SQL

(подготовлено для курсов Вадима Ксендзова)

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ.

SQL (Structured Query Language)- это язык программирования структурированных запросов.

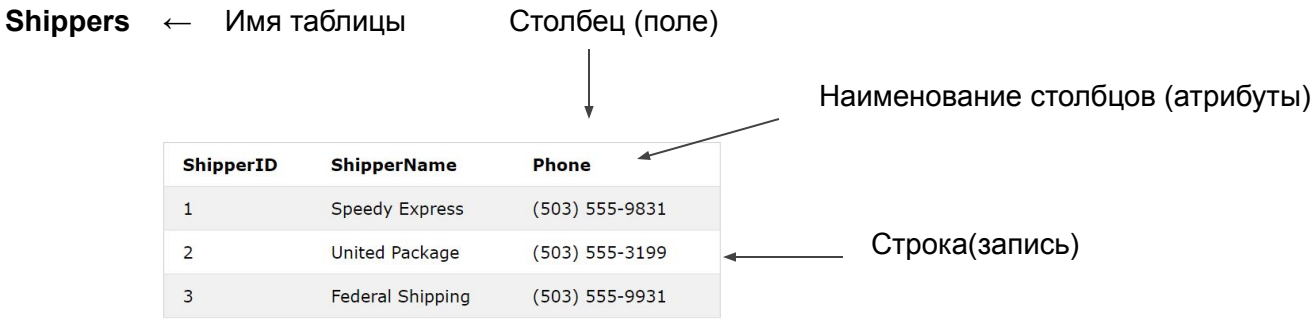
SQL- это стандартный язык для доступа к базам данных и управления ими. SQL стал стандартом ANSI (Американского национального института стандартов) в 1986 году и ISO (Международной организации по стандартизации) в 1987 году. Хотя SQL является стандартом ANSI / ISO, в большинстве программ баз данных SQL имеют свои собственные расширения, в дополнение к SQL стандарту. Однако, чтобы соответствовать стандарту ANSI, все они одинаково поддерживают как минимум основные команды (такие как SELECT, UPDATE, DELETE, INSERT, WHERE).

SQL сегодня активно применяется:

- разработчиками баз данных (обеспечивают функциональность приложений),
- тестировщиками (в ручном и автоматическом режиме),
- администраторами (выполняют поддержание работоспособности среды).

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

База данных - это набор информации, организованный таким образом, который облегчает доступ и эффективное управление и обновление информации. База данных состоит из **таблиц**, хранящих необходимую информацию. Таблицы хранят и отображают информацию в структурированном формате, состоящим из строк или записей (это горизонтальный объект в таблице) и столбцов или полей (вертикальный объект в таблице), похожим на таблицы Excel.



ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

Данные, хранящиеся в базе данных, могут быть любых типов, таких как: строки, числа, даты, изображения и т. д.

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

SQL (Structured Query Language)

СУБД: MySQL, MS SQL Server, ORACLE, MS ACCESS (Microsoft Access), Postgres и др.

Операторы SQL делятся на:

- операторы определения данных **(DDL): CREATE, ALTER, DROP**
- операторы манипуляции данными **(DML): SELECT, INSERT, UPDATE, DELETE**
- операторы определения доступа к данным **(DCL): GRANT, REVOKE, DENY**
- операторы управления транзакциями **(TCL): COMMIT, ROLLBACK, SAVEPOINT**

Система управления базами данных (СУБД) — программное обеспечение, которое взаимодействует с пользователем, приложениями и самой базой данных для сбора и анализа данных. Основой для многих современных систем баз данных является RDBMS (Relational Database Management System) - система управления реляционными базами данных.

Реляционные системы управления базами данных используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Есть много популярных RDBMS, доступных для работы. Самые популярные среди них: MySQL, MS SQL Server, ORACLE, MS ACCESS (Microsoft Access), Postgres. Пример нереляционной системы управления базами данных — Mongo.

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

Взаимосвязи в SQL обеспечиваются с помощью первичного и вторичного ключа.

Первичный ключ (PRIMARY KEY)- это поле таблицы, которое уникально идентифицирует записи таблицы. Это основной ключ и он может быть использован для связи с дочерней таблицей, содержащей внешний ключ.

Первичные ключи *должны содержать УНИКАЛЬНЫЕ* значения и *не могут содержать значения NULL*.

Таблица может иметь *только ОДИН* первичный ключ; а в таблице этот первичный ключ может состоять из одного или нескольких столбцов (полей).

Внешний ключ (Foreign key) также обеспечивает связь между данными в двух таблицах. По сути, это поле или несколько полей, которые ссылаются на PRIMARY KEY в родительской таблице.

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

При работе с SQL полезно знать:

1. Некоторые системы баз данных требуют наличия **точки с запятой** в конце каждой SQL инструкции. Точка с запятой - это стандартный способ разделения каждой SQL инструкции в системах баз данных, которые позволяют выполнять несколько SQL инструкций за один вызов сервера.

2. Одно SQL выражение может быть размещено на одной, или на нескольких строках. Кроме того, несколько SQL выражений могут быть скомбинированы на одной строке. **Пробелы и множества строк игнорируются в SQL**. Комбинация правильного использования пробелов и отступов, разбивания команд на логические строки, делает SQL выражения гораздо проще для чтения и управления.

3.Ключевые слова SQL **НЕ чувствительны к регистру**: select идентично SELECT. Для удобства чтения иногда пишут все ключевые слова SQL в верхнем регистре.

4. При работе с текстовыми столбцами, **заключайте любой текст в выражении в одинарные кавычки (')**. Если ваш текст содержит апостроф (одинарная кавычка), вы должны использовать два символа одиночной кавычки, чтобы избежать апострофа. Например: 'Can''t'. Большинство систем баз данных также допускают двойные кавычки.

5. **В имени таблице лучше не использовать пробелов**. Можно задавать имя таблицы с пробелом, но только в апострофах(в кавычках нельзя). Также зависит от СУБД.

6. В SQL, вы можете указать имя таблицы перед именем столбца, разделяя их точкой. Термин вышеупомянутого синтаксиса называется **"полным именем"** этого столбца.

