# Часть 1. Базы данных и SQL

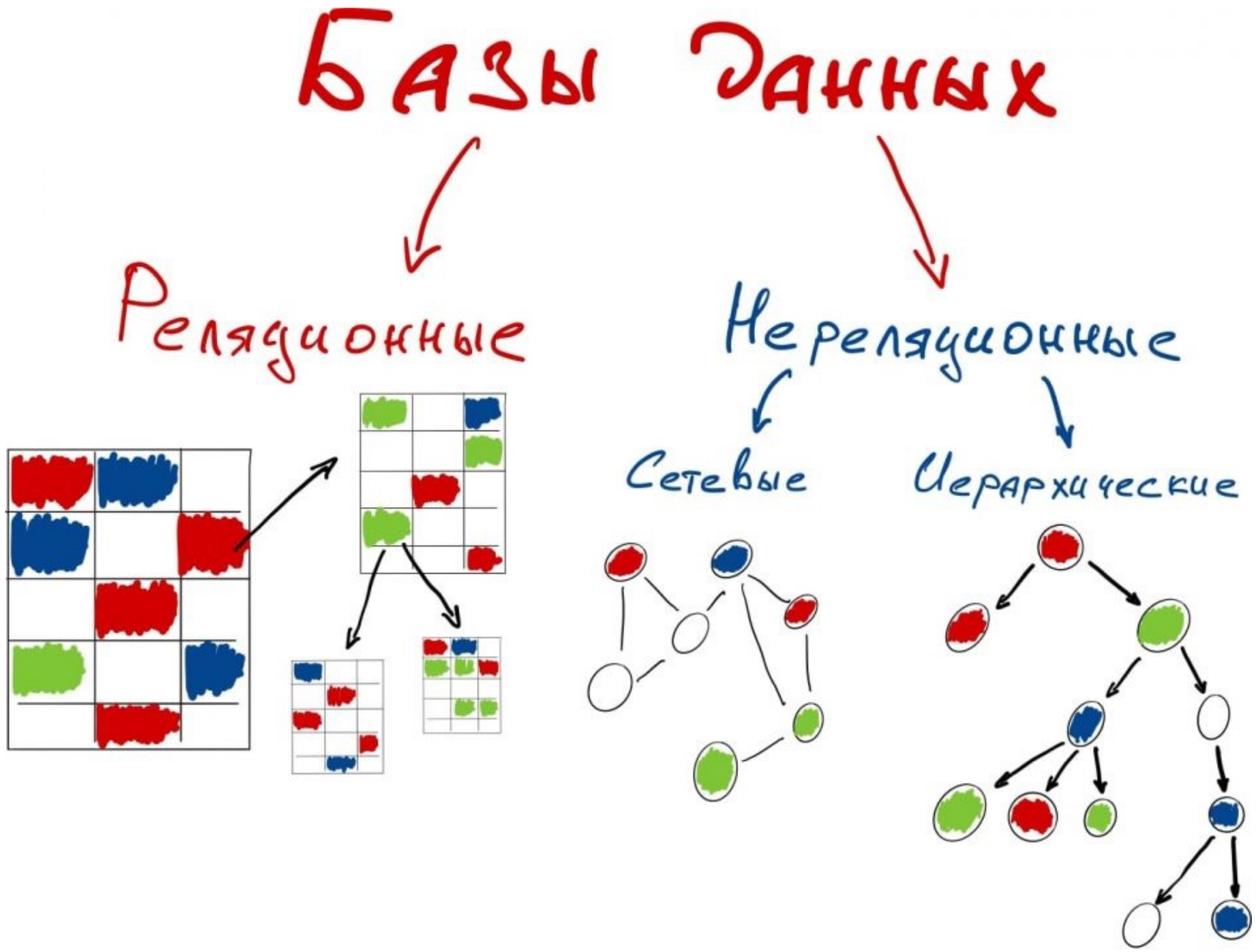
08.12.2020

# План

- Какие существуют типы БД
- типы данных в SQL
- Структура запроса в SQL
- Основные операторы манипулирования данными
- Работа с БД

