



Райффайзен
БАНК

ОСНОВЫ SQL



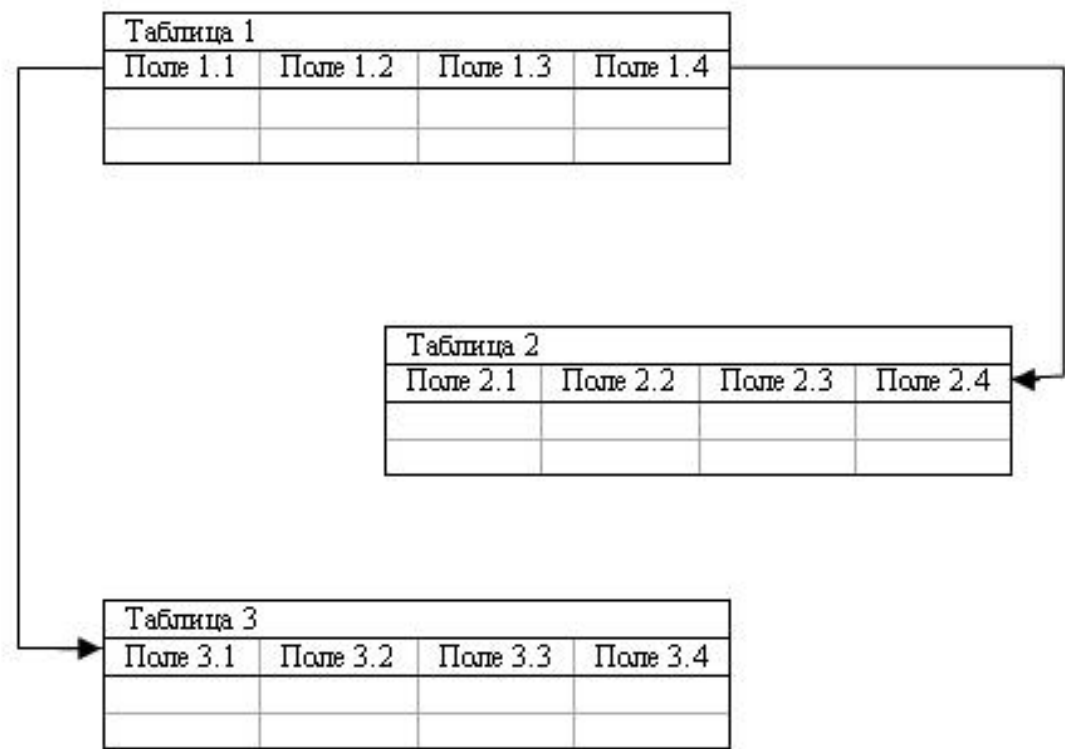
Понятие БД

База данных - комплекс информации, данных, которые структурированы и связаны между собой.



Реляционная модель данных

Реляционные базы данных (БД) имеют табличную форму организации т.е. реализованы в виде таблиц, связанных между собой отношениями с помощью кодов (ключей)





Особенности реляционных БД

- таблице присваивается имя, уникальное в пределах БД
- каждому столбцу присваивается имя, уникальное в пределах таблицы
- каждый столбец таблицы содержит данные одного типа
- строки таблицы не имеют имен (но имеют уникальный идентификатор - ключ)
- в таблице нет одинаковых строк



Таблица БД

Таблица представляет из себя набор **столбцов**.

Столбцы таблицы могут называть **полями** или **колонками**, все эти слова будут использоваться как синонимы.

Строки, записи – тоже синонимы

Поле или столбец

Запись или строка

Код	Фамилия	Имя	Отчество	Дата рождения
1	Иванов	Иван	Петрович	20.07.2000
2	Петров	Илья	Иванович	12.11.1988
3	Сидорова	Анна	Сергеевна	20.06.1989
4	Смирнов	Лев	Владимирович	01.01.2000



Первичный ключ (PRIMARY KEY)

PRIMARY KEY (первичный ключ) – это поле (или комбинация полей), которое однозначно идентифицирует запись.

В таблице не может быть двух записей с одинаковым значением первичного ключа.

PRIMARY KEY должен:

- содержать уникальные значения
- не может содержать NULL значения

Таблица может иметь **только один первичный ключ!**



Могут ли эти данные быть ключом?

фамилия

номер и серия паспорта

номер дома

регистрационный номер автомобиля



ВНЕШНИЙ КЛЮЧ

Внешний ключ (FOREIGN KEY) – поле (или группа полей), значение которого может повторяться для нескольких записей (например, столбец «PersonID» в таблице Orders (Заказы)).

Внешний ключ содержит ссылку на поле первичного ключа в другой таблице.

Таблица содержащая внешний ключ называется дочерней (Orders), содержащая первичный ключ – родительской (Person)

«Person» Персоны

PersonID	Last Name	First Name	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

