**SQL**

**DDL:**

DDL is short name of **Data Definition Language,** which deals with database schemas and descriptions, of how the data should reside in the database.

● CREATE - to create a database and its objects like (table, index, views, store procedure, function, and triggers)

● ALTER - alters the structure of the existing database

● DROP - delete objects from the database

● TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

● RENAME - rename an object

**DML:**

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

● SELECT - retrieve data from a database

● INSERT - insert data into a table

● UPDATE - updates existing data within a table

● DELETE - Delete all records from a database table

● MERGE - UPSERT operation (insert or update)

**DCL:**

DCL is short name of **Data Control Language** which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

● GRANT - allow users access privileges to the database

● REVOKE - withdraw users access privileges given by using the GRANT command

**TCL:**

TCL is short name of Transaction Control Language which deals with a transaction within a database.

● COMMIT - commits a Transaction

● ROLLBACK - rollback a transaction in case of any error occurs

● SAVEPOINT - to roll back the transaction making points within groups **SQL:**

SQL is a standard language for storing, manipulating and retrieving data in databases.

**SELECT:**

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. If you want to select all the fields available in the table, use the following syntax: ● SELECT \* FROM *table\_name*;

**Ex –**

● SELECT CustomerName, City FROM Customers;

**SELECT DISTINCT:**

The SELECT DISTINCT statement is used to return only distinct (different) values. **Syntax –**

● SELECT DISTINCT *column1*, *column2, ...*

FROM *table\_name*;

**Ex –**

● SELECT DISTINCT Country FROM Customers;

**WHERE:**

The WHERE clause is used to filter records.

**Syntax –**

● SELECT *column1*, *column2, ...*

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT \* FROM Customers

WHERE Country='Mexico';

|  |  |
| --- | --- |
| **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 != |

**AND, OR and NOT:**

The WHERE clause can be combined with AND, OR, and NOT operators.

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

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

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

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

**Syntax –**

● SELECT *column1*, *column2, ...*

FROM *table\_name*

WHERE *condition1* AND *condition2* AND *condition3 ...*;

● SELECT *column1*, *column2, ...*

FROM *table\_name*

WHERE *condition1* OR *condition2* OR *condition3 ...*;

● SELECT *column1*, *column2, ...*

FROM *table\_name*

WHERE NOT *condition*;

**Ex –**

● SELECT \* FROM Customers

WHERE Country='Germany' AND City='Berlin';

● SELECT \* FROM Customers

WHERE Country='Germany' AND (City='Berlin' OR City='München');

**ORDER BY:**

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.

**Syntax –**

● SELECT *column1*, *column2, ...*

FROM *table\_name*

ORDER BY *column1, column2, ...* ASC|DESC;

**Ex –**

● SELECT \* FROM Customers

ORDER BY Country;

● SELECT \* FROM Customers

ORDER BY Country ASC, CustomerName DESC;

**INSERT INTO:**

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

**Syntax –**

● INSERT INTO *table\_name* (*column1*, *column2*, *column3*, ...)

VALUES (*value1*, *value2*, *value3*, ...);

● INSERT INTO *table\_name*

VALUES (*value1*, *value2*, *value3*, ...);

\*In the second syntax, make sure the order of the values is in the same order as the columns in the table.

**Ex –**

● INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

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

**NULL Value:**

It is not possible to test for NULL values with comparison operators, such as =, <, or <>. We will have to use the IS NULL and IS NOT NULL operators instead.

**Syntax –**

● SELECT *column\_names*

FROM *table\_name*

WHERE *column\_name* IS NULL;

● SELECT *column\_names*

FROM *table\_name*

WHERE *column\_name* IS NOT NULL;

**Ex –**

● SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NULL;

**UPDATE:**

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

● UPDATE *table\_name*

SET *column1* = *value1*, *column2* = *value2*, ...