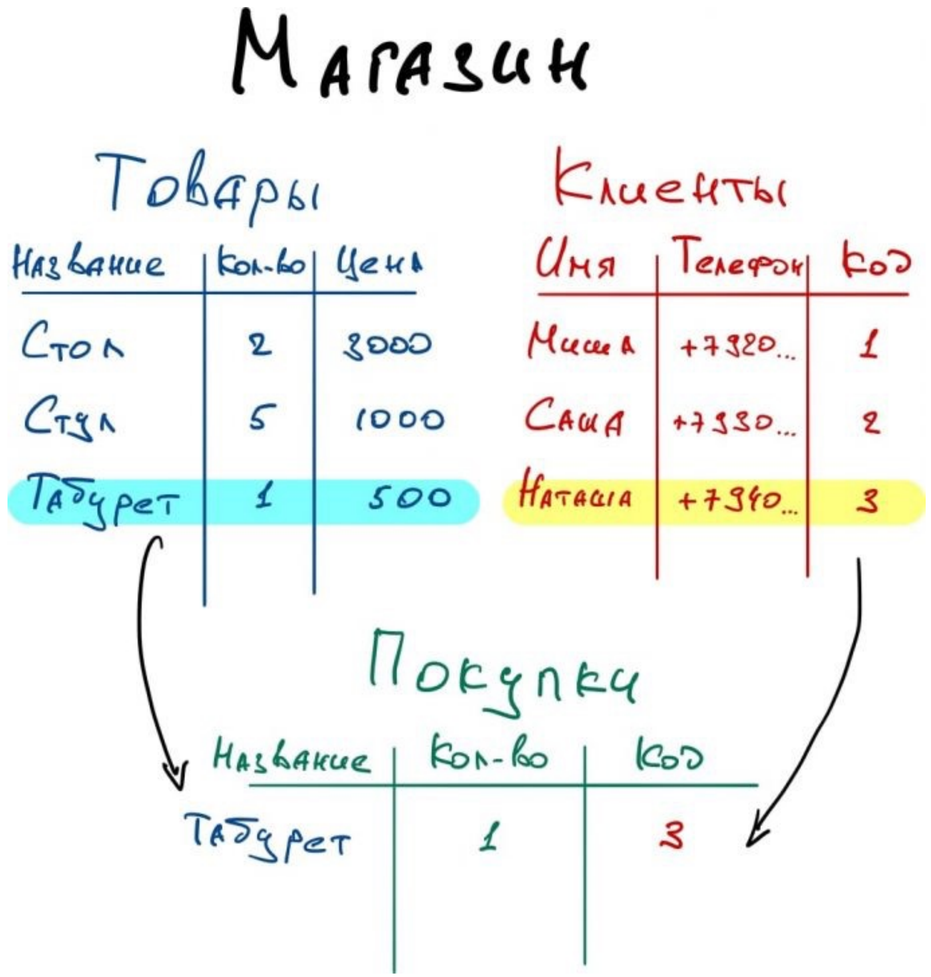
# Базы данных

# Базы даных


Базы Данных
Реляционные
Нереляционные
Сетевые
Иерархические

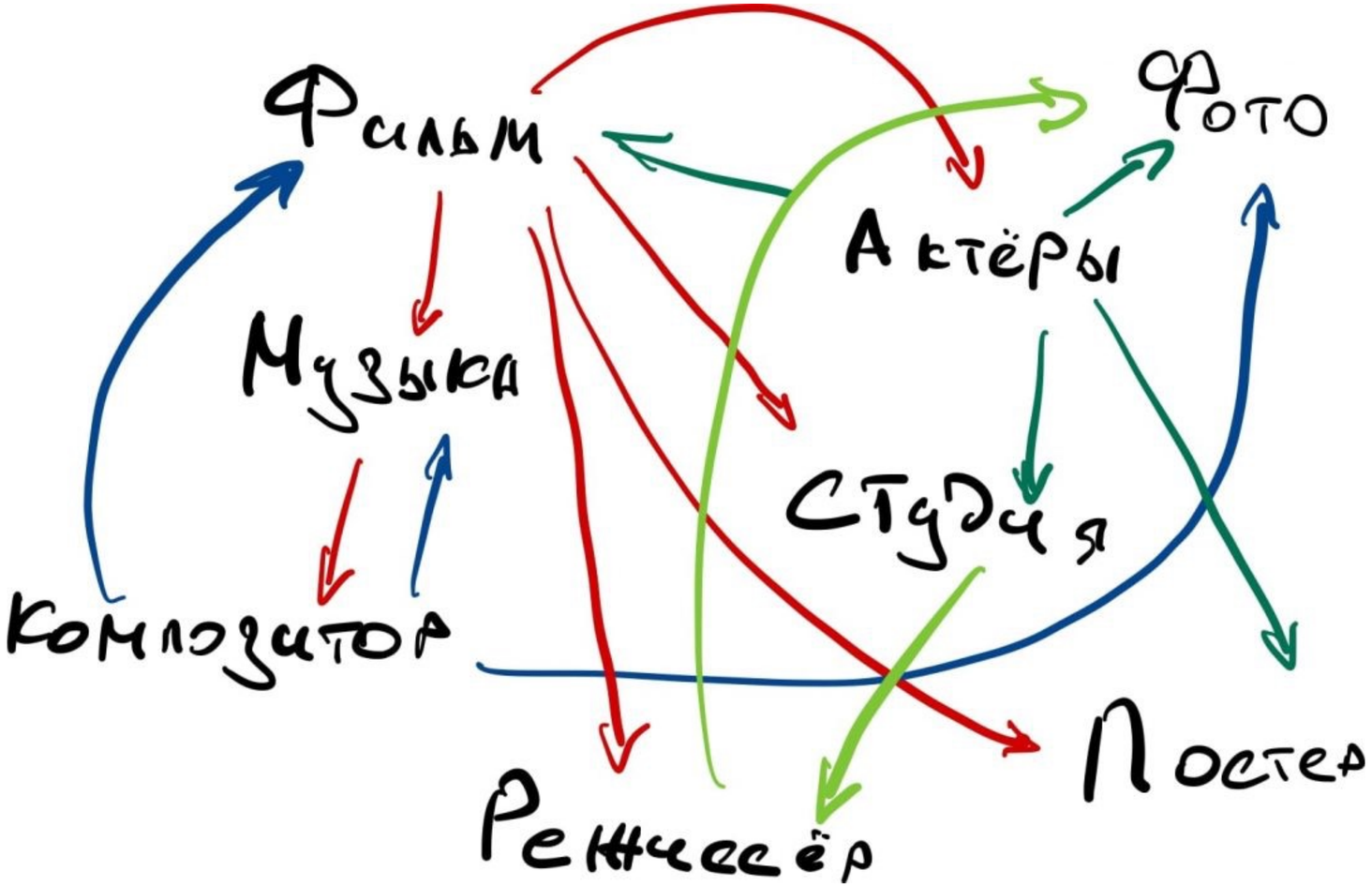
# Реляционные БД

**Реляционные БД** ещё называют табличными, потому что все данные в них можно представить в виде разных таблиц. Одни таблицы связаны с другими, а другие — с третьими. Например, база данных покупок в магазине может выглядеть так:



# Сетевые БД

В отличие от реляционных баз, в сетевых между таблицами и записями может быть несколько разных связей, каждая из который отвечает за что-то своё.
Если мы возьмём базу данных с сайта Кинопоиска, то она может выглядеть так:



