**📄 Lab Assignment 4 – Data Exploration Using Joins in SQL**

**Introduction:**

Relational Database Management Systems (RDBMS) are central to modern data management and organization. This assignment explores the practical application of SQL joins to retrieve and analyze data spread across multiple related tables in a relational database. By leveraging different types of joins, we can draw meaningful connections between entities, perform multi-table queries, and generate insights that support business analysis and decision-making.

In this assignment, we apply SQL operations on a pre-built relational database system (Company Management System) developed during Lab Assignment 1. The focus is on understanding and performing **JOIN operations**, demonstrating how relational data can be combined, filtered, and aggregated to derive actionable insights.

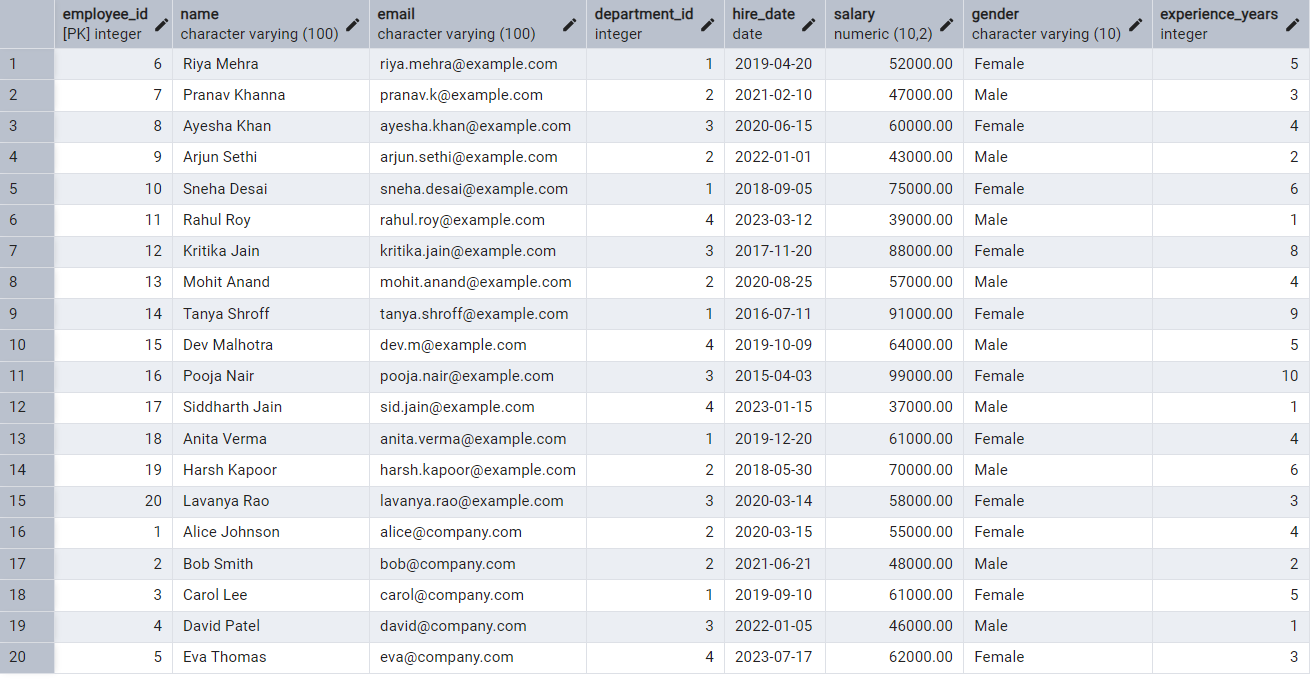
**Assignment Requirements:**

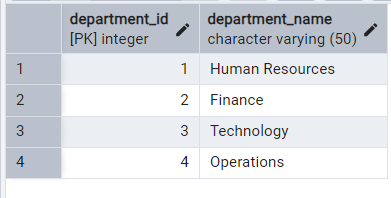
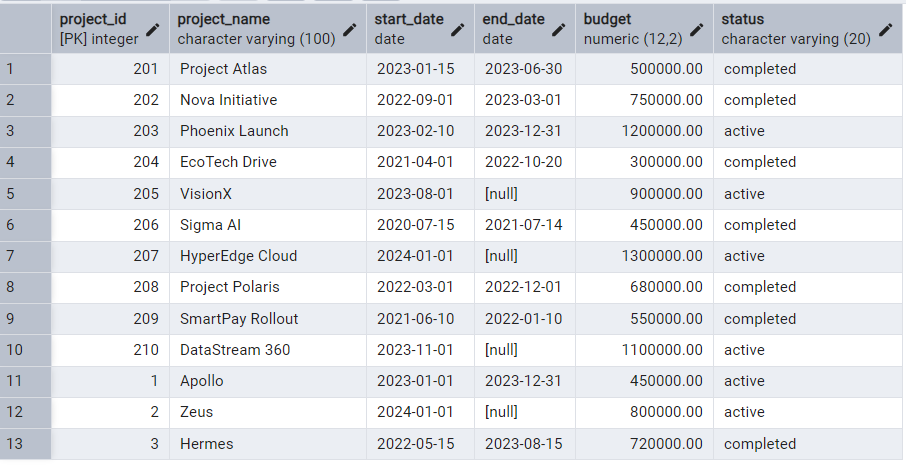
* Perform **data exploration** through SQL by joining tables from the selected relational database system.
* Use a variety of SQL **JOIN operations** (e.g., INNER JOIN, LEFT JOIN) to combine tables.
* Include **aggregate functions** and **grouping** operations where appropriate.
* Each query must be clearly explained in terms of its purpose and output.

We will be working on our company management system database from the previous assignment

Here is the schema overview

