**SQL Project – Employee management system**

**Ross Dipple – 09/09/2024**

Table of Contents

Introduction 2

Requirements 2

Tables and Attributes 5

Relationships 7

Future steps 7

Table creation 8

Data insertion 12

Query building and testing 20

Conclusion 24

**1. Introduction**

The **Employee Management System (EMS)** is designed to efficiently manage essential information about employees, departments, job titles, projects, and salaries within an organisation.

The system will provide essential functionalities such as:

* Storing employee personal details and work information.
* Assigning employees to different departments and job titles.
* Managing the salary history and recording pay changes over time.
* Tracking employee participation in various projects.

This project is built using **PostgreSQL**

The purpose of this project is twofold:

1. **Practical Application**

To develop a functional database system that meets the typical needs of an organisation’s HR department, specifically around employee data management.

1. **Skills Demonstration**

To showcase proficiency in key PostgreSQL concepts, including but not limited to:

* + **Database Design**
  + **SQL Querying**
  + **Performance Optimization**
  + **Advanced PostgreSQL Features**:
  + **Data Integrity**

**2. Requirements**

The Employee Management System (EMS) must fulfil the following requirements to ensure comprehensive management of employee data within an organisation:

**1. Employee Management**

* The system should store personal and professional information about employees, including:
  + Full name
  + Contact details (email and phone number)
  + Date of hire
  + Current salary
  + Job title and department
* Each employee should have a unique identifier (employee ID) for database integrity.

**2. Department Management**

* The system should support multiple departments within the organisation (e.g., Human Resources, IT, Marketing).
* Each department should have a unique identifier and name.
* Employees must be assigned to one department at a time, but departments can contain multiple employees.

**3. Job Title Management**

* Employees must be assigned a job title (e.g., Software Engineer, Marketing Manager).
* The system should allow for multiple employees to share the same job title.
* Job titles should have a unique identifier.

**4. Project Management**

* The system must track employee assignments to projects.
* A single employee can be assigned to multiple projects, and each project may have multiple employees.
* Project data should include:
  + Project name
  + Start and end dates
  + Assigned employees
* Employee assignments to projects should include the date they were assigned.

**5. Salary History**

* The system must track changes to employee salaries over time.
* A salary history must be maintained, recording the effective date of each salary change.
* Employees can have multiple salary records, but only one salary is active at any given time.

**6. Data Integrity and Relationships**

* Ensure that all data relationships are properly maintained:
  + Employees should always belong to a valid department.
  + Job titles and departments should be consistently assigned to employees.
  + Employee-project assignments should reflect actual project involvement.
* Constraints should be applied to ensure referential integrity between tables (e.g., foreign keys between employees, departments, projects, and job titles).

**7. Performance Considerations**

* The system must allow for efficient querying, especially for retrieving:
  + Employees by department or job title
  + Project assignments and employee contributions
  + Salary history for each employee
* Indexing should be implemented on frequently queried fields such as employee ID, department ID, and project ID to optimise performance.

**3. Tables and Attributes**

**1. Employee**

* The Employee table stores all personal and professional information about each employee in the organisation.
* **Attributes:**
  + employee\_id (Primary Key): **SERIAL** – A unique identifier for each employee (auto-incrementing integer).
  + first\_name: **VARCHAR(50)** – The employee’s first name.
  + last\_name: **VARCHAR(50)** – The employee’s last name.
  + email: **VARCHAR(100)** – The employee’s email address for professional communication.
  + phone\_number: **VARCHAR(20)** – The employee’s contact phone number.
  + hire\_date: **DATE** – The date the employee was hired by the organisation.
  + salary: **NUMERIC(10, 2)** – The employee’s current salary, allowing for up to 10 digits with 2 decimal places.
  + department\_id (Foreign Key): **INT** – A reference to the employee’s department in the Department table.
  + job\_title\_id (Foreign Key): **INT** – A reference to the employee’s job title in the Job Title table.

**2. Department**

* The Department table defines the various departments within the organisation.
* **Attributes:**
  + department\_id (Primary Key): **SERIAL** – A unique identifier for each department (auto-incrementing integer).
  + department\_name: **VARCHAR(100)** – The name of the department (e.g., IT, HR, Marketing).

**3. Job Title**

* The Job Title table stores information about the different job titles assigned to employees.
* **Attributes:**
  + job\_title\_id (Primary Key): **SERIAL** – A unique identifier for each job title (auto-incrementing integer).
  + job\_title\_name: **VARCHAR(100)** – The title or position of an employee (e.g., Software Engineer, HR Manager).

**4. Project**

* The Project table records details about the various projects to which employees are assigned.
* **Attributes:**
  + project\_id (Primary Key): **SERIAL** – A unique identifier for each project (auto-incrementing integer).
  + project\_name: **VARCHAR(150)** – The name of the project (e.g., Website Redesign, Marketing Campaign).
  + start\_date: **DATE** – The start date of the project.
  + end\_date: **DATE** – The expected or actual end date of the project.

**5. Employee\_Project**

* The Employee\_Project table manages the many-to-many relationship between employees and projects.
* **Attributes:**
  + employee\_id (Foreign Key): **INT** – A reference to the employee in the Employee table.
  + project\_id (Foreign Key): **INT** – A reference to the project in the Project table.
  + assigned\_date: **DATE** – The date on which the employee was assigned to the project.

**6. Salary\_History**

* The Salary\_History table maintains a historical record of salary changes for each employee.
* **Attributes:**
  + salary\_id (Primary Key): **SERIAL** – A unique identifier for each salary record (auto-incrementing integer).
  + employee\_id (Foreign Key): **INT** – A reference to the employee in the Employee table.
  + salary: **NUMERIC(10, 2)** – The salary amount for that specific period, allowing for up to 10 digits with 2 decimal places.
  + effective\_date: **DATE** – The date on which the salary change took effect.

NOTE: I will be using the NOT NULL constraint for various columns where I think it is essential for data integrity, however I will leave some columns without a NOT NULL where it could seem appropriate to have them. The reason for this is practicality and application in the real world, where someone charged with inputting data may wish to begin an entry to log some essential details, but can then come back and fill in the rest of these columns later when further information has been acquired. This may be practical in both time efficiency and readiness of company information available, particularly if it is for a very large company, so it makes to leave certain ‘non-essential’ columns without the constraint however subjective that may be.

**4. Relationships**

### **Summary of Key Relationships between tables:**

* **Employee and Department**: An employee belongs to one department, but a department can have many employees (one-to-many).
* **Employee and Job Title**: Each employee has one job title, but a job title can be assigned to multiple employees (one-to-many).
* **Employee and Project**: An employee can be assigned to many projects, and a project can have many employees (many-to-many, managed through Employee\_Project).
* **Employee and Salary History**: Each employee can have multiple salary records over time (one-to-many).
* **Employee** acts as the central link between **Department**, **Job Title**, **Salary\_History**, and **Project**.
* There are **no direct relationships** between tables like **Department** and **Job Title** or **Department** and **Project**. They are all connected through the **Employee** table.