# Сетевые БД

Например, вы посмотрели «Начало» Кристофера Нолана и вам понравился этот фильм. Когда вы перейдёте к списку фильмов, которые он ещё снял, база на сайте сделает так:

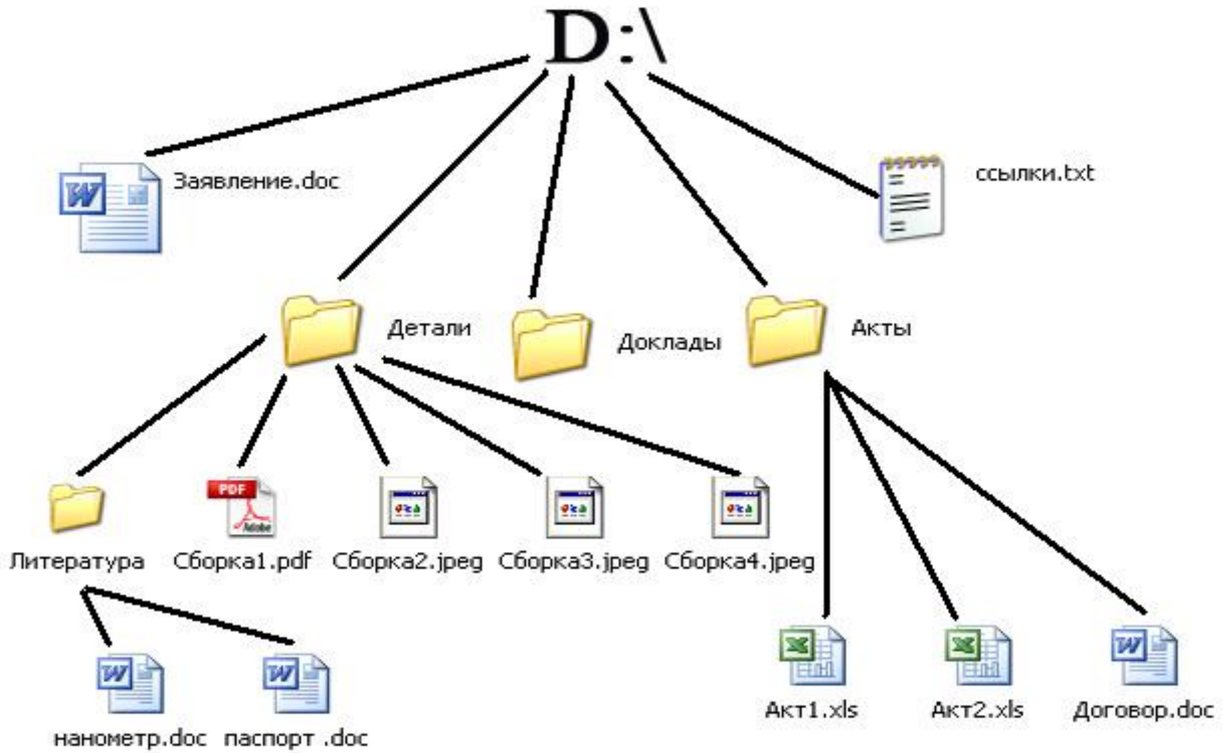
- возьмёт имя режиссёра;
- посмотрит, какие связи и с чем у него есть;
- выдаст список фильмов;
- к этим фильмам может сразу подгрузить список актёров, которые там играют;
- и сразу же показать постеры к каждому фильму.

А главное — база сделает это очень быстро, потому что ей не нужно просматривать всю базу в поисках нужных фильмов. Она сразу видит, какие фильмы с чем связаны, и выдаёт ответ.

