# **SQL and Database Objects Assignment**

**Name:** Omar Abdulrahim

**Course:** Database 2

**Date:** Friday, June 20, 2025

Specialization: Information systems 5

## **Part 1: Create the Table and Insert Data**

### **1.1. CREATE TABLE Statement**

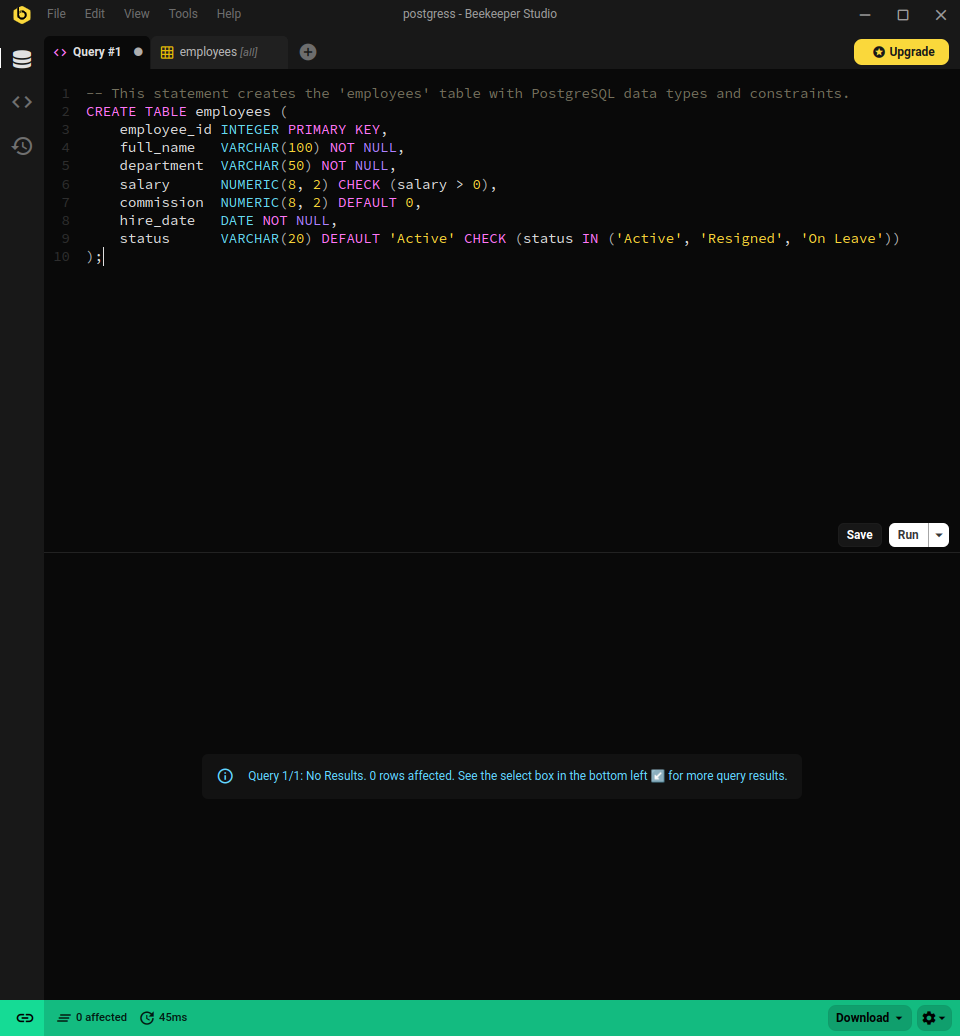
#### **SQL Query**

-- This statement creates the 'employees' table with PostgreSQL data types and constraints.  
CREATE TABLE employees (  
 employee\_id INTEGER PRIMARY KEY,  
 full\_name VARCHAR(100) NOT NULL,  
 department VARCHAR(50) NOT NULL,  
 salary NUMERIC(8, 2) CHECK (salary > 0),  
 commission NUMERIC(8, 2) DEFAULT 0,  
 hire\_date DATE NOT NULL,  
 status VARCHAR(20) DEFAULT 'Active' CHECK (status IN ('Active', 'Resigned', 'On Leave'))  
);

#### **Explanation**

This query creates the employees table. It defines each column with an appropriate PostgreSQL data type and applies constraints like PRIMARY KEY for uniqueness, NOT NULL for required fields, DEFAULT for automatic values, and CHECK to ensure data integrity.

