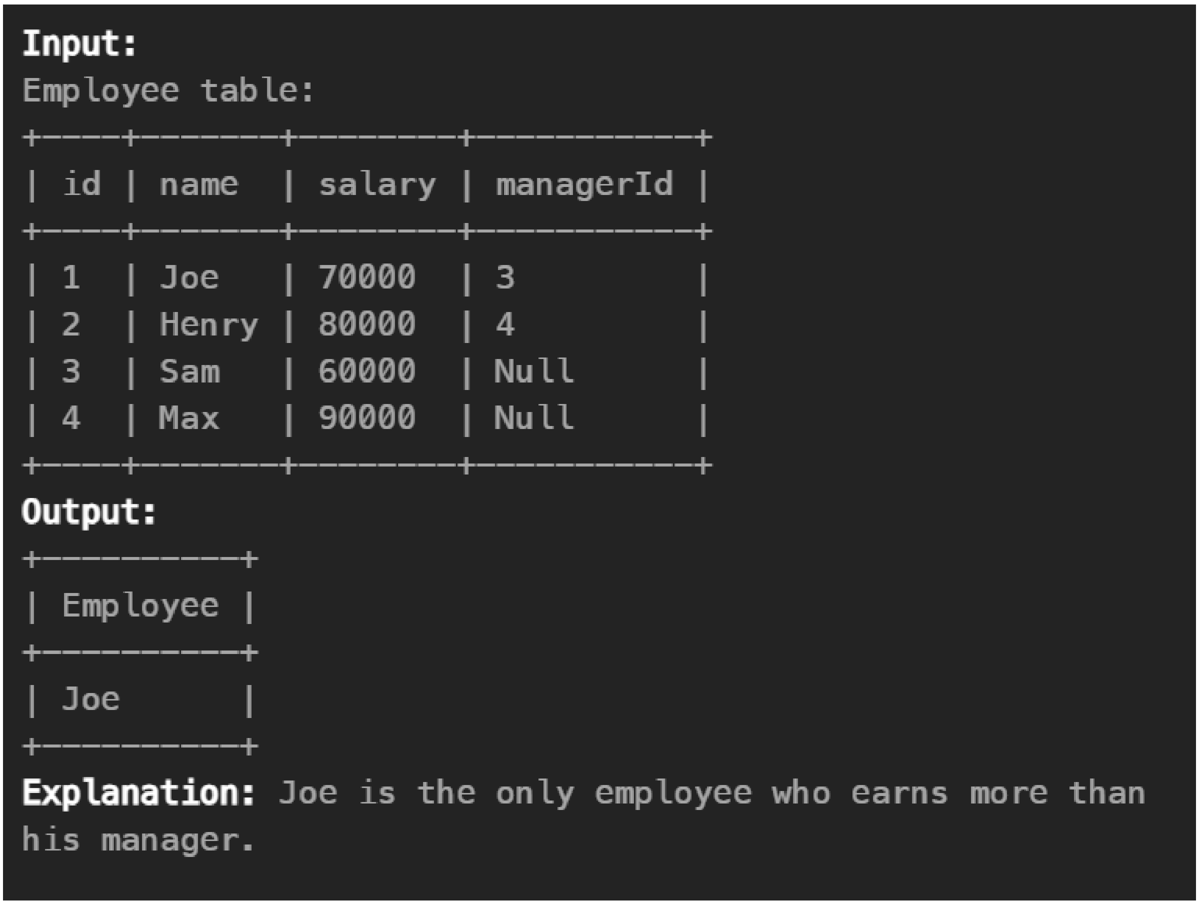
SQL Self Join

Self Join

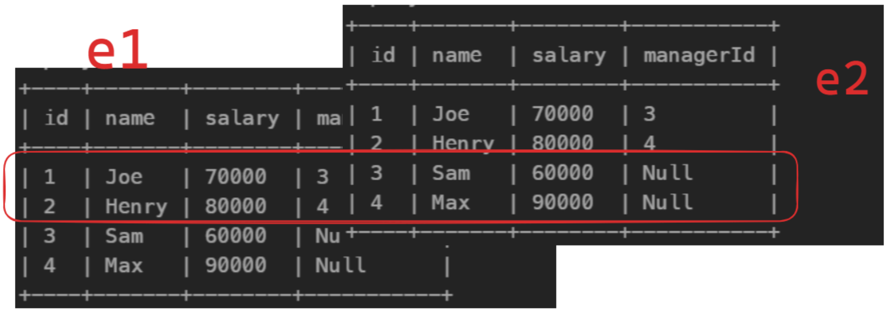


```
Input:
Employee table:
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
Output:
+----------+
| Employee |
+----------+
| Joe      |
+----------+
Explanation: Joe is the only employee who earns more than
his manager.
```

Sample Table

id	name	salary	managerId
1	Joe	70000	3
2	Henry	80000	4
3	Sam	60000	NULL
4	Max	90000	NULL
5	Emma	75000	3
6	Olivia	72000	5
7	Liam	85000	4
8	Noah	50000	3
9	Ava	95000	NULL
10	Sophia	68000	7
11	Mason	88000	9
12	Ethan	55000	11
13	Isabella	63000	3

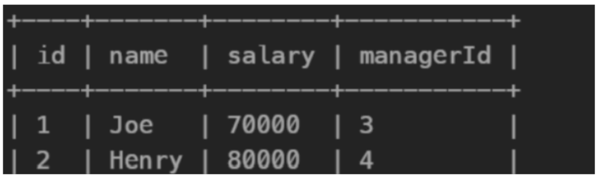
id	name	salary	managerId
14	Logan	77000	9
15	Mia	91000	11



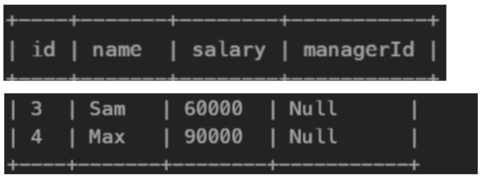
```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```

```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```

```
select e1.name as emp_name, e2.name as mgr_name,
e1.salary as emp_salary, e2.salary as mgr_salary
from employee e1
INNER JOIN
employee e2
on e1.managerid = e2.id
where e1.salary > e2.salary;
```



```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
```



```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```

```SQL
select e1.name as emp_name, e2.name as mgr_name,
e1.salary as emp_salary, e2.salary as mgr_salary
from employee e1
INNER JOIN
employee e2
on e1.managerid = e2.id
where e1.salary > e2.salary;
```

emp_name	mgr_name	emp_salary	mgr_salary
Joe	Sam	70000	60000
Emma	Sam	75000	60000
Isabella	Sam	63000	60000
Mia	Mason	91000	88000

Self Join in SQL

Overview

