр, то он указывает на количество извлекаемых строка. 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:

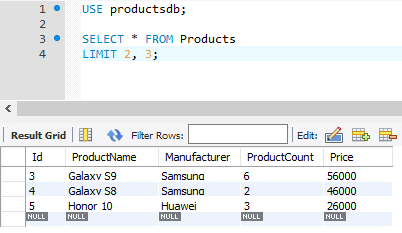
|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  LIMIT 3; |



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

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  LIMIT 2, 3; |

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



Как правило, оператор LIMIT используетс вместе с оператором ORBER BY:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM Products  ORDER BY ProductName  LIMIT 2, 3; |

## **Агрегатные функции**

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

Все агрегатные функции принимают в качестве параметра выражение, которое представляет критерий для определения значений. Зачастую, в качестве выражения выступает название столбца, над значениями которого надо проводить вычисления.

Выражения в функциях **AVG** и **SUM** должно представлять числовое значение (например, столбец, который хранит числовые значения). Выражение в функциях **MIN**, **MAX** и **COUNT** может представлять числовое или строковое значение или дату.

Все агрегатные функции за исключением COUNT(\*) игнорируют значения NULL.

### **Avg**

Функция **Avg** возвращает среднее значение на диапазоне значений столбца таблицы.

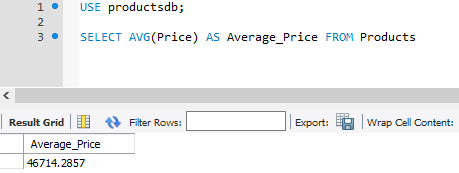
Например, пусть есть следующая таблица товаров Products:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 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 NOT NULL  );    INSERT INTO Products(ProductName, Manufacturer, ProductCount, Price)  VALUES  ('iPhone X', 'Apple', 3, 76000),  ('iPhone 8', 'Apple', 2, 51000),  ('iPhone 7', 'Apple', 5, 32000),  ('Galaxy S9', 'Samsung', 2, 56000),  ('Galaxy S8', 'Samsung', 1, 46000),  ('Honor 10', 'Huawei', 5, 28000),  ('Nokia 8', 'HMD Global', 6, 38000) |

Найдем среднюю цену товаров из базы данных:

|  |  |
| --- | --- |
| 1 | SELECT AVG(Price) AS Average\_Price FROM Products |

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.



На этапе выборки можно применять фильтрацию. Например, найдем среднюю цену для товаров определенного производителя:

|  |  |
| --- | --- |
| 1  2 | SELECT AVG(Price) FROM Products  WHERE Manufacturer='Apple' |

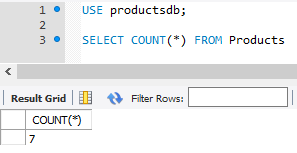
Также можно находить среднее значение для более сложных выражений. Например, найдем среднюю сумму всех товаров, учитывая их количество:

|  |  |
| --- | --- |
| 1 | SELECT AVG(Price \* ProductCount) FROM Products |

### **Count**

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

|  |  |
| --- | --- |
| 1 | SELECT COUNT(\*) FROM Products |



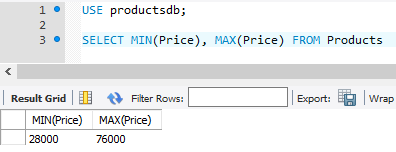
Вторая форма функции вычисляет количество строк по определенному столбцу, при этом строки со значениями NULL игнорируются:

|  |  |
| --- | --- |
| 1 | SELECT COUNT(Manufacturer) FROM Products |

### **Min и Max**

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

|  |  |
| --- | --- |
| 1 | SELECT MIN(Price), MAX(Price) FROM Products |



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

### **Sum**

Функция **Sum** вычисляет сумму значений столбца. Например, подсчитаем общее количество товаров:

|  |  |
| --- | --- |
| 1 | SELECT SUM(ProductCount) FROM Products |

Также вместо имени столбца может передаваться вычисляемое выражение. Например, найдем общую стоимость всех имеющихся товаров:

|  |  |
| --- | --- |
| 1 | SELECT SUM(ProductCount \* Price) FROM Products |

### **All и Distinct**

По умолчанию все вышеперечисленных пять функций учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющие значения. Если необходимо выполнить вычисления только над уникальными значениями, исключив из набора значений повторяющиеся данные, то для этого применяется оператор **DISTINCT**.

|  |  |
| --- | --- |
| 1 | SELECT COUNT(DISTINCT Manufacturer) FROM Products |

По умолчанию вместо DISTINCT применяется оператор **ALL**, который выбирает все строки:

|  |  |
| --- | --- |
| 1 | SELECT COUNT(ALL Manufacturer) FROM Products |

В данном случае мы видим, что производители могут повторяться в таблице, так как некоторые товары могут иметь одних и тех же производителей. Поэтому чтобы подсчитать количество уникальных производителей, необходимо использовать оператор DISTINCT.

