1) Explain the difference between inner join and left join.
2) What is a JOIN in SQL, and how does it work?
3) Describe the various types of JOINs and provide examples.
4) What is a self-join, and when would you use it?

Joins are used to combine data from two or more tables based on a related column between them.

Types of Joins

1. INNER JOIN

- **Definition**: Returns only the rows that have matching values in both tables.
- **Use case**: When you want records that exist in **both** tables.

Example:

sql
Copy code

```
SELECT e.employee_name, d.department_name
FROM employees e
INNER JOIN departments d
ON e.department_id = d.department_id;
```

- This query returns employees who belong to a department.

2. LEFT OUTER JOIN (or LEFT JOIN)

- **Definition**: Returns all rows from the **left** table and the matched rows from the right table. If there is no match, `NULL` values are returned for columns from the right table.
- **Use case**: When you want **all records from the left table**, regardless of whether they have a match in the right table.

Example:

sql
Copy code

```
SELECT e.employee_name, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```

- This query returns all employees, even those who don't belong to any department.

3. RIGHT OUTER JOIN (or RIGHT JOIN)

- **Definition**: Returns all rows from the **right** table and the matched rows from the left table. If there is no match, `NULL` values are returned for columns from the left table.
- **Use case**: When you want **all records from the right table**, regardless of whether they have a match in the left table.

Example:

sql
Copy code

```
SELECT e.employee_name, d.department_name
FROM employees e
RIGHT JOIN departments d
ON e.department_id = d.department_id;
```

- This query returns all departments, even those with no employees.

4. CROSS JOIN

- **Definition**: Returns the Cartesian product of both tables, meaning each row from the first table is combined with every row from the second table.
- **Use case**: When you need every possible combination of rows.

Example:

sql
Copy code

```
SELECT e.employee_name, p.project_name
FROM employees e
CROSS JOIN projects p;
```

- This query returns every combination of employee and project.

5. SELF JOIN

- **Definition**: Joins a table to itself. Useful when you need to compare rows within the same table.

- **Use case**: To find relationships within the same table (e.g., employees reporting to other employees).

Example:

sql
Copy code

```
SELECT e1.employee_name AS employee, e2.employee_name AS manager
FROM employees e1
JOIN employees e2
ON e1.manager_id = e2.employee_id;
```

- This query returns employees and their respective managers from the same table.

6. NATURAL JOIN

- **Definition**: Automatically joins tables based on columns with the same name and compatible data types.
- **Use case**: When the tables have columns with identical names and types, and you want to join them automatically.

Example:

sql
Copy code

```
SELECT employee_name, department_name
FROM employees
NATURAL JOIN departments;
```

- This query joins `employees` and `departments` based on columns with the same name.

- **INNER JOIN**: Matches in **both** tables.
- **LEFT JOIN**: All from left table + matching from right.
- **RIGHT JOIN**: All from right table + matching from left.
- **CROSS JOIN**: Every combination of rows.
- **SELF JOIN**: Table joined to itself.
- **NATURAL JOIN**: Joins on columns with the same name automatically.

1. Explanation of self join

This query retrieves employees along with their managers from the **employees** table. Let's break it down step by step with an example.

Understanding the Query:

1. **Self-Join Concept:**
    - The `employees` table is being joined with itself (`e1` and `e2` are just aliases).
    - Each employee (`e1`) has a `manager_id`, which refers to another employee's `employee_id` (`e2`).
2. **Join Condition:**
    - `e1.manager_id = e2.employee_id` → This means:
        - The `manager_id` in the employee's row matches the `employee_id` of another row (which is the manager's row).

Example Data

employees Table

employee_id	employee_name	manager_id
1	Alice	NULL
2	Bob	1

3	Charlie	1
4	David	2

How the Query Works

1. `e1` refers to **all employees**.
2. `e2` refers to **their managers**.
3. It finds rows where `e1.manager_id = e2.employee_id`, meaning:
    - **Bob's `manager_id` is 1**, which matches **Alice's `employee_id`** → Bob's manager is Alice.
    - **Charlie's `manager_id` is 1**, which matches **Alice's `employee_id`** → Charlie's manager is Alice.
    - **David's `manager_id` is 2**, which matches **Bob's `employee_id`** → David's manager is Bob.

Query Output

employee	manager
Bob	Alice
Charlie	Alice
David	Bob

Visualization (Hierarchy Representation)

Alice (CEO)

├── Bob (Manager)

│   └── David (Employee)

├── Charlie (Employee)

- Alice has **two direct reports**: **Bob and Charlie**.
- Bob manages **David**.
- Charlie has **no direct reports**.

Summary

- The query **self-joins** the table to get employee-manager relationships.
- `e1` represents the **employee**, `e2` represents the **manager**.
- The `ON` condition ensures that each employee’s `manager_id` matches a manager’s `employee_id`.
- The result shows **who reports to whom**.