**SELECT Syntax**

**SELECT column1, column2, ...FROM table\_name;**

2.SELECT \* FROM table\_name;

3.1.SELECT column FROM table\_name;

**SELECT DISTINCT Syntax**

**SELECT DISTINCT column1, column2, ...FROM table\_name;**

2.SELECT COUNT(DISTINCT column) FROM table\_name;

3.SELECT DISTINCT column FROM table\_name;

4. select count ( country), count( distinct city) from customers;

**WHERE Syntax**

1.SELECT column1, column2, ...FROM table\_name WHERE condition;

2.SELECT column FROM table\_name WHERE condition;

e.g.

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

2.SELECT \* FROM Customers WHERE CustomerID=1;

**Note:-Text Fields vs. Numeric Fields**

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**

Operator Description

= 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

e.g.

1.SELECT \* FROM Products WHERE Price = 18;

2. SELECT \* FROM Products WHERE Price > 30;

3.SELECT \* FROM Products WHERE Price < 30;

4. SELECT \* FROM Products WHERE Price >= 30;

5.SELECT \* FROM Products WHERE Price <= 30;

6.SELECT \* FROM Products WHERE Price <> 18;

7.SELECT \* FROM Products WHERE Price BETWEEN 50 AND 60;

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

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

10. SELECT \* FROM [Customers] where Country IN ('Germany' ,'Mexico','UK','Sweden');

**SQL AND, OR and NOT Operators**

The AND and OR operators are used to filter records based on more than one condition:

1.The AND operator displays a record if all the conditions separated by AND are TRUE.

2.The OR operator displays a record if any of the conditions separated by OR is TRUE.

3.The NOT operator displays a record if the condition(s) is NOT TRUE.

**AND Syntax**

SELECT column1, column2, ...FROM table\_name WHERE condition1 AND condition2 AND condition3 ...;

e.g -->SELECT \* FROM Customers WHERE Country='Germany' AND City='Berlin';

**OR Syntax**

SELECT column1, column2, ...FROM table\_name WHERE condition1 OR condition2 OR condition3 ...;

e.g. SELECT \* FROM Customers WHERE City='Berlin' OR City='München';

SELECT \* FROM Customers WHERE Country='Germany' OR Country='Spain';

**NOT Syntax**

SELECT column1, column2, ...FROM table\_name WHERE NOT condition;

e.g. SELECT \* FROM Customers WHERE NOT Country='Germany';

**Combining AND, OR and NOT e.g.**

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

SELECT \* FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';

**The SQL ORDER BY Keyword**

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

The ORDER BY keyword sorts the records in ascending order by default.

To sort the records in descending order, use the DESC keyword.

**ORDER BY Syntax**

SELECT column1, column2, ...FROM table\_name ORDER BY column1, column2, ... ASC|DESC;

e.g. SELECT \* FROM Customers ORDER BY Country;

SELECT Country FROM Customers ORDER BY Country;

SELECT \* FROM Customers ORDER BY Country DESC;

SELECT \* FROM Customers ORDER BY Country, CustomerName;

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

SELECT Price FROM [Products] ORDER BY Price DESC LIMIT 5 ;

**The SQL INSERT INTO Statement**

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

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, ...);

e.g.1 INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

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

e.g.2 INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');

**SQL NULL Values**

**What is a NULL Value?**

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

If a field in a table is optional, it is possible to insert a new record or update a record without

adding a value to this field. Then, the field will be saved with a NULL value.

