**How to insert data in a table-**

INSERT INTO <Table Name>

(<Column List>) VALUES (<Values>)

Example:

INSERT INTO Orders

(FirstName, LastName, OrderDate) VALUES

('John', 'Smith', '10/10/2010')

**SQL SELECT statement**

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

Syntax-

SELECT \* FROM table\_name;

It collets all data.

We can also write it as-

SELECT column\_name,column\_name  
FROM table\_name;

It collects particular columns.

**How to select data from a table**

SELECT <Column List>

FROM <Table Name>

WHERE <Search Condition>

Example:

SELECT FirstName, LastName, OrderDate

FROM Orders WHERE OrderDate > '10/10/2010'

**How to select data from more than one table**

SELECT <Column List>

FROM <Table1> JOIN <Table2>

ON <Table1>.<Column1> = <Table2>.<Column1>

Example:

SELECT Orders.LastName, Countries.CountryName

FROM Orders JOIN Countries ON

Orders.CountryID = Countries.ID

**How to update data in a table**

UPDATE <Table Name>

SET <Column1> = <Value1>, <Column2> = <Value2>, …

WHERE <Search Condition>

Example:

UPDATE Orders

SET FirstName = 'John', LastName = 'Who' WHERE LastName='Wo'

**How to delete data from a table**

DELETE FROM <Table Name>

WHERE <Search Condition>

Example:

DELETE FROM Orders

WHERE OrderDate < '10/10/2010'

**How to group data and use aggregates**

SELECT <Column List>, <Aggregate Function>(<Column Name>)

FROM <Table Name>

WHERE <Search Condition>

GROUP BY <Column List>

Example:

SELECT LastName, SUM(OrderValue)

FROM Orders

WHERE OrderDate > '10/10/2010'

GROUP BY LastName

**How to order data**

You can use a column alias in the ORDER BY clause for sorting.

SELECT <Column List>

FROM <Table Name>

WHERE <Search Condition>

ORDER BY <Column List>

Example:

SELECT FirstName, LastName, OrderDate

FROM Orders

WHERE OrderDate > '10/10/2010'

ORDER BY OrderDate

**Using UNION**

Union, Intersect or Minus operators are called set operators.

SELECT <Column List> FROM <Table1>

UNION

SELECT <Column List> FROM <Table2>

Example:

SELECT FirstName, LastName FROM Orders2010

UNION

SELECT FirstName, LastName FROM Orders2011

**SQL SELECT DISTINCT Statement**

The SELECT DISTINCT statement is used to return only different values.

Syntax-

SELECT DISTINCT column\_name,column\_name  
FROM table\_name;

Example-

Select distinct city from customers;

**SQL where clause-**

The WHERE clause is used to filter records.

Syntax-

SELECT column\_name,column\_name  
FROM table\_name  
WHERE column\_name operator value;

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

There are many operators that we also use here-

= Equal

!= Not equal. Note: In some versions of SQL this operator may be written as !=

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

BETWEEN Between an inclusive range

LIKE Search for a pattern

IN To specify multiple possible values for a column

**Text Fields vs. Numeric Fields**

SQL requires single quotes around text values and in numeric fields no quotes are required.

SELECT \* FROM Customers  
WHERE CustomerID=1;

**The SQL AND & OR Operators-**

The AND operator displays a record if both the first condition AND the second condition are true.

The OR operator displays a record if either the first condition OR the second condition is true.

e.g

SELECT \* FROM Customers  
WHERE Country='Germany'  
AND (City='Berlin' OR City='München');

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

**SQL ORDER By command-**

The ORDER BY keyword is used to sort the result-set. In this it set the data in ascending order.

Syntax-

SELECT column\_name, column\_name  
FROM table\_name  
ORDER BY column\_name;

SELECT \* FROM Customers  
ORDER BY Country;

It give all country names in ascending order.

**ORDER BY DESC -**

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

SELECT \* FROM Customers  
ORDER BY Country DESC;

This give results in descending order.

