SQL vs NoSQL

SQL databases are table based databases whereas
NoSQL databases can be document based, key-value pairs, graph databases.
SQL databases are vertically scalable while
NoSQL databases are horizontally scalable.
SQL databases have a predefined schema whereas
NoSQL databases use dynamic schema for unstructured data**.**
NoSQL - does not use SQL, no query language is declared

Tutorial https://www.w3schools.com/sql/

Quiz - https://sqlbolt.com/

Some of The Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

```
SELECT * FROM Customers;
// * selects all the records from the Customers table
```

SELECT - select Two Columns from data from a table of a database

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

```
SELECT CustomerName, City FROM Customers;
// selects the "CustomerName" and "City" columns from the "Customers" table
```

```
SELECT DISTINCT Country FROM Customers;
// selects only the DISTINCT values from the "Country" column in the "Customers" table:
```

```
SELECT COUNT(DISTINCT Country) FROM Customers;
// lists the number of different (distinct) customer countries
```

```
SELECT * FROM Customers
WHERE CustomerID=1;
// where is condition - returns all info from ID=1
```

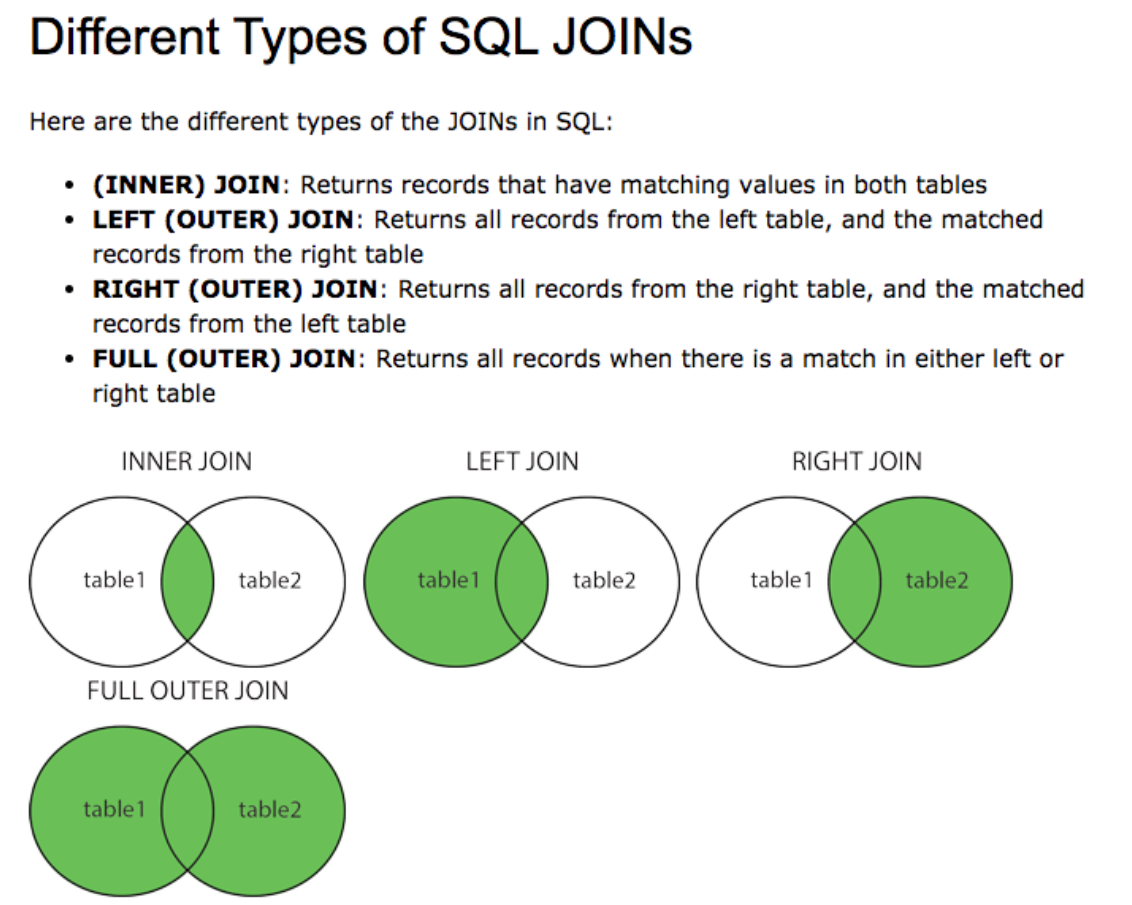
```
AND OR NOT // used for && || !

SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;

SELECT * FROM Customers
ORDER BY Country DESC;

SELECT * FROM Customers
ORDER BY Country, CustomerName;
// orders by Country, but if some rows have the same Country, it orders them by CustomerName

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

SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
// show all non null address values

UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

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

SELECT MIN(Price) AS SmallestPrice
FROM Products;
// MIN MAX

SELECT SUM(column_name)
FROM table_name
WHERE condition;
// SUM, AVG, COUNT

SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
// all customers with a ContactName that starts with "a" and ends with "o":

SELECT CustomerName FROM Customers
```

WHERE Country NOT IN ('Germany', 'France', 'UK');
// selects all CustomerName from Customers, where NOT IN countries: Germ, Fra, UK

SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
// The BETWEEN operator selects values within a given range

JOINS

// A JOIN clause is used to combine rows from two or more tables, based on a related column between them.