A **self join** is a type of **join operation** where a table is joined with itself.

It allows you to compare rows within the *same table* — for example, finding relationships among rows of the same dataset (like employees and their managers, categories and subcategories, or parent-child hierarchies).

In a self join, we conceptually treat the same table as if it were **two separate tables**, using **table aliases** to distinguish between them.

Why We Need Self Joins

A **self join** is necessary when a dataset has **recursive or hierarchical relationships** — where one record references another record within the same table.

Common Use Cases

Use Case	Example
Employee → Manager Relationship	Each employee has a `manager_id` pointing to another employee's `id`.
Category Hierarchies	A category can have a `parent_category_id` referencing another category.
Friendships or References	A person can "follow" another person, both existing in the same table.
Comparative Queries	Comparing rows of the same table — e.g., find products with higher prices than other products in the same category.

In all these scenarios, **self join** enables you to relate an entity to another instance of itself.

Conceptual Understanding

Let's say we have a table `Employee`:

id	name	salary	managerId
1	Joe	70000	3
2	Henry	80000	4
3	Sam	60000	NULL
4	Max	90000	NULL

Here, `managerId` refers to another `id` in the same table.

We can visualize this as:

```
Joe   →  Sam
Henry →  Max
```

This is a **self-referential relationship**.
To query such data, we must join the table to itself.

Syntax