ORDER BY Several Columns Example

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

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

**Desc order-**

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

**SQL INSERT INTO statement-**

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

The first form does not specify the column names where the data will be inserted, only their values:

INSERT INTO table\_name  
VALUES (value1,value2,value3,...);

The second form specifies both the column names and the values to be inserted:

INSERT INTO table\_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);

e.g

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

We can also insert it directly-

INSERT INTO Customers VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

**UPDATE statement-**

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

UPDATE table\_name  
SET column1=value1,column2=value2,...  
WHERE some\_column=some\_value;

Example-

UPDATE Customers  
SET ContactName='Alfred Schmidt', City='Hamburg'  
WHERE CustomerName='Alfreds Futterkiste';

**SQL DELETE Statement**

The DELETE statement is used to delete records in a table.

SQL DELETE Syntax

DELETE FROM table\_name  
WHERE some\_column=some\_value;

Example-

DELETE FROM Customers  
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

We can also delete all records by using this command-

DELETE FROM table\_name;  
or  
DELETE \* FROM table\_name

**SQL SELECT TOP Clause-**

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

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

SQL SELECT TOP Equivalent in MySQL and Oracle

MySQL Syntax

SELECT column\_name(s)  
FROM table\_name  
LIMIT number;

Example

SELECT \*  
FROM Persons  
LIMIT 5;

Oracle Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE ROWNUM <= number;

Example

SELECT \*  
FROM Persons  
WHERE ROWNUM <=5;

**SQL LIKE operator-**

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

SQL LIKE Syntax

SELECT column\_name(s)  
FROM table\_name

WHERE column\_name LIKE pattern;

Example-

SELECT \* FROM Customers  
WHERE City LIKE 's%';

IN THIS IT GIVE the string that is started with s.

In this it give string that is end with s.

SELECT \* FROM Customers

WHERE City LIKE '%s';

The following SQL statement selects all customers with a Country containing the pattern "land":

SELECT \* FROM Customers  
WHERE Country LIKE '%land%';

Using the NOT keyword allows you to select records that do NOT match the pattern.

The following SQL statement selects all customers with Country NOT containing the pattern "land":

SELECT \* FROM Customers  
WHERE Country NOT LIKE '%land%';

**SQL Wildcard Property-**

In this we select any string from table.

Example-

SELECT \* FROM Customers  
WHERE City LIKE '\_erlin';

By this command it selects all customers with a City starting with any character, followed by "erlin".

Next is

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

This SQL statement selects all customers with a City starting with "b", "s", or "p":

We can also use this

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

It selects all customers with a City starting with "a", "b", or "c".

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';  
or  
SELECT \* FROM Customers  
WHERE City NOT LIKE '[bsp]%';

This SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

SQL IN Operator-

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

SQL IN Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1,value2,...);

SELECT \* FROM Customers  
WHERE City IN ('Paris','London');

SQL BETWEEN Operator-

The BETWEEN operator is used to select values within a range.The values can be numbers, text, or dates.

SQL BETWEEN Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

Example-

The following SQL statement selects all products with a price BETWEEN 10 and 20:

Example

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20;

**NOT BETWEEN Operator Example-**

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

Example

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

**BETWEEN Operator with IN Example-**

The following SQL statement selects all products with a price BETWEEN 10 and 20, but products with a CategoryID of 1,2, or 3 should not be displayed:

Example-

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

**BETWEEN Operator with Text Value Example-**

The following SQL statement selects all products with a ProductName beginning with any of the letter BETWEEN 'C' and 'M':

Example

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'C' AND 'M';

**NOT BETWEEN Operator with Text Value Example-**

The following SQL statement selects all products with a ProductName beginning with any of the letter NOT BETWEEN 'C' and 'M':

Example

SELECT \* FROM Products  
WHERE ProductName NOT BETWEEN 'C' AND 'M';

**SQL Aliases-**

SQL aliases are used to temporarily rename a table or a column heading.

Syntax-

SELECT column\_name AS alias\_name  
FROM table\_name;

SQL Alias Syntax for Tables

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

Example-

SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
FROM Customers;

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

In this it create temporary column Address that contains a address,city,postal code,country.

**SQL Joins-**

IN this operation we join the two tables.In these two table atleast one column name is need to be same.