«Orders» Заказы

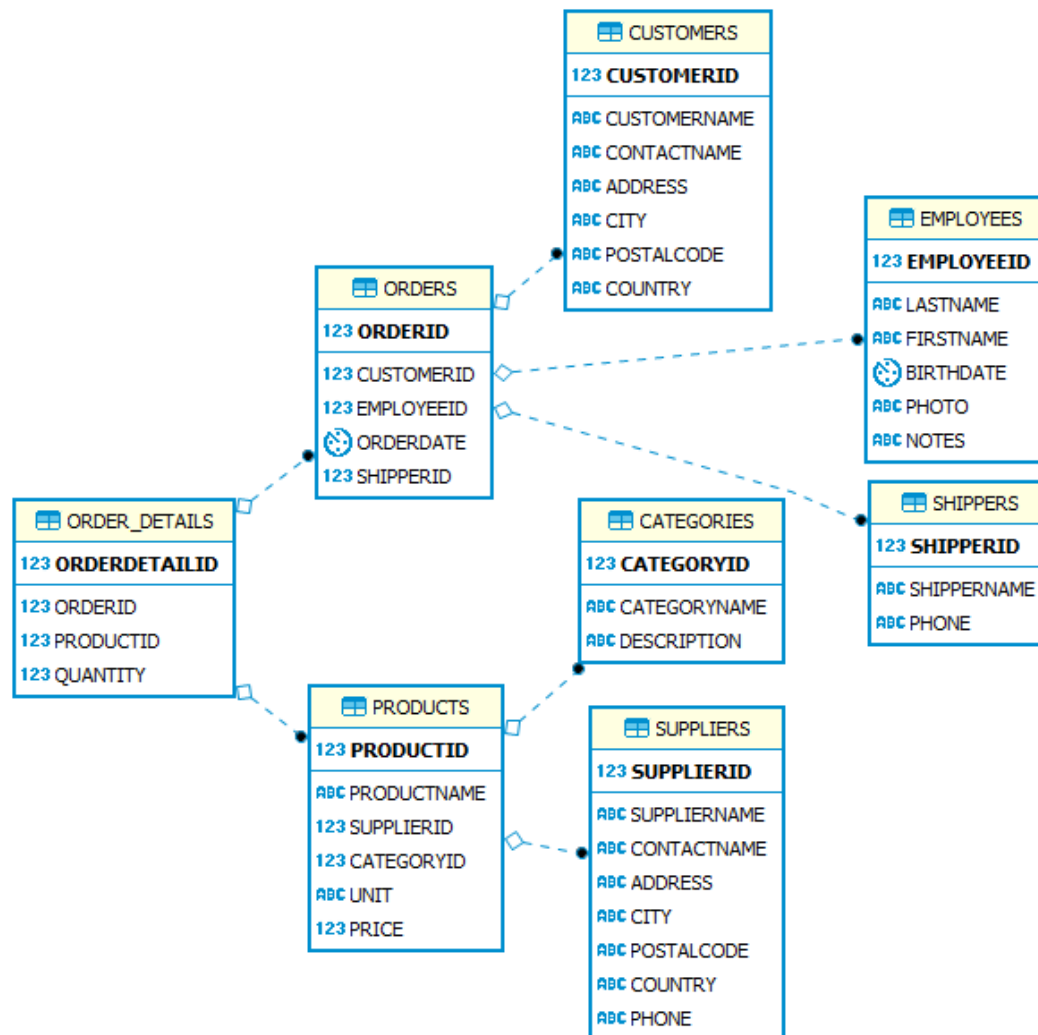
OrderID	Order Number	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1





Схема БД

Схема БД - это
графический образ БД





Типы связей между таблицами

Существует три типа связей между таблицами:

Связь “ОДИН-КО-МНОГИМ”


Связь “ОДИН-К-ОДНОМУ”

Связь “МНОГИЕ-КО-МНОГИМ”

Связи между таблицами

Один к одному («1-1») – одной записи в первой таблице соответствует ровно одна запись во второй.

Применение: выделение часто используемых данных.



Код	Фамилия	Имя
1	Иванов	Кузьма
2	Петров	Василий
...		

Код	Год рожд.	Адрес
1	1992	Суворовский, д.20, кв. 6
2	1993	Кирочная, д. 30, кв 18
...		

Один ко многим («1-∞») – одной записи в первой таблице соответствует сколько угодно записей во второй.



Код	Название
1	Монитор
2	Винчестер
...	

Код	Код товара	Цена
123	1	10 999
345	1	11 999
...		



Связи между таблицами

Многие ко многим («∞ - ∞») – одной записи в первой таблице соответствует сколько угодно записей во второй, и наоборот.

учителя

Код	Фамилия
1	Иванов
2	Петров
...	...

∞

∞

Код	Название
1	История
2	География
3	Биология
...	...

предметы

Реализация – через третью таблицу и две связи «1-∞».

1

∞

∞

1

Код	Фамилия
1	Иванов
2	Петров
...	...

Код	Код учителя	Код предмета	Класс
1	1	1	9-А
2	1	2	8-Б
3	2	3	7-В
...

расписание

Код	Название
1	История
2	География
3	Биология
...	...



Немного истории

В начале 1970-х годов в одной из исследовательских лабораторий компании IBM была разработана экспериментальная реляционная СУБД IBM System R, для которой был создан специальный язык **SEQUEL**, позволявший управлять данными в этой СУБД.

Аббревиатура SEQUEL расшифровывалась как Structured English Query Language — **«структурированный английский язык запросов»**.

Позже язык SEQUEL был переименован в SQL (Structured Query Language)



Создатели SQL

Дональд Д. Чемберлин



Раймонд Ф. Бойс





Как работает SQL

SQL сам по себе не является ни СУБД, ни отдельным продуктом. Это язык, применяемый для взаимодействия с СУБД.





С помощью чего можно выполнить SQL запрос

Все современные СУБД содержат в своем составе утилиты, позволяющие выполнить SQL запрос и посмотреть его результат.



Структура языка SQL

Операторы определения данных
(DDL Data Definition Language)



CREATE TABLE
DROP TABLE
ALTER TABLE
CREATE VIEW
ALTER VIEW
DROP VIEW
CREATE INDEX
DROP INDEX

**Операторы манипулирования данными
(DML Data Manipulation Lanquage)**



DELETE
INSERT
UPDATE
SELECT

Средства администрирования данных
(DCL Data Control Language)



GRANT
REVOKE
DENY

Средства управления транзакциями
(TCL Transaction Control Language)



COMMIT
ROLLBACK
SAVEPOINT



SELECT

Используется для выбора данных из базы данных (БД). Данные, возвращаемые в результате запроса, называются множеством результатов.

```
SELECT column_list  
FROM table_name  
[WHERE условие]  
[GROUP BY условие]  
[HAVING условие]  
[ORDER BY условие]
```

SELECT - ключевое слово, которое сообщает базе данных о том, что оператор является запросом. Все запросы начинаются с этого слова, за ним следует пробел.

FROM - ключевое слово, которое должно присутствовать в каждом запросе. После него через пробел указывается имя таблицы (**table_name**), являющейся источником данных.

Column_list - список столбцов таблицы, которые выбираются запросом. Столбцы, не указанные в операторе, не будут включены в результат.

SELECT column1, column2, ...FROM table_name

Если необходимо вывести данные всех столбцов, можно использовать (*).

*SELECT * FROM table_name*

Операторы в скобках являются не обязательными в SELECT.

SQL является регистронезависимым.



Задачи

- 1. Вывести все поля и записи таблицы Customers
- 2. Необходимо вывести список клиентов (таблица Customers), отобразить их имена (CustomerName), город (City), страну (Country).