**5. Future Steps**

* The next steps for this project include the following:
  + Table creation
  + Inserting data
  + Query building
  + Finalised documentation and uploading to GitHub

**6. Table Creation**

**1. Employee table**

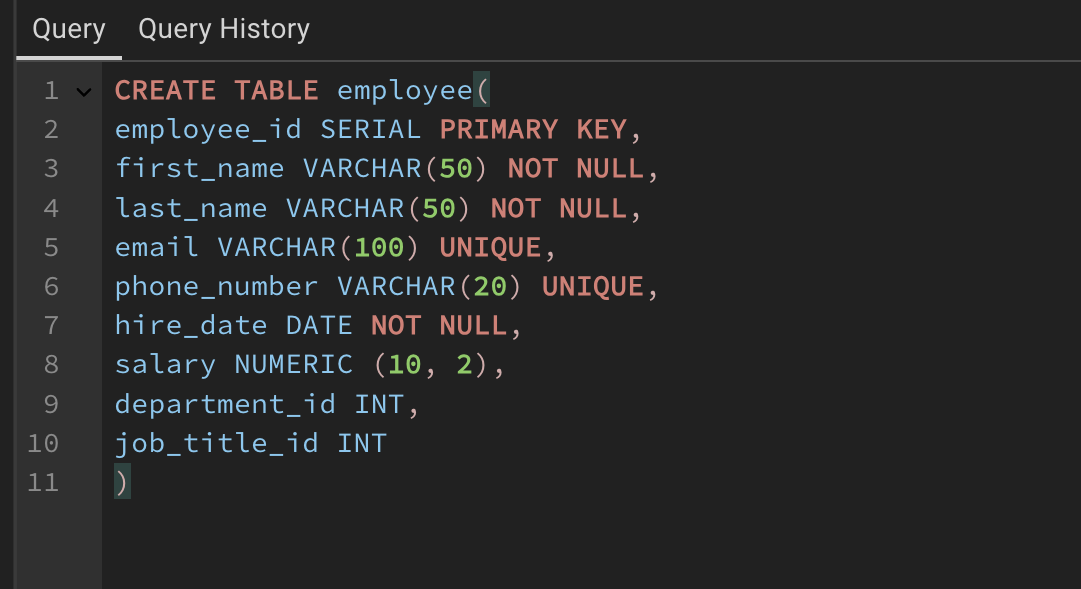
**Purpose:**

The Employee table holds the primary details of each employee in the system, including their unique identifier (employee\_id), name, department, job title, and date of hire. This table is central to the system, as most other tables reference it.

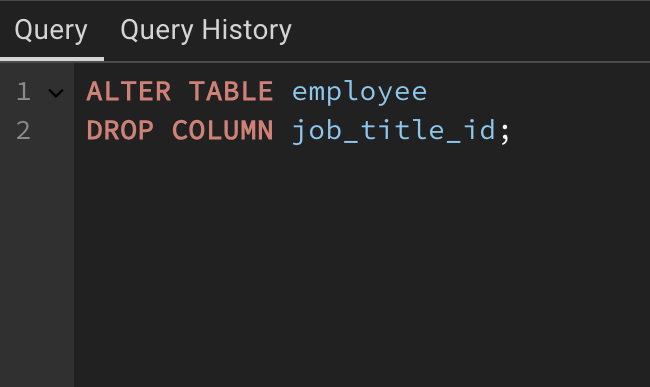
**Key Parts:**

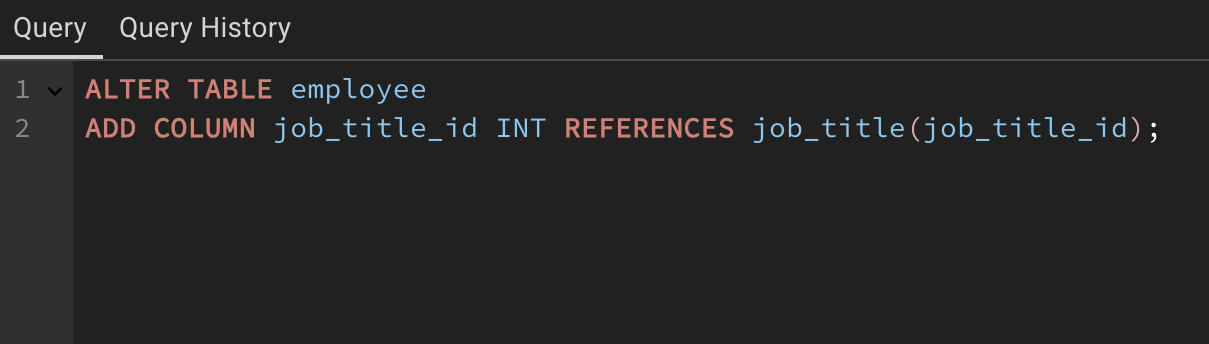
* Primary Key: employee\_id
* Foreign Keys: department\_id, job\_title\_id — these establish relationships with the Department and Job Title tables.
* Constraints: Ensures data integrity by restricting the possible values for certain fields (e.g., a valid department and job title must be assigned to each employee).

**SQL script:**



Note: I noticed after table creation that I forgot to reference the department\_id and job\_title\_id as foreign keys, so I dropped those columns from the table using an ALTER clause and then added them as new columns with the correct reference constraints, as demonstrated on the job\_title\_id column below:

****

****

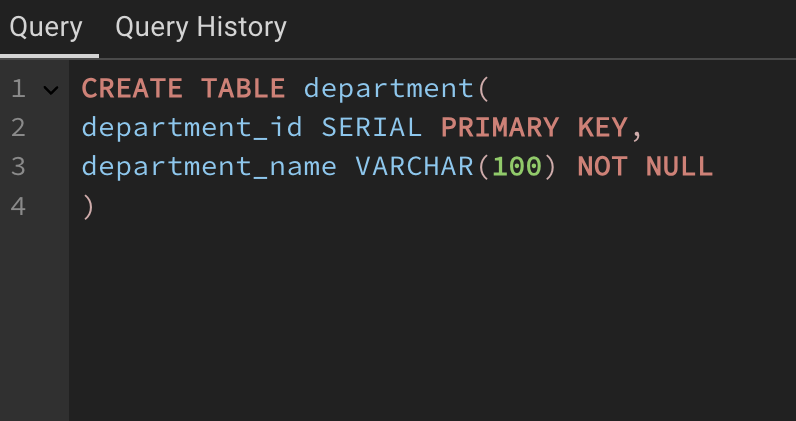
**2. Department table  
  
Purpose:**

The Department table holds information about the various departments within the organisation. Each department can have many employees assigned to it.

**Key Parts:**

* Primary Key: department\_id
* One-to-Many Relationship with Employee: Each department can have multiple employees, but each employee belongs to only one department.

**SQL script:**



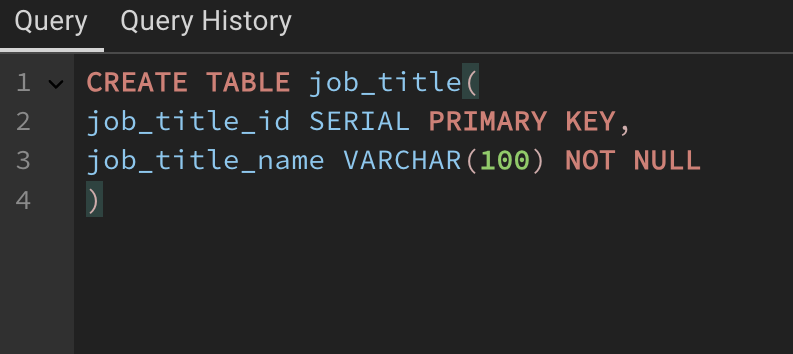
**3. Job Title table  
  
Purpose:**

The Job Title table stores all possible job titles within the company. Multiple employees can share the same job title, creating a one-to-many relationship.