# Иерархические БД

Иерархия — это когда есть вышестоящий, а есть его подчинённые, кто ниже. У них могут быть свои подчинённые и так далее.

Самый простой пример такой базы данных — хранение файлов
и папок на компьютере:



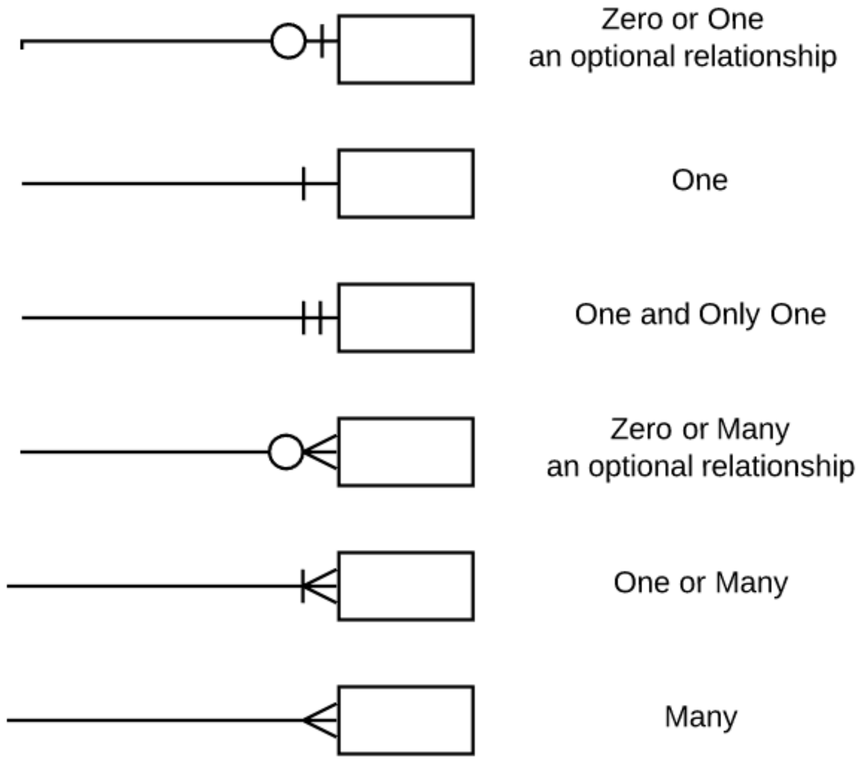
# Выводы по БД

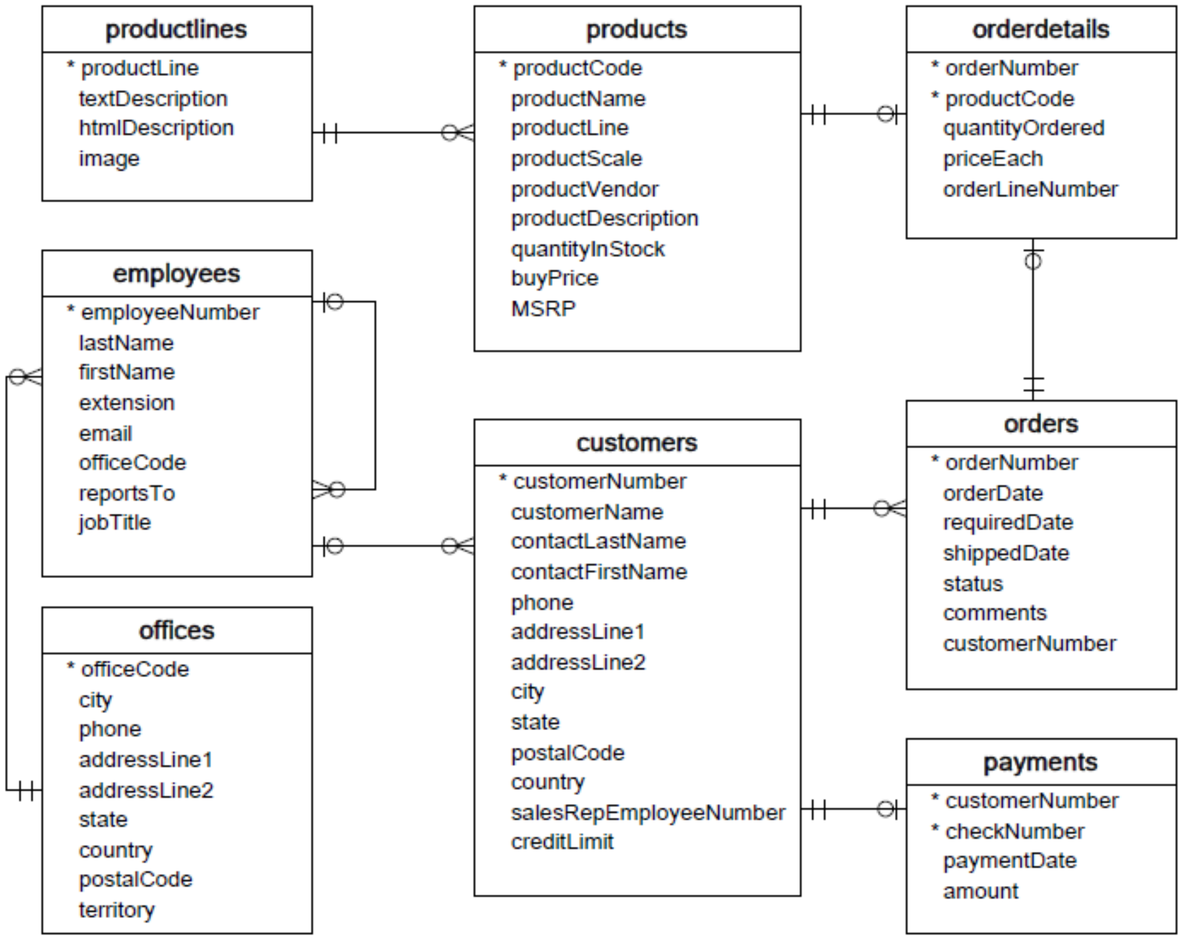
- Чаще всего базы данных напоминают таблицы: в них одному параметру соответствует один набор данных. Например, один клиент — одно имя, один телефон, один адрес.
- Такие «табличные» базы данных называются реляционными.
- Чтобы строить сложные связи, разные таблицы в реляционных базах можно связывать между собой: ставить ссылки.
- Реляционная база — не единственный способ хранения данных. Есть ситуации, когда нам нужна большая гибкость в хранении.
- Бывают сетевые базы данных: когда нужно хранить много связей между множеством объектов. Например, каталог фильмов: в одном фильме может участвовать много человек, а каждый из них может участвовать во множестве фильмов.
- Бывают иерархические базы, или «деревья». Пример — наша файловая система.
- Какую выбрать базу — зависит от задачи. Одна база не лучше другой, но они могут быть более или менее подходящими для определённых задач.