SELECT Syntax:
SELECT column1, column2, ...
FROM table_name

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



Задачи

1. Вывести все поля и записи таблицы Customers
*SELECT * FROM Customers*
2. Необходимо вывести список клиентов (таблица Customers), отобразить их имена (CustomerName), город (City), страну (Country).
*SELECT CustomerName, City, Country
FROM Customers*

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



DISTINCT

Столбец таблицы часто содержит много повторяющихся значений.

DISTINCT возвращает только уникальные значения (т.е отсеивает дубли).

DISTINCT Syntax:

```
SELECT DISTINCT column1, column2, ...  
FROM table_name
```



Задачи

DISTINCT Syntax:

```
SELECT DISTINCT column1, column2, ...  
FROM table_name
```

- 1. Вывести без повторов значения из столбца Country таблицы Customers

```
SELECT DISTINCT Country FROM Customers
```

- 2. Вывести без повторов значения из столбца City таблицы Customers

```
SELECT DISTINCT City FROM Customers
```

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



WHERE

Предложение WHERE используется для извлечения только тех записей, которые удовлетворяют заданному условию (фильтрация записей).

WHERE также используется в конструкциях UPDATE, DELETE

WHERE Syntax:

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition
```

SQL требует при задании условий WHERE:

для **ТЕКСТОВЫХ** значений - одинарные кавычки

для **ЧИСЛОВЫХ** значений - отсутствие кавычек



Примеры

PostalCode - числовое поле

```
SELECT * FROM Customers WHERE PostalCode=67000
```

Country - текстовое поле

```
SELECT * FROM Customers WHERE Country='France'
```




Операторы в WHERE

Следующие операторы могут быть использованы в предложении **WHERE**:

Оператор	Описание
=	Равно
<>	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
BETWEEN	Диапазон значений
LIKE	Поиск по шаблону
IN	Для указания нескольких возможных значений столбца



Задачи

1. Выбрать из таблицы «Клиенты» (Customers) клиентов из города «Париж» (Paris)

***SELECT * FROM Customers
WHERE City='Paris'***

2. Вывести продукты (таблица Products) с ценой (Price) больше 55

SELECT * FROM PRODUCTS WHERE Price>55

PRODUCTS (Продукты)	
ProductID	
ProductName	
SupplierID	
CategoryID	
Unit	
Price	

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



AND, OR и NOT Операторы

Предложение WHERE может быть объединено с AND, OR и NOT операторами. OR и AND операторы используются для фильтрации записей основанной на более чем одном условии.

AND является логическим «И». Отображает запись, если **все условия**, разделенные «AND» ИСТИНА

OR является логическим «ИЛИ». Отображает запись, если **какое-либо из условий** разделенных «OR» ИСТИНА

NOT отображается запись, если **условие не соответствует** действительности



AND, OR, NOT Syntax

AND Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```



Задачи

- 1. Выбрать строки из таблицы «Customers» (Клиенты), где страна (поле Country)='Germany' и Город (поле City)='Berlin'
- 2. Выбрать строки из таблицы «Customers» (Клиенты), где город (поле City)='Berlin' или 'London'
- 3. Выбрать строки из таблицы «Customers» (Клиенты), где страна (поле Country) не 'France'

AND Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...  
FROM table_name WHERE NOT condition;
```

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



Задачи

1. Выбрать строки из таблицы «Customers» (Клиенты), где страна (поле Country) 'Germany' и Город (поле City)='Berlin'

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

2. Выбрать строки из таблицы «Customers» (Клиенты), где город (поле City)='Berlin' или 'London'

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

3. Выбрать строки из таблицы «Customers» (Клиенты), где страна (поле Country) не 'France'

SELECT * FROM Customers WHERE NOT Country='France'

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



NULL

Поле со значением **NULL** представляет собой поле без значения. Если поле в таблице не является обязательным, то можно вставить новую запись или обновить без добавления значения в поле, оно будет сохранено со значением NULL.

Значение NULL отличается от нулевого значения (0) или поля, которое содержит пробелы (« »).

Невозможно проверить значения NULL операторами сравнения, такими как =, < или <>.

Нужно использовать **IS NULL** или **IS NOT NULL**



IS NULL / IS NOT NULL

IS NULL Syntax

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

IS NOT NULL Syntax

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




Задачи

IS NULL Syntax

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

IS NOT NULL Syntax

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

- 1. Найти записи в таблице Customers, в которых не указан адрес, но указана контактная информация (поле ContactName)

```
SELECT * FROM Customers WHERE  
Address IS NULL and ContactName IS NOT NULL
```

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



ORDER BY

ORDER BY используется для сортировки множества результатов в порядке возрастания или убывания.

ORDER BY сортирует записи в возрастающем порядке по умолчанию. Чтобы отсортировать записи в порядке убывания используется **DESC**.

ORDER BY Syntax

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... DESC;
```



Задачи

ORDER BY Syntax

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... DESC;
```

- 1. Отсортировать строки таблицы «Customers» (Клиенты) по полю Country (страна) по возрастанию

```
SELECT * FROM Customers ORDER BY Country
```

- 2. Отсортировать данные таблицы «Customers» (Клиенты) по 2м полям: Страна по возрастанию, Город по убыванию

```
SELECT * FROM Customers  
ORDER BY Country, City DESC
```

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



SELECT TOP

TOP используется для указания количества возвращаемых записей.

MS SQL Server / MS Access Syntax:

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

MySQL Syntax:

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

Oracle Syntax:

```
SELECT column_name(s)  
FROM table_name  
WHERE ROWNUM <= number;
```



Задачи (SQL Server / MS Access Syntax)

1. Написать запрос, который выведет 5 любых записей из таблицы Customers

*SELECT TOP 5 * FROM Customers*

2. Написать запрос, который выведет 10 % записей из таблицы Customers

*SELECT TOP 10 PERCENT * FROM Customers*



SQL MIN() and MAX() Functions

Функция **MIN ()** возвращает наименьшее значение в выбранном столбце.
Функция **MAX ()** возвращает наибольшее значение в выбранном столбце.

