SQL (Standard Language for Storing)

```
SELECT * FROM Customers;    //Print ths Entier DataBase Named "Customer"
```

- MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems are Subparts of SQL.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"), and contain records (rows) with data.

- SQL keywords are NOT case sensitive: `select` is the same as `SELECT`

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

SQL SELECT Statement:-

The `SELECT` statement is used to select data from a database.

Syntax:

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

Here, column1, column2, ... are the *field names* of the table you want to select data from.The table_name represents the name of the *table* you want to select data from.

Example:

```
SELECT CustomerName, City FROM Customers;
```

Select ALL columns

To return all columns, without specifying every column name, you can use the `SELECT *` syntax:

```
SELECT * FROM Customers;
```

SQL SELECT DISTINCT Statement:-

`SELECT DISTINCT` statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax

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

SQL WHERE Clause:-

The `WHERE` clause is used to filter records.It is used to extract only those records that fulfill a specified condition.

Syntax

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

Example:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

SQL requires single quotes around text values (most database systems will also allow double quotes).However, numeric fields should not be enclosed in quotes:

Operators in The WHERE Clause:

=	Equal	
>	Greater than	
<	Less than	
>=	Greater than or equal	
<=	Less than or equal	
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=	
BETWEEN	Between a certain range	
LIKE	Search for a pattern	
IN	To specify multiple possible values for a column	

Example For Each Operator:

```
SELECT * FROM Customers
WHERE customer_id = 1

SELECT * FROM Customers
WHERE customer_id > 1

SELECT * FROM Customers
WHERE customer_id < 1

SELECT * FROM Customers
WHERE customer_id >= 1

SELECT * FROM Customers
WHERE customer_id <= 1

SELECT * FROM Customers
WHERE customer_id <> 1    //Customer ID=1 will not be present in Output

SELECT * FROM Customers
WHERE customer_id BETWEEN 1 AND 5
```

```
SELECT * FROM Customers
WHERE customer_name LIKE 's%' //Output with 's' present in name

SELECT * FROM Customers
WHERE City IN ('Paris','London'); //Output Rows with only Paris&London in City Colomn
```

SQL ORDER BY:-

The `ORDER BY` keyword is used to sort the result-set in ascending or descending order.

Syntax:

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

Example:

```
SELECT * FROM Products
ORDER BY Price ASC;  //Not mentioning ASC|DESC with take ASC by Default
```

ORDER BY Several Columns:

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

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName.

Using Both ASC and DESC:

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

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column.

SQL AND Operator:-

The `AND` operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G'.

The `WHERE` clause can contain one or many `AND` operators.

Syntax

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

AND vs OR

`AND` operator displays a record if *all* the conditions are TRUE.

`OR` operator displays a record if *any* of the conditions are TRUE.

Combining AND and OR:

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

```
SELECT * FROM Customers
WHERE Country = 'Spain'
AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

SQL OR Operator:-

The `OR` operator is used to filter records based on more than one condition

`WHERE` clause can contain one or more `OR` operators.

Syntax

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

SQL NOT Operator:-

The `NOT` operator is used in combination with other operators to give the opposite result, also called the negative result.

Syntax

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

Example:

```
SELECT * FROM Customers
WHERE NOT Country = 'Spain';
```

SQL INSERT INTO Statement:-

The `INSERT INTO` statement is used to insert new records in a table.

Syntax: INSERT INTO

It is possible to write the `INSERT INTO` statement in two ways:

1. Specify both the column names and the values to be inserted:

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

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the `INSERT INTO` syntax would be as follows:

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

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Insert Multiple Rows

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');
```

SQL NULL Values:-

A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

`IS NULL` and `IS NOT NULL` operators

Syntax : IS NULL

`IS NULL` operator is used to test for empty values (NULL values).

```
SELECT column_names FROM table_name
WHERE column_name IS NULL;  //column parameter which is null
```

Syntax : IS NOT NULL

`IS NOT NULL` operator is used to test for non-empty values (NOT NULL values).

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

SQL UPDATE Statement:-

The `UPDATE` statement is used to modify the existing records in a table.

Syntax:

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

when updating records in a table! Notice the `WHERE` clause in the `UPDATE` statement. The `WHERE` clause specifies which record(s) that should be updated. If you omit the `WHERE` clause, all records in the table will be updated!

Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

If you omit the `WHERE` clause, ALL records will be updated!

Example

```
UPDATE CustomersSET
ContactName='Juan';
```

SQL DELETE Statement:-

`DELETE` statement is used to delete existing records in a table.

Syntax:

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

when deleting records in a table! Notice the `WHERE` clause in the `DELETE` statement. The `WHERE` clause specifies which record(s) should be deleted. If you omit the `WHERE` clause, all records in the table will be deleted!