SELECT City FROM customers;

SELECT customers.City FROM customers;

Эта форма написания особенно полезна при работе со множеством таблиц, которые могут иметь одинаковые названия столбцов.

7. Операторы в разных СУБД могут отличаться. Поэтому их лучше перепроверять.

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

Что может делать SQL?

- SQL может создавать новые базы данных, создавать новые таблицы
- SQL может вставлять, обновлять, удалять записи в базу данных
- SQL может выполнять запросы к базе данных, извлекать данные из базы данных
- SQL может создавать хранимые процедуры в базе данных
- SQL может создавать представления в базе данных
- SQL может устанавливать разрешения для таблиц, процедур и представлений
- указания прав доступа, схем отношений (в том числе, их удаления и изменения),
- взаимодействие с другими языками программирования,
- проверку целостности, задание начала и завершения транзакций.

ЧАСТЬ 1. SQL. ОСНОВНЫЕ ПОНЯТИЯ

Операторы(ключевые слова) SQL делятся на:

- операторы определения данных **(Data Definition Language, DDL).** DDL даёт возможность независимо создавать базу данных, определять её структуру, использовать, а затем сбрасывать по завершению манипуляций.:
    - **CREATE** создаёт объект базы данных (саму базу, таблицу, представление, пользователя, индекс(ключ поиска) и так далее),
    - **ALTER** изменяет объект (БД, таблицу и др.),
    - **DROP** удаляет объект (таблицу, индекс);
- операторы манипуляции данными **(Data Manipulation Language, DML)**- DML позволяет поддерживать уже существующие базы данных, измененять и извлекать данные из баз данных.:
    - **SELECT** извлекает данные, удовлетворяющие заданным условиям,
    - **INSERT** добавляет новые данные,
    - **UPDATE** изменяет существующие данные,
    - **DELETE** удаляет данные;
- операторы определения доступа к данным **(Data Control Language, DCL).** Язык DCL позволяет контролировать данные, когда нужно защитить свою базу данных от повреждения и неправильного использования.:
    - **GRANT** предоставляет пользователю (группе) разрешения на определённые операции с объектом,
    - **REVOKE** отзывает ранее выданные разрешения,
    - **DENY** задаёт запрет, имеющий приоритет над разрешением;
- операторы управления транзакциями **(Transaction Control Language, TCL)**:
    - **COMMIT** применяет транзакцию,
    - **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции,
    - **SAVEPOINT** делит транзакцию на более мелкие участки.

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT

Выражение **SELECT** используется для выбора информации из базы данных. Результат сохраняется в результирующей таблице, которая называется **result-set** или **набором результатов.**

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

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

* - все колонки (столбцы) из таблицы

```
SELECT column1, column2, … FROM table;
SELECT ShipperName, Phone FROM  Shippers;
```

ShipperName	Phone
Speedy Express	(503) 555-9831
United Package	(503) 555-3199
Federal Shipping	(503) 555-9931

```
SELECT ShipperName FROM  Shippers;
SELECT Phone FROM  Shippers;
```

ShipperName
Speedy Express
United Package
Federal Shipping

Phone
(503) 555-9831
(503) 555-3199
(503) 555-9931

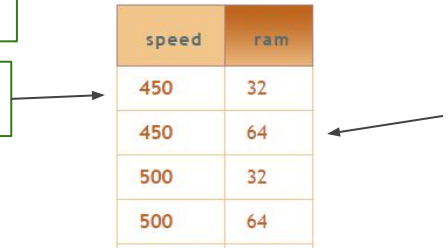
DISTINCT – команда для вывода только уникальных значений (без повторов).

```
SELECT DISTINCT столбец1, столбец2, …
FROM имя_таблицы;
```

```
SELECT DISTINCT City FROM customers;
```

```
SELECT DISTINCT speed, ram FROM PC;
```

speed	ram
450	32
450	64
500	32
500	64



speed	ram
450	32
450	64
500	32
500	32
500	64
500	64

ORDER BY используется для сортировки набора результатов в порядке возрастания или убывания. По умолчанию сортировка происходит по возрастанию числовых значений и в алфавитном порядке строковых. Если необходимо получить **обратный порядок**, используйте **DESC.** Для выбора **прямого порядка** используется **ASC .**

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

LIMIT / TOP / FETCH FIRST *n* ROWS ONLY используют для уменьшения количества результатов.

MySQL

```
SELECT column_name(s) FROM table_name
WHERE condition LIMIT number;
```

```
SELECT * FROM Customers LIMIT 5;
```

```
SELECT ContactName, Address FROM
Customers LIMIT 5;
```

```
SELECT * FROM Customers
WHERE Country='Germany'  LIMIT 3;
```

```
SELECT ID, FirstName, LastName, City FROM
customers OFFSET 3 LIMIT 4;
```

```
SELECT id, name FROM customers LIMIT 4,12
```

```
SELECT DISTINCT City FROM Customers
ORDER BY City DESC Limit 5;
```

SQL Server / MS Access

```
SELECT TOP number|percent column_name(s)
FROM table_name  WHERE condition;
```

```
SELECT TOP 3 * FROM Customers;
```

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

```
SELECT TOP 50 PERCENT * FROM Customers;
```

Oracle 12

```
SELECT column_name(s)  FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

WHERE

Условие WHERE используется для выборки только тех записей, которые соответствуют указанному условию. В качестве условия (предиката) можно использовать следующие типы выражений: сравнение, диапазон, перечень, текстовой шаблон и поиск пустых значений.

Оператор	Описание
=	равно
<> / !=	не равно
> / >=	больше / больше или равно
< / <=	меньше / меньше или равно)