MIN() Syntax

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

MAX() Syntax

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



Задачи

1. Вывести максимальное значение из поля Price таблицы Products

SELECT MAX(Price) FROM Products

2. Вывести минимальное значение из поля Price таблицы Products

SELECT MIN(Price) FROM Products

3. Вывести одним запросом максимальное и минимальное значение по столбцу Price

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

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



COUNT(), AVG() and SUM() Functions

Функция **COUNT()** возвращает количество строк, которое соответствует указанным критериям

Функция **AVG()** возвращает среднее значение **ЧИСЛОВОГО** столбца

Функция **SUM()** возвращает общую сумму по **ЧИСЛОВОМУ** столбцу



COUNT(), AVG(), SUM() Syntax

COUNT() Syntax

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

AVG() Syntax

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

SUM() Syntax

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



Задачи

1. Посчитать кол-во записей в таблице *Products*
SELECT COUNT() FROM Products*
2. Посчитать среднюю цену товаров в таблице *Products* с категорией 2 или 5 (столбец *CategoryID*)

SELECT AVG(Price) FROM Products where CategoryID=2 or CategoryID=5

PRODUCTS (Продукты)	
ProductID	
ProductName	
SupplierID	
CategoryID	
Unit	
Price	



LIKE

Оператор LIKE используется в предложении WHERE для поиска указанного шаблона в столбце.

LIKE Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE columnName LIKE pattern
```

Подстановочные знаки, используемые в сочетании с LIKE:

Символ	Пояснение
%	Соответствует любой строке любой длины (в том числе нулевой длины)
_	Соответствует одному символу

Знак процента и подчеркивание могут быть использованы в комбинации!



Задачи

1. Найти всех клиентов с именем клиента (CustomerName), начинающиеся с "М":

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

2. Найти клиентов с именем клиента (CustomerName), заканчивающиеся на «Я»:

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

3. Найти клиентов, у которых в имени (CustomerName) в любой позиции содержится «Иван»:

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

4. Выбрать клиентов с CUSTOMERNAME, которое начинается с «W» и имеет длину 6 символов

```
SELECT * FROM Customers WHERE CustomerName LIKE 'W_____'
```



IN

Оператор IN выбирает записи из таблицы по определенным значениям поля. Оператор IN является сокращением для нескольких условий OR.

IN Syntax

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



Задачи

IN Syntax

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

- 1. Выбрать всех клиентов (таблица *Customers*), которые находятся в странах «*Brazil*», «*Mexico*» или «*Canada*» (столбец *Country*)
- 2. Выбрать всех клиентов, которые НЕ находятся в «*Brazil*», «*Mexico*» или «*Canada*»(столбец *Country*)
- 3. Выбрать товары из таблицы *Products* с категорией 2 или 5 (столбец *CategoryID*)

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



Задачи

1. Выбрать всех клиентов (таблица *Customers*), которые находятся в странах «*Brazil*», «*Mexico*» или «*Canada*» (столбец *Country*):

```
SELECT * FROM Customers  
WHERE Country IN ('Brazil', 'Mexico', 'Canada')
```

2. Выбрать всех клиентов, которые НЕ находятся в «*Brazil*», «*Mexico*» или «*Canada*» (столбец *Country*):

```
SELECT * FROM Customers  
WHERE Country not IN ('Brazil', 'Mexico', 'Canada')
```

3. Выбрать товары из таблицы *Products* с категорией 2 или 5 (столбец *CategoryID*)

```
SELECT ProductName FROM Products where CategoryID in (2,5)  
SELECT ProductName FROM Products where CategoryID=2 or  
CategoryID=5
```



BETWEEN

Оператор BETWEEN выбирает значения в заданном диапазоне. Значения могут быть числами, текстом или датами.

BETWEEN Syntax

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




Задачи

BETWEEN Syntax

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

Выбрать продукты(таблица *Products*) с ценой(*Price*) от 50 до 150

```
SELECT * FROM Products WHERE Price BETWEEN 50 AND 150
```

PRODUCTS (Продукты)	
ProductID	
ProductName	
SupplierID	
CategoryID	
Unit	
Price	



Aliases

Псевдонимы используются для присвоения таблице или столбцу временного имени.

Alias Column Syntax

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

Alias Table Syntax