Так как этот оператор неявно подразумевается при отсутствии DISTINCT, то его можно не указывать.

### **Комбинирование функций**

Объединим применение нескольких функций:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT COUNT(\*) AS ProdCount,  SUM(ProductCount) AS TotalCount,  MIN(Price) AS MinPrice,  MAX(Price) AS MaxPrice,  AVG(Price) AS AvgPrice  FROM Products |

## **Группировка**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT столбцы  FROM таблица  [WHERE условие\_фильтрации\_строк]  [GROUP BY столбцы\_для\_группировки]  [HAVING условие\_фильтрации\_групп]  [ORDER BY столбцы\_для\_сортировки] |

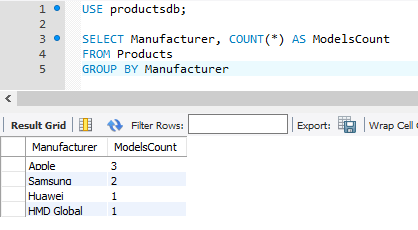
### **GROUP BY**

Оператор **GROUP BY** определяет, как строки будут группироваться.

Например, сгруппируем товары по производителю

|  |  |
| --- | --- |
| 1  2  3 | 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.



И если в выражении SELECT производится выборка по одному или нескольким столбцам и также используются агрегатные функции, то необходимо использовать выражение GROUP BY. Так, следующий пример работать не будет, так как он не содержит выражение группировки:

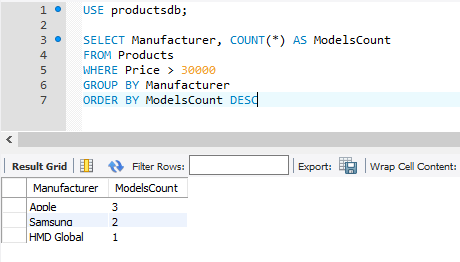
|  |  |
| --- | --- |
| 1  2 | SELECT Manufacturer, COUNT(\*) AS ModelsCount  FROM Products |

Оператор GROUP BY может выполнять группировку по множеству столбцов. Так, добавим группировку по количеству товаров:

|  |  |
| --- | --- |
| 1  2  3 | SELECT Manufacturer, ProductCount, COUNT(\*) AS ModelsCount  FROM Products  GROUP BY Manufacturer, ProductCount |

Следует учитывать, что выражение GROUP BY должно идти после выражения WHERE, но до выражения ORDER BY:

|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT Manufacturer, COUNT(\*) AS ModelsCount  FROM Products  WHERE Price > 30000  GROUP BY Manufacturer  ORDER BY ModelsCount DESC |



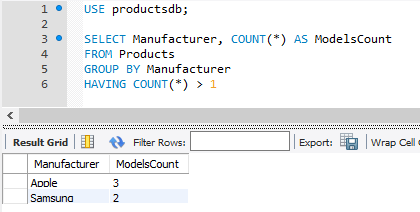
### **Фильтрация групп. HAVING**

Оператор **HAVING** позволяет выполнить фильтрацию групп, то есть определяет, какие группы будут включены в выходной результат.

Использование HAVING во многом аналогично применению WHERE. Только есть WHERE применяется для фильтрации строк, то HAVING - для фильтрации групп.

Например, найдем все группы товаров по производителям, для которых определено более 1 модели:

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



В одной команде также можно сочетать выражения WHERE и HAVING:

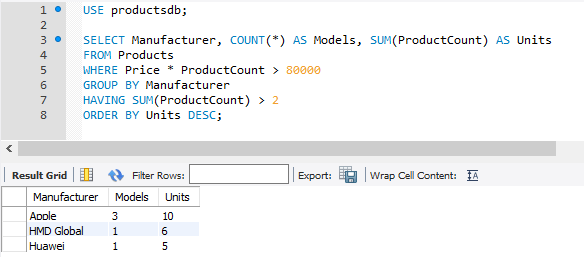
|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT Manufacturer, COUNT(\*) AS ModelsCount  FROM Products  WHERE Price \* ProductCount > 80000  GROUP BY Manufacturer  HAVING COUNT(\*) > 1; |

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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | SELECT Manufacturer, COUNT(\*) AS Models, SUM(ProductCount) 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.