**Key Parts:**

* Primary Key: job\_title\_id
* One-to-Many Relationship with Employee: One job title can apply to many employees, while each employee has only one job title.

**SQL Script:**

****

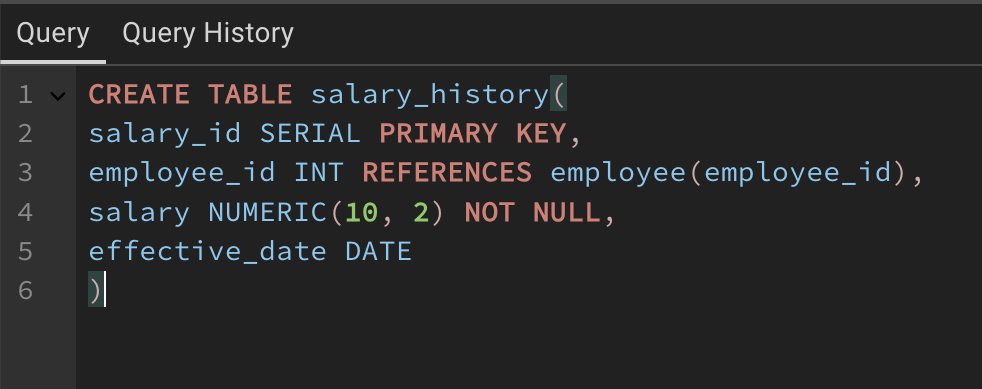
**4. Salary History table  
  
Purpose:**

The Salary\_History table keeps track of the salary changes for each employee over time, storing when a salary becomes effective and the new salary amount. It helps maintain a record of employee salary progression.

**Key Parts:**

* Primary Key: salary\_id
* Foreign Key: employee\_id — links the salary record to the specific employee.
* One-to-Many Relationship with Employee: One employee can have multiple salary history records, but each record refers to a single employee.
* Constraints: Ensures salaries are recorded with a valid employee and a valid date.

**SQL Script:**



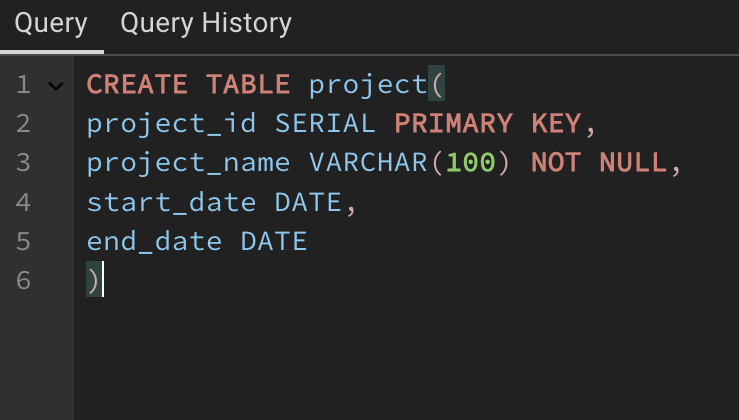
**5. Project table   
  
Purpose:**

The Project table holds details about all the projects employees are assigned to. Projects are not linked directly to employees but through the Employee\_Project intersection table.

**Key Parts:**

* Primary Key: project\_id
* Many-to-Many Relationship with Employee: Employees can work on multiple projects, and projects can have multiple employees, managed through the Employee\_Project table.

**SQL Script:**



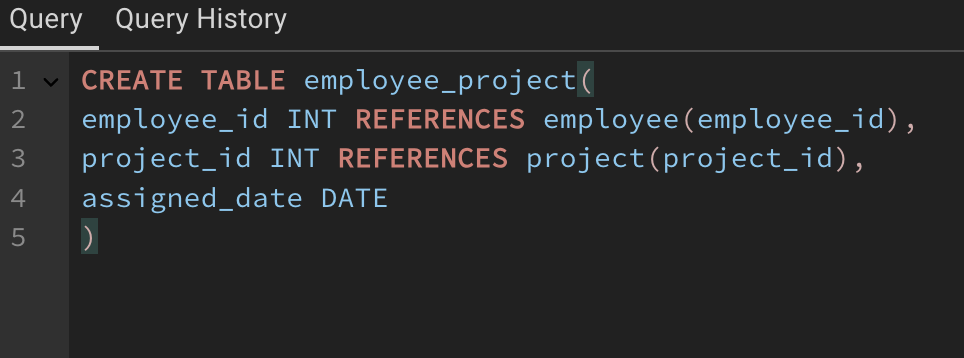
**6. Employee project table  
  
Purpose:**

The Employee\_Project table is an intersection table that links employees to the projects they work on. It manages the many-to-many relationship between the Employee and Project tables.

**Key Parts:**

* Primary Key: Composite primary key consisting of employee\_id and project\_id.
* Foreign Keys: employee\_id and project\_id reference the Employee and Project tables, respectively.
* Constraints: Ensures that each combination of employee and project is unique, meaning an employee can’t be assigned to the same project more than once.

**SQL Script:**



**7. Inserting Data**

**Data Insertion Plan:**

**Department Table (3-5 records)**

Insert a few departments like HR, Sales, etc.

**Job Title Table** (3-5 records)  
Add different job titles to cover various roles within departments.

**Employee Table** (5-10 records)  
Insert employees, each belonging to a department and having a job title. Ensure employees are spread across multiple departments and job titles.

NOTE: Will use UPDATE clause to obtain salary column info from salary\_history table once data has been inserted.

**Project Table** (3-5 records)  
Add a few projects that employees can work on, with start and end dates. Will add Website Design, App Development and New Year Sales Campaign.

**Employee\_Project Table** (8-10 records)  
Assign employees to multiple projects to demonstrate the many-to-many relationship. Each employee should work on at least one project, and some employees should be on multiple projects. Project allocation explained below:

**Project: Website Design**

* Shane McMahon (Chief Financial Officer): Overseeing financial and strategic decisions related to the project.
* Scott Hall (Head Designer): Leading the creative and visual aspects of the website design.
* Dallas Page (Junior Designer): Assisting Scott Hall with design-related tasks for the website.
* Liv Morgan (Sales Representative): Contributing from a sales perspective, possibly advising on the user experience and customer-facing elements.
* Hunter Helmsley (Sales Representative): Assisting with aspects of the project that may impact customer engagement and website sales strategies.

**Project: App Development**

* Shane McMahon (Chief Financial Officer): Ensuring budget adherence and overseeing the project at a high level.
* Scott Hall (Head Designer): Leading the design of the app's user interface and user experience.
* Dallas Page (Junior Designer): Assisting Scott Hall in developing the design and UI components for the app.
* Liv Morgan (Sales Representative): Providing insights from a sales and customer engagement perspective, helping design a user-friendly experience.
* Hunter Helmsley (Sales Representative): Offering sales-driven insights into the app development process, likely focused on customer interaction features.