```
SELECT column_name(s)  
FROM table_name AS <alias_name>
```

«AS» - необязателен в большинстве СУБД.

Aliases могут быть полезны, когда:

В запросе задействовано более одной таблицы

В запросе используются функции

Имена столбцов являются большими или не очень читаемыми

Два или более столбцов объединяются вместе



Задачи (MS SQL)

1. Вывести все строки таблицы *Customers* и все столбцы (используя *) с присвоением псевдонима таблице «*Customers*».

*SELECT * FROM Customers C*

2. Вывести из таблицы *Customers* столбцы *CustomerName*, *Country*.
Заголовок столбца *Country* переименовать в «Страна»

SELECT CustomerName, Country 'Страна' FROM Customers

CustomerName	Страна
Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taqueria	Mexico
Around the Horn	UK



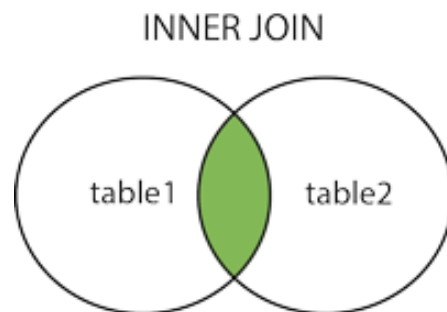
JOIN

JOIN- операция горизонтального соединения данных, при которой таблицы сравниваются между собой построчно. Т.е суть операции объединения состоит в том, чтобы склеить разбитые по таблицам данные **на основе связей между ними (ключей)** и привести их в удобочитаемый вид.



INNER JOIN

INNER JOIN выбирает записи, которые имеют совпадающие значения в обеих таблицах.



INNER JOIN Syntax

`SELECT column_name(s)`

`FROM table1`

`INNER JOIN table2 ON table1.column_name = table2.column_name`

Примечание:

Внутреннее соединение выбирает все строки из обеих таблиц до тех пор, пока существует соответствие между столбцами.

В большинстве СУБД ключевое слово INNER является необязательным, поэтому можно просто не указывать его.



INNER JOIN

INNER JOIN выбирает записи, которые имеют совпадающие значения (по ключу) в обеих таблицах.

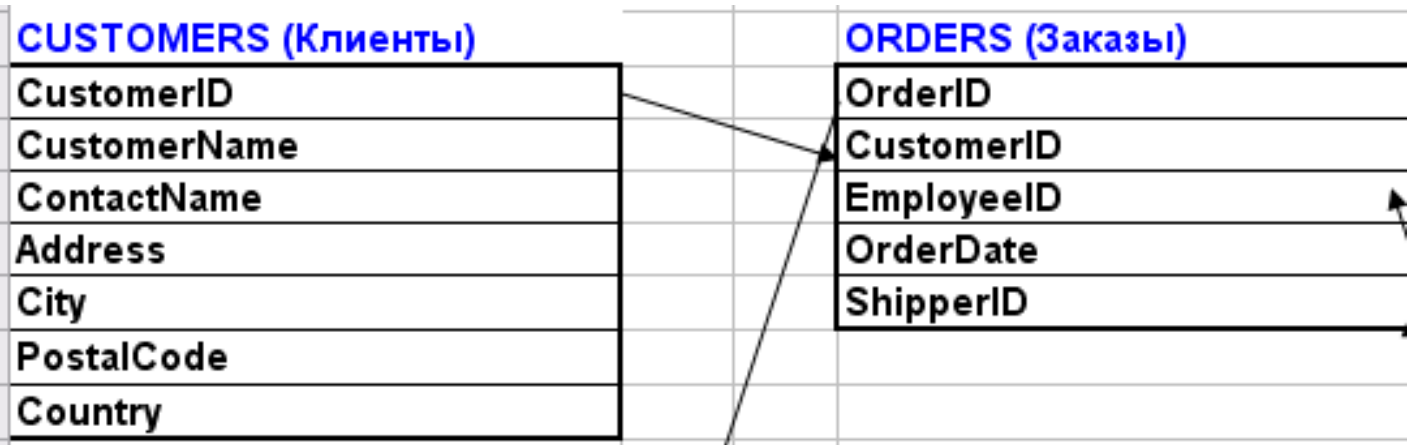




Задачи

1. Выбрать клиентов, которые делали заказы
Из таблицы Orders вывести поля: *OrderID, OrderDate*
Из таблицы Customers вывести поля: *CustomerName, ContactName*

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



INNER JOIN Syntax

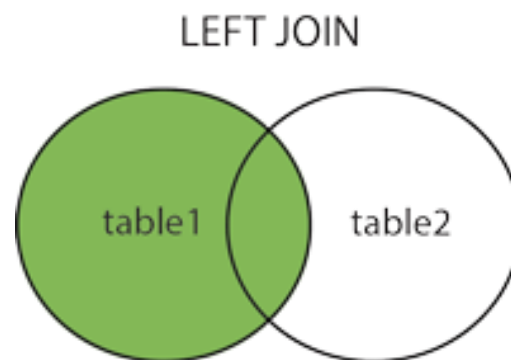
```
SELECT column_name(s)  
FROM table1  
INNER JOIN table2 ON table1.column_name = table2.column_name
```



LEFT JOIN

LEFT JOIN возвращает **ВСЕ** записи из левой таблицы (Table1), а также совпавшие записи из правой таблицы (Table2). В результате будет NULL с правой стороны, если нет совпадения.

В некоторых базах данных LEFT JOIN называется LEFT OUTER JOIN



LEFT JOIN Syntax

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




LEFT OUTER JOIN

LEFT JOIN возвращает все записи из левой таблицы, даже если нет совпадений в таблице справа





Задачи

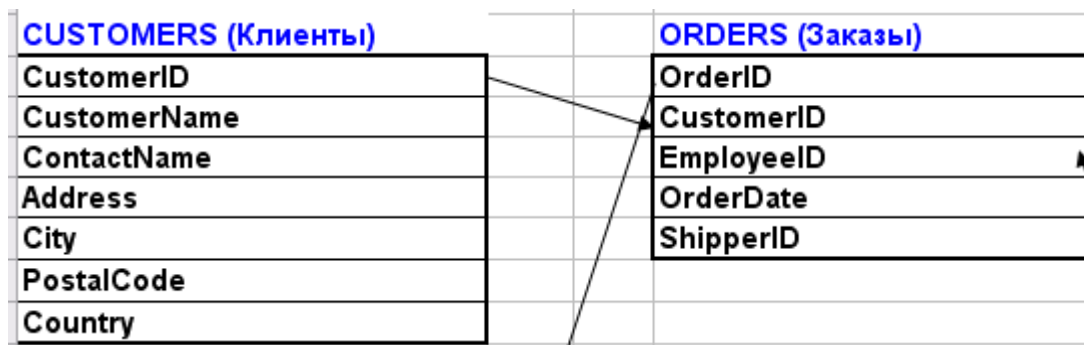
1. Вывести ВСЕХ клиентов (таблица Customers), и если клиент делал хоть 1 заказ, то отобразить детали заказа (таблица Orders).

Из таблицы Customers вывести поля: *CustomerName*, *ContactName*

Из таблица Orders вывести поля: *OrderID*, *OrderDate*

```
SELECT Orders.OrderID, Orders.OrderDate,  
Customers.CustomerName, Customers.ContactName  
FROM Customers
```

```
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
```



LEFT JOIN Syntax

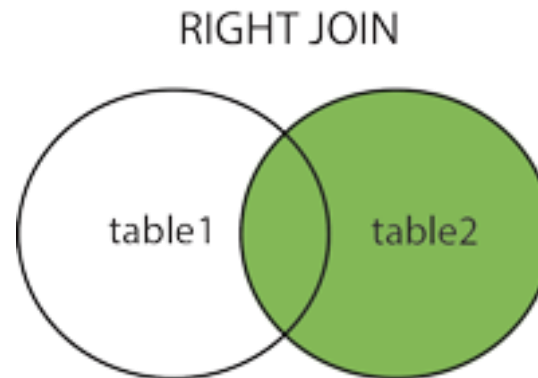
```
SELECT column_name(s)  
FROM table1
```

```
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

RIGHT JOIN

RIGHT JOIN возвращает все записи из правой таблицы (table2) и совпавшие записи из левой таблицы (table1). В результате отображается NULL с левой стороны, когда нет соответствия.

