SQL

SQL : Standard language for accessing and manipulating database. It is an ANSI (American National Standards Institute) standard

RDBMS : Relational Database Management System.
It is basis for SQL and all modern database systems like MS SQL Server, IBM DB2, Oracle, MySql, MS Access.

A **FIELD** is the column in table that is designed to maintain a specific information about every record in table.

A database more often contains one or more tables. Each table is identified by a name(eg. "Customers" or "Orders"). Tables contain **records** (rows) with data.

Some of the most important **SQL commands** :

1) SELECT - Extracts data from database.
2) UPDATE - Updates data in database.
3) DELETE - Deletes data from database.
4) INSERT INTO - Inserts new data into database.
5) CREATE DATABASE - Creates new database.
6) ALTER DATABASE - Modifies a database.
7) CREATE TABLE - Creates new table.
8) ALTER TABLE - Modifies a table.
9) DROP TABLE - Deletes a table.
10)CREATE INDEX - Creates an index(search key).
11)DROP INDEX - Deletes an index.

SQL STATEMENTS

1) SELECT :

a) Simple SELECT :

The data returned is stored in a result table known as result-set.

Syntax : SELECT column1, column2, column3, ...
FROM table_name;

Here column1,column2,... are field names of the table you want to select data from.

If you want to select all the fields available in table use following syntax.

Syntax : SELECT *
FROM table_name;

b) SELECT DISTINCT :

SELECT DISTINCT statement is used to returned only distinct(unique and avoids dublicate) values.

Syntax : SELECT DISTINCT column1, column2, column3, ...
FROM table_name;

The following sql statement lists the number of distinct entries in the column.

Syntax : SELECT COUNT(DISTINCT column_name)
FROM table_name;

NOTE : ABOVE EXAMPLE WILL NOT WORK IN FIREFOX AND MICROSOFT EDGE.

For MS Access :
Syntax : SELECT COUNT (*) AS new_column_name
FROM (SELECT DISTINCT column_name
FROM table_name);

2) WHERE clause :

The WHERE clause is used to filter records.
The records which satisfies the condition will be extracted.

Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE condition;
eg. : SELECT *
FROM EmpDetails
WHERE Country="Mexico";

OPERATORS in WHERE clause :

=, <>, !=, <, >, <=, >=, BETWEEN, LIKE, IN
BETWEEN - Bwtween an inclusive range.
LIKE - Search for a pattern.
IN - To specifi multiple possible values for a column.

AND, OR, NOT operators :
AND Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE condition AND condition AND condition AND .....;

OR Syntax : SELECT column1, column2, column3, ...
FROM table_name

WHERE condition OR condition OR condition OR .....;

NOT Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE condition NOT condition NOT condition NOT .....;

3) ORDER BY keyword :

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

Syntax : SELECT column1, column2, column3, ...
FROM table_name
ORDER BY column1, column2, column3, ... ASC | DESC

ASC - Ascending order (by default)
DESC - Descending order

NOTE : for single column it will sort the data in single column in ascending order(by default)
for multiple columns it will firstly arrange the data in column1 in ascending order(by default) if it got some dublicate data in column1 then it will sort that entries containing dublicate data according to column2 and so on.....

4) INSERT INTO :

Used to insert new records in a row.

Syntax1 : This is for when you are NOT entering for all the columns(Data Fields)
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

Syntax2 : This is for when you are entering for all the columns(Data Fields)
INSERT INTO table_name
VALUES (value1, value2, value3, ...);

5) NULL Values :

A field with NULL value is a field with no value.

If a field in a table is optional, it is possible to insert new record or update a record without adding a vlaue to this field. Then the fiels will be saved with a NULL value.

NULL value cannot be test by comparison opertors. (<, >, =, !=, etc.)

IS NULL :

Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE column_name IS NULL;

IS NOT NULL :

Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE column_name IS NOT NULL;

(Tip : always use IS NULL for checking NULL values)

6) UPDATE :

Used to Modify existing records in the table.

Syntax : UPDATE table_name
SET column1 = value1, column2 = value2, column3 = value3, ...
WHERE condition;

Important to note the WHERE clause if you forgot to write the WHERE clause all records will be updated.

eg : UPDATE multiple records

```
UPDATE Customers
SET ContactName = 'John'
WHERE Country = 'Mexico';
```

7) DELETE :

Used to delete existing record in a table.

Syntax : DELETE FROM table_name
WHERE condition;

Important to note the WHERE clause if you forgot to write the WHERE clause all records will be deleted.