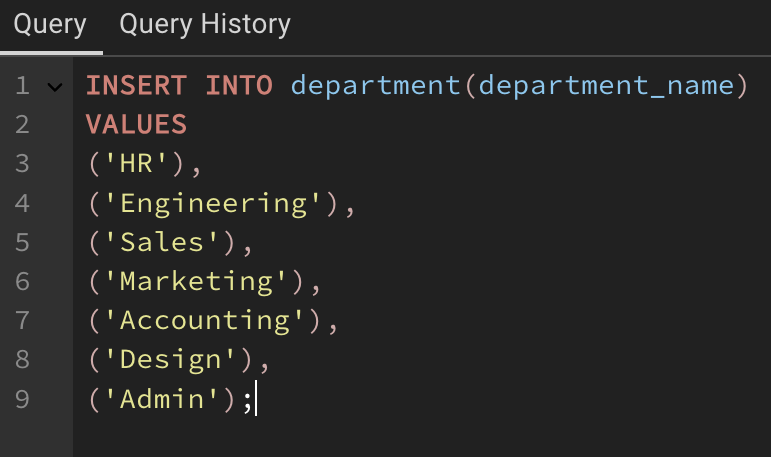
**Project: New Year Sales Campaign**

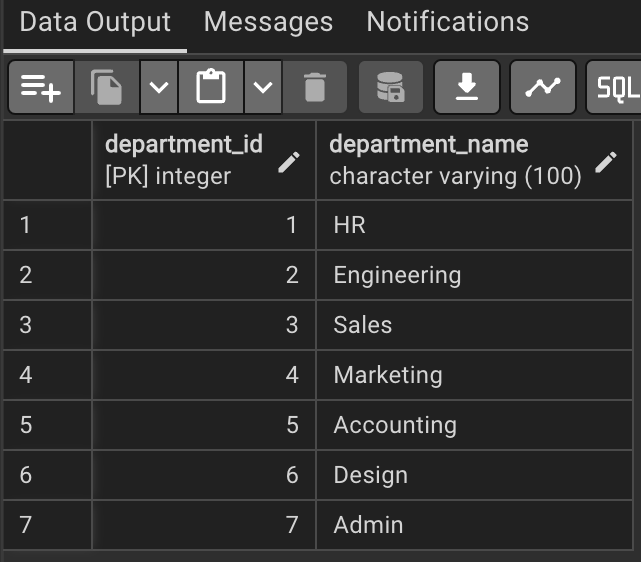
* Stephanie McMahon (Secretary): Likely involved in campaign coordination and administrative support.
* Bill Goldberg (Marketing Specialist): Working on campaign execution and promotional strategies for the sales event.
* Charlotte Flair (Marketing Specialist): Collaborating with Bill Goldberg on the marketing and promotional aspects of the campaign.

**Salary\_History Table** (8-10 records)  
Insert multiple salary records for each employee, showing salary progression over time.

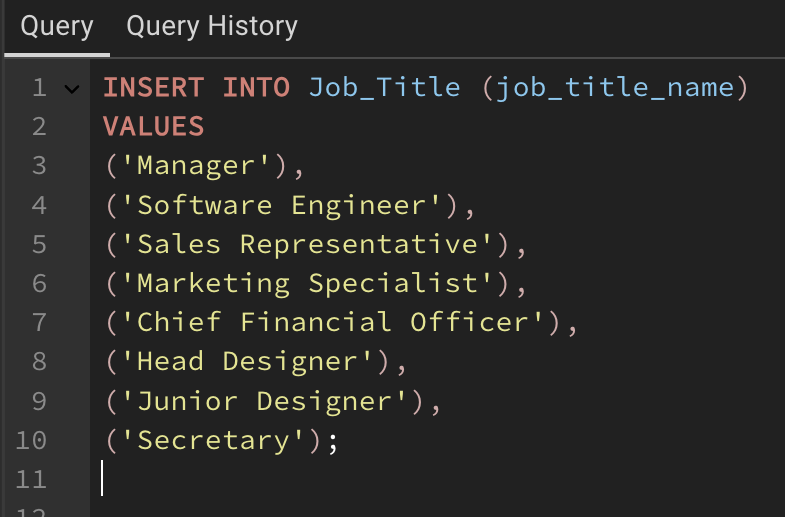
**Data insertion SQL scripts and outputs:**

