**SQL AND RDBMS**

1. **What is RDBMS?**

RDBMS (Relational Database Management System) is a type of database management system that stores data in tables (rows and columns) and allows relationships between different tables using keys.

* It uses SQL (Structured Query Language) to manage and query data.
* Ensures data integrity using constraints like primary keys, foreign keys, etc.

1. **What is database table?**

A table is a structured format in a database where data is stored in rows and columns.

* Rows represent records (also called tuples).
* Columns represent fields (attributes) of the data.

1. **What are the features of RDBMS?**

* Data is stored in tabular form with rows and columns.
* Supports primary keys and foreign keys for maintaining relationships.
* Allows ACID properties (Atomicity, Consistency, Isolation, Durability).
* Multi-user access with proper security and transaction control.

1. **Where is RDBMS used in customer management relationship?**

* To store **customer data** like name, contact, purchase history, etc.
* Helps in **generating reports**, analyzing buying patterns, and managing leads.
* Supports **sales and marketing automation** through structured and relational data.

1. **What are the advantages of RDBMS?**

* **Data Integrity and Accuracy**: Enforces rules to prevent data duplication or inconsistency.
* **Scalability**: Can handle large amounts of data and concurrent users.
* **Security**: Provides user access control and permission management.

1. **What are the disadvantages of RDBMS?**

* **Complexity**: Can be hard to design and manage as data grows.
* **Cost**: Licensing and hardware can be expensive for enterprise systems.
* **Performance**: Slower than NoSQL for unstructured data or high-volume read/writes.

1. **Difference between DBMS & RDBMS?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **DBMS** | **RDBMS** |
| **1. Data Storage** | **Stores data in files or simple formats** | **Stores data in tabular form** |
| **2. Relationship Support** | **No support for relationships** | **Supports relationships using keys** |
| **3. Normalization** | **Not supported** | **Supports normalization** |
| **4. Keys** | **No concept of primary/foreign keys** | **Uses primary and foreign keys** |
| **5. Integrity Constraints** | **Doesn’t enforce integrity constraints** | **Enforces integrity constraints** |
| **6. Multi-user Support** | **Limited** | **Full multi-user support** |
| **7. Security** | **Basic security** | **Advanced user-level access control** |
| **8. Data Redundancy** | **High** | **Controlled via relationships and normalization** |
| **9. Query Language** | **Not standardized** | **Uses SQL as a standard** |
| **10. Examples** | **Microsoft Access (small scale)** | **MySQL, Oracle, PostgreSQL, SQL Server** |

1. **What is SQL & What are the common types?**

**SQL (Structured Query Language)** is a standardized programming language used to manage and manipulate **relational databases**. It allows users to perform operations like querying data, updating records, inserting data, and deleting records in a database.

|  |  |
| --- | --- |
| **Type** | **Description** |
| **DDL** – Data Definition Language | Used to define and modify the structure of database objects (tables, schemas).Examples: CREATE, ALTER, DROP |
| **DML** – Data Manipulation Language | Used to manipulate data in tables.Examples: SELECT, INSERT, UPDATE, DELETE |
| **DCL** – Data Control Language | Controls access to data in the database.Examples: GRANT, REVOKE |
| **TCL** – Transaction Control Language | Manages transactions and the changes made by DML commands.Examples: COMMIT, ROLLBACK, SAVEPOINT |
| **DQL** – Data Query Language | Retrieves data from the database (mostly just SELECT). |

1. **What is SQL select query and exp with eg**

The **SELECT** statement is used to retrieve data from one or more tables.

**Syntax:**

SELECT column1, column2 FROM table\_name;

**Eg:**

SELECT name, age FROM students;

1. **Exp select statement with HAVING clause**

The **HAVING** clause is used to filter results of groups created by the GROUP BY clause.

Example:

SELECT department, COUNT(\*) AS employee\_count

FROM employees

GROUP BY department

HAVING COUNT(\*) > 5;

This returns departments with more than 5 employees.

1. **Exp select statement with GROUP BY clause**

The **GROUP BY** clause is used to group rows that have the same values in specified columns.

**Example:**

SELECT department, AVG(salary)

FROM employees

GROUP BY department;

This gives the average salary for each department.

1. **Exp select statement with WHERE clause?**

The **WHERE** clause is used to filter records that meet a specific condition.

SELECT name, salary FROM employees

WHERE salary > 50000;

This returns employees whose salary is more than 50,000.

1. **Exp select statement with ORDER BY clause**

The **ORDER BY** clause is used to sort the result in ascending or descending order.

**Example:**

SELECT name, age FROM students

ORDER BY age DESC;

This returns student names and ages sorted in descending order of age.