**IS NULL Syntax**

SELECT column\_names FROM table\_name WHERE column\_name IS NULL;

e.g. SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL;

**IS NOT NULL Syntax**

SELECT column\_names FROM table\_name WHERE column\_name IS NOT NULL;

SELECT CustomerName, ContactName, Address

FROM Customers WHERE Address IS NOT NULL;

The SQL UPDATE Statement

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

**UPDATE Syntax**

UPDATE table\_name SET column1 = value1, column2 = value2, ...WHERE condition;

E.G. UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;

E.G.UPDATE Customers SET ContactName='Juan' WHERE Country='Mexico';

**Update Warning!**

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

E.G.UPDATE Customers SET ContactName='Juan';

**The SQL DELETE Statement**

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

**DELETE Syntax**

DELETE FROM table\_name WHERE condition;

E.G.DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

**Delete All Records**

It is possible to delete all rows in a table without deleting the table.

This means that the table structure, attributes, and indexes will be intact:

SYNTAX---DELETE FROM table\_name;

E.G. DELETE FROM Customers;

**The SQL SELECT TOP Clause.**

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

**Syntax::-**

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

SELECT TOP number \* FROM table\_name ;

E.G. SELECT TOP 6 \* FROM CUSTOMERS;

**LIMIT SYNTAX**

SELECT column\_name(s) FROM table\_name LIMIT NO ;

SELECT \* FROM table\_name LIMIT NO ;

E.G. SELECT \* FROM Customers LIMIT 3;

**The 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.

**MIN() Syntax**

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

E.G. SELECT MIN(Price) FROM Products;

E.G. SELECT Price FROM [Products] ORDER BY Price LIMIT 5 ;---MIN 5 VALUES

**MAX() Syntax**

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

E.G. SELECT MAX(Price) FROM Products;

E.G. SELECT Price FROM [Products] ORDER BY Price DESC LIMIT 5 ;TOP 5 VALUES

**The SQL COUNT(), AVG() and SUM() Functions**

**COUNT() Syntax**

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

E.G. SELECT COUNT(ProductID) FROM Products;

**AVG() Syntax**

SELECT AVG(column\_name) FROM table\_name WHERE condition;

**SELECT AVG(Price) FROM Products;**

**SUM() Syntax**

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

SELECT SUM(Quantity) FROM OrderDetails;

**LIKE Syntax**

SELECT column1, column2, ...FROM table\_name WHERE columnN LIKE pattern;

**LIKE Operator Description**

**WHERE CustomerName LIKE 'a%'---Finds any values that start with "a"**

E.G. SELECT \* FROM Customers WHERE CustomerName LIKE 'a%';

**WHERE CustomerName LIKE '%a'--- Finds any values that end with "a"**

SELECT \* FROM Customers WHERE CustomerName LIKE '%a';

**WHERE CustomerName LIKE '%or%'---Finds any values that have "or" in any position**

SELECT \* FROM Customers WHERE CustomerName LIKE '%or%'

**WHERE CustomerName LIKE '\_r%'---Finds any values that have "r" in the second position**

SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%';

**WHERE CustomerName LIKE 'a\_%'---Finds any values that start with "a" and are at least 2 characters in length**

SELECT \* FROM Customers WHERE CustomerName LIKE 'a\_\_%';

**WHERE ContactName LIKE 'a%o'--- Finds any values that start with "a" and ends with "o"**

SELECT \* FROM Customers WHERE ContactName LIKE 'a%o';

**SQL Wildcard Characters**

A wildcard character is used to substitute one or more characters in a string.

E.G. SELECT \* FROM Customers WHERE City LIKE 'ber%';

​

**The SQL IN Operator**

The IN operator is a shorthand for multiple OR conditions.

**IN Syntax**

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

E.G. SELECT \* FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

**The SQL BETWEEN Operator**

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

**BETWEEN Syntax**

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

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

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

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

E.G. SELECT \* FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;

E.G. SELECT \* FROM Products WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;

E.G. 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.

An alias only exists for the duration of that query.

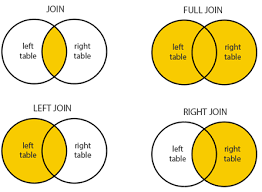
An alias is created with the AS keyword.