## **Подзапросы**

Подзапросы представляют выражения SELECT, которые встроены в другие запросы SQL. Рассмотрим простейший пример применения подзапросов.

Например, создадим таблицы для товаров и заказов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | 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 NOT NULL  );  CREATE TABLE Orders  (  Id INT AUTO\_INCREMENT PRIMARY KEY,  ProductId INT NOT NULL,  ProductCount INT DEFAULT 1,  CreatedAt DATE NOT NULL,  Price DECIMAL NOT NULL,  FOREIGN KEY (ProductId) REFERENCES Products(Id) ON DELETE CASCADE  ); |

Таблица Orders содержит данные о купленным товарам из таблицы Products.

Добавим в таблицы некоторые данные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)  VALUES ('iPhone X', 'Apple', 2, 76000),  ('iPhone 8', 'Apple', 2, 51000),  ('iPhone 7', 'Apple', 5, 42000),  ('Galaxy S9', 'Samsung', 2, 56000),  ('Galaxy S8', 'Samsung', 1, 46000),  ('Honor 10', 'Huawei', 2, 26000),  ('Nokia 8', 'HMD Global', 6, 38000);    INSERT INTO Orders (ProductId, CreatedAt, ProductCount, Price)  VALUES  (  (SELECT Id FROM Products WHERE ProductName='Galaxy S8'),  '2018-05-21',  2,  (SELECT Price FROM Products WHERE ProductName='Galaxy S8')  ),  (  (SELECT Id FROM Products WHERE ProductName='iPhone X'),  '2018-05-23',  1,  (SELECT Price FROM Products WHERE ProductName='iPhone X')  ),  (  (SELECT Id FROM Products WHERE ProductName='iPhone 8'),  '2018-05-21',  1,  (SELECT Price FROM Products WHERE ProductName='iPhone 8')  ); |

При добавлении данных в таблицу Orders как раз используются подзапросы. Например, первый заказ был сделан на товар Galaxy S8. Соответственно в таблицу Orders нам надо сохранить информацию о заказе, где поле ProductId указывает на Id товара Galaxy S8, поле Price - на его цену. Но на момент написания запроса нам может быть неизвестен ни Id покупателя, ни Id товара, ни цена товара. В этом случае можно выполнить подзапрос в виде

|  |  |
| --- | --- |
| 1 | (SELECT Price FROM Products WHERE ProductName='iPhone 8') |

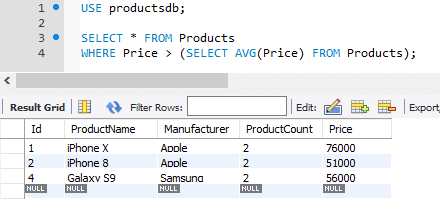
Подзапрос выполняет команду SELECT и заключается в скобки. В данном же случае при добавлении одного товара выполняется два подзапроса. Каждый подзапрос возвращает одного скалярное значение, например, числовой идентификатор.

В примере выше подзапросы выполнялись к другой таблице, но могут выполняться и к той же, для которой вызывается основной запрос. Например, найдем товары из таблицы Products, которые имеют минимальную цену:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price = (SELECT MIN(Price) FROM Products); |

Или найдем товары, цена которых выше средней:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price > (SELECT AVG(Price) FROM Products); |



### **Коррелирующие и некоррелирующие подзапросы**

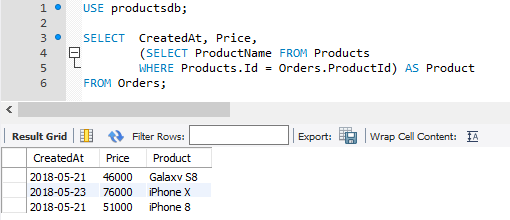
Подзапросы бывают коррелирующими и некоррелирующими. В примерах выше команды SELECT фактически выполняли один подзапрос для всех строк, извлекаемых командой. Например, подзапрос возвращает минимальную или среднюю цену, которая не изменится, сколько бы мы строк не выбирали в основном запросе. То есть результат подзапроса не зависел от строк, которые выбираются в основном запросе. И такой подзапрос выполняется один раз для всего внешнего запроса.