# SQL

# SQL

## SQL - язык запросов для реляционных БД




ER-диаграмма в crow-foot нотации

# SQL vs noSQL

Преимущества **SQL**-базы **(напр. MySQL)** :

- **Проверено временем:** MySQL — крайне развитая СУБД, что означает наличие большого сообщества вокруг неё, множество примеров и высокую надёжность;
- **Совместимость:** MySQL доступна на всех основных платформах, включая Linux, Windows, Mac, BSD и Solaris. Также у неё есть библиотеки для языков вроде Node.js, Ruby, C#, C++, Java, Perl, Python и PHP;
- **Окупаемость:** Это СУБД с открытым исходным кодом, находящаяся в свободном доступе;

Преимущества **NoSQL (напр. MongoDB)**:

- **Динамическая схема:** Как упоминалось выше, эта СУБД позволяет гибко работать со схемой данных без необходимости изменять сами данные;
- **Масштабируемость:** MongoDB горизонтально масштабируема, что позволяет легко уменьшить нагрузку на сервера при больших объёмах данных;
- **Удобство в управлении:** СУБД не нуждается в отдельном администраторе базы данных. Благодаря достаточному удобству в использовании, ей легко могут пользоваться как разработчики, так и системные администраторы;
- **Скорость:** Высокая производительность при выполнении простых запросов;

noSQL «not only SQL» - неструктурированный формат хранения данных

# Диалекты SQL






# Типы данных

| Тип | Описание | Пример |
|---|---|---|
| INT, NUMBER | Целое число | 11, 53, 0, -4 |
| FLOAT, DECIMAL | Дробное число | 11.12, 42.0, -14.2 |
| VARCHAR, TEXT | Текст | «Mexico»<br>«»<br>«12.4» |
| TIME, DATETIME, TIMESTAMP | Дата и время | 2008-10-29<br>14:56:59<br>1225292219 |