```
SELECT * FROM Customers WHERE CustomerID=1;
```

```
SELECT * FROM Customers WHERE PostalCode = 44000 Limit 5;
```

```
SELECT * FROM Customers WHERE NOT City='Berlin';
```

```
SELECT * FROM Products WHERE Price <> 18;
```

```
SELECT * FROM Products WHERE Price > 30;
```

Диапазон (Range). BETWEEN

Диапазон (Range). BETWEEN . Оператор BETWEEN выбирает значения внутри диапазона (числа, текст или даты).

Начальные и конечные значения включаются в диапазон.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT * FROM customers WHERE ID BETWEEN 3
AND 7;
```

```
SELECT*FROM Customers WHERE PostalCode
BETWEEN  12200 AND 12210 LIMIT 5;
```

```
SELECT * FROM Products WHERE Price BETWEEN
10 AND 20;
```

```
SELECT * FROM Products WHERE ProductName
BETWEEN 'C' AND 'M';
```

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon
Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

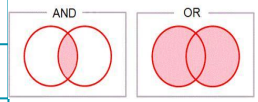
```
SELECT * FROM Orders WHERE OrderDate
BETWEEN '1996-07-01' AND '1996-07-31';
```

```
SELECT * FROM Products WHERE Price NOT
BETWEEN 10 AND 20;
```

Логические операторы: AND, OR, IN, NOT

Логические операторы могут использоваться для комбинации двух булевых значений и возврата результата верно(true), ложь(false), или null.

Оператор	Описание
And	ВЕРНО, если **оба** выражения ВЕРНЫ
Or	ВЕРНО, если **одно из** выражений ВЕРНО
in ()	ВЕРНО, если операнд равен одному из выражений в списке (То есть можно указать несколько значений)
Not	Возвращает ВЕРНО, если выражение НЕ ВЕРНО (Like, In)



SELECT *столбец1*, *столбец2, …* FROM *имя_таблицы* WHERE *условие1* AND *условие2* AND *условие3 ...*;

SELECT * FROM Customers WHERE Country = 'Germany' **AND** City = 'Berlin';

SELECT ID, FirstName, LastName, Age FROM customers WHERE Age >= 30 AND Age <= 40;

SELECT *столбец1*, *столбец2, …* FROM *имя_таблицы* WHERE *условие1* OR *условие2* OR *условие3 ...*;

SELECT * FROM Customers WHERE City = 'Berlin' **OR** City = 'London';

SELECT *столбец1*, *столбец2, …* FROM *имя_таблицы* WHERE NOT *условие*;

SELECT * FROM Customers WHERE NOT Country='Germany';

Логические операторы: AND, OR, IN, NOT

При комбинации AND, OR, NOT желательно использовать скобки, для упорядочивания вычислений всех известных условий.

NOT имеет высший приоритет, затем AND.

AND, OR

```
SELECT * FROM Customers WHERE Country = ‘Mexico' AND (City = ‘Mexico' OR City = ‘Hanalulu');
```

AND, OR

```
SELECT * FROM customers WHERE City = 'New York' AND (Age=30 OR Age=35);
```

NOT, AND, NOT

```
SELECT * FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';
```

Диапазон данных. Оператор IN

Оператор IN используется тогда, когда вы хотите сравнить столбец больше чем с одним значением, то есть указать больше одного значения. IN Оператор является обобщающим для нескольких OR условий.

```
SELECT column_name(s) FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s) FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

```
SELECT * FROM Customers WHERE PostalCode
IN (12209, 44000, 78000)  limit 5;
```

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

```
SELECT * FROM customers
WHERE City = 'New York' OR City = 'Los Angeles' OR City = 'Chicago';
```

→

```
SELECT * FROM customers
WHERE City IN ('New York', 'Los Angeles', 'Chicago');
```

NOT IN

```
SELECT * FROM customers WHERE City NOT IN ('New York', 'Los
Angeles', 'Chicago');
```

IN, AND, IN

```
SELECT model, speed, hd FROM PC WHERE hd IN (10, 20) AND
model IN (SELECT model FROM product WHERE maker = 'A');
```

→

model	speed	hd
1233	750	20
1232	500	10
1232	450	10
1233	800	20

BETWEEN, AND, AND, NOT IN

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

LIKE, подстановочные операторы

Ключевое слово **LIKE** используется с оператором WHERE для поиска по шаблону. Для создания шаблонов используются **подстановочные операторы.** Подстановочные знаки используются в сочетании с оператором сравнения LIKE или оператором сравнения **NOT LIKE**.

Подстановочные знаки (Wildcards). Основные команды.

Что заменяет подстановочный знак	MySQL	MS Access	SQL Server	Примеры
любое количество символов (включая 0 символов)	%	*	%	bl*, bl%
1 символ	_	?	_	h?t
любой отдельный символ в скобках	[abc]	[abc]	[abc]	h[oa]t
диапазон символов	[a-c]	[a-c]	[a-c]	c[a-b]t
любой символ, не заключенный в скобки		!	^	h[!oa]t, h[^oa]t
любой отдельный числовой символ		#		2#5

```
SELECT column1, column2, …
FROM table_name
WHERE columnN LIKE pattern;
```

```
SELECT * FROM Customers WHERE City LIKE 'ber%';
SELECT * FROM Customers WHERE City LIKE '_erlin';
SELECT * FROM Customers WHERE City LIKE '[bsp]%';
```

LIKE, подстановочные операторы

Примеры.

```
SELECT * FROM Customers WHERE City LIKE '[a-c]%';
```

```
SELECT * FROM Customers WHERE City LIKE '[!bsp]%';
```

```
SELECT * FROM Customers WHERE City NOT LIKE '[bsp]%
```

```
SELECT*FROM Customers WHERE CustomerName LIKE '%or%'
```

```
SELECT*FROM Customers WHERE ContactName LIKE 'a%o'
```

```
SELECT * FROM Customers WHERE CustomerName NOT LIKE 'a%'
```

```
SELECT * FROM Customers WHERE CustomerName LIKE 'a__%';
```

```
SELECT*FROM Customers WHERE CustomerName LIKE '_r%'
```

```
SELECT * FROM Customers WHERE PostalCode LIKE
‘122%’ OR  PostalCode LIKE ‘82%’ Limit 5;
```