```sql
SELECT e.column_name, m.column_name
FROM Employee e
JOIN Employee m
  ON e.managerId = m.id;
```

- `Employee e` → represents the "employee" table instance.
- `Employee m` → represents the "manager" table instance.
- The join condition `e.managerId = m.id` connects each employee to their manager.

Aliases (`e`, `m`) are **mandatory** because otherwise the SQL engine cannot differentiate between two instances of the same table.

Example: Find Each Employee and Their Manager

```sql
SELECT
    e.name AS Employee,
    m.name AS Manager
FROM Employee e
LEFT JOIN Employee m
    ON e.managerId = m.id;
```

Result:

Employee	Manager
Joe	Sam
Henry	Max
Sam	NULL
Max	NULL

Explanation:

- `LEFT JOIN` ensures all employees are shown even if they don't have managers.
- Managers appear as values from the joined instance `m`.

Example: Find Employees Who Earn More Than Their Manager

```sql
SELECT
    e.name AS Employee
FROM Employee e
JOIN Employee m
  ON e.managerId = m.id
WHERE e.salary > m.salary;
```

Result:

Employee
Joe

Here, Joe earns more than Sam (his manager).

Self Join Types

You can apply any **join type** to a self join — the concept remains the same.

Join Type	Description	Example Use
INNER JOIN	Only rows with matching relationships appear.	Employees with managers.
LEFT JOIN	Returns all rows from the left side even if no match.	Show all employees, even without a manager.
RIGHT JOIN	Opposite of LEFT JOIN.	Show all managers even if no subordinates.
FULL JOIN	Combines both sides.	Complete mapping of hierarchy.

Core Concepts and Deep Dive

1. Aliasing

- Self joins are impossible without aliases because you reference the same table twice.
- Use meaningful aliases like `child`, `parent`, `manager`, `employee`, etc.

2. Self-Referencing Keys

- A foreign key column (like `managerId`) that points to the same table's primary key (`id`) is called a **self-referencing foreign key**.
- It establishes hierarchical integrity.

3. Hierarchy Traversal

- A single self join reveals **one level** of hierarchy (employee → manager).
- Recursive CTEs (Common Table Expressions) can extend this for **multi-level hierarchies** (employee → manager → director → VP).

Example:

```sql
WITH RECURSIVE hierarchy AS (
    SELECT id, name, managerId, 1 AS level
    FROM Employee
    WHERE managerId IS NULL
    UNION ALL
    SELECT e.id, e.name, e.managerId, h.level + 1
    FROM Employee e
    JOIN hierarchy h ON e.managerId = h.id
)
SELECT * FROM hierarchy;
```

Performance Considerations

- **Indexing:**
  Ensure the self-referencing column (`managerId`) is indexed for efficient joins.
- **Join Complexity:**
  The cost of a self join is similar to any other join — but can become expensive if the dataset is large and recursive traversal is deep.
- **Null Relationships:**
  Use `LEFT JOIN` instead of `INNER JOIN` if the foreign key can be null.

Real-World Examples

1. Organizational Structure

Find all employees who report directly to a manager:

```sql
SELECT e.name, m.name AS manager
FROM Employee e
JOIN Employee m ON e.managerId = m.id;
```

2. Category Hierarchy

For a `Category` table with `parent_id`:

```sql
SELECT c.name AS Category, p.name AS ParentCategory
FROM Category c
LEFT JOIN Category p ON c.parent_id = p.id;
```

3. Product Comparison

Compare products with others in the same category:

```sql
SELECT a.name AS ProductA, b.name AS ProductB
FROM Product a
JOIN Product b
  ON a.category_id = b.category_id
WHERE a.price > b.price;
```

Key Takeaways

- A **self join** joins a table to itself to compare or relate its own rows.
- It is critical for **hierarchical** or **recursive** data relationships.
- Always use **table aliases** to differentiate instances.
- Combine with **recursive CTEs** for multi-level hierarchy traversal.
- Optimize with **indexes** and **appropriate join types**.

Summary Table

Concept	Description
Definition	Join of a table with itself
Key Use Case	Hierarchical or self-referential relationships
Requires Aliases	✅ Yes
Join Condition	Typically a self-referencing key relationship
Common Example	Employee–Manager, Category–Subcategory
Advanced Extension	Recursive CTE for multi-level hierarchies