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);
// selects all orders with customer and shipper information

LEFT JOIN - (SEE PIC) returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

SELECT Customers.CustomerName, Orders.OrderID

```
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
// select all customers, and any orders they might have

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

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

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;
//SELF JOIN - self JOIN is a regular join, but the table is joined with itself.
```

Unions

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
// UNION order ascending by city

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

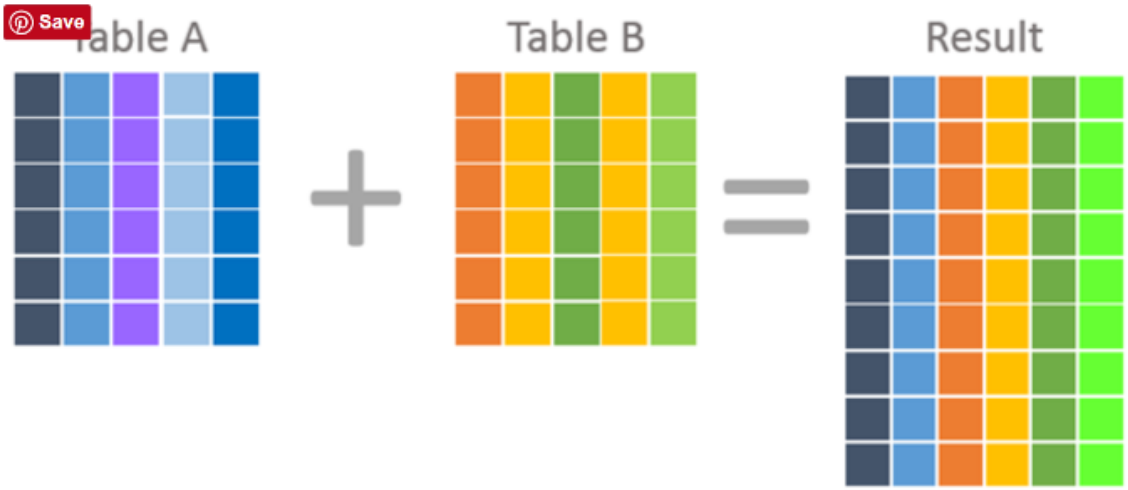
The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

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

ORDER BY COUNT(CustomerID) DESC;
// lists the number of customers in each country, sorted high to low:

JOIN vs UNION

The **JOIN** clause combines the attributes of two relations to form the resultant tuples whereas, **UNION** clause combines the result of two queries. **JOIN**-Combine into new columns. **Union**-Combine into new rows

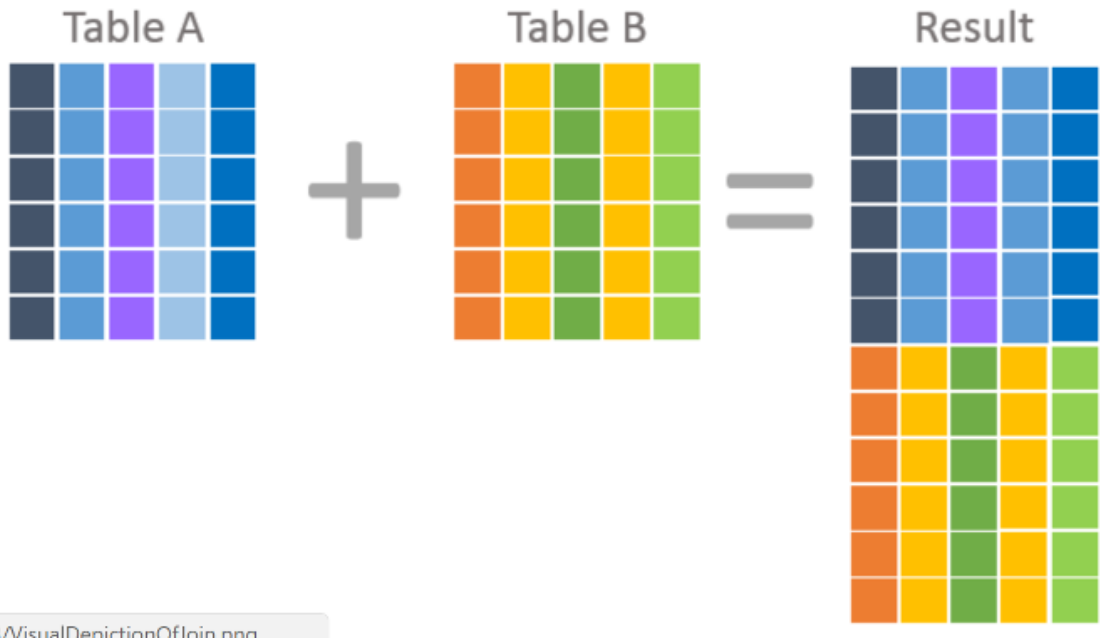


Joins Combine Columns

Each row in the result contains columns from BOTH table A and B. Rows are created when columns from one table match columns from another. This match is called the join condition.

This makes joins really great for looking up values and including them in results. This is usually the result of denormalizing (reversing normalization) and involves using the foreign key in one table to look up column values by using the primary key in another.

Now compare the above depiction with that of a union. In a union each row within the result is from one table OR the other. In a union, columns aren't combined to create results, rows are combined.



/2015/08/VisualDepictionOfJoin.png

https://www.chegg.com/homework-help/questions-and-answers/answer-given-question-difference-union-join-sql-hard-time-understanding-answer-could-pleas-q29653366