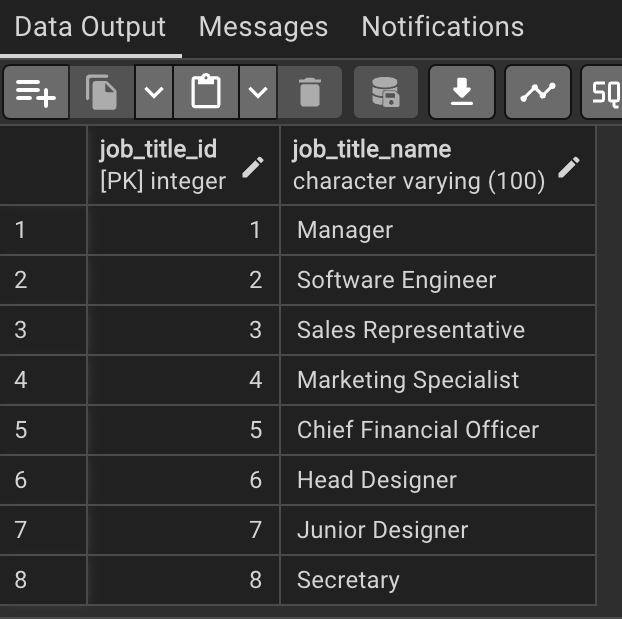
**Department Table:**



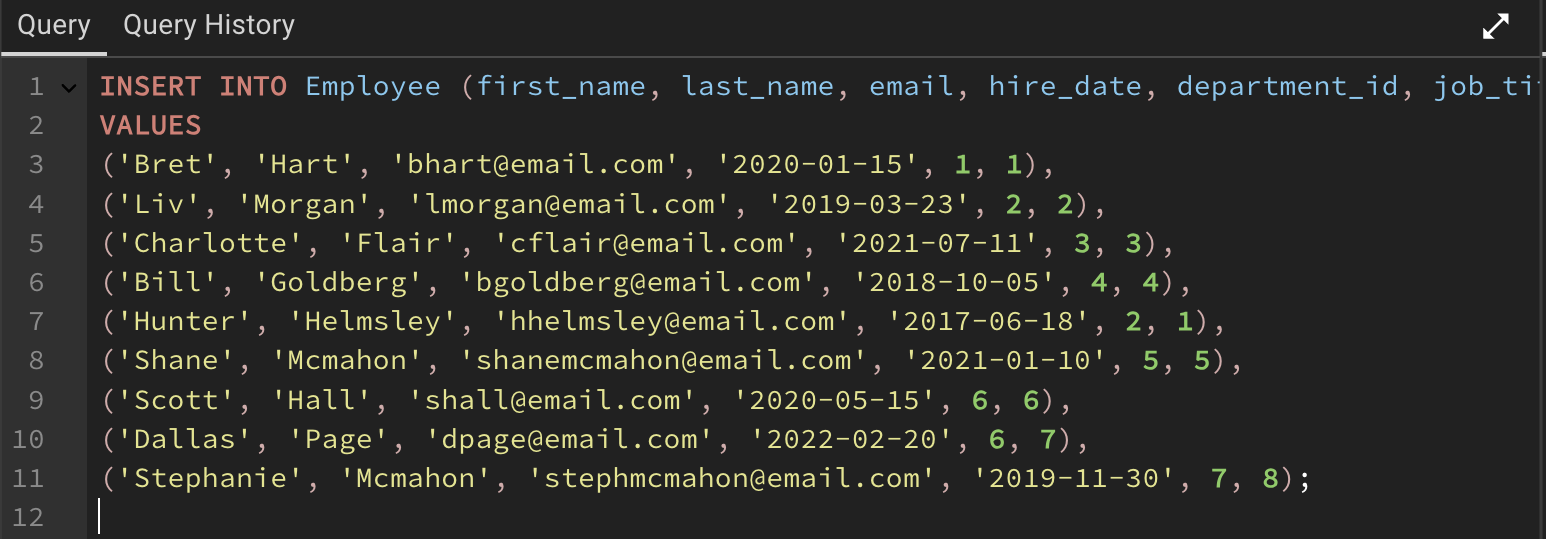


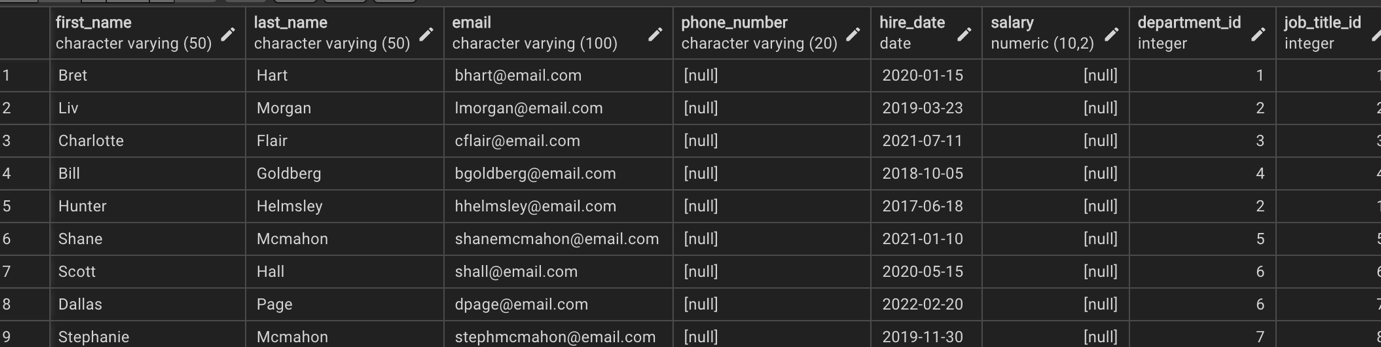
**Job Title Table:**



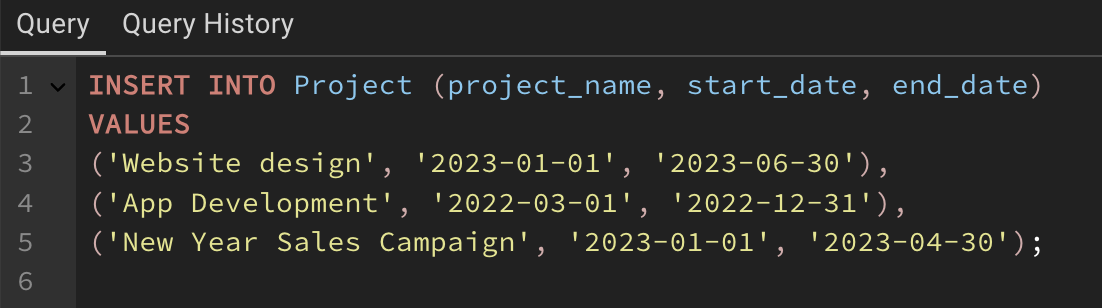


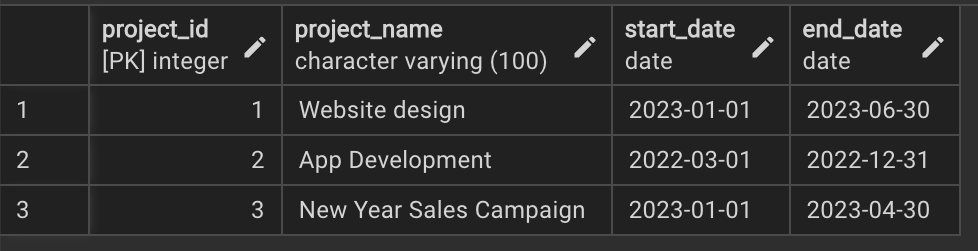
**Employee Table:**



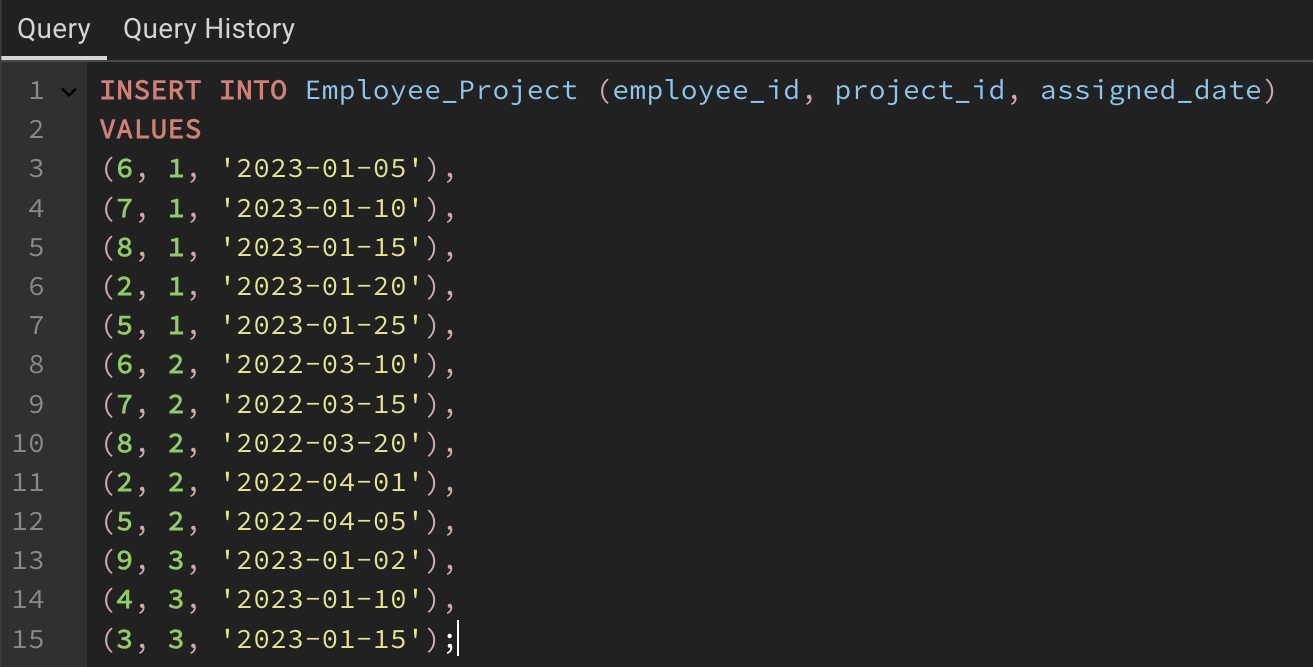


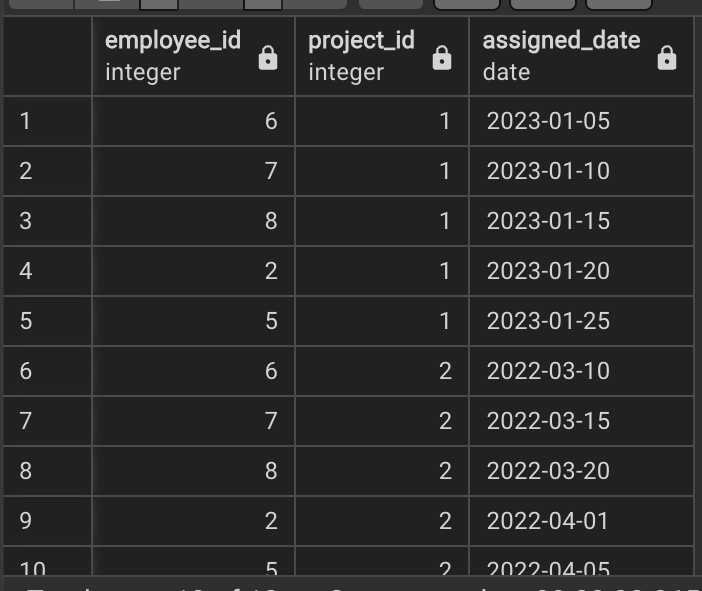
**Project Table:**



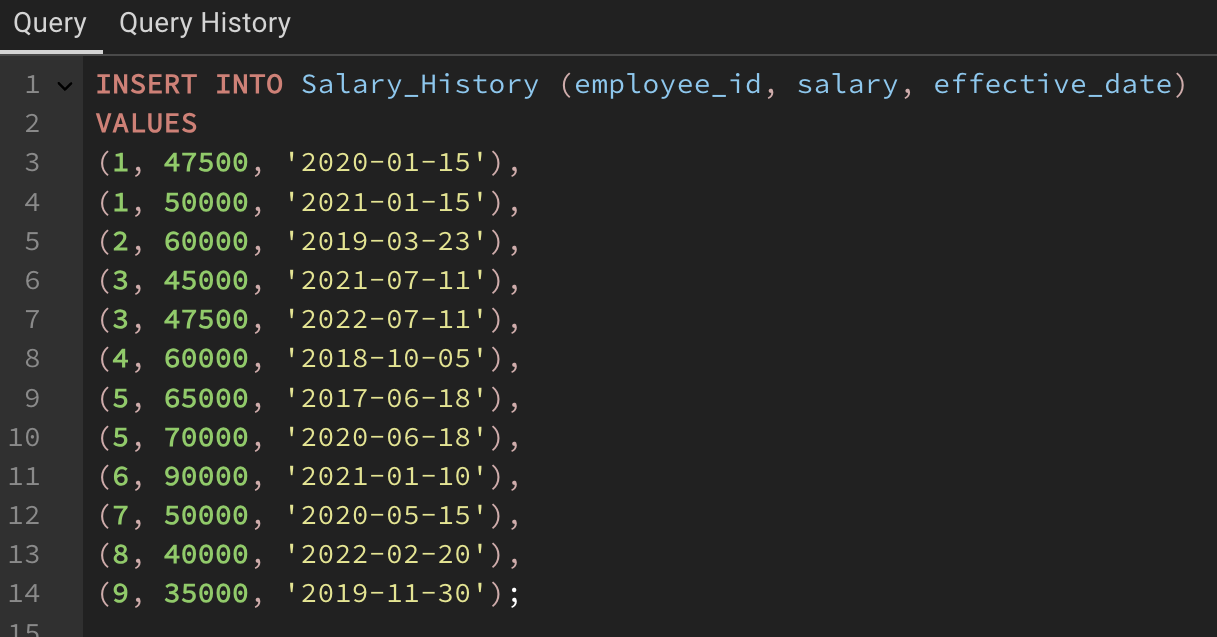


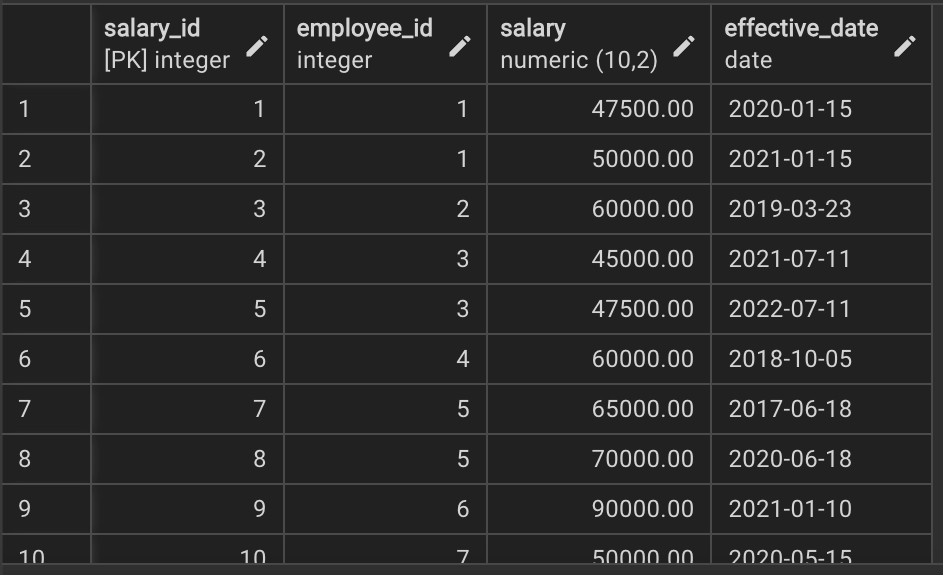