WHERE *condition*;

**Ex –**

● UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;

**DELETE:**

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

**Syntax –**

● DELETE FROM *table\_name* WHERE *condition*;

● DELETE FROM *table\_name*;

In 2ndsyntax, all rows are deleted. The table structure, attributes, and indexes will be intact **Ex –**

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

**SELECT TOP:**

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

● SELECT TOP *number*|*percent column\_name(s)*

FROM *table\_name*

WHERE *condition*;

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE *condition*

LIMIT *number*;

● SELECT *column\_name(s)*

FROM *table\_name*

ORDER BY *column\_name(s)*

FETCH FIRST *number* ROWS ONLY;

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE ROWNUM <= *number*;

\*In case the interviewer asks other than the TOP, rest are also correct. (Diff. DB Systems) **Ex –**

● SELECT TOP 3 \* FROM Customers;

● SELECT \* FROM Customers

LIMIT 3;

● SELECT \* FROM Customers

FETCH FIRST 3 ROWS ONLY;

**Aggregate Functions:**

**MIN():**

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

● SELECT MIN(*column\_name*)

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT MIN(Price) AS SmallestPrice

FROM Products;

**MAX():**

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

● SELECT MAX(*column\_name*)

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT MAX(Price) AS LargestPrice

FROM Products;

**COUNT():**

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

● SELECT COUNT(*column\_name*)

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT COUNT(ProductID)

FROM Products;

**AVG():**

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

**Syntax –**

● SELECT AVG(*column\_name*)

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT AVG(Price)

FROM Products;

**SUM():**

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

**Syntax –**

● SELECT SUM(*column\_name*)

FROM *table\_name*

WHERE *condition*;

**Ex –**

● SELECT SUM(Quantity)

FROM OrderDetails;

**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*;

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |

|  |  |
| --- | --- |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

**IN:**

The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

**Syntax –**

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name* IN (*value1*, *value2*, ...);

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name* IN (*SELECT STATEMENT*);

**Ex –**

● SELECT \* FROM Customers

WHERE Country IN ('Germany', 'France', 'UK');

● SELECT \* FROM Customers

WHERE Country IN (SELECT Country FROM Suppliers);

**BETWEEN:**

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 –**

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name* BETWEEN *value1* AND *value2;*

**Ex –**

● SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20;

**Joins:**

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

**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*;

**Ex –**

● SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

**LEFT (OUTER) 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*;

**Ex –**

● SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

**RIGHT (OUTER) 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*;

**Ex –**

● SELECT Orders.OrderID, Employees.LastName, Employees.FirstName FROM Orders

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

**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*;

**Ex –**

● SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;

**UNION:**

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

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

**Syntax –**

● SELECT *column\_name(s)* FROM *table1*

UNION

SELECT *column\_name(s)* FROM *table2*;

● SELECT *column\_name(s)* FROM *table1*

UNION ALL

SELECT *column\_name(s)* FROM *table2*;

**Ex –**

● SELECT City FROM Customers

UNION

SELECT City FROM Suppliers

ORDER BY City;

**GROUP BY:**

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

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);*

**Ex –**

● SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;

**HAVING:**

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

\*WHERE is given priority over HAVING.

**Syntax –**

● SELECT *column\_name(s)*

FROM *table\_name*

WHERE *condition*

GROUP BY *column\_name(s)*

HAVING *condition*

ORDER BY *column\_name(s);*

**Ex –**

● SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

**CREATE DATABASE:**

The CREATE DATABASE statement is used to create a new SQL database. **Syntax –**

● CREATE DATABASE *databasename*;

**DROP DATABASE:**

The DROP DATABASE statement is used to drop an existing SQL database. **Syntax –**

● DROP DATABASE *databasename*;

**CREATE TABLE:**

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

**Syntax –**

● CREATE TABLE *table\_name* (

*column1 datatype*,

*column2 datatype*,

*column3 datatype*,

 ....

);

**DROP TABLE:**

The DROP TABLE statement is used to drop an existing table in a database. **Syntax –**

● DROP TABLE *table\_name*;

**TRUNCATE TABLE:**

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself. **Syntax –**

● TRUNCATE TABLE *table\_name*;

**ALTER TABLE:**

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

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

**Syntax –**

● ALTER TABLE *table\_name*

ADD *column\_name datatype*;

● ALTER TABLE *table\_name*

DROP COLUMN *column\_name*;

● ALTER TABLE *table\_name*

MODIFY COLUMN *column\_name datatype*;

**Ex –**

● ALTER TABLE Customers

ADD Email varchar(255);

● ALTER TABLE Customers

DROP COLUMN Email;

● ALTER TABLE Persons

ALTER COLUMN DateOfBirth year;