```
SELECT * FROM employees WHERE FirstName LIKE
'A% ORDER BY LastName';
```

```
SELECT*FROM Customers WHERE (PostalCode LIKE
'82%') AND PostalCode NOT LIKE '%23';
```

Недостающие (пустые) значения. Null

IS NULL Оператор используется для проверки пустых значений (значений NULL). Значение NULL вовсе не равно нулю или пробелу. Значение NULL представляет значение, которое недоступно, неизвестно, присвоено или неприменимо, тогда как ноль — это число, а пробел — символ. Для **Null** всегда используется **IS** вместо **=** или **Like.**

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

```
SELECT*FROM Customers
WHERE PostalCode IS NULL;
```

```
SELECT CustomerName, ContactName,
Address
FROM Customers
WHERE Address IS NOT NULL;
```

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT Агрегатные функции (*aggregate functions*): MIN(), MAX(), AVG(), SUM(), COUNT().

Агрегатные функции (*aggregate functions*) выполняют вычисление на наборе значений в столбце и возвращают единственное значение.

MIN() - Наименьшее значение (*Returns the smallest value)*

MAX() - Наибольшее значение (*Returns the largest value)*

AVG() - Среднее значение (*Returns the average value*)

SUM() – Сумму *(Returns the sum*)

COUNT() - Количество строк (*Returns the number of rows)*

При этом функции **COUNT**, **MIN** и **MAX** применимы к данным любого типа, в то время как **SUM** и **AVG** используются только для данных числового типа. Агрегатные функции при подсчете не учитывает **NULL**-значения (исключение **COUNT**(*)).

```
SELECT (поля+агрегатная функция)
FROM таблица
GROUP BY поле;
```

```
SELECT COUNT(model) AS Qty_model
FROM Product
WHERE maker = 'A';
```

Qty_model
7

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT Агрегатные функции (*aggregate functions*):

MIN(), MAX(), AVG(), SUM(), COUNT().

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

```
SELECT MIN(Salary) AS Salary
FROM employees;
```

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

```
SELECT AVG(Price)
FROM Products;
```

```
SELECT AVG(Salary) FROM employees;
```

Number of Records: 1

SmallestPrice
2.5

Number of Records: 1

LargestPrice
263.5

Number of Records: 1

AVG(Price)
28.866363636363637

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT Агрегатные функции (*aggregate functions):* MIN(), MAX(), AVG(), SUM(), COUNT().

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(Salary) FROM employees;
```

```
SELECT SUM(Quantity) FROM OrderDetails;
```

Number of Records: 1

SUM(Quantity)
12743

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

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

```
SELECT COUNT(ProductID) FROM Products;
```

COUNT(ProductID)
77

```
SELECT COUNT(DISTINCT Country) FROM
Customers;
```

Приведенный выше пример не будет работать в Firefox! Потому что COUNT(DISTINCT *column_name*) не поддерживается в базах данных Microsoft Access. Вот обходной путь для MS Access:

```
SELECT Count(*) AS DistinctCountries
FROM (SELECT DISTINCT Country FROM
Customers);
```

Оператор **GROUP BY** используется вместе с агрегатными функциями для группировки результата по признакам одного или более столбцов.

Предложение **GROUP BY** используется для определения групп строк, к которым могут применяться агрегатные функции (**COUNT**, **MIN**, **MAX, AVG** и **SUM**).

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Number of Records: 21

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France

```
SELECT DISTINCT City FROM Customers ORDER BY City;
```

=

```
SELECT City FROM Customers GROUP BY City;
```

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
```

```
SELECT COUNT(Country), Country FROM Customers GROUP BY Country;
```

COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil

```
SELECT model, COUNT(model) AS Qty_model, AVG(price) AS Avg_price FROM PC GROUP BY model;
```

model	Qty_model	Avg_price
1260	1	350
1232	4	425
1121	3	850
1233	3	843,333333333333

```
SELECT MIN(price) AS min_price,MAX(price) AS max_price, AVG(price) avg_price FROM PC;
```

min_price	max_price	avg_price
350.00	980.00	675.00

Ключевое слово HAVING используется в качестве условия в SQL-запросе для ограничения записей, обрабатываемых агрегатными функциями. **HAVING** был добавлен в SQL потому, что оператор WHERE не может быть использован вместе с агрегатной функцией. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении **FROM**, а из групп таких строк.

```
SELECT column(s),  statement(column) AS alias
FROM table  WHERE condition GROUP BY column(s)
HAVING condition ORDER BY column(s);
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

```
SELECT Country, COUNT (Country) FROM
Customers GROUP BY Country HAVING
COUNT (Country) > 10;
```

Number of Records: 5

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK

```
+---------+----------------+
| country | count(country) |
+---------+----------------+
| France  |             11 |
| Germany |             11 |
| USA     |             13 |
+---------+----------------+
```

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT Скалярные функции (*scalar functions*): LENGTH(), ROUND(), NOW(), FORMAT().

Скалярные функции (*scalar functions*) SQL возвращают одно значение, основанное на входном значении.

LENGTH() - Возвращает длину текстового поля в символах, включая пробелы

ROUND() - Возвращает числовое значение, округленное до указанной длины или точности.

NOW() - Возвращает текущее время и дату

FORMAT() - Форматирование для определенного отображения поля.

```
SELECT DISTINCT City,
CHAR_LENGTH(City) AS 'Word_Lenght'
FROM Customers ORDER BY City Limit 5;
```

City	Word_Lenght
Aachen	6
Albuquerque	11
Anchorage	9
Barcelona	9
Barquisimeto	12

```
SELECT CustomerName,
LENGTH(CustomerName) AS
LengthOfName FROM Customers;
```

CustomerName	LengthOfName
Alfreds Futterkiste	19
Ana Trujillo Emparedados y helados	34
Antonio Moreno Taquería	24
Around the Horn	15

Арифметические операторы выполняют арифметические операции с числовыми операндами. SQL позволяет делать математические и другие операции прямо в процессе выборки данных. Арифметические операторы включают в себя сложение (+), вычитание (-), умножение (*) и деление (/), модуль (%).

```
SELECT ID, FirstName, LastName,
Salary+500 AS Salary
FROM employees;
```

```
SELECT 30+20;
```

Number of Records: 1

30 + 20
50

```
SELECT ProductName, CONCAT (Price, 'US') AS 'USD_Price', CONCAT ((Price*1.97), 'BYN') AS 'BYN_Price' FROM Products ORDER BY ProductName LIMIT 5;
```

ProductName	USD_Price	BYN_Price
Alice Mutton	39.00 US	76.8300 BYN
Aniseed Syrup	10.00 US	19.7000 BYN
Boston Crab Meat	18.40 US	36.2480 BYN
Camembert Pierrot	34.00 US	66.9800 BYN
Carnarvon Tigers	62.50 US	123.1250 BYN

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT Функции SQRT, UPPER, LOWER

Функция SQRT возвращает квадратный корень заданного значения в аргументе.