В некоторых базах данных RIGHT JOIN называется RIGHT OUTER JOIN



RIGHT JOIN Syntax

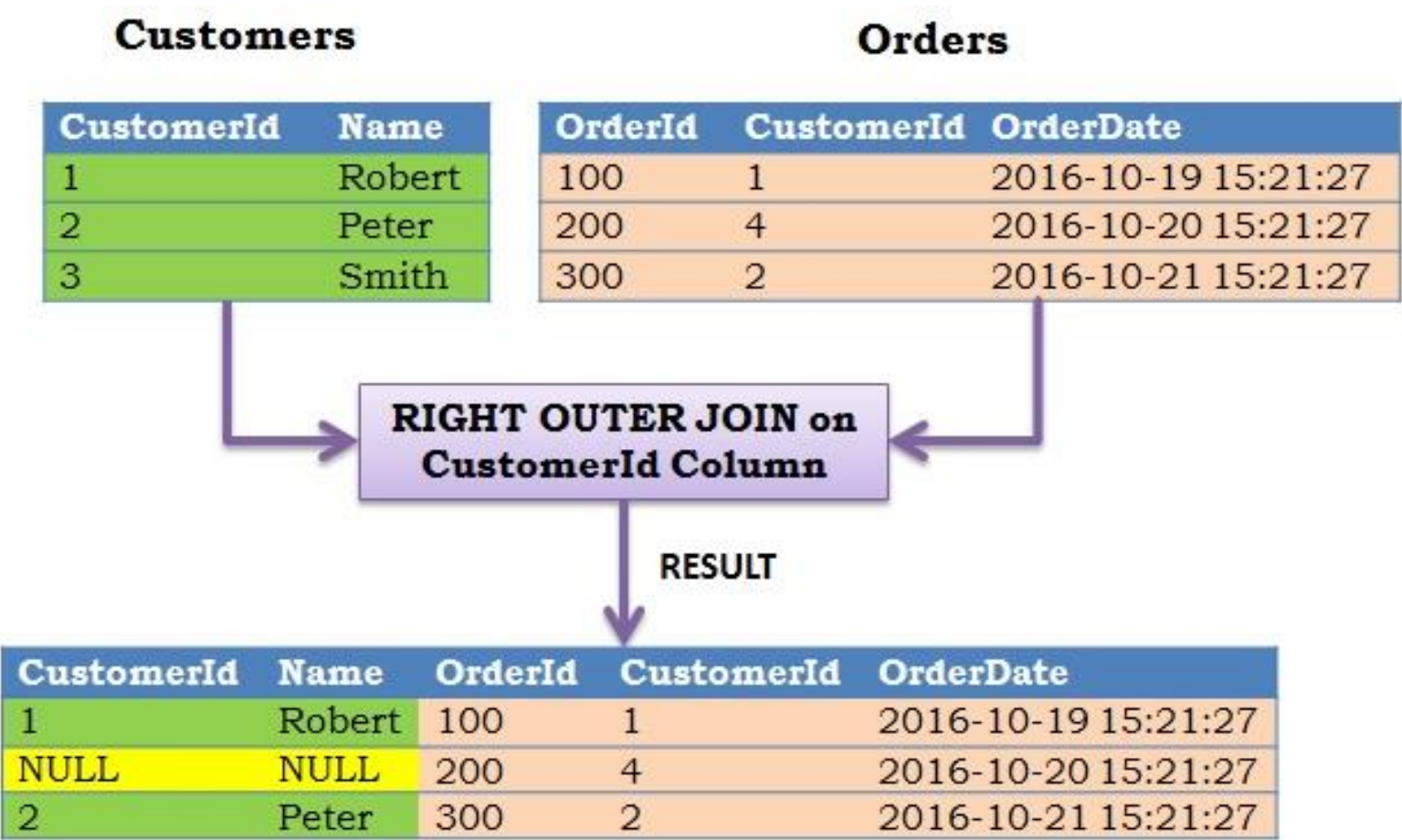
```
SELECT column_name(s)  
FROM table1
```

```
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```



RIGHT OUTER JOIN

RIGHT JOIN возвращает все записи из **правой** таблицы, даже если нет совпадений в таблице слева





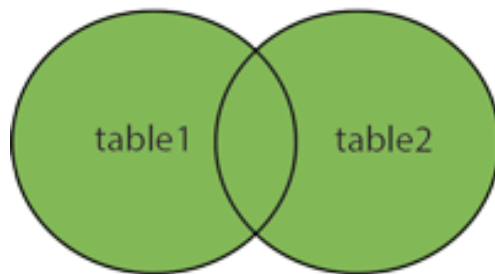
FULL JOIN

FULL JOIN возвращает строку из любой таблицы, когда условия выполняются, и возвращает нулевое значение, когда нет совпадения. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

Объединяет левое внешнее соединение (**LEFT JOIN**) и правое внешнее соединение (**RIGHT JOIN**)

FULL OUTER JOIN потенциально может вернуть очень большие результирующие наборы!