Но также можно использовать и **коррелирующие подзапросы** (correlated subquery), результаты которых зависят от строк, которые выбираются в основном запросе.

Например, выберем все заказы из таблицы Orders, добавив к ним информацию о товаре:

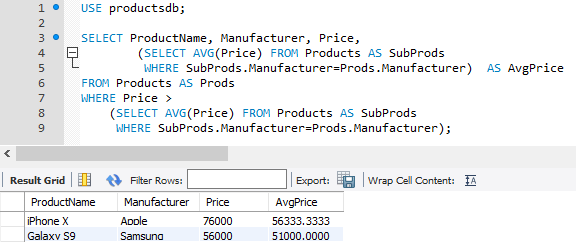
|  |  |
| --- | --- |
| 1  2  3  4 | SELECT CreatedAt, Price,  (SELECT ProductName FROM Products  WHERE Products.Id = Orders.ProductId) AS Product  FROM Orders; |

В данном случае для каждой строки из таблицы Orders будет выполняться подзапрос, результат которого зависит от столбца ProductId. И каждый подзапрос может возвращать различные данные.



Коррелирующий подзапрос может выполняться и для той же таблицы, к которой выполняется основной запрос. Например, выберем из таблицы Products те товары, стоимость которых выше средней цены товаров для данного производителя:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | SELECT ProductName,  Manufacturer,  Price,  (SELECT AVG(Price) FROM Products AS SubProds  WHERE SubProds.Manufacturer=Prods.Manufacturer) AS AvgPrice  FROM Products AS Prods  WHERE Price >  (SELECT AVG(Price) FROM Products AS SubProds  WHERE SubProds.Manufacturer=Prods.Manufacturer); |



Здесь определено два коррелирующих подзапроса. Первый подзапрос определяет спецификацию столбца AvgPrice. Он будет выполняться для каждой строки, извлекаемой из таблицы Products. В подзапрос передается производитель товара и на его основе выбирается средняя цена для товаров именно этого производителя. И так как производитель у товаров может отличаться, то и результат подзапроса в каждом случае также может отличаться.

Второй подзапрос аналогичен, только он используется для фильтрации извлекаемых из таблицы Products. И также он будет выполняться для каждой строки.

Чтобы избежать двойственности при фильтрации в подзапросе при сравнении производителей (SubProds.Manufacturer=Prods.Manufacturer) для внешней выборки установлен псевдоним Prods, а для выборки из подзапросов определен псевдоним SubProds.

Следует учитывать, что коррелирующие подзапросы выполняются для каждой отдельной строки выборки, то выполнение таких подзапросов может замедлять выполнение всего запроса в целом.

## **Подзапросы в основных командах SQL**

### **Подзапросы в SELECT**

В выражении SELECT мы можем вводить подзапросы четырьмя способами:

1. В условии в выражении WHERE
2. В условии в выражении HAVING
3. В качестве таблицы для выборки в выражении FROM
4. В качестве спецификации столбца в выражении SELECT

Рассмотрим некоторые из этих случаев. Например, получим все товары, у которых цена выше средней:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price > (SELECT AVG(Price) FROM Products) |

Чтобы получить нужные товары, нам вначале надо выполнить подзапрос на получение средней цены товара: SELECT AVG(Price) FROM Products.

#### **Оператор IN**

Нередко подзапросы применяются вместе с оператором **IN**, который выбирает из набора значений. И подзапрос как раз может предоставить требуемый набор значений. Например, выберем все товары из таблицы Products, на которые есть заказы в таблице Orders:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Id IN (SELECT ProductId FROM Orders) |

То есть подзапрос в данном случае выбирает все идентификаторы товаров из Orders, затем по этим идентификаторам извлекаютя товары из Products.

Добавив оператор **NOT**, мы можем выбрать те товары, на которые нет заказов в таблице Orders:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Id NOT IN (SELECT ProductId FROM Orders) |

Стоит отметить, что это не самый эффективный способ для извлечения связанных данных из других таблиц, так как для сведения данных из разных таблиц можно использовать оператор JOIN, который рассматривается в следующей главе.

#### **Получение набора значений**

При использовании в операторах сравнения подзапросы должны возвращать одно скалярное значение. Но иногда возникает необходимость получить набор значений. Чтобы при использовании в операторах сравнения подзапрос мог возвращать набор значений, перед ним необходимо использовать один из операторов: **ALL**, **SOME** или **ANY**.