**Employee project table:**

****



**Salary history table:**



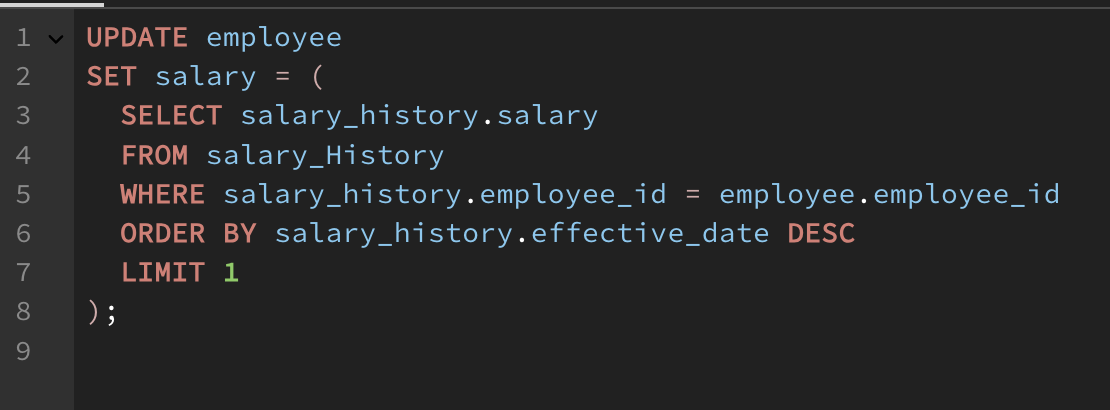


**Updating the Salary column in the Employee Table:**

Since the salary history for each employee is now in place, I can run the query to update the salary column in the Employee table based on the latest salary from Salary\_History.  
This will ensure that the Employee table is updated with each employee's current salary based on the most recent record in the Salary\_History table.