Delete all records :

Syntax1 : DELETE
FROM table_name;

Syntax2 : DELETE *
FROM table_name;

8) TOP, LIMIT(MySql), ROWNUM(Oracle) clause :

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

SQL Server / MS Access Syntax :

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

eg. :
```
SELECT TOP 3 *
FROM table_name;
```

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;
```

eg. :
```
SELECT *
From Customers
WHERE Country = "Germany" AND ROWNUM <= 3;
```

9) MIN(),MAX() :

MIN() returns smallest value of selected column.
MAX() returns biggest value of selected column.

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

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

10) COUNT(), AVG(), SUM() :

COUNT() returns the number of rows that matches the specified criteria.
AVG() returns the average value of numeric column.
SUM() returns the total sum of numeric column.

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;

11) LIKE operator :

Used in WHERE clause to search a specified pattern in column.

There are 2 wildcard used in conjunction with LIKE operator :

1) % - The percent sign represent 0,1 or multiple characters.

2) _ - The underscore represent the single character.

MS Access uses a question mark(?) insted of (_)

Both can be used in combines.

Syntax : SELECT column1, column2, column3, ...
FROM table_name
WHERE columnN LIKE pattern;

Tip : you can also combine any number of conditions using AND or OR operators.

eg. :

WHERE City LIKE 'a%' - Find any values that starts with 'a';

WHERE City LIKE '%a' - Find any values that ends with 'a';

WHERE City LIKE '%or%' - Find any values that have 'or' in any position;

WHERE City LIKE '_r%' - Find any values that have 'r' in second position;

WHERE City LIKE 'a_%_%' - Find any values that starts with 'a' and with at least 3 characters in length;

WHERE City LIKE 'a%o' - Find any values that starts with 'a' and ends with 'o';

WHERE City LIKE '[bsp]%' - Find any values with City starting with 'b','s' or 'p';

WHERE City LIKE '[a-c]%' - Find any values with City starting with 'a','b' or 'c';

WHERE City LIKE '[!bsp]%' - Find any values with City NOT starting with 'b','s' or 'p';

WHERE City NOT LIKE '[bsp]%' - Find any values with City NOT starting with 'b','s' or 'p';

12) IN operator :

Allows you to specify multiple values in WHERE clause, it is an shorthand to multiple OR conditions.

Syntax1 : SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, value3, ...)

eg. : SELECT *
FROM Customers
WHERE Country IN ("Germany", "France", "NY");

Syntax2 : SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, value3, ...);

eg. : SELECT *
FROM Customers
WHERE Country IN (SELECT Country
From Suppliers);

Above example will select all customers that are from same countries as suppliers.

13) BETWEEN operator :

For selecting values in given range. Values can be numbers, texts or dates.
The BETWEEN operator is inclusive i.e. begin and end values are included.

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

You can use "NOT BETWEEN" instead of "BETWEEN" for selecting all values except given range.

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

BETWEEN Text eg. :

```
SELECT *
FROM Products
WHERE ProductName BETWEEN 'Tiger' AND 'GoodDay' ORDER BY
```

Products;

BETWEEN Date eg. :

```
SELECT *
FROM Orders
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;
```

14) SQL Aliases :

Used to give a table or a column in a table, a Temporary Name.
Aliases are used to make column names more readable.
An Alias only exists for the duration of the query.

Alias Column Syntax :

```
SELECT column_name1 AS alias_name1, column_name2 AS alias_name2, ...
FROM table_name;
```

Alias Table Syntax :

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

NOTE : It requires double quotation marks or square brackets if the alias name contain spaces.