Функция UPPER конвертирует все буквы в указанной строке в верхний регистр.

Функция LOWER конвертирует строку в нижний регистр.

```
SELECT Salary, SQRT(Salary) FROM
employees;
```

```
SELECT FirstName, UPPER(LastName) AS
LastName
FROM Employees  LIMIT 3;
```

Number of Records: 3

FirstName	LastName
Nancy	DAVOLIO
Andrew	FULLER
Janet	LEVERLING

```
SELECT CONCAT(UPPER(LEFT(ContactName,2)),
SUBSTRING_INDEX(ContactName,
LEFT(ContactName, 1), -1)) AS RESULT FROM
Customers;
```

Number of Records: 91

RESULT
MAaria Anders
ANna Trujillo
ANntonio Moreno

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT **Функции** CONCAT, SUBSTRING_INDEX

Функция **CONCAT** используется для конкатенации двух и более текстовых значений, и для возврата полученной строки. Функция CONCAT() принимает два параметра или более. Конкатенация образует новый столбец. Названием столбца по умолчанию будет функция CONCAT.

SUBSTRINGS_INDEX (column_name, ‘ разделитель’, номер разделителя)

- возвращает все символы до разделителя. Например, можно выбрать пробел в качестве разделителя. Если номер разделителя положительный, то берётся текст слева от него, если отрицательный, то берётся текст справа от него.

```
SELECT CONCAT(CustomerName,', ', City)
AS new_column FROM Customers;
```

new_column
Alfreds Futterkiste, Berlin
Ana Trujillo Emparedados y helados, México D.F.
Antonio Moreno Taquería, México D.F.

```
SELECT SUBSTRING_INDEX(Description, '
', 1) FROM Categories;
```

SUBSTRING_INDEX(Description, ' ', 1)
Soft
Sweet
Desserts,
Cheeses
Breads,
Prepared
Dried
Seaweed

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses

Соединение двух таблиц.

Обратите внимание, что выражение WHERE "объединяет" таблицы, при условии, что ID таблицы customers **должно быть равно** customer_ID таблицы orders.

```
SELECT Customers.CustomerID,
Customers.CustomerName,Orders.OrderDate
FROM Customers, Orders
WHERE
Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerID;
```

CustomerID	CustomerName	OrderDate
1	Alfreds Futterkiste	1997-10-03
1	Alfreds Futterkiste	1998-04-09
1	Alfreds Futterkiste	1998-03-16
1	Alfreds Futterkiste	1997-10-13

Подзапрос. Вложенные запросы.

Подзапрос это запрос внутри другого запроса. В подзапросе внешний запрос называется основным запросом, тогда как внутренний запрос называется подзапросом. Подзапросы всегда выполняются первыми, а результат подзапроса передается в основной запрос. Он может быть вложен в SELECT, UPDATE или любой другой запрос.

Подзапрос можно делать и к той же таблице, что и основной запрос или к другой таблице.

Варианты использования подзапросов:

1.SELECT, FROM, WHERE, =/>/<(Подзапрос вместо условия для сравнения) Сравнение с результатом агрегатной функции.

```
SELECT FirstName, Salary FROM employees
WHERE  Salary > (SELECT AVG(Salary) FROM
employees)
ORDER BY Salary DESC;
```

=

```
SELECT AVG(Salary) FROM employees;
```

AVG(Salary) =3100

+

```
SELECT FirstName, Salary FROM employees
WHERE  Salary > 3100
ORDER BY Salary DESC;
```

```
SELECT*FROM Products WHERE Price=(SELECT MIN(Price)
FROM Products);
```

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
33	Geitost	15	4	500 g	2.5

```
SELECT ProductName, MIN(Price) FROM Products GROUP BY
ProductName ORDER BY MIN(Price) LIMIT 1;
```

ProductName	MIN(Price)
Geitost	2.50

```
SELECT ProductName, Price AS ‘Proucts_With_Price_Above_Average’
FROM Products WHERE Price>(SELECT AVG(Price) FROM Products)
ORDER BY ProductName Limit 5;
```

ProductName	Products_With_Price_Above_Average
Alice Mutton	39.00
Camembert Pierrot	34.00
Carnarvon Tigers	62.50
Cute de Blaye	263.50
Gnocchi di nonna Alice	38.00

2. SELECT, FROM, WHERE, IN (Подзапрос вместо условия для выбора из перечня)

Например, когда надо связать несколько таблиц по взаимосвязанным столбцам, чтобы выбрать условие из той таблицы, в которой оно есть.