1. **What is SQL insert into statement ? exp with eg**

The INSERT INTO statement is used to add new records into a table.

**Syntax:**

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

**Example:**

INSERT INTO students (name, age) VALUES ('John', 20);

1. **What is SQL update statement?**

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

* It updates one or more columns for one or more rows based on a given condition.
* Always use the WHERE clause to avoid updating all rows unintentionally.

**Example:**

UPDATE employees SET salary = 60000 WHERE id = 101;

1. **What are the key points about update statement?**

Key Points:

* Use with WHERE clause: To target specific rows. Without it, all rows in the table will be updated.
* Can update multiple columns: You can update more than one column in a single query using commas.

**Example:**

UPDATE students SET name = 'John', grade = 'A' WHERE student\_id = 10;

1. **What is delete statement?**

The DELETE statement is used to remove one or more records from a table.

* + The data deleted using DELETE can be rolled back if used inside a transaction.
  + Without a WHERE clause, all records from the table will be deleted (but the table itself remains).

**Example:**

DELETE FROM orders WHERE order\_id = 5001;

1. **What are the best practises for using delete statement?**

* Always use a WHERE clause to prevent accidental deletion of all data.
* Take a backup or use transactions before executing a delete, especially on production data.

Example:

BEGIN TRANSACTION;

DELETE FROM users WHERE inactive = 1;

COMMIT;

1. **What is duplicate records in table?**

Duplicate records are rows in a table that have identical values in one or more columns, typically where uniqueness is expected.

* Duplicates can occur due to missing constraints (like primary key or unique key).
* Duplicates may lead to incorrect query results, data inconsistency, or calculation errors.
* Example:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| **1** | **Alice** |
| **1** | **Alice** |

1. **What are duplicate rows in RDBMS?**

Rows in a database table that have identical values in all or specific columns.

* Often occur due to incorrect data entry or missing constraints.
* Can cause incorrect query results.

1. **Why we should remove duplicate rows?**

* Prevents inaccurate data analysis.
* Saves storage space.
* Improves query performance.
* Maintains data integrity.

1. **How to identify duplicate rows?**

* Use GROUP BY with HAVING COUNT(\*) > 1 on specific columns.
* Use ROW\_NUMBER() or RANK() window functions.
* Example:

SELECT column1, column2, COUNT(\*)

FROM table\_name

GROUP BY column1, column2

HAVING COUNT(\*) > 1;

1. **What are the best practices to prevent duplicate rows**

* Use **PRIMARY KEY** or **UNIQUE constraints**.
* Validate data before inserting.
* Implement proper indexes.
* Use database triggers to block duplicates.
* Perform regular data cleanup checks.

1. **What is the difference between update and delete statement**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **UPDATE** | **DELETE** |
| **Purpose** | Modifies existing data in one or more rows | Removes one or more rows from a table |
| **Effect on Structure** | Keeps the row structure, only changes values | Removes entire rows from the table |
| **Keyword** | SET is used to specify new values | No SET; uses WHERE to specify which rows to delete |
| **Can Work Without WHERE?** | Yes — updates all rows if WHERE is omitted | Yes — deletes all rows if WHERE is omitted |
| **Rollback** | Can be rolled back (if in transaction) | Can be rolled back (if in transaction) |

1. **What is join in SQL?**

Combines rows from two or more tables based on related columns.  
**Example:**

SELECT e.name, d.department\_name

FROM Employees e

JOIN Departments d

ON e.department\_id = d.department\_id;

1. **What is inner join in SQL?**

Returns only rows with matching values in both tables.  
**Example:**

SELECT e.name, d.department\_name

FROM Employees e

INNER JOIN Departments d

ON e.department\_id = d.department\_id;

1. **What is left join in SQL?**

Returns all rows from left table + matching rows from right table (NULL for unmatched).

**Example:**

SELECT e.name, d.department\_name

FROM Employees e

LEFT JOIN Departments d

ON e.department\_id = d.department\_id;

1. **What is right join in SQL?**

Returns all rows from right table + matching rows from left table (NULL for unmatched).  
**Example:**

SELECT e.name, d.department\_name

FROM Employees e

RIGHT JOIN Departments d

ON e.department\_id = d.department\_id;

1. **What is full join in SQL?**

Returns all rows from both tables, filling with NULL where no match exists.  
**Example:**

SELECT e.name, d.department\_name

FROM Employees e

FULL JOIN Departments d

ON e.department\_id = d.department\_id;

1. **What is cross join in SQL?**

Returns the Cartesian product — all possible combinations.  
**Example:**

SELECT e.name, d.department\_name

FROM Employees e

CROSS JOIN Departments d;