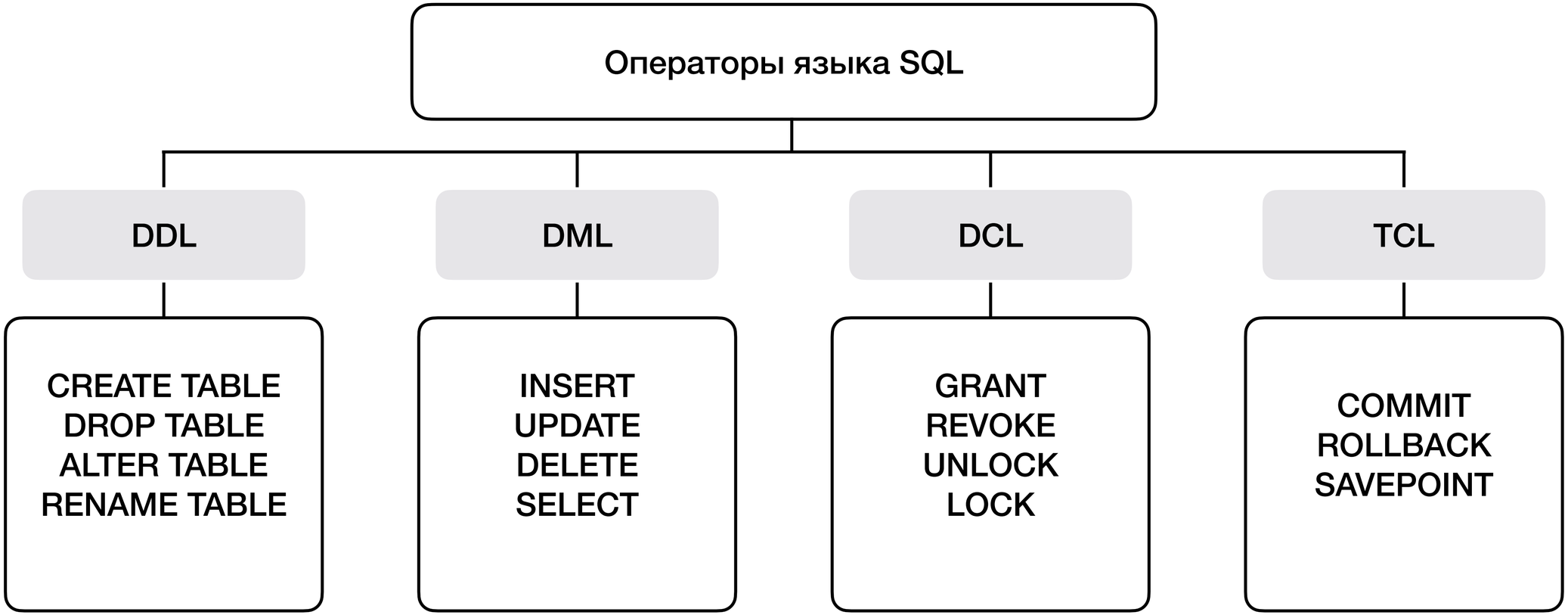
# Структура SQL-запроса

Общая структура запроса выглядит следующим образом:

```
SELECT ('столбцы или * для выбора всех столбцов; обязательно')
FROM ('таблица; обязательно')
WHERE ('условие/фильтрация, например, city = 'Moscow'; необязательно')
GROUP BY ('столбец, по которому хотим сгруппировать данные; необязательно')
HAVING ('условие/фильтрация на уровне сгруппированных данных; необязательно')
ORDER BY ('столбец, по которому хотим отсортировать вывод; необязательно')
```

Разберем структуру. Для удобства текущий изучаемый элемент в запроса выделяется CAPS'ом.

# Операторы SQL




**DDL** - операторы описания данных
**DML** - операторы манипулирования данными
**DCL** - операторы задания прав в БД
**TCL** - операторы защиты, восстановления данных и тд

# Работа с БД

Допустим имеется следующий табличный набор данных «customers»:

Number of Records: 91

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|---|---|---|---|---|---|---|
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constituciyn 2222 | Мйxico D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquerна | Antonio Moreno | Mataderos 2312 | Мйxico D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbkцp | Christina Berglund | Berguvsvдgen 8 | Lulee | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel риre et fils | Frйdйrique Citeaux | 24, place Klйber | Strasbourg | 67000 | France |
| 8 | Bylido Comidas preparadas | Martнn Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |

что выдаст следующий запрос?

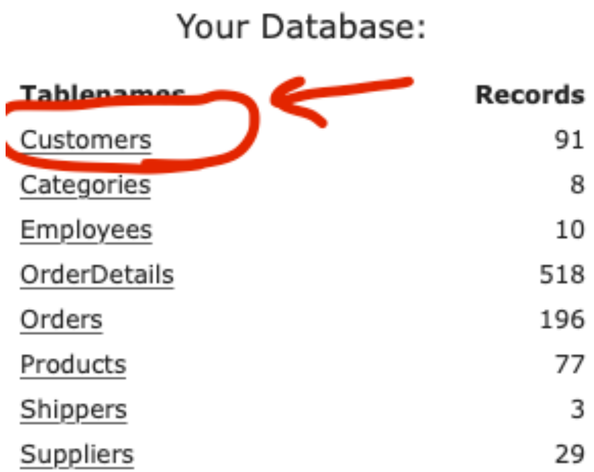
**SELECT * FROM Customers WHERE Country = "Mexico"**

# Пример

1. Перейдите по ссылке:

https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in

2. Откройте таблицу Customers



# 1. SELECT, FROM

SELECT, FROM — обязательные элементы запроса, которые определяют выбранные столбцы, их порядок и источник данных.

1. Выбрать все (обозначается как *) из таблицы Customers:

```
SELECT * FROM Customers
```

2. Выбрать столбцы CustomerID, CustomerName из таблицы Customers:

```
SELECT CustomerID, CustomerName FROM Customers
```

# 2. WHERE

WHERE — необязательный элемент запроса, который используется, когда нужно отфильтровать данные по нужному условию. Очень часто внутри элемента where используются IN / NOT IN для фильтрации столбца по нескольким значениям, AND / OR для фильтрации таблицы по нескольким столбцам.

