Database Management Systems Part-II

Class- XII

1. DATABASE COMMANDS

a) CREATE DATABASE: Used to create a new database.

Syntax: `CREATE DATABASE database_name;`

Example: `CREATE DATABASE SchoolDB;`

b) USE DATABASE: Selects an existing database to work with.

Syntax: `USE database_name;`

Example: `USE SchoolDB;`

c) SHOW DATABASES: Displays the list of all databases on the server.

Syntax: `SHOW DATABASES;`

d) DROP DATABASE: Deletes an existing database permanently.

Syntax: `DROP DATABASE database_name;`

Example: `DROP DATABASE SchoolDB;`

2. TABLE COMMANDS

a) SHOW TABLES: Displays all tables in the currently selected database.

Syntax: `SHOW TABLES;`

b) CREATE TABLE: Used to create a new table.

Syntax:

```
CREATE TABLE table_name (
    column_name data_type [constraints],
    ...
);
```

Example:

```
CREATE TABLE Student (
    RollNo INT PRIMARY KEY,
    Name VARCHAR(50),
```

```
    Marks INT
);
```

c) DESCRIBE TABLE: Displays structure (columns, data types, constraints) of a table.

Syntax: `DESCRIBE table_name;`

d) ALTER TABLE: Used to **modify** an existing table.

(i) Add a new column:

```
ALTER TABLE table_name ADD column_name data_type;
```

Example: `ALTER TABLE Student ADD Address VARCHAR(100);`

(ii) Remove a column:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Example: `ALTER TABLE Student DROP COLUMN Address;`

(iii) Add a Primary Key:

```
ALTER TABLE table_name ADD PRIMARY KEY (column_name);
```

(iv) Remove a Primary Key:

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

e) DROP TABLE: This command deletes a table permanently.

Syntax: `DROP TABLE table_name;`

3. DATA MANIPULATION COMMANDS (DML)

a) INSERT: This command is used to add new records (rows) to a table.

Syntax:

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

Example:

```
INSERT INTO Student (RollNo, Name, Marks) VALUES (1, 'Amit', 85);
```

b) DELETE: This SQL command deletes records from a table.

Syntax: `DELETE FROM table_name WHERE condition;`

Example: `DELETE FROM Student WHERE RollNo = 1;`

c) UPDATE: This SQL command is used to modifies existing records in a table.

Syntax:

```
UPDATE table_name SET column_name = value WHERE condition;
```

Example: `UPDATE Student SET Marks = 90 WHERE RollNo = 2;`

d) SELECT: This SQL command is used to retrieves data from one or more tables.

Syntax: `SELECT column1, column2 FROM table_name;`

Example: `SELECT Name, Marks FROM Student;`

---

4. OPERATORS

Operators are special symbols used in SQL to perform operations on data values.
They are mainly classified into **three categories:**

a) Mathematical Operators

Operator	Meaning	Example
+	Addition	Marks + 10
-	Subtraction	Marks - 5
*	Multiplication	Marks * 2
/	Division	Marks / 2

Example: SELECT Name, Marks + 5 AS BonusMarks FROM Student;

b) Relational Operators

Operator	Meaning	Example
=	Equal to	Marks = 80
<> or !=	Not equal to	Marks <> 50
<	Less than	Marks < 50
>	Greater than	Marks > 50
<=	Less than or equal to	Marks <= 80
>=	Greater than or equal to	Marks >= 60

Example: SELECT Name FROM Student WHERE Marks >= 85;

c) Logical Operators

Operator	Meaning	Example
AND	Both conditions true	Marks > 50 AND Marks < 90
OR	Any condition true	Marks < 40 OR Grade = 'C'
NOT	Negation	NOT (Marks > 70)

Example: SELECT Name, Marks FROM Student WHERE Marks > 70 AND Grade = 'A';

5. ALIASING

This command gives a temporary name to a column or table within a query.

Syntax: `SELECT column_name AS alias_name FROM table_name;`

Example: `SELECT Name AS StudentName FROM Student;`

6. DISTINCT CLAUSE:

This SQL command is used to removes duplicate values from query results.

Syntax: `SELECT DISTINCT column_name FROM table_name;`

Example: `SELECT DISTINCT address FROM Student;`

---

7. WHERE CLAUSE:

This is the most useful SQL command used to filters records based on any conditions.

Syntax: `SELECT * FROM table_name WHERE condition;`

Example: `SELECT * FROM Student WHERE Marks > 80;`

8. IN and BETWEEN:

a) IN / NOT IN: This checks if a value matches with any value from the given list.

```
SELECT * FROM Student WHERE city IN ("Delhi", "Mumbai", "Kolkata");
```

b) BETWEEN: It Checks if a value lies within a range.

```
SELECT * FROM Student WHERE Marks BETWEEN 60 AND 90;
```

Note: here 60 and 90 are also included.