#### **Screenshot of Output**



### **1.2. INSERT Statements**

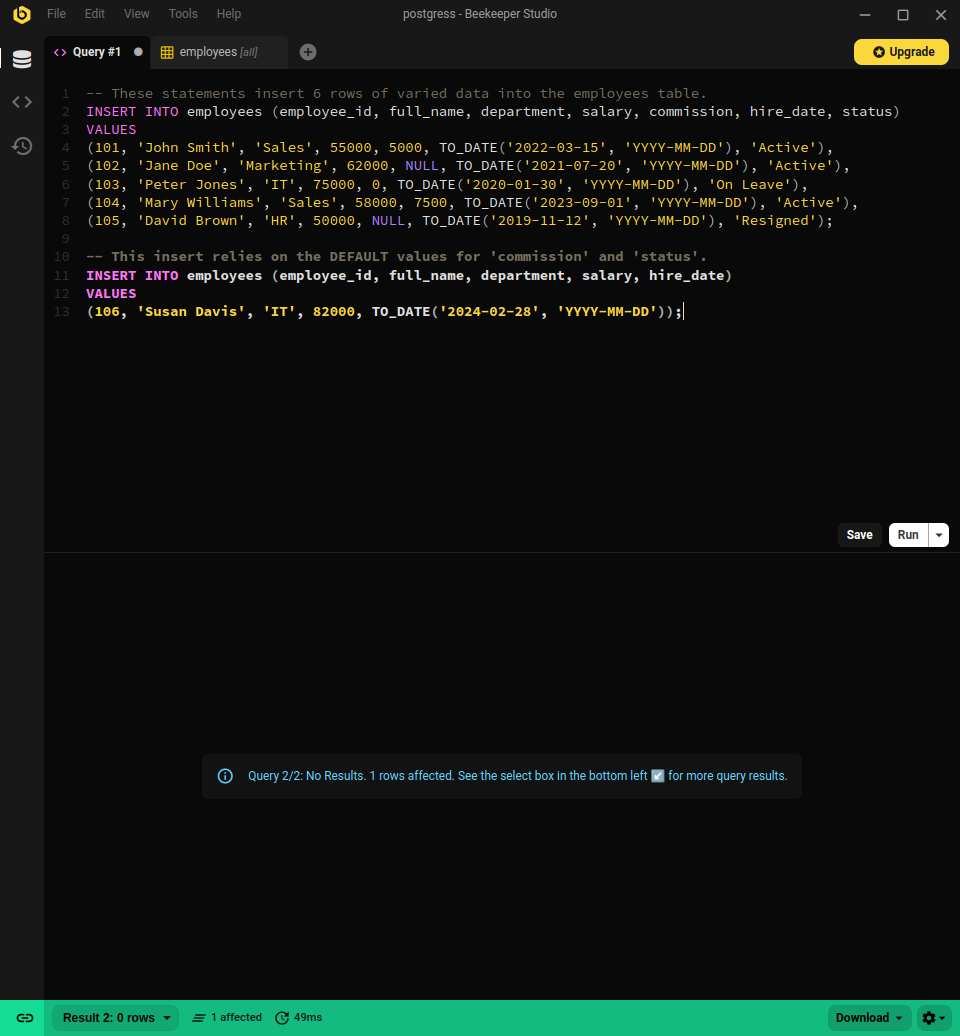
#### **SQL Query**

-- These statements insert 6 rows of varied data into the employees table.  
INSERT INTO employees (employee\_id, full\_name, department, salary, commission, hire\_date, status)  
VALUES  
(101, 'John Smith', 'Sales', 55000, 5000, TO\_DATE('2022-03-15', 'YYYY-MM-DD'), 'Active'),  
(102, 'Jane Doe', 'Marketing', 62000, NULL, TO\_DATE('2021-07-20', 'YYYY-MM-DD'), 'Active'),  
(103, 'Peter Jones', 'IT', 75000, 0, TO\_DATE('2020-01-30', 'YYYY-MM-DD'), 'On Leave'),  
(104, 'Mary Williams', 'Sales', 58000, 7