I will do this using UPDATE, SET and a SELECT FROM WHERE statement, also implementing ORDER BY to make sure it takes the most recent value from those who have had their salary change.

SQL script:



**8. Query building and testing**

Now that the database is filled with some data, I will build and test some basic and more advanced queries to showcase and check the functionality of the system.

The queries I will run:

**Basic queries:**

* Retrieve basic employee information
* List all departments and their employees

**Intermediate queries:**

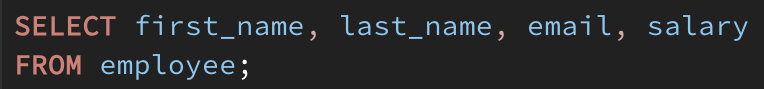
* Show which employees are working on a specific project
* Show employee salary history

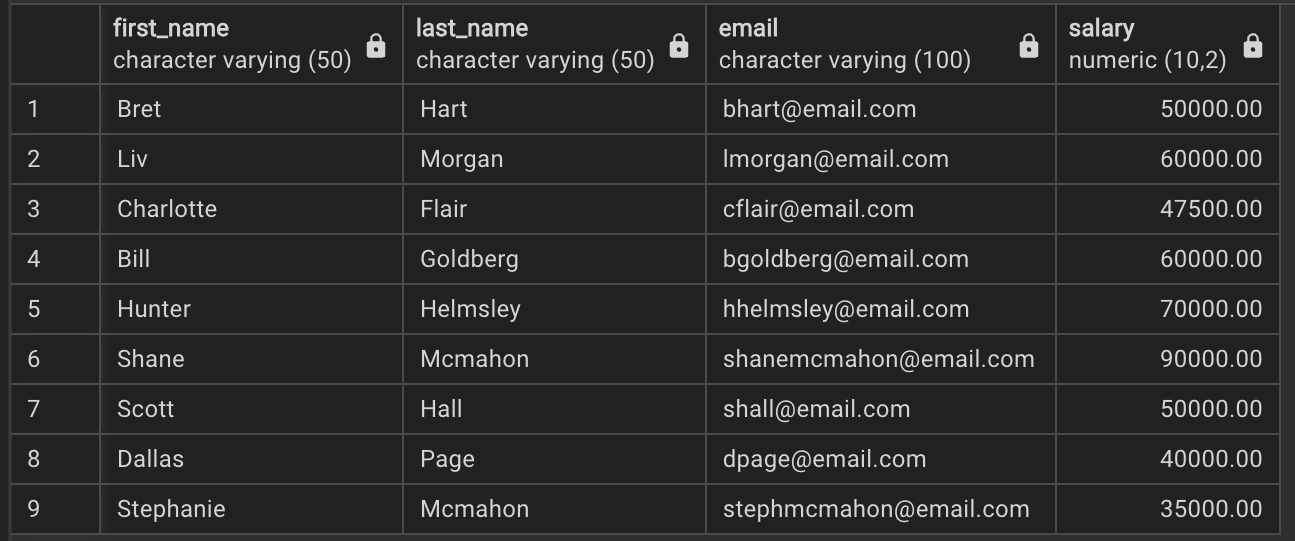
**Advanced queries:**

* Show employees with multiple projects
* Show salary expenses for each department

**Basic queries:**

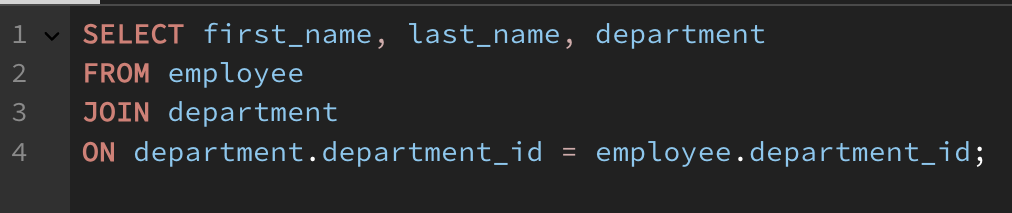
**Basic query 1 - Retrieve basic employee information:**