1. Фильтрация по одному условию и одному значению:

```
select * from Customers
WHERE City = 'London'
```

2. Фильтрация по одному условию и нескольким значениям с применением IN (включение) или NOT IN (исключение):

```
select * from Customers
where City IN ('London', 'Berlin')
```

```
select * from Customers
where City NOT IN ('Madrid', 'Berlin','Bern')
```

3. Фильтрация по нескольким условиям с применением AND (выполняются все условия) или OR (выполняется хотя бы одно условие) и нескольким значениям:

```
select * from Customers
where Country = 'Germany' AND City not in ('Berlin', 'Aachen') AND CustomerID > 15
```

# 3. GROUP BY

GROUP BY — необязательный элемент запроса, с помощью которого можно задать агрегацию по нужному столбцу (например, если нужно узнать какое количество клиентов живет в каждом из городов).

При использовании GROUP BY обязательно:

- перечень столбцов, по которым делается разрез, был одинаковым внутри SELECT и внутри GROUP BY,
- агрегатные функции (SUM, AVG, COUNT, MAX, MIN) должны быть также указаны внутри SELECT с указанием столбца, к которому такая функция применяется.

1. Группировка количества клиентов по городу:

```
select City, count(CustomerID) from Customers
GROUP BY City
```

2. Группировка количества клиентов по стране и городу:

```
select Country, City, count(CustomerID) from Customers
GROUP BY Country, City
```

# 3. GROUP BY

3. Группировка продаж по ID товара с разными агрегатными функциями: количество заказов с данным товаром и количество проданных штук товара:

```
select ProductID, COUNT(OrderID), SUM(Quantity) from OrderDetails
GROUP BY ProductID
```

4. ЗАДАНИЕ: Вывести кол-во клиентов по городам Германии:

# TODO

5. Переименование столбца с агрегацией с помощью оператора AS. По умолчанию название столбца с агрегацией равно примененной агрегатной функции, что далее может быть не очень удобно для восприятия.

```
select City, count(CustomerID) AS Number_of_clients from Customers
group by City
```

# 4. HAVING

HAVING — необязательный элемент запроса, который отвечает за фильтрацию на уровне сгруппированных данных (по сути, WHERE, но только на уровень выше).

1. Фильтрация агрегированной таблицы с количеством клиентов по городам, в данном случае оставляем в выгрузке только те города, в которых не менее 5 клиентов:

```
select City, count(CustomerID) from Customers
group by City
HAVING count(CustomerID) >= 5
```

2. В случае с переименованным столбцом внутри HAVING можно указать как и саму агрегирующую конструкцию count(CustomerID), так и новое название столбца number_of_clients:

```
select City, count(CustomerID) as number_of_clients from Customers
group by City
HAVING number_of_clients >= 5
```

3. ЗАДАНИЕ: Отфильтруйте таблицу по пользователям (CustomerName NOT IN 'Around the Horn','Drachenblut Delikatessend'), рассчитайте кол-во клиентов по городам и оставьте только те города, где кол-во клиентов не менее 5.

# TODO

# 5. ORDER BY

ORDER BY — необязательный элемент запроса, который отвечает за сортировку таблицы.

1. Простой пример сортировки по одному столбцу. В данном запросе осуществляется сортировка по городу, который указал клиент:

```
select * from Customers
ORDER BY City
```

2. Осуществлять сортировку можно и по нескольким столбцам, в этом случае сортировка происходит по порядку указанных столбцов:

```
select * from Customers
ORDER BY Country, City
```

3. Если нужна обратная сортировка, то в конструкции ORDER BY после названия столбца надо добавить DESC:

```
select * from Customers
order by CustomerID DESC
```

4. ЗАДАНИЕ: осуществите обратную сортировку по столбцу Country и сортировка по умолчанию по столбцу  City

# TODO

# 6. JOIN

JOIN — необязательный элемент, используется для объединения таблиц по ключу, который присутствует в обеих таблицах. Перед ключом ставится оператор ON.

1. Запрос, в котором соединяем таблицы Order и Customer по ключу CustomerID, при этом перед названиям столбца ключа добавляется название таблицы через точку:

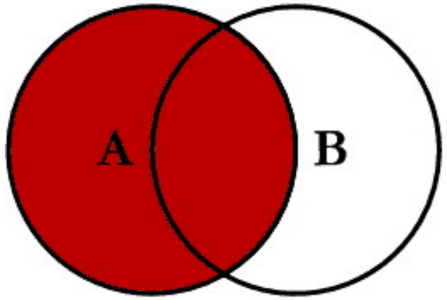
```
select * from Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
```

Нередко может возникать ситуация, когда надо промэппить одну таблицу значениями из другой. В зависимости от задачи, могут использоваться разные типы присоединений. INNER JOIN — пересечение, RIGHT/LEFT JOIN для мэппинга одной таблицы значениями из другой,