При использовании ключевого слова **ALL** условие в операции сравнения должно быть верно для всех значений, которые возвращаются подзапросом. Например, найдем все товары, цена которых меньше чем у любого товара фирмы Apple:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price < ALL(SELECT Price FROM Products WHERE Manufacturer='Apple') |

Если бы мы в данном случае опустили бы ключевое слово ALL, то мы бы столкнулись с ошибкой.

Допустим, если данный подзапрос возвращает значения vаl1, val2 и val3, то условие фильтрации фактически было бы аналогично объединению этих значений через оператор AND:

|  |  |
| --- | --- |
| 1 | WHERE Price < val1 AND Price < val2 AND Price < val3 |

В тоже время подобный запрос гораздо проще переписать другим образом:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price < (SELECT MIN(Price) FROM Products WHERE Manufacturer='Apple') |

Как работает оператор ALL:

* x > ALL (1, 2) эквивалентно x > 2
* x < ALL (1, 2) эквивалентно x < 1
* x = ALL (1, 2) эквивалентно (x = 1) AND (x = 2)
* x <> ALL (1, 2) эквивалентно x NOT IN (1, 2)

Операторы ANY и SOME условие в операции сравнения должно быть истинным для хотя бы одного из значений, возвращаемых подзапросом. По своему действию оба этих оператора аналогичны, поэтому можно применять любой из них. Например, в следующем случае получим товары, которые стоят меньше самого дорогого товара компании Apple:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price < ANY(SELECT Price FROM Products WHERE Manufacturer='Apple') |

И также стоит отметить, что данный запрос можно сделать проще, переписав следующим образом:

|  |  |
| --- | --- |
| 1  2 | SELECT \* FROM Products  WHERE Price < (SELECT MAX(Price) FROM Products WHERE Manufacturer='Apple') |

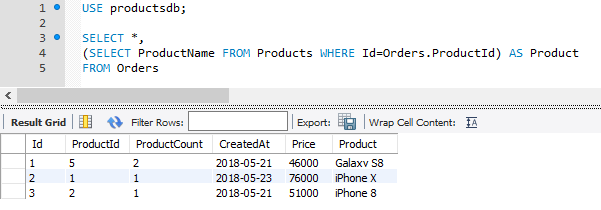
Как работает оператор ANY (а также SOME):

* x > ANY (1, 2) эквивалентно x > 1
* x < ANY (1, 2) эквивалентно x < 2
* x = ANY (1, 2) эквивалентно x NOT (1, 2)
* x <> ANY (1, 2) эквивалентно (x <> 1) OR (x <> 2)

#### **Подзапрос как спецификация столбца**

Результат подзапроса может представлять отдельный столбец в выборке. Например, выберем все заказы и добавим к ним информацию о названии товара:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \*,  (SELECT ProductName FROM Products WHERE Id=Orders.ProductId) AS Product  FROM Orders |



### **Подзапросы в команде INSERT**

В команде INSERT подзапросы могут применяться для определения значения, которое вставляется в один из столбцов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | INSERT INTO Orders (ProductId, CreatedAt, ProductCount, Price)  VALUES  (  (SELECT Id FROM Products WHERE ProductName='Galaxy S8'),  '2018-05-23',  2,  (SELECT Price FROM Products WHERE ProductName='Galaxy S8')  ) |

### **Подзапросы в команде UPDATE**

В команде UPDATE подзапросы могут применяться:

1. В качестве устанавливаемого значения после оператора SET
2. Как часть условия в выражении WHERE

Так, увеличим в таблице Orders количество купленных товаров компании Apple на 2:

|  |  |
| --- | --- |
| 1  2  3 | UPDATE Orders  SET ProductCount = ProductCount + 2  WHERE ProductId IN (SELECT Id FROM Products WHERE Manufacturer='Apple'); |

Или установим для заказа цену товара, полученную в результате подзапроса:

|  |  |
| --- | --- |
| 1  2  3 | UPDATE Orders  SET Price = (SELECT Price FROM Products WHERE Id=Orders.ProductId) + 3000  WHERE Id=1; |

### **Подзапросы в команде DELETE**

В команде DELETE подзапросы также применяются как часть условия. Так, удалим все заказы на Galaxy S8:

|  |  |
| --- | --- |
| 1  2 | DELETE FROM Orders  WHERE ProductId=(SELECT Id FROM Products WHERE ProductName='Galaxy S8'); |