FULL OUTER JOIN



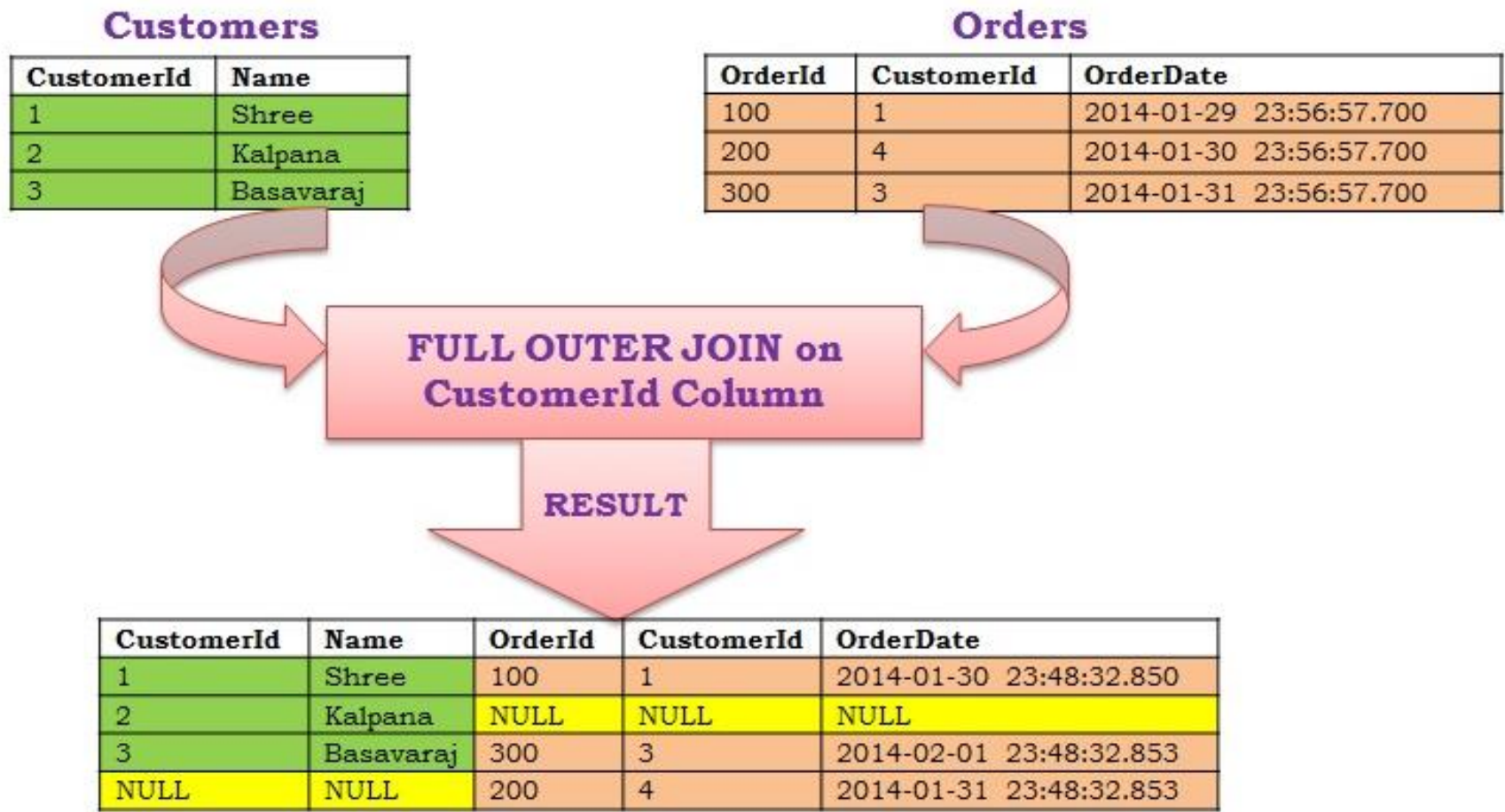
FULL OUTER JOIN Syntax

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



FULL OUTER JOIN

FULL OUTER JOIN возвращает **все** строки из левой таблицы *Customers* (Клиенты) и **все** строки из правой таблицы *Orders* (Заказы).

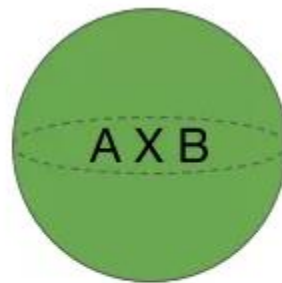




CROSS JOIN

CROSS JOIN – это декартово произведение. Результатом такого соединения будет сцепление каждой строки первой таблицы с каждой строкой второй таблицы.

Если в *Table1* содержится $N1$ записей, а в *Table2* – $N2$ записей, в результате будет $N1 \times N2$ записей.



CARTESIAN
(CROSS) JOIN

FULL OUTER JOIN Syntax

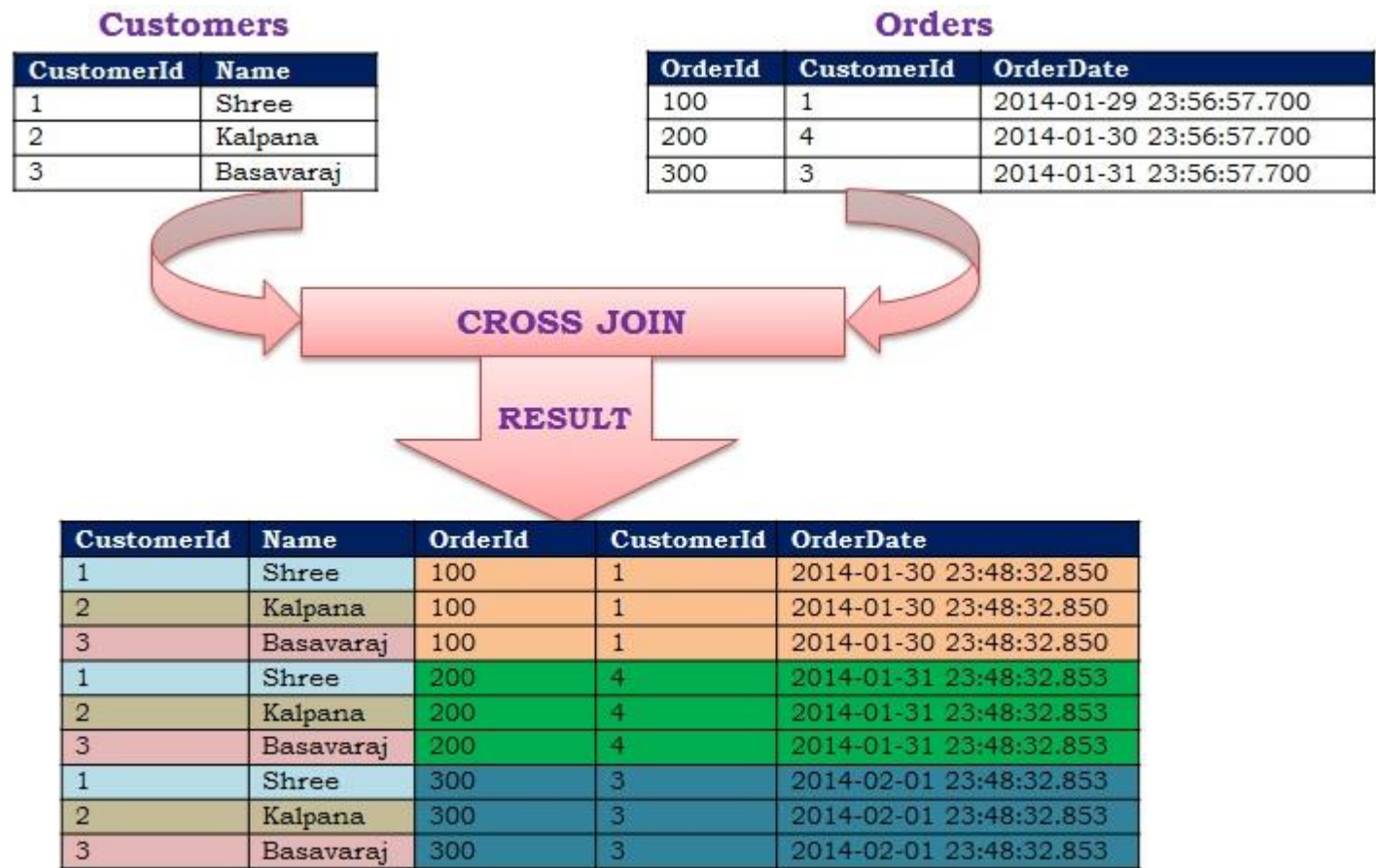
```
SELECT column_name(s)  
FROM table1  
CROSS JOIN table2
```