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

Delete All Records:

Delete all rows in a table without deleting the table

```
DELETE FROM table_name;
```

Delete a Table:

To delete the table completely, use the `DROP TABLE` statement:

Example:

```
DROP TABLE Customers;
```

SQL SELECT TOP Clause:-

The `SELECT TOP` clause is used to specify the number of records to return.

The `SELECT TOP` clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Example:

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

Not all database systems support the `SELECT TOP` clause. MySQL supports the `LIMIT` clause to select a limited number of records, while Oracle uses `FETCH FIRST n ROWS ONLY`

LIMIT (MySQL) :

```
SELECT * FROM Customers
LIMIT 3;
```

FETCH FIRST (Oracle) :

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

SQL TOP PERCENT (SQL Server/MS Access) :

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

ADD a WHERE CLAUSE:-

Example

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

SQL MIN() and MAX() Functions:-

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

Syntax:

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

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

Set Column Name (Alias):

When you use `MIN()` or `MAX()`, the returned column will be named `MIN(field)` or `MAX(field)` by default. To give the column a new name, use the `AS` keyword:

Example

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

SQL COUNT() Function:-

The `COUNT()` function returns the number of rows that matches a specified criterion.

Syntax :

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

Example:

```sql
SELECT COUNT(*)
FROM Products;
```

Ignore Duplicates:

If `DISTINCT` is specified, rows with the same value for the specified column will be counted as one.

Example:

```sql
SELECT COUNT(DISTINCT Price)
FROM Products;
```

Use an Alias:

```sql
SELECT COUNT(*) AS [number of records]
FROM Products;
```

SQL SUM() Function:

The `SUM()` function returns the total sum of a numeric column.

Syntax

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

Example

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

SQL AVG() Function:

The `AVG()` function returns the average value of a numeric column.

Syntax

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

Higher Than Average:

To list all records with a higher price than average, we can use the `AVG()` function in a sub query:

Example

Return all products with a higher price than the average price:

```
SELECT * FROM Products
WHERE price > (SELECT AVG(price) FROM Products);
```

SQL LIKE Operator:

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign `%` represents zero, one, or multiple characters
- The underscore sign `_` represents one, single character

Syntax

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

The _ Wildcard :

It can be any character or number, but each `_` represents one, and only one, character.

Example:

```
SELECT * FROM Customers
WHERE city LIKE 'L_nd__';
```

The % Wildcard :

The `%` wildcard represents any number of characters, even zero characters.

Example:

```
SELECT * FROM Customers
WHERE city LIKE '%L%';
```

To return records that starts with a specific letter or phrase, add the `%` at the end of the letter or phrase. //`LIKE 'La%'`

To return records that ends with a specific letter or phrase, add the `%` at the beginning of the letter or phrase. //`LIKE '%a'`

To return records that contains a specific letter or phrase, add the `%` both before and after the letter or phrase. //`LIKE '%or%'`

the [] Wildcard:

The `[]` wildcard returns a result if *any* of the characters inside gets a match.

Example :

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

Return all customers starting with either "b", "s", or "p".

the - Wildcard :

The `-` wildcard allows you to specify a range of characters inside the `[]` wildcard.

Example :

```
SELECT * FROM Customers
WHERE CustomerName LIKE '[a-f]%';
```

Return all customers starting with "a", "b", "c", "d", "e" or "f".

Without Wildcard :

If no wildcard is specified, the phrase has to have an exact match to return a result.

```sql
SELECT * FROM Customers
WHERE Country LIKE 'Spain';
```

SQL IN Operator :

The `IN` operator allows you to specify multiple values in a `WHERE` clause.

The `IN` operator is a shorthand for multiple `OR` conditions.

Syntax

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

Example

Return all customers from 'Germany', 'France', or 'UK'

```sql
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

NOT IN :

By using the `NOT` keyword in front of the `IN` operator, you return all records that are NOT any of the values in the list.

Example

Return all customers that are NOT from 'Germany', 'France', or 'UK':

```sql
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

IN (SELECT):

```sql
SELECT * FROM Customers
WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

NOT IN (SELECT) :

```sql
SELECT * FROM Customers
WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);
```

SQL BETWEEN Operator :

The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.

The `BETWEEN` operator is inclusive: begin and end values are included.

Syntax

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

NOT BETWEEN :

To display the products outside the range of the previous example, use `NOT BETWEEN`:

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

BETWEEN with IN :

Example :

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID IN (1,2,3);
```

```sql
SELECT * FROM Products
WHERE ProductName
BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"
ORDER BY ProductName;
```

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

SQL Aliases :

SQL aliases are used to give a table, or a column in a table, a temporary name.Aliases are often used to make column names more readable.