There are four types if join-

INNER JOIN: Returns all rows when there is at least one match in BOTH tables

LEFT JOIN: Return all rows from the left table, and the matched rows from the right table

RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table

FULL JOIN: Return all rows when there is a match in ONE of the tables

Example-

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

\*\*

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
INNER JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

Second is SQL LEFT JOIN Keyword-

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

Syntax-

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
LEFT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

Example-

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword-

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

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

SQL FULL OUTER JOIN Keyword-

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

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

**The SQL UNION operator-**

The SQL UNION operator combines the result of two or more SELECT statements.

Syntax-

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

It select only distinct values to select all records use union all operator.

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

Example1-

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

Example2-

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

Example3-

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

SQL SELECT INTO Statement-

With SQL, you can copy information from one table into another.

Syntax-

SELECT \*  
INTO newtable [IN externaldb]  
FROM table1;

Example-

SELECT \*  
INTO CustomersBackup  
FROM Customers;

We can also copy particular columns that we need-

SELECT CustomerName, ContactName  
INTO CustomersBackup  
FROM Customers;

SQL INSERT INTO SELECT Statement-

The INSERT INTO SELECT statement copies data from one table and inserts it into an existing table.

Syntax-

INSERT INTO table2  
SELECT \* FROM table1;

Example1-

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers;

Example2-

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers  
WHERE Country='Germany';

**SQL CREATE DATABASE Statement-**

The CREATE DATABASE statement is used to create a database.

CREATE DATABASE dbname;

SQL CREATE TABLE Statement-

The CREATE TABLE statement is used to create a table in a database.

Syntax-

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size),  
column\_name2 data\_type(size),  
column\_name3 data\_type(size),  
....  
);

Example-

CREATE TABLE Persons  
(  
PersonID int,  
LastName varchar(255),  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
);

**SQL NOT NULL Constraint-**

The NOT NULL constraint enforces a column to NOT accept NULL values.

Example-

CREATE TABLE PersonsNotNull  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

**SQL UNIQUE Constraint-**

The UNIQUE constraint uniquely identifies each record in a database table.

Example-

CREATE TABLE Persons  
(  
P\_Id int NOT NULL UNIQUE,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

**SQL PRIMARY KEY Constraint-**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain UNIQUE values.

A primary key column cannot contain NULL values.

Most tables should have a primary key, and each table can have only ONE primary key.

Example-

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
PRIMARY KEY (P\_Id)  
)

SQL FOREIGN KEY Constraint-

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

P\_Id LastName FirstName Address City

1 Hansen Ola Timoteivn 10 Sandnes

2 Svendson Tove Borgvn 23 Sandnes

3 Pettersen Kari Storgt 20 Stavanger

The "Orders" table:

O\_Id OrderNo P\_Id

1 77895 3

2 44678 3

3 22456 2

4 24562 1

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
FOREIGN KEY (P\_Id) REFERENCES Persons(P\_Id)  
)

**SQL CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CHECK (P\_Id>0)  
)

**SQL DROP INDEX, DROP TABLE, and DROP DATABASE-**

Indexes, tables, and databases can easily be deleted/removed with the DROP statement.

The DROP TABLE Statement-

DROP TABLE table\_name

The DROP DATABASE Statement

The DROP DATABASE statement is used to delete a database.

DROP DATABASE database\_name

**The TRUNCATE TABLE Statement**

If we only want to delete the data inside the table, and not the table itself.

TRUNCATE TABLE table\_name

**The ALTER TABLE Statement**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

Syntax-

To add a column in a table, use the following syntax:

ALTER TABLE table\_name  
ADD column\_name datatype

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name

SQL ALTER TABLE Example-

ALTER TABLE Persons  
ADD DateOfBirth date

**Change Data Type Example-**

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year

**DROP COLUMN Example**

ALTER TABLE Persons  
DROP COLUMN DateOfBirth

**MySQL NOW() Function-**

By this function you get date and time.