Выбрать все категории товаров, который были отправлены в Torino, отсортировать в алфавитном порядке:

```
SELECT Categories.CategoryName  FROM Categories
WHERE Categories.CategoryID IN (SELECT Products.CategoryID FROM Products
WHERE Products.ProductID IN (SELECT OrderDetails.ProductID FROM OrderDetails
WHERE OrderDetails.OrderID IN (SELECT Orders.OrderID FROM Orders
WHERE Orders.CustomerID IN (SELECT Customers.CustomerID FROM Customers
WHERE Customers.City='Torino'))))
ORDER BY Categories.CategoryName;
```

Если бы все необходимые данные были в одной таблице, то запрос бы выглядел так:

```
SELECT Categories.CategoryName
 FROM Categories
WHERE Categories.City='Torino'
ORDER BY Categories.CategoryName;
```

```
SELECT COUNT(*) AS Qty FROM PC
WHERE model IN
(SELECT model FROM Product
WHERE maker = 'A');
```

Qty
8

```
SELECT COUNT(DISTINCT model) AS Qty FROM PC
WHERE model IN
(SELECT model FROM Product
WHERE maker = 'A');
```

3.SELECT (Подзапрос вместо столбца) FROM

Например, для выведения столбца со значениями, полученными в результате использования подзапроса.

```
SELECT model, price, (SELECT MIN(price) FROM Printer) min_price, (SELECT MAX(price) FROM Printer) max_price
FROM printer;
```

4. SELECT, FROM (Подзапрос вместо имени таблицы для основного запроса)

```
? SELECT prod.maker, lap.*FROM
(SELECT 'laptop' AS type, model, speed FROM laptop WHERE speed > 600) AS lap
INNER JOIN
(SELECT maker, model FROM product) AS prod
ON lap.model = prod.model;
```

maker	type	model	speed
B	laptop	1750	750
A	laptop	1752	750

```
SELECT product.maker, 'laptop' AS type, product.model, laptop.speed FROM
product
INNER JOIN laptop ON laptop.model = product.model AND speed > 600;
```

ПОЛЬЗОВАТЕЛЬСКИЕ ИМЕНА. ПРОЗВИЩА. Псевдонимы (Aliases)

Псевдонимы SQL используются для присвоения таблице или столбцу в таблице временного имени. Псевдоним существует только на время этого запроса.Псевдоним создается с AS ключевым словом или без ключевого слова AS через пробел после названия таблицы. Псевдонимы могут быть полезны, когда:

1)Для сокращения выражения, когда в запросе участвует более одной таблицы.
2)Имена столбцов большие или плохо читаемые для сокращения имени или хотим вывести колонку с другим именем.
3)Для названия объединенного столбца
4)В запросе используются функции
5) Для вывода колонки с дополнительной информацией
6)для решения проблемы конфликта имён.

1)Для сокращения выражения. Переименование таблиц.

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn'
AND c.CustomerID=o.CustomerID;
```

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn'
AND Customers.CustomerID=Orders.CustomerID;
```

OrderID	OrderDate	CustomerName
10355	1996-11-15	Around the Horn
10383	1996-12-16	Around the Horn

2)Имена столбцов большие или плохо читаемые для сокращения имени. Или просто хотим вывести колонку с другим именем.

```
SELECT CustomerID AS ID, CustomerName AS
Customer
FROM Customers;
```

ID	Customer
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería

```
SELECT CustomerName AS Customer,
ContactName AS [Contact Person]
FROM Customers;
```

Number of Records: 91

Customer	Contact Person
Alfreds Futterkiste	Maria Anders

3)Псевдонимы для названия объединенного столбца.

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' +
City + ', ' + Country AS Address
FROM Customers;
```

Number of Records: 91

CustomerName	Address
Alfreds Futterkiste	Obere Str. 57, 12209 Berlin, Germany
Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222, 05021 México D.F., Mexico

```
SELECT CustomerName, CONCAT(Address,', '
,PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

В MySQL

4)псевдонимы для столбцов с результатами вычислений или при использовании в предложении SELECT выражений для вычисления значения.

```
SELECT Country, COUNT (Country) AS 'Customer Number'
FROM Customers GROUP BY Country;
```

Country	Customer Number
Argentina	3
Austria	2
Belgium	2
Brazil	9

```
mysql> SELECT country, count(country) AS THE_NAME_I_WANTED FROM customers
    -> GROUP BY country
    -> HAVING THE_NAME_I_WANTED > 10;
+---------+--------------------+
| country | THE_NAME_I_WANTED  |
+---------+--------------------+
| France  |                 11 |
| Germany |                 11 |
| USA     |                 13 |
+---------+--------------------+
```

```
SELECT ram*1024 AS Kb, hd Gb
FROM PC
WHERE cd = '24x';
```

Kb	Gb
65536	8
32768	10

5)Псевдонимы для вывода колонки с дополнительной информацией рядом со значениями

```
SELECT ram, 'Mb' AS ram_units, hd, 'Gb' AS hd_units
FROM PC
WHERE cd = '24x';
```

ram	ram_units	hd	hd_units
64	Mb	8	Gb
32	Mb	10	Gb

6)Иногда в предложении *FROM* требуется указать одну и ту же таблицу несколько раз. Например, если надо сравнивать между собой значения внутри одной колонки и для этого создаём две таблицы из одной. В этом случае обязательным является переименование. Вывести пары моделей, имеющих одинаковые цены:

```
SELECT DISTINCT A.model AS model_1, B.model AS model_2
FROM PC AS A, PC B
WHERE A.price = B.price AND A.model < B.model;
```

model_1	model_2
1232	1233
1232	1260

7) В предложении FROM используется подзапрос, так как, в противном случае, у нас нет возможности уточнения имени столбца из подзапроса.

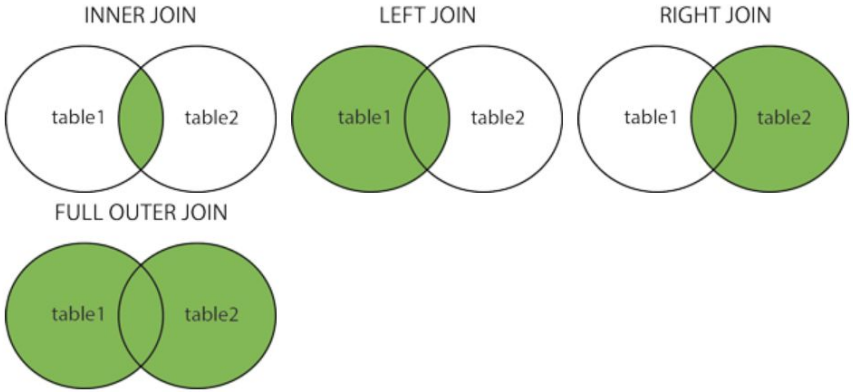
```
? SELECT DISTINCT PC.model, maker FROM PC, (SELECT maker, model
FROM Product ) AS Prod WHERE PC.model = Prod.model AND price < 600;
```

```
SELECT DISTINCT PC.model, maker
FROM PC, Product WHERE PC.model = Product.model
AND price < 600
```

SQL JOIN (соединение, объединение таблиц, логическое связывание таблиц). JOIN используется для объединения строк из двух или более таблиц на основе связанного столбца между ними. Выбирает значения из нескольких таблиц одновременно. При необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно).