An alias is created with the `AS` keyword.

Syntax

When alias is used on column:

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

When alias is used on table:

```sql
SELECT column_name(s)
FROM table_name AS alias_name;
```

If you want your alias to contain one or more spaces, like "`My Great Products`", surround your alias with square brackets or double quotes.

```sql
SELECT ProductName AS [My Great Products]
FROM Products;
```

```sql
SELECT ProductName AS "My Great Products"
FROM Products;
```

Concatenate Columns

The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):

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

MySQL Example :

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

Oracle Example :

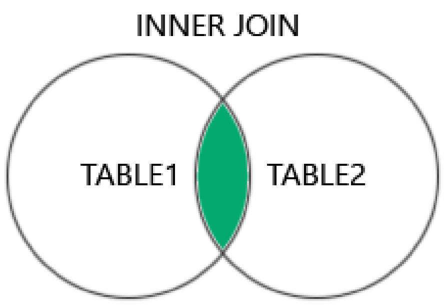
```sql
SELECT CustomerName, (Address || ', ' || PostalCode || ' ' || City || ', ' || Country)
AS Address
FROM Customers;
```

SQL JOIN :

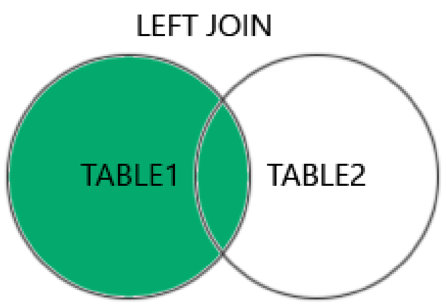
A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs :

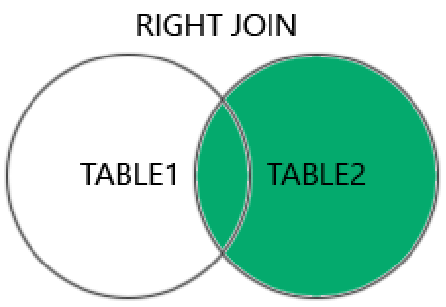
- `(INNER) JOIN`: Returns records that have matching values in both tables



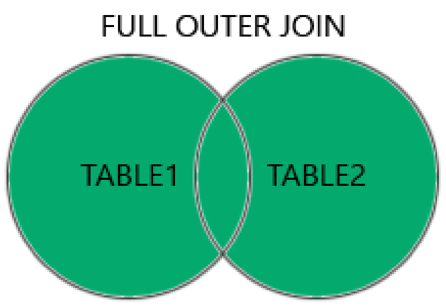
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table



- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table



- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



INNER JOIN :

The `INNER JOIN` keyword selects records that have matching values in both tables.

Syntax

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

The `INNER JOIN` keyword returns only rows with a match in both tables. Which means that if you have a product with no CategoryID, or with a CategoryID that is not present in the Categories table, that record would not be returned in the result.

Example

Join Products and Categories with the INNER JOIN keyword:

```
SELECT ProductID, ProductName, CategoryName
FROM ProductsINNER
JOIN Categories
ON Products.CategoryID = Categories.CategoryID;
```

`JOIN` and `INNER JOIN` will return the same result.

JOIN Three Tables

Example

```
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);
```

LEFT JOIN :

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

Syntax

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

RIGHT JOIN :

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

Syntax

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

In some databases `RIGHT JOIN` is called `RIGHT OUTER JOIN`.

FULL OUTER JOIN :

The `FULL OUTER JOIN` keyword returns all records when there is a match in left (table1) or right (table2) table records

Syntax

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

Example :

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

SQL Self Join :

A self join is a regular join, but the table is joined with itself.

Self Join Syntax :

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

Example :

```sql
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;
```

SQL UNION Operator :

The `UNION` operator is used to combine the result-set of two or more `SELECT` statements.

- Every `SELECT` statement within `UNION` must have the same number of columns
- The columns must also have similar data types
- The columns in every `SELECT` statement must also be in the same order

UNION Syntax :

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

The `UNION` operator selects only distinct values by default. To allow duplicate values, use `UNION ALL`

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

SQL UNION With WHERE :

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

SQL GROUP BY Statement :

The `GROUP BY` statement groups rows that have the same values into summary rows

The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.

Syntax :

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

SQL HAVING Clause :

The `HAVING` clause was added to SQL because the `WHERE` keyword cannot be used with aggregate functions.

Syntax :

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

Example :

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

SQL EXISTS Operator :

The `EXISTS` operator is used to test for the existence of any record in a subquery.

The `EXISTS` operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax :

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```