9. ORDER BY:

This command sorts/arranges the result in ascending (ASC) or descending (DESC) order.

Syntax: `SELECT * FROM Student ORDER BY Marks DESC;`

10. NULL VALUES: **NULL** means *no value / missing data*. It is **not equal to 0 or empty string**.

(a) IS NULL

```
SELECT * FROM Student WHERE Address IS NULL;
```

(b) IS NOT NULL

```
SELECT * FROM Student WHERE Address IS NOT NULL;
```

11. LIKE OPERATOR (Pattern Matching)

The **`LIKE` operator** is used to **search for a specific pattern** in a column. It helps when you don't know the exact text — only part of it. For example you want see the details of those students whose name starts with 'A'.

Symbol	Meaning
%	Any number of characters
_	One character

Examples:

```
SELECT * FROM Student WHERE Name LIKE 'A%';      -- starts with A
SELECT * FROM Student WHERE Name LIKE '_a%';     -- second letter is 'a'
```

---

12. AGGREGATE FUNCTIONS: Used to perform calculations on data columns.

Function	Description
MAX()	Returns the maximum value
MIN()	Returns the minimum value
AVG()	Returns the average value
SUM()	Returns the total sum
COUNT()	Returns the number of rows

Example: `SELECT MAX(Marks), AVG(Marks) FROM Student;`

13. GROUP BY CLAUSE: The **`GROUP BY` clause** in SQL is used to **group rows** that have the **same values** in one or more columns. It is **often used with aggregate functions** such as `SUM()`, `AVG()`, `COUNT()`, `MAX()`, or `MIN()` to perform calculations on each group of data.

Syntax:

```
SELECT column_name(s), aggregate_function(column_name)
FROM table_name
GROUP BY column_name(s);
```

Example 1: The following example of Group By groups students according to their class and calculates the **average marks** for each class.

```
SELECT Class, AVG(Marks)
FROM Student
GROUP BY Class;
```

Example 2: It `Counts how many students are in each class.`

```
SELECT Class, COUNT(*) AS NoOfStudents
FROM Student
GROUP BY Class;
```

Example 3: Grouping by Multiple Columns

```
SELECT Class, Gender, COUNT(*) AS NoOfStudents
FROM Student
GROUP BY Class, Gender;
```

Result:

Class	Gender	NoOfStudents
12A	M	2
12A	F	1
12B	M	1
12B	F	1

Explanation: Groups records **first by Class**, and then **by Gender** inside each class.

Example with HAVING:

```
SELECT Class, AVG(Marks)
FROM Student
GROUP BY Class
HAVING AVG(Marks) > 80;
```

Purpose of GROUP BY command:

- To **summarize** data.
- To **arrange** data into **groups** based on one or more columns.
- To use **aggregate functions** (like `SUM`, `AVG`, `COUNT`) **on each group**.

14. HAVING CLAUSE: It is used to apply a condition on groups (after `GROUP BY`).

Example:

```
SELECT Class, AVG(Marks)
FROM Student
GROUP BY Class
HAVING AVG(Marks) > 75;
```

15. JOINS

A **JOIN** in SQL is used to **combine data from two or more tables** based on a **related column** between them. It helps you see **connected information** stored across multiple tables.

a) Cartesian Product: The **Cartesian Product** of two tables gives **all possible combinations** of rows — every row from the first table is combined with **every row** from the second table.

Syntax:

```
SELECT * FROM Table1, Table2;
```

Result: Each row of Table1 is paired with each row of Table2. If Table1 has 3 rows and Table2 has 2 rows → result = 3 × 2 = 6 rows.

b) Equi-Join: An **Equi-Join** combines rows from two tables **based on a common column** using an **equality (=) condition**.

Syntax:

```
SELECT Table1.column, Table2.column
FROM Table1, Table2
WHERE Table1.common_column = Table2.common_column;
```

Example:

```
SELECT Student.Name, Class.ClassName
FROM Student, Class
WHERE Student.ClassID = Class.ClassID;
```

Explanation:
Only rows where Student.ClassID equals Class.ClassID are joined and displayed.

c) Natural Join: A **Natural Join** automatically joins two tables **based on all columns with the same name** and **data type**. It removes duplicate columns from the result.

Syntax:

```
SELECT * FROM Student NATURAL JOIN Class;
```

Result: The output is s**ame as equi-join, but** no need to specify the condition **and** duplicate columns **appears only once.**

☑**Note:**

- Always use **WHERE** with **DELETE** and **UPDATE** to avoid accidental removal or modification of all records.
- Use **aliases**, **ORDER BY**, and **DISTINCT** to make results more readable and meaningful.