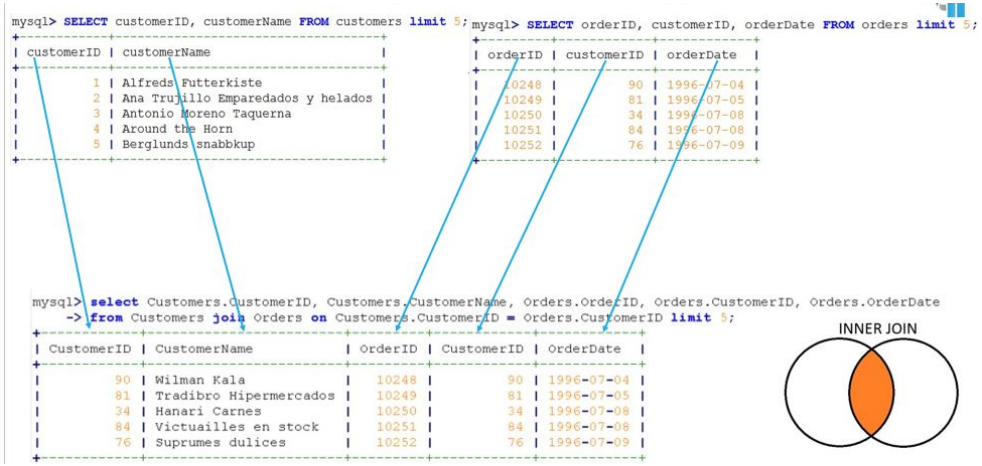
Наиболее часто используемых 4:

- (INNER) JOIN: Возвращает записи, значения которых совпадают в обеих таблицах (внутреннее соединение/объединение).
- LEFT (OUTER) JOIN: Возвращает все записи из левой таблицы и соответствующие записи из правой таблицы.
- RIGHT (OUTER) JOIN: Возвращает все записи из правой таблицы и соответствующие записи из левой таблицы.
- FULL (OUTER) JOIN: Возвращает все записи, если есть совпадения в левой или правой таблице (полное внешнее объединение)



Внутреннее объединение **INNER JOIN** (ключевое слово INNER можно опустить). Выбираются **только** совпадающие данные из двух объединяемых таблиц.

JOIN - это горизонтальное объединение



INNER JOIN Синтаксис

SELECT *column_name(s)*

FROM *table1*

INNER JOIN *table2*

ON *table1.column_name* =

table2.column_name;

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) SELECT JOIN

```
SELECT Orders.OrderID, Customers.CustomerName FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

OrderID	CustomerName
10248	Wilman Kala
10249	Tradição Hipermercados
10250	Hanari Carnes
10251	Victuailles en stock

```
SELECT Products.ProductsName, Suppliers.SupplierName FROM Products
JOIN Suppliers ON Products.SupplierID=Suppliers.SupplierID WHERE
Products.ProductsName  LIKE 'Z%' ORDER BY  Products.ProductsName;
```

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

Number of Records: 1

LastName	NumberOfOrders
Davolio	29

LEFT OUTER JOIN (**LEFT JOIN**) возвращает все записи из левой таблицы (табл.1), а также совпадающие записи из таблицы справа (table2).

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT *
FROM  Person -- Левая таблица
LEFT OUTER JOIN  City   -- Правая таблица
ON Person.CityId = City.Id
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

```
mysql> SELECT customers.customerID, customers.customerName, orders.orderDate FROM customers
    -> LEFT JOIN orders ON customers.customerID = orders.customerID
    -> ORDER BY customers.customerID
    -> LIMIT 5;
+------------+------------------------------------+------------+
| customerID | customerName                       | orderDate  |
+------------+------------------------------------+------------+
|          1 | Alfreds Futterkiste                | NULL       |
|          2 | Ana Trujillo Emparedados y helados | 1996-09-18 |
|          3 | Antonio Moreno Taquerna            | 1996-11-27 |
|          4 | Around the Horn                    | 1996-12-16 |
|          4 | Around the Horn                    | 1996-11-15 |
+------------+------------------------------------+------------+
```

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Number of Records: 3

ShipperName	NumberOfOrders
Federal Shipping	68
Speedy Express	54
United Package	74

RIGHT OUTER JOIN. (RIGHT JOIN) возвращает все записи из таблицы справа (table2) и совпадающие записи из левой таблицы (table1).

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT *
FROM   Person -- Левая таблица
RIGHT OUTER JOIN   City   -- Правая таблица
ON Person.CityId = City.Id
```

```
SELECT Orders.OrderID, Employees.LastName,
Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret

В **FULL OUTER JOIN** возвращаются все записи , когда Есть совпадения в левой (table1) или правой таблице (table2) записях.

Слово OUTER здесь можно пропускать.

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Оператор **UNION** используется для объединения результата 2-х и более SELECT запросов. UNION это вертикальное объединение. **Union** обьединяет только уникальные (distinct) данные.

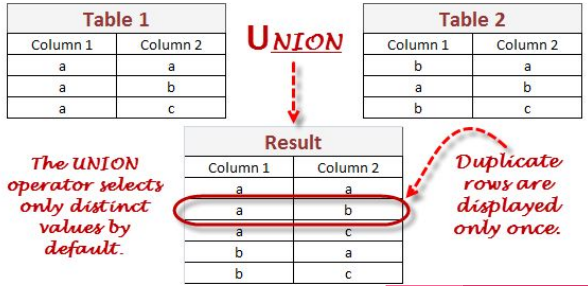
Тип данных в колонках, по которым делается select для Union должен совпадать (текст, число...). Каждый из этих запросов:

1)должен иметь одинаковое количество столбцов

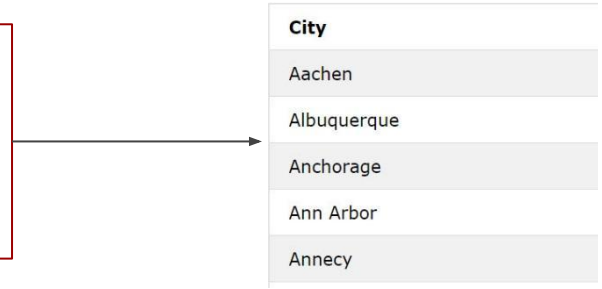
2)с одинаковыми типами данных и

3)расположенных в том же порядке.

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;;
```



```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```



City
Aachen
Albuquerque
Anchorage
Ann Arbor
Annecy

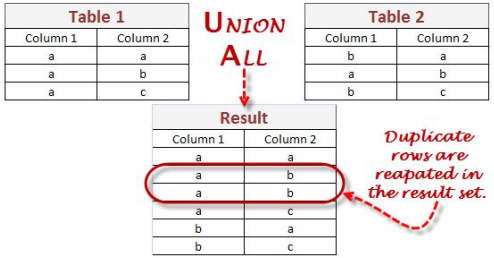
```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

```
SELECT 'Customer' AS Type, ContactName, City, Country FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country FROM Suppliers;
```

```
SELECT FirstName, LastName, Company FROM businessContacts
UNION
SELECT FirstName, LastName, NULL FROM otherContacts;
```