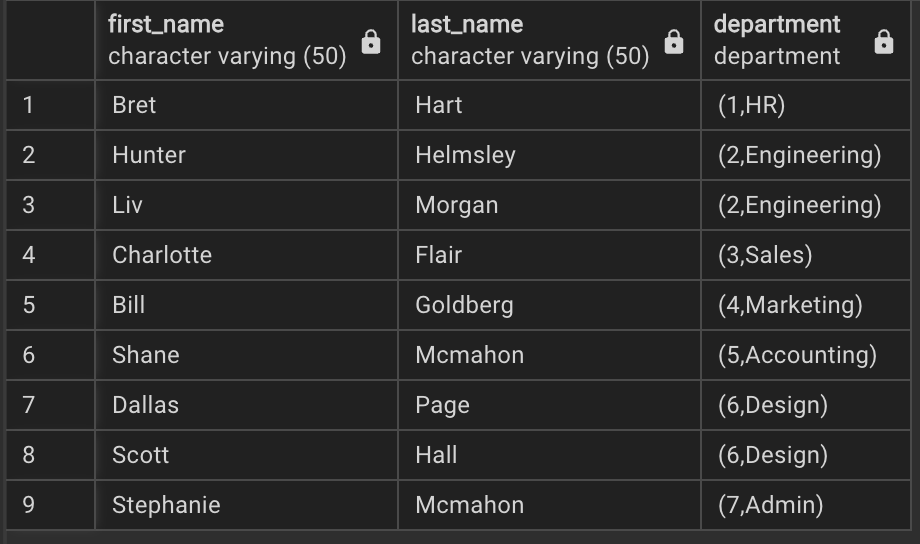
****

****

**Summary**: Retrieve basic employee details using SELECT and FROM,

**Basic query 2 - List all departments and their employees:**

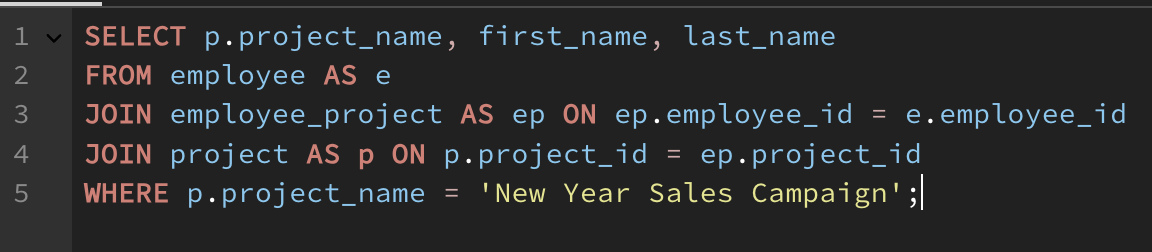
****

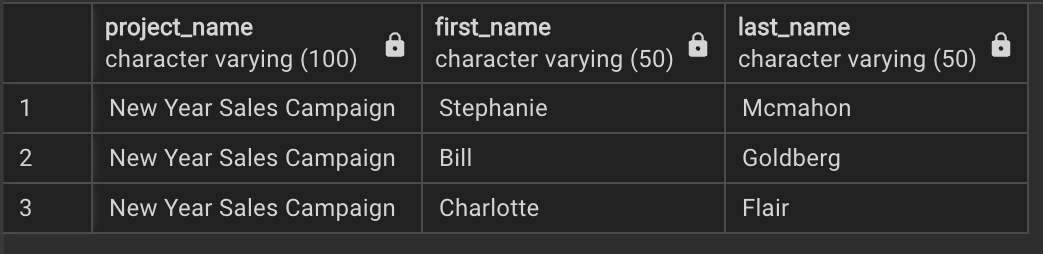
****

**Summary**: Demonstrate a simple JOIN to list employees by department.

**Intermediate queries:**

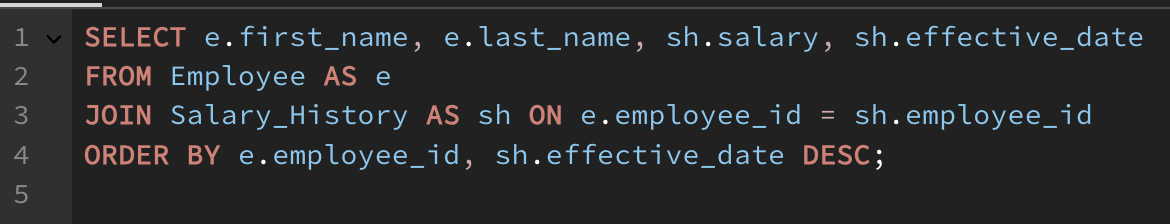
**Intermediate query 1 - Show which employees are working on a specific project:**

****

****

**Summary**: Display all employees working on a specific project using a JOIN between multiple tables and WHERE to stipulate which project.

**Intermediate query 2 - Show employee salary history:**

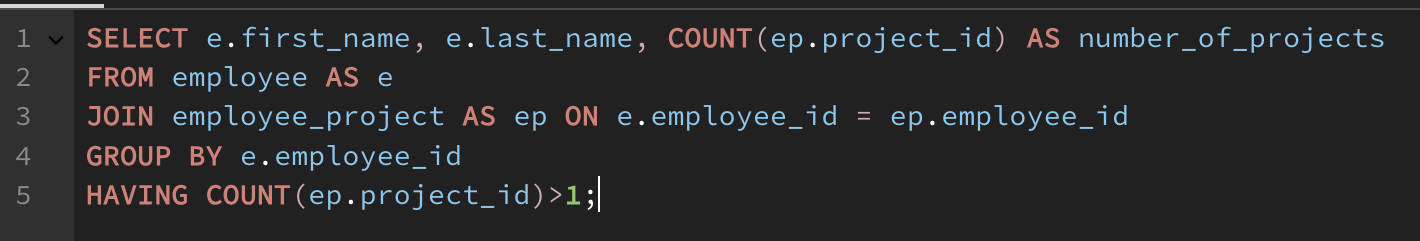
****

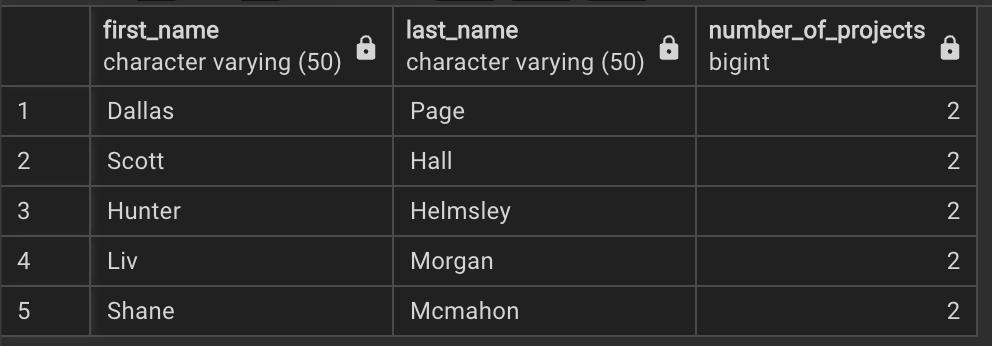
****

**Summary**: Display each employee’s salary history, sorted by the latest effective date (individually). Use JOIN and demonstrate use of ORDER BY.

**Advanced queries:**

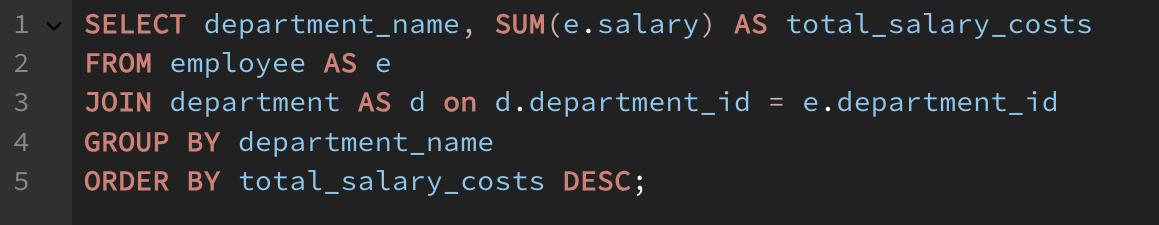
**Advanced query 1 - Show employees with multiple projects:**

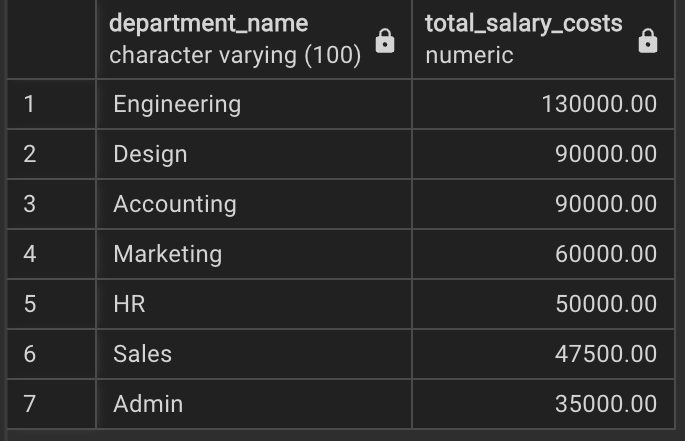




**Summary**: Show employees working on more than one project, showcasing aggregation COUNT and HAVING.

**Advanced query 2 - Show salary expenses for each department:**





**Summary**: Calculate the total salary expenditure by department, showcasing SUM.

### **9. Conclusion**

In this project, I created an Employee Management System using PostgreSQL. I designed the database by creating tables for employees, departments, projects, and salary history. I then inserted data into these tables to reflect a realistic work environment. I also wrote and tested several queries to retrieve information, such as which employees are working on different projects and tracking salary changes. This project helped me practice the basics of SQL, including creating tables, inserting data, and using JOIN queries to link information from different tables for realistic database queries. Overall, it has helped me better understand how databases work and how to manage data effectively.