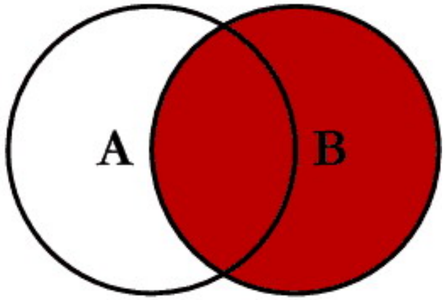
2. Внутри всего запроса JOIN встраивается после элемента from до элемента where, пример запроса:

```
select * from Orders
join Customers on Orders.CustomerID = Customers.CustomerID
where Customers.CustomerID >10
```

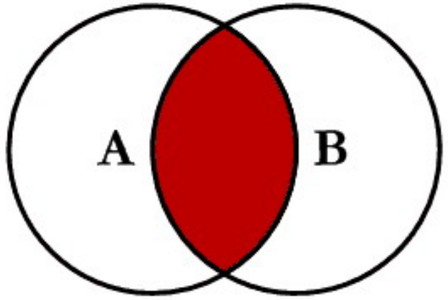
# SQL JOINS



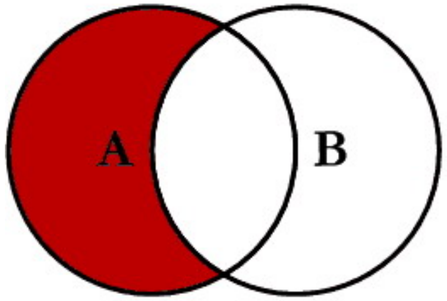

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



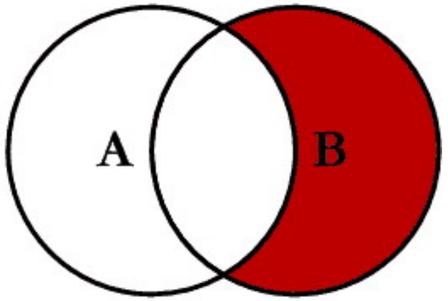

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



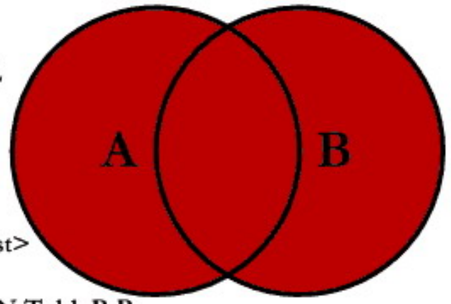

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



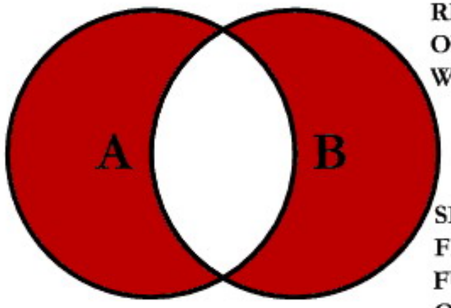

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

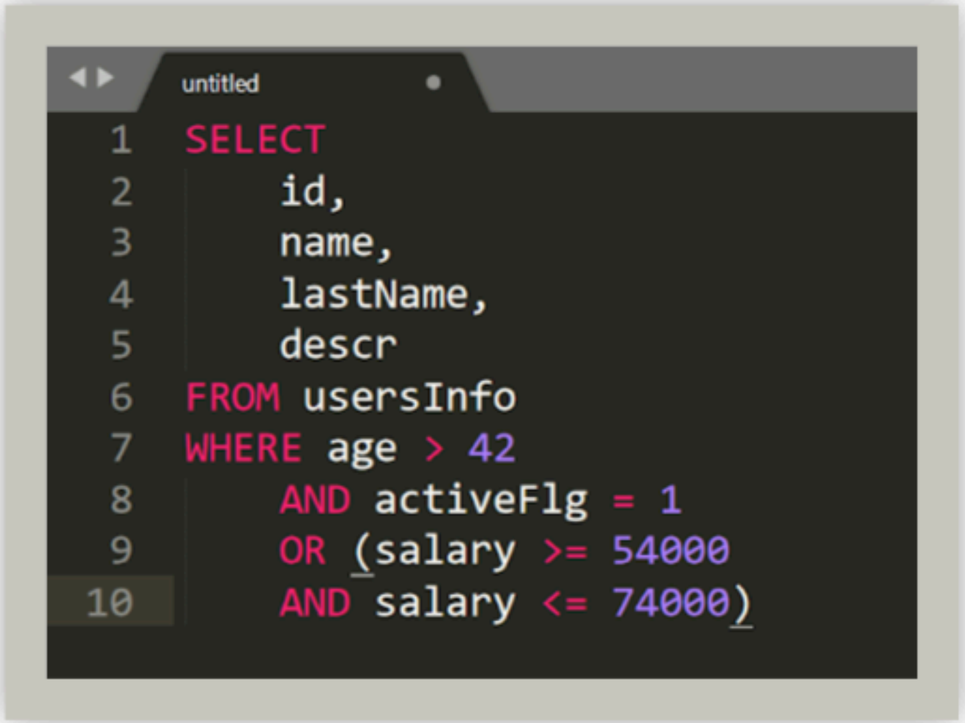



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# Операторы SQL


Пример логических операторов