```
SELECT column_name(s)  
FROM table1, table2
```




CROSS JOIN

Результатом такого соединения будет сцепление каждой строки первой таблицы с каждой строкой второй таблицы.





GROUP BY

Выражение **GROUP BY** используется для разбиения данных на группы с одинаковыми значениями в заданном столбце, к которым могут применяться агрегатные функции (**COUNT**, **MIN**, **MAX**, **AVG** и **SUM**).

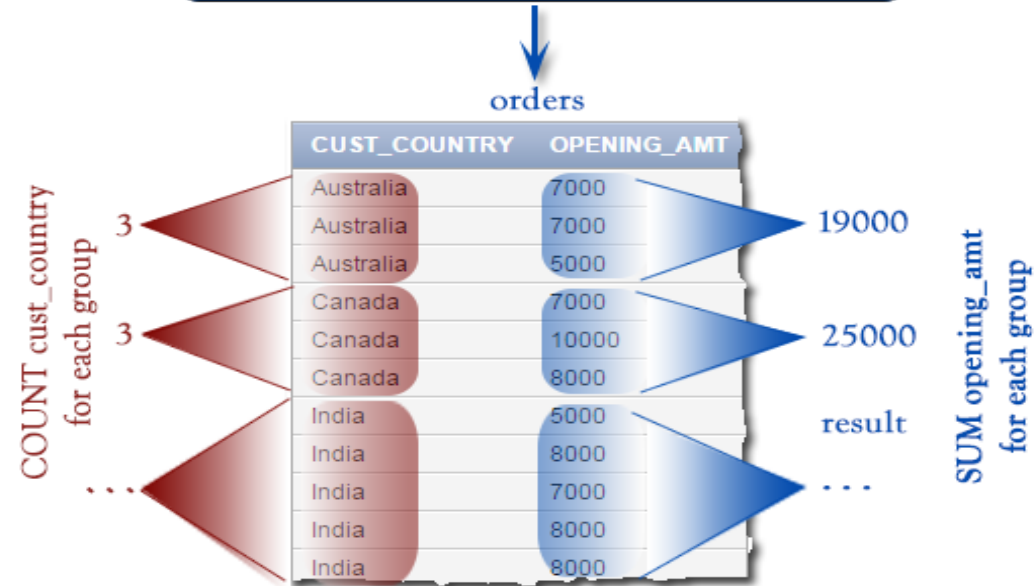
Агрегатные функции (**COUNT**, **MIN**, **MAX**, **AVG** и **SUM**) используются для подсчета значений по каждой группе (или по всей выборке, если GROUP BY не задано)

GROUP BY Syntax

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

GROUP BY

```
SELECT cust_country, SUM(opening_amt),  
COUNT(cust_country)  
FROM customer  
GROUP BY cust_country;
```



CUST_COUNTRY	SUM(OPENING_AMT)	COUNT(CUST_COUNTRY)
India	73000	10
USA	18000	4
Australia	19000	3
Canada	25000	3
UK	26000	5



Задачи

- 1. Посчитать количество клиентов (таблица *Customers*) в каждой стране, результаты отсортировать по убыванию значений столбца с количеством клиентов

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

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



HAVING

HAVING позволяет фильтровать результаты группировки, сделанной с помощью команды GROUP BY.

WHERE - используют относительно **всех выбираемых данных**

HAVING - только по отношению **к группам**, определенным в параметре GROUP BY

- используется только в связке с параметром GROUP BY, указывается сразу же после него и перед ORDER BY.
- в условии этого параметра можно использовать только агрегатные функции и поля, указанные в параметре GROUP BY.

HAVING Syntax

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



Задачи

1. Вычислить количество клиентов из каждой страны, показать страны, где зарегистрировано больше 5 клиентов

```
SELECT Country, COUNT(Country)
FROM Customers
GROUP BY Country
HAVING COUNT(Country) > 5
```

Country	Count(country)
Brazil	9
France	11
Germany	11
UK	7
USA	13

CUSTOMERS (Клиенты)	
CustomerID	
CustomerName	
ContactName	
Address	
City	
PostalCode	
Country	



UPDATE

UPDATE используется для изменения существующих записей в таблице.

Будьте осторожны при обновлении записей в таблице! Обратите внимание на пункт WHERE в операторе UPDATE. Предложение WHERE определяет, какая запись/записи, которые должны быть обновлены. Если опустить предложение WHERE, все записи в таблице будут обновлены!

UPDATE Syntax

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



Задачи

- 1. Изменить поле *City* клиента, у которого ключевое поле *CustomerID*=85 на город= 'Париж'

```
UPDATE Customers SET City='Париж'  
WHERE CustomerID=85
```

CUSTOMERS (Клиенты)

CustomerID
CustomerName
ContactName
Address
City
PostalCode
Country



DELETE

Будьте осторожны при удалении записей в таблице!
Предложение WHERE определяет , какие записи должны быть удалены. Если опустить предложение WHERE, все записи в таблице будут удалены!

DELETE Syntax

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

Пример:

Следующая конструкция удаляет клиента «Alfreds» из таблицы Customers:

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




INSERT INTO

Добавляет строку в таблицу

Syntax:

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

Пример:

```
INSERT INTO Customers  
(CustomerName, ContactName, Address, City, PostalCode)  
VALUES ('ИП Петров', 'Петров А', 'Успешная 51', 'Омск', '644006')
```



Рекомендуемые сайты по SQL

- <http://www.sql-ex.ru/>
- <https://www.codecademy.com/>
- <https://www.udemy.com/>
- <https://www.w3schools.com/>
- www.sql-tutorial.ru



Рекомендуемые книги по SQL

Алан Бьюли «Изучаем SQL»

Мартин Грубер «Понимание SQL»

Крис Фиайли «SQL»

Тейлор Аллен Г. «SQL для «чайников»



Райффайзен
БАНК

Желаем успехов!