```
eg. : SELECT CustomerName AS Customers,
ContactNumber AS [Contact Person]
From Customers;
```

Following SQL statement creates an Alias named "Address" that combine 4 columns (Address, PostalCode, City, Country)

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

For MYSQL use the following instead of above statement.

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

Aliases can be useful when :

a) There are more table involve a query.
b) Functions are used in query.
c) Column names are big or not easily readable.
d) Two or more columns are combined together.

15) JOIN :

Used to combine rows from two or more tables, based on related columns between them.

Types : NOTE : Table1=LEFT TABLE, Table2=RIGHT TABLE

a) (INNER) JOIN : Return records that have matching values in both tables.
Table1 intersection Table2 (In terms of SETs)

b) LEFT (OUTER) JOIN : Return all records from the LEFT table and the matched records from RIGHT table.
Table1 - Table2 (In terms of SETs)

c) RIGHT (OUTER) JOIN : Return all records from the RIGHT table and the matched records from LEFT table.
Table2 - Table1 (In terms of SETs)

d) FULL (OUTER) JOIN : Return all records either matched in LEFT or RIGHT table.
Table1 union Table2 (In terms of SETs)

a) INNER JOIN :

Syntax : SELECT table1.column_name, table2.column_name, ...
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;

INNER JOIN on 3 tables :

SELECT table1.column_name, table2.column_name, table3.column_name, ...
FROM ((table1 INNER JOIN table2 ON table1.col = table2.col)
INNER JOIN table3 ON table1.col = table3.col);

b) LEFT JOIN :

Syntax : SELECT table1.column_name, table2.column_name, ...
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;

The result is "null" from RIGHT side if there is no match.

c) RIGHT JOIN :

Syntax : SELECT table1.column_name, table2.column_name, ...
FROM table1

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

The result is "null" from LEFT side if there is no match.

d) FULL OUTER JOIN :

Syntax : SELECT table1.column_name, table2.column_name, ...
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;

16) SELF JOIN : why?

table is joined with itself.

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

17) UNION operator :

used to combine the result-set of two or more SELECT statements.

a) Each SELECT statement within UNION must have the same number of columns.

b) The columns must have the similar data types.

c) The coolumns in each SELECT statement must also be in the same ORDER.

Syntax : SELECT column_name(s)
FROM table1
UNION
SELECT column_name(s)
FROM table2;

UNION operator selects only distinct values by default to allow dublicate entries use UNION ALL

Syntax : SELECT column_name(s)
FROM table1
UNION ALL
SELECT column_name(s)
FROM table2;

NOTE : The column names in the result-set are usually equal to the column names in the first
SELECT statement in the UNION.

UNION with WHERE eg. : SELECT column_name(s)

FROM table1
WHERE condition

UNION

SELECT column_name(s)
FROM table2
WHERE condition;

UNION ALL with WHERE eg. : SELECT column_name(s)
FROM table1
WHERE condition

UNION ALL

SELECT column_name(s)
FROM table2
WHERE condition;

18) GROUP BY :

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)
GROUP BY column_name(s);

19) HAVING clause :

Added to SQL because WHERE keyword could not be used with aggregate functions (COUNT, MAX, MIN, SUM, AVG)

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

eg. : SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;

20) EXISTS operator :

Used to test foe the existence of any record in a subquery.

EXISTS Operator returns true iif subquery returns 1 or more records.

Syntax : SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name
FROM table_name
WHERE condition);

21) ANY and ALL operators :

Used with WHERE or HAVING clause.

ANY operator returns TRUE if any subquery values meet the condition.

ALL operator returns TRUE if all subquery values meet the condition.

ANY Syntax : SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY (SELECT column_name

FROM table_name

WHERE condition);

ALL Syntax : SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL (SELECT column_name

FROM table_name

WHERE condition);

operator must be standard comparison operators(=, <>, !=, <, <=, >, >=)

22) INTO :

Used to copy data from 1 table to another table.

Syntax : SELECT column1, column2, ...
INTO newtable [IN externalDB]
FROM oldtable
WHERE condition;

copy data from more than 1 table into new table.

```
SELECT table1.col, table2.column2
INTO newtable
FROM oldtable
JOIN table2 ON table1.col = table2.col;
```

Tip : SELECT INTO can also be used to create a new empty table using the schema of another.

Just add a WHERE clause that causes the query to return no data.

```
SELECT *
INTO newtable
WHERE 1=0;
```

23) INSERT INTO SELECT :

Used to copy data from one table and inserts into another table.

a) INSERT INTO SELECT requires that data types in source and target tables match.

b) The existing records in the target table are unaffected.

```
Syntax1 : INSERT INTO table2
                SELECT *
                FROM table1
                WHERE condition;

Syntax2 : INSERT INTO (column1, column2, ....)
                SELECT (column1, column2, ....)
                FROM table1
                WHERE condition;
```

24) IFNULL(), ISNULL(), COALESCE(), NVL() :

All functions will replace any NULL values with any other value.

```
MySql Syntax1 : SELECT ProductName, UnitPrice *
                        (UnitsInBlock + IFNULL(UnitsOnOrder, 0))
                FROM Products;

MySql Syntax2 : SELECT ProductName, UnitPrice *
                        (UnitsInBlock + COALESCE(UnitsOnOrder, 0))
                FROM Products;

Sql Server    : SELECT ProductName, UnitPrice *
                        (UnitsInBlock + ISNULL(UnitsOnOrder, 0))
                FROM Products;

MS Access     : SELECT ProductName, UnitPrice *
                        (UnitsInBlock + IIF(IFNULL(UnitsOnOrder), 0,
UnitsOnOrder))
                FROM Products;
```

```
Oracle      : SELECT ProductName, UnitPrice *
                     (UnitsInBlock + NVL(UnitsOnOrder, 0))
              FROM Products;
```