**Alias Column Syntax**

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

SELECT CustomerID AS ID, CustomerName AS Customer FROM Customers;

**SQL JOIN-is used to join two table**

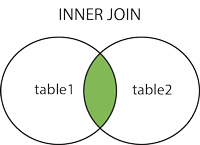
****

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

**SQL INNER JOIN Keyword**

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

**INNER JOIN / join Syntax**

****

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

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

we match primary key with foreign key. first column id is primarykey and in another table it is refer as foreign key

e.g. SELECT \* FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

e.g. SELECT \* FROM customers INNER JOIN orders ON Orders.CustomerID = Customers.CustomerID;

e.g. SELECT \* FROM customers INNER JOIN orders ON Customers.CustomerID = Orders.CustomerID;

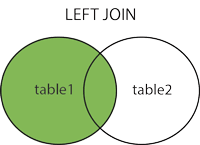
**JOIN Three Tables for exam**

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);

**SQL LEFT JOIN Keyword**

****

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.

**LEFT JOIN Syntax--**

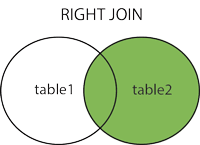
SELECT column\_name(s) FROM table1 LEFT JOIN table2 ON table1.column\_name = table2.column\_name;

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

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

e.g. SELECT \* FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID ORDER BY Customers.CustomerName;

**SQL RIGHT JOIN Keyword**

****

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.

**RIGHT JOIN Syntax--**

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

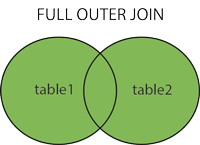
e.g. SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

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

e.g. SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID ;

**SQL FULL OUTER JOIN Keyword**

****

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

Tip: FULL OUTER JOIN and FULL JOIN are the same.

**FULL OUTER JOIN Syntax**

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

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

The columns must also have similar data types

**UNION Syntax**

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

e.g.SELECT City FROM Customers UNION SELECT City FROM Suppliers ORDER BY City;

**UNION ALL Syntax**

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

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

e.g. SELECT City FROM Customers UNION ALL SELECT City FROM Suppliers ORDER BY City;

**The SQL GROUP BY Statement**

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.

**GROUP BY Syntax**

SELECT column\_name(s) FROM table\_name GROUP BY column\_name(s);

e.g. SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;

**The SQL HAVING Clause:- when we used group by then only we can use having**

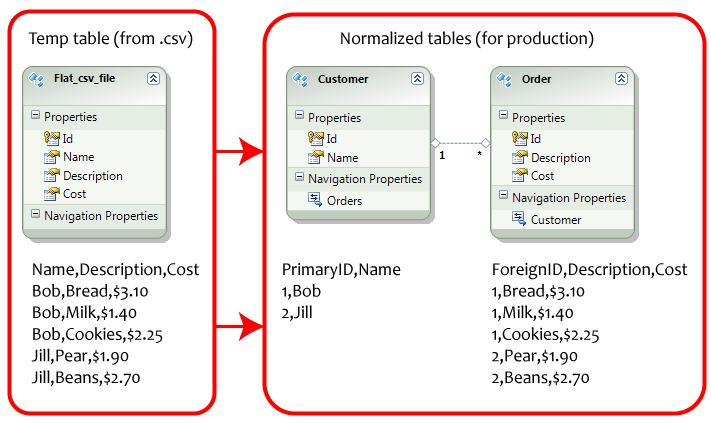
**synatx---**

SELECT column\_name(s) FROM table\_name GROUP BY column\_name(s) HAVING condition;

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

**Normalization---**

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.



Find Maximum salary

SELECT Price FROM Products order by Price desc limit 0,1;

Find 2nd Maximum salary

Index

SELECT Price FROM Products order by Price desc limit 1,1;

How many records we wants

Find 3rd Maximum salary

SELECT Price FROM Products order by Price desc limit 2,1;

Find 4th Maximum salary

SELECT Price FROM Products order by Price desc limit 3,1;