| **Table Name** | **Column Name** | **Data Type** | **Description** |
| --- | --- | --- | --- |
| departments | department\_id | INTEGER, NOT NULL | Primary key of the department |
|  | department\_name | VARCHAR(50), NOT NULL | Name of the department |
| employees | employee\_id | INTEGER, NOT NULL | Primary key of the employee |
|  | name | VARCHAR(100) | Full name of the employee |
|  | email | VARCHAR(100) | Email address |
|  | department\_id | INTEGER | Foreign key referencing departments |
|  | hire\_date | DATE | Date of hiring |
|  | salary | NUMERIC(10,2) | Monthly salary |
|  | gender | VARCHAR(10) | Gender of the employee |
|  | experience\_years | INTEGER | Total years of experience |
| projects | project\_id | INTEGER, NOT NULL | Primary key of the project |
|  | project\_name | VARCHAR(100) | Name of the project |
|  | start\_date | DATE | Project start date |
|  | end\_date | DATE | Project end date |
|  | budget | NUMERIC(12,2) | Project budget |
|  | status | VARCHAR(20) | Project status (e.g., ongoing, closed) |
| employee\_project | employee\_id | INTEGER, NOT NULL | FK referencing employees |
|  | project\_id | INTEGER, NOT NULL | FK referencing projects |
|  | role | VARCHAR(50) | Role of the employee in the project |
| salaries | salary\_id | INTEGER, NOT NULL | Primary key of the salary record |
|  | employee\_id | INTEGER | FK referencing employees |
|  | base\_salary | NUMERIC(10,2) | Basic monthly salary |
|  | bonus | NUMERIC(10,2) | Bonus amount |
|  | pay\_date | DATE | Date of payment |