UNION ALL выбирает все строки из каждой таблицы и комбинирует их в одну таблицу.

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```




```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

```
SELECT ID, FirstName, LastName, City FROM First
UNION ALL
SELECT ID, FirstName, LastName, City FROM Second;
```

CROSS JOIN. Оператор *перекрёстного соединения*, или *декартова произведения* CROSS JOIN соединяет две таблицы. Каждая строка одной таблицы соединяется с каждой строкой второй таблицы, давая тем самым в результате все возможные сочетания строк двух таблиц.

```
SELECT model, price, min_price, max_price
FROM printer CROSS JOIN
(SELECT MIN(price) min_price, MAX(price)
max_price
FROM printer) X;
```

model	price	min_price	max_price
1276	400.0000	150.0000	400.0000
1288	400.0000	150.0000	400.0000
1401	150.0000	150.0000	400.0000
1408	270.0000	150.0000	400.0000
1433	270.0000	150.0000	400.0000
1434	290.0000	150.0000	400.0000

Self JOIN. Это выражение используется для того, чтобы таблица объединилась сама с собой, словно это две разные таблицы. Чтобы такое реализовать, одна из таких «таблиц» временно переименовывается.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID AND A.City = B.City
ORDER BY A.City;
```

INSERT INTO Добавление новых записей в таблицу:

```
INSERT INTO table_name (column1, column2, column3, ...,columnN)
VALUES (value1, value2, value3,...valueN);
column1, column2, ..., columnN
```

```
INSERT INTO table_name
VALUES (value1, value2, value3,...);
```

```
INSERT INTO clients (id, first_name, last_name
VALUES (2, 'Vasil', 'Pupkin');
```

```
INSERT INTO Employees
VALUES (8, 'Anthony', 'Young', 35);
```

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

ЧАСТЬ 2. SQL. ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML) UPDATE

UPDATE - изменение данных в таблице.

Вы указываете столбец и его новое значение в разделенном запятыми списке после ключевого слова SET. Можно определять порядок столбцов любым образом в выражении SET.

```
UPDATE table_name
SET column1=value1, column2=value2, ...
WHERE condition;
```

```
UPDATE Shippers
SET ShipperName='AAA BBB', Phone='(503) 111-1111'
WHERE ShipperID=1;
```

ShipperID	ShipperName	Phone
1	AAA BBB	(503) 111-1111
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

```
UPDATE Shippers
SET ShipperName='AAA BBB'
WHERE Phone LIKE '%1';
```

ShipperID	ShipperName	Phone
1	AAA BBB	(503) 555-9831
2	United Package	(503) 555-3199
3	AAA BBB	(503) 555-9931

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

UPDATE

```
UPDATE Shippers
SET ShipperName='AAA BBB';
```

ShipperID	ShipperName	Phone
1	AAA BBB	(503) 555-9831
2	AAA BBB	(503) 555-3199
3	AAA BBB	(503) 555-9931

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

```
UPDATE Таблица
SET Изменяемое поле
CASE
WHEN поле=значение THEN значение
…
ELSE значение
END;
```

```
mysql> UPDATE users
    -> SET rank = CASE
    -> WHEN reputation > 350 THEN 'Элита'
    -> WHEN reputation > 100 THEN 'Популярный'
    -> WHEN reputation > 0 THEN 'Пользователь'
    -> WHEN reputation = 0 THEN 'Новичок'
    -> ELSE 'Нуб'
    -> END;
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM users;
+----+-------------+-------------+------------+-------------------------+
| id | fnames_list | lnames_list | reputation | rank                    |
+----+-------------+-------------+------------+-------------------------+
|  1 | Peter       | Yan         |        500 | Элита                   |
|  2 | Lola        | Vin         |        324 | Популярный              |
|  3 | Jimmy       | Voode       |         21 | Пользователь            |
|  4 | Грег        | Эйтсон      |          0 | Новичок                 |
|  6 | Майк        | Гайден      |          2 | Пользователь            |
+----+-------------+-------------+------------+-------------------------+
5 rows in set (0.00 sec)
```

DELETE

Выражение DELETE используется для удаления данных из вашей таблицы.

Примечание: будьте осторожны при удалении записей в таблице! Обратите внимание на WHERE пункт в DELETE заявлении. В WHERE предложении указывается, какие записи следует удалить. Если вы опустите WHERE предложение, все записи в таблице будут удалены!

```
DELETE FROM table_name WHERE condition;
```

```
DELETE FROM table_name;
```

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

```
DELETE FROM Customers;
```