Example—

SELECT NOW(),CURDATE(),CURTIME()

Example2-

CREATE TABLE Orders  
(  
OrderId int NOT NULL,  
ProductName varchar(50) NOT NULL,  
OrderDate datetime NOT NULL DEFAULT NOW(),  
PRIMARY KEY (OrderId)  
)

**SQL Functions**

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

**Useful aggregate functions:**

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum

**SQL Scalar functions**

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

UCASE() - Converts a field to upper case

LCASE() - Converts a field to lower case

MID() - Extract characters from a text field

LEN() - Returns the length of a text field

ROUND() - Rounds a numeric field to the number of decimals specified

NOW() - Returns the current system date and time

FORMAT() - Formats how a field is to be displayed

**The AVG() Function**

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

Example1-

SELECT AVG(Price) AS PriceAverage FROM Products;

Example2-

SELECT ProductName, Price FROM Products  
WHERE Price>(SELECT AVG(Price) FROM Products);

**SQL COUNT() Function-**

The COUNT(column\_name) function returns the number of values (NULL values will not be counted) of the specified column:

SELECT COUNT(column\_name) FROM table\_name;

SQL COUNT(\*) Syntax

The COUNT(\*) function returns the number of records in a table:

SELECT COUNT(\*) FROM table\_name;

SELECT COUNT(\*) AS NumberOfOrders FROM Orders;

SELECT COUNT(DISTINCT CustomerID) AS NumberOfCustomers FROM Orders;

**The FIRST() Function-**

The FIRST() function returns the first value of the selected column.

SELECT FIRST(column\_name) FROM table\_name;

Example-

SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers;

The LAST() Function

The LAST() function returns the last value of the selected column.

SELECT LAST(column\_name) FROM table\_name;

Example-

SELECT LAST(CustomerName) AS LastCustomer FROM Customers;

**The MAX() Function**

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

Syntax-

SELECT MAX(column\_name) FROM table\_name;

Example-

SELECT MAX(Price) AS HighestPrice FROM Products;

The MIN() Function-

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

Syntax-

SELECT MIN(column\_name) FROM table\_name;

Example-

SELECT MIN(Price) AS SmallestOrderPrice FROM Products;

The SUM() Function

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

Syntax-

SELECT SUM(column\_name) FROM table\_name;

Example-

SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;

**The GROUP BY Statement**

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Syntax-

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name;

Example-

SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders  
LEFT JOIN Shippers  
ON Orders.ShipperID=Shippers.ShipperID  
GROUP BY ShipperName;

The HAVING Clause

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

Syntax-

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name  
HAVING aggregate\_function(column\_name) operator value;

Example-

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders  
INNER JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;

**The UCASE() Function**

The UCASE() function converts the value of a field to uppercase.

SELECT UCASE(column\_name) FROM table\_name;

Example-

SELECT UCASE(CustomerName) AS Customer, City  
FROM Customers;

**The LCASE() Function**

The LCASE() function converts the value of a field to lowercase.

Syntax-

SELECT LCASE(column\_name) FROM table\_name;

Example-

SELECT LCASE(CustomerName) AS Customer, City  
FROM Customers;

**The MID() Function**

The MID() function is used to extract characters from a text field.

SELECT MID(column\_name,start[,length]) AS some\_name FROM table\_name;

Parameter Description

column\_name Required. The field to extract characters from

start Required. Specifies the starting position (starts at 1)

length Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

Example-

SELECT MID(City,1,4) AS ShortCity  
FROM Customers;

**The LEN() Function**

The LEN() function returns the length of the value in a text field.

SELECT LEN(column\_name) FROM table\_name;

Example-

SELECT CustomerName,LEN(Address) as LengthOfAddress  
FROM Customers;

**The ROUND() Function**

The ROUND() function is used to round a numeric field to the number of decimals specified.

Example-

SELECT ProductName, ROUND(Price,0) AS RoundedPrice  
FROM Products;

**The NOW() Function**

The NOW() function returns the current system date and time.

SELECT NOW() FROM table\_name;

Example-

SELECT ProductName, Price, Now() AS PerDate  
FROM Products;

**The FORMAT() Function**

The FORMAT() function is used to format how a field is to be displayed.

SELECT FORMAT(column\_name,format) FROM table\_name;

Example-

SELECT ProductName, Price, FORMAT(Now(),'YYYY-MM-DD') AS PerDate  
FROM Products;