Table overview



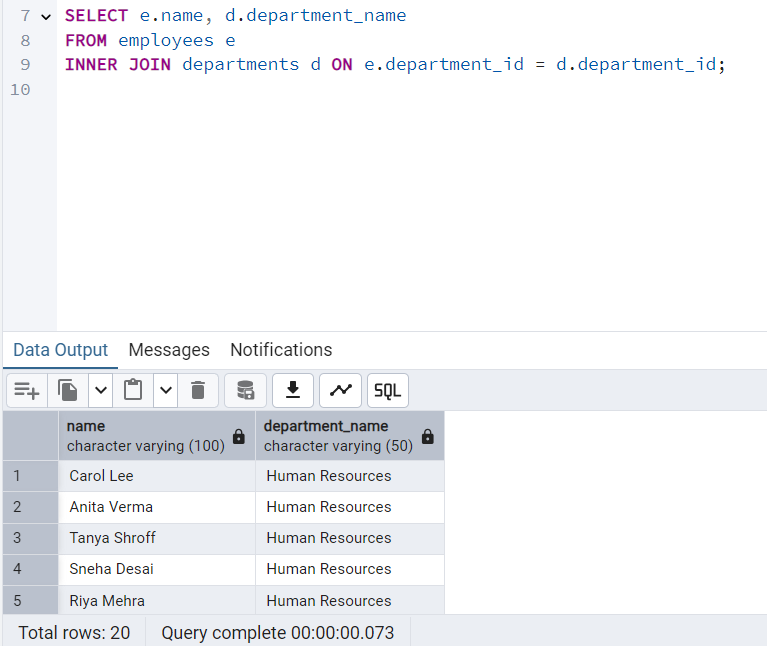


**🔗 Types of SQL Joins – Explained with Relevance to Our Schema**

In relational databases, **JOIN operations** are essential for combining records from two or more tables based on related columns. Our **Company Management System** database uses foreign key relationships (e.g., department\_id, employee\_id, project\_id) to connect entities. Here's how different types of JOINs operate and interact with our schema:

**1. INNER JOIN**

* **Description:** Returns only the rows where there is a match in **both** tables.
* **Use Case in Our DB:** Fetch employee details **with a valid department**.



* **Output:** Only employees who are assigned to a department.

**2. LEFT JOIN (or LEFT OUTER JOIN)**

* **Description:** Returns **all records from the left table**, and matched records from the right. If no match, NULLs are returned for the right table.
* **Use Case in Our DB:** List **all employees**, even those not assigned to any department.

SELECT e.name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id;

**3. RIGHT JOIN (or RIGHT OUTER JOIN)**

* **Description:** Opposite of LEFT JOIN. Returns **all records from the right table**, with matching records from the left.
* **Use Case in Our DB:** List **all departments**, even those with **no employees** assigned.
* **Example:**

SELECT d.department\_name, e.name

FROM employees e

RIGHT JOIN departments d ON e.department\_id = d.department\_id;

* **Output:** Includes departments without employees (employee name will be NULL).



**4. FULL OUTER JOIN**

* **Description:** Returns **all records** from both tables, with NULLs where there is no match.
* **Use Case in Our DB:** Audit mismatches like unassigned employees and empty departments.
* **Example:**

SELECT e.name, d.department\_name

FROM employees e

FULL OUTER JOIN departments d ON e.department\_id = d.department\_id;



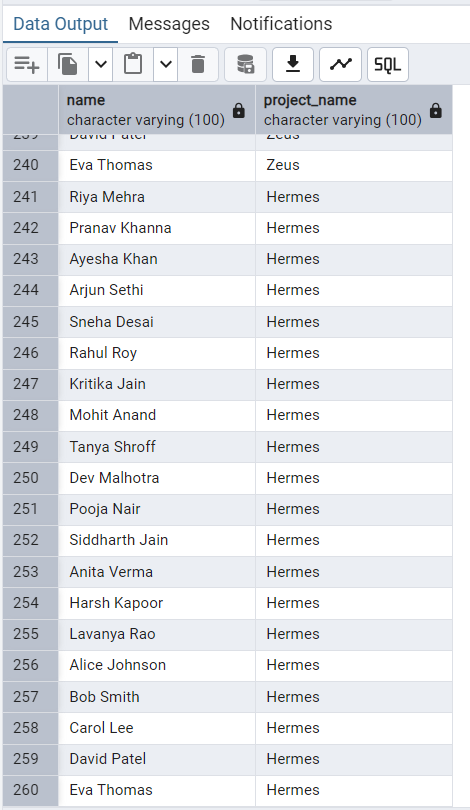
**5. CROSS JOIN**

* **Description:** Returns the **Cartesian product** of two tables — every row of the first with every row of the second.
* **Use Case in Our DB:** Not common — but useful for generating pairing possibilities like project-team matchups.
* **Example:**