1. **What is self join in SQL?**

Joins a table to itself using aliases.  
**Example:**

SELECT e1.name AS Employee, e2.name AS Manager

FROM Employees e1

JOIN Employees e2

ON e1.manager\_id = e2.employee\_id;

1. **What is the difference between inner and left join?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **INNER JOIN** | **LEFT JOIN** |
| **Definition** | **Returns only rows where there is a match in both tables** | **Returns all rows from the left table and matched rows from the right table** |
| **Unmatched Rows** | **Unmatched rows are discarded** | **Unmatched rows from the left table are kept, with NULLs for missing right table data** |
| **Result Size** | **Generally smaller** | **Generally larger than INNER JOIN** |
| **NULL Handling** | **No NULLs from unmatched rows** | **NULLs appear for columns of unmatched rows from the right table** |
| **Use Case** | **When you need only matching data** | **When you want all data from one table regardless of matches** |
| **Example** | **SELECT \* FROM A INNER JOIN B ON A.id = B.id;** | **SELECT \* FROM A LEFT JOIN B ON A.id = B.id;** |

1. **How are null values handled in joins?**

* In standard joins, NULL ≠ NULL (no match).
* In outer joins, missing values are filled with NULLs.

Example:

SELECT e.name, d.department\_name

FROM Employees e

LEFT JOIN Departments d

ON e.department\_id = d.department\_id

WHERE d.department\_id IS NULL; -- employees without department

1. **Difference between left and right join?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **LEFT JOIN** | **RIGHT JOIN** |
| **Definition** | **Returns all rows from left table and matched rows from right table** | **Returns all rows from right table and matched rows from left table** |
| **Unmatched Rows** | **Keeps unmatched left rows, fills right with NULLs** | **Keeps unmatched right rows, fills left with NULLs** |
| **Primary Focus** | **Left table’s data is prioritized** | **Right table’s data is prioritized** |
| **NULL Placement** | **NULLs in right table columns when no match** | **NULLs in left table columns when no match** |
| **Use Case** | **Keep all records from the left table** | **Keep all records from the right table** |
| **Example** | **SELECT \* FROM A LEFT JOIN B ON A.id = B.id;** | **SELECT \* FROM A RIGHT JOIN B ON A.id = B.id;** |

1. **Why joins are important in SQL?**

Joins are important in SQL to:

* Combine data from multiple tables.
* Reduce redundancy.
* Provide comprehensive results.

1. **Difference between full join and cross join**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **FULL JOIN** | **CROSS JOIN** |
| **Definition** | **Combines LEFT and RIGHT JOIN; returns all rows from both tables with NULLs where there is no match** | **Returns the Cartesian product of two tables (all combinations)** |
| **NULL Handling** | **NULLs appear for unmatched columns** | **No NULLs unless they exist in the original tables** |
| **Matching Criteria** | **Based on a condition (usually ON)** | **No condition — every row of table A joins with every row of table B** |
| **Result Size** | **Depends on matches; max = sum of both tables' rows** | **Always = rows in table A × rows in table B** |
| **Use Case** | **To get all data from both tables regardless of matches** | **To generate all possible combinations of rows** |
| **Example** | **SELECT \* FROM A FULL JOIN B ON A.id = B.id;** | **SELECT \* FROM A CROSS JOIN B;** |

1. **What is natural join?**

Joins tables automatically using columns with the same name.  
Example:

SELECT \*

FROM Employees

NATURAL JOIN Departments;

1. **What is anti join?**

Returns rows in one table without matches in another.  
Example:

SELECT e.name

FROM Employees e

LEFT JOIN Departments d

ON e.department\_id = d.department\_id

WHERE d.department\_id IS NULL;

1. **What is conditional join?**

A join with additional conditions besides matching columns.  
Example:

SELECT e.name, o.order\_id

FROM Employees e

JOIN Orders o

ON e.employee\_id = o.employee\_id

AND o.amount > 1000;

1. **Difference between self join and cross join?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **SELF JOIN** | **CROSS JOIN** |
| **Definition** | **Joins a table with itself** | **Joins every row of one table with every row of another table** |
| **Use Case** | **Compare rows within the same table** | **Generate combinations between two datasets** |
| **Condition** | **Usually uses an alias with a join condition** | **No condition is required** |
| **Result Size** | **Depends on join condition** | **Always rows in table A × rows in table B** |
| **NULL Handling** | **NULLs may appear if using OUTER variant** | **NULLs appear only if originally present** |
| **Example** | **SELECT a.name, b.name FROM Employees a JOIN Employees b ON a.manager\_id = b.id;** | **SELECT \* FROM A CROSS JOIN B;** |