SELECT e.name, p.project\_name

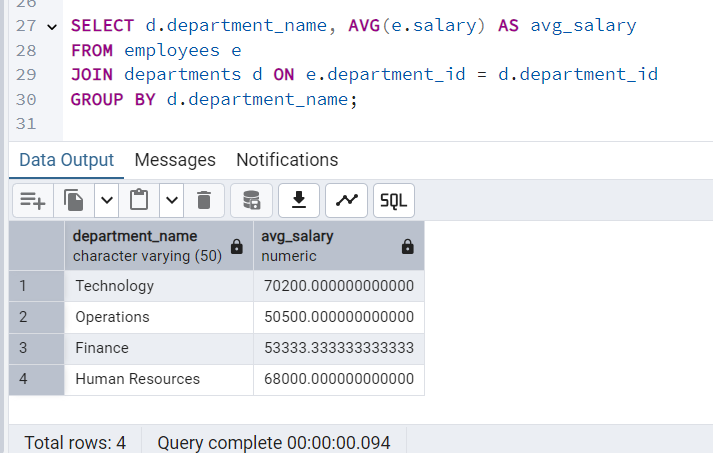
FROM employees e

CROSS JOIN projects p;



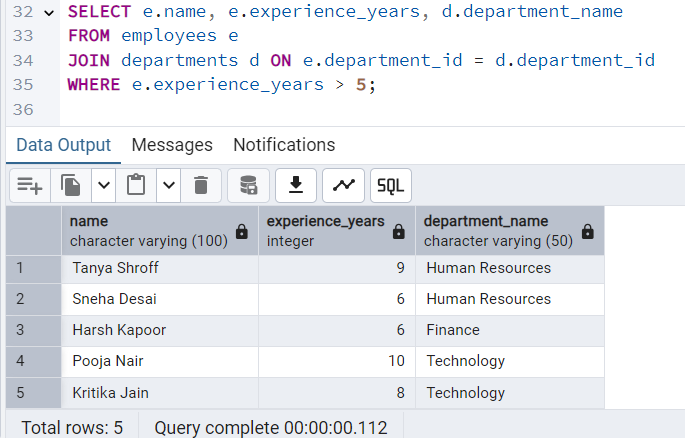
Note here: cross join produces 260 results it is the cartesian product of the 2 tables

10 employees26 projects,Then:10 employees × 26 projects = 260 rows

1. **JOIN with Aggregate Function**

**2. JOIN + Filtering with WHERE**

List employees who are in a department and have **more than 5 years** of experience:



**3. Multi-level JOIN (3 Tables)**

List all employees, their department, and the projects they are working on:

SELECT e.name, d.department\_name, p.project\_name

FROM employees e

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

JOIN employee\_project ep ON e.employee\_id = ep.employee\_id

JOIN projects p ON ep.project\_id = p.project\_id;

**💡 4. LEFT JOIN + NULL Filter**

Find employees who are **not assigned to any project**:

sql

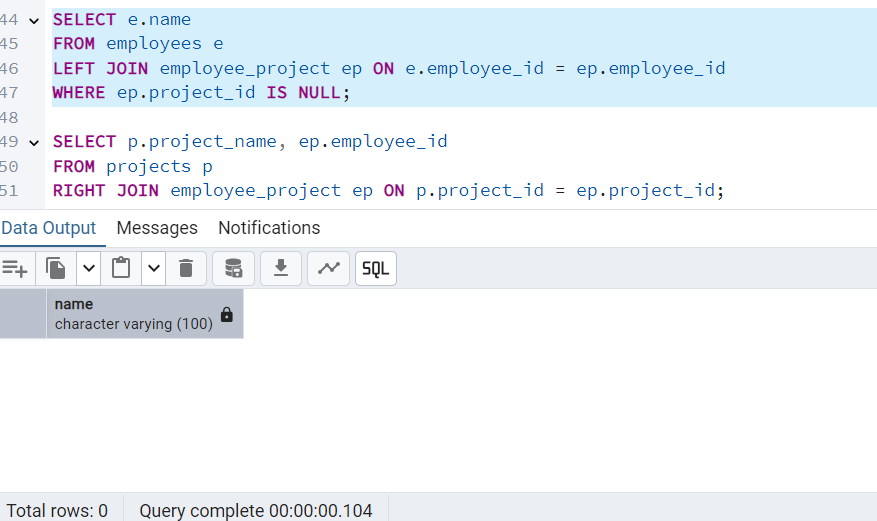
CopyEdit

SELECT e.name

FROM employees e

LEFT JOIN employee\_project ep ON e.employee\_id = ep.employee\_id

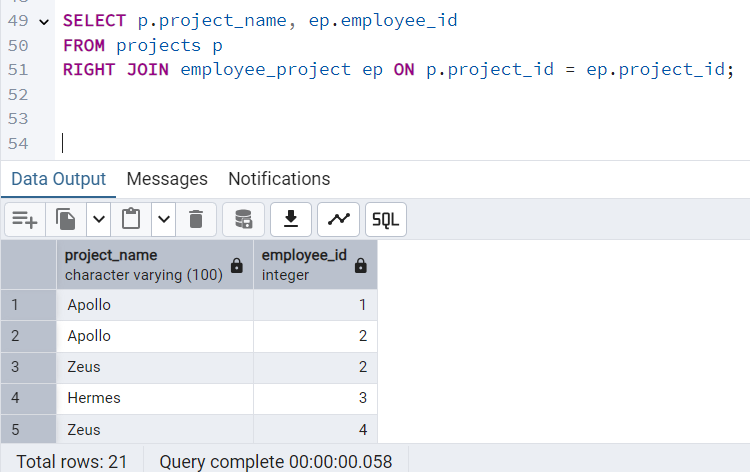
WHERE ep.project\_id IS NULL;



No cases of unassigned employees

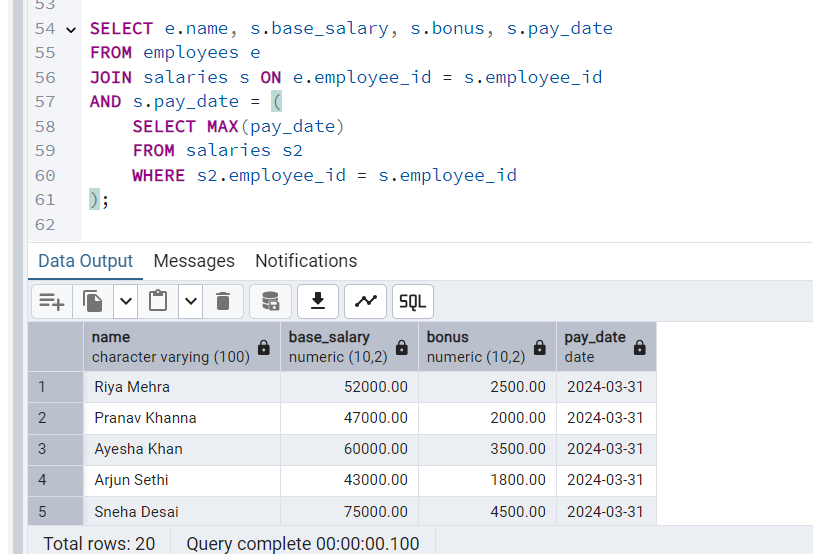
**5. RIGHT JOIN + Filter**

List all projects, including those with **no employees assigned**:

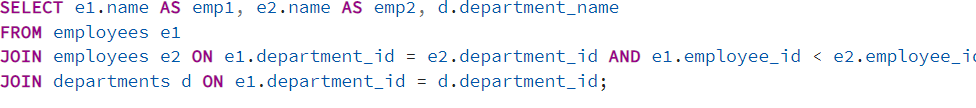


**6. JOIN with Subquery in ON Clause**

Get latest pay for each employee (from salaries) and join with employee details:



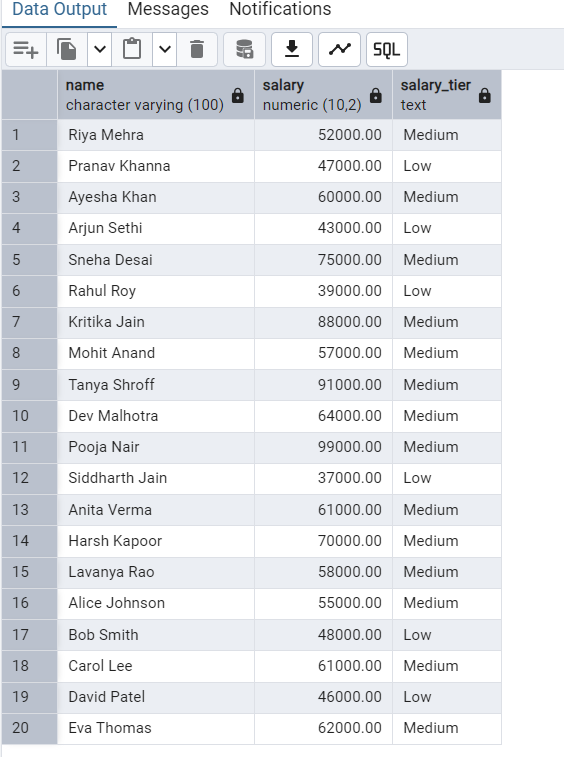
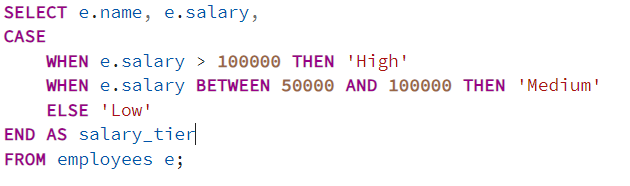
**7. Self JOIN**

Find pairs of employees in the same department:



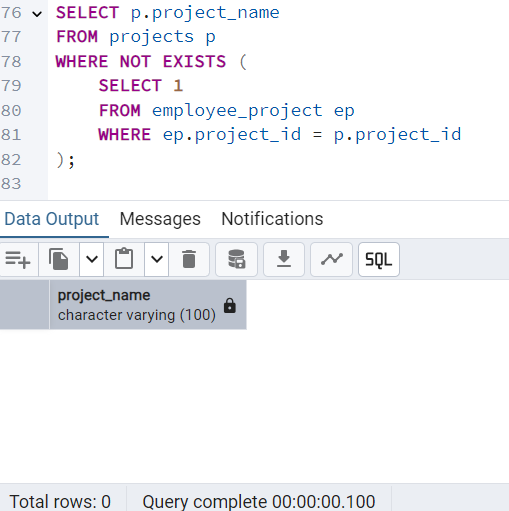
**8. JOIN with CASE Statement**

Show employees and classify them based on salary tiers:



**9. Anti-Join (NOT EXISTS / NOT IN)**

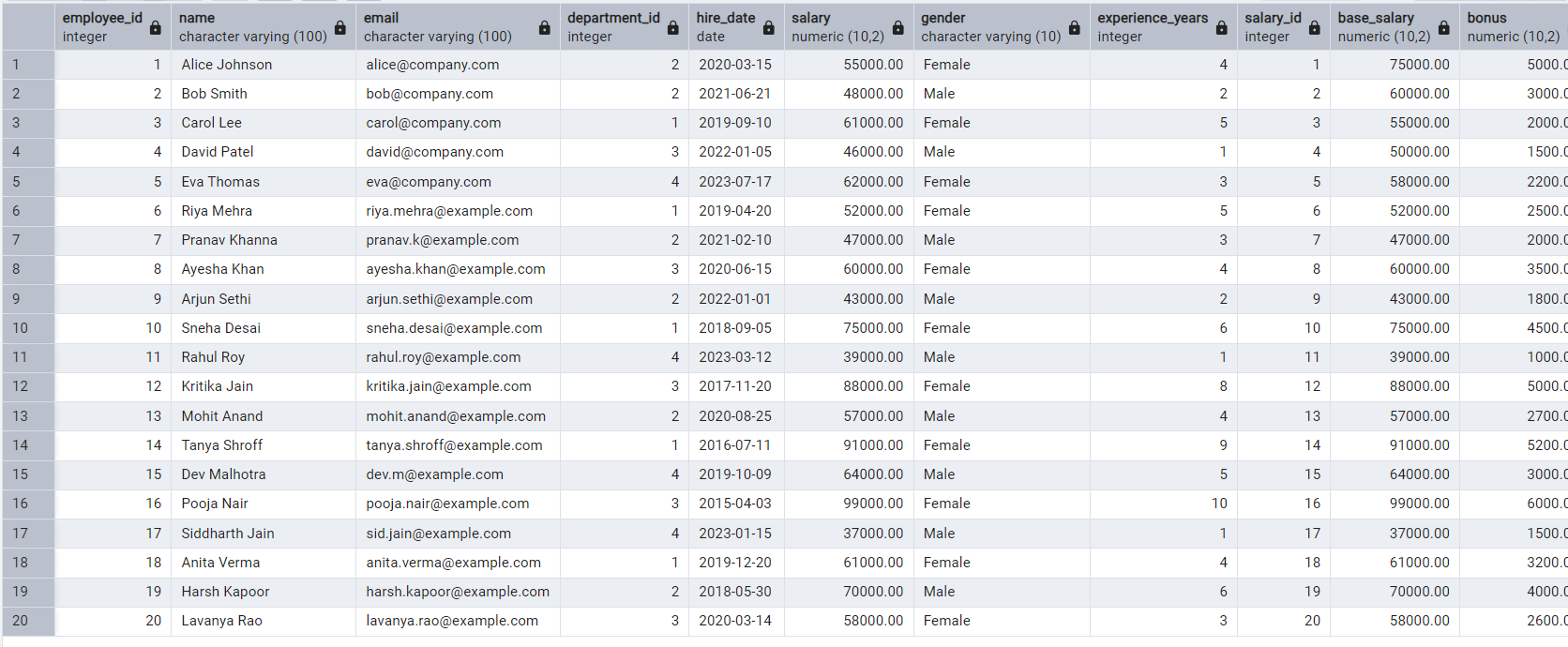
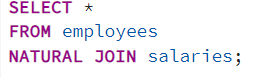
List all projects that **have no employees assigned**:



No such employee exist, all working

**10. Natural JOIN**

If employee\_id is the same in both:



**✅ Conclusion – Lab Assignment 4**

In this assignment, we explored the power of **SQL JOIN operations** within a relational database context, using a Company Management System as our foundation. Through various types of JOINs—such as INNER JOIN, LEFT JOIN, RIGHT JOIN, and even CROSS JOIN—we successfully demonstrated how interconnected tables can be combined to extract meaningful, real-world insights.

We analyzed the relationships between employees, departments, and projects, and used advanced filtering techniques to:

* Identify employees not assigned to any project
* Determine departments with or without assigned employees
* Merge and aggregate multi-table data for clearer business understanding

While some queries returned empty results, these outcomes were themselves insightful, reflecting the **integrity and completeness of the existing database**. They also illustrated how JOINs can be used not just for combining data, but for **identifying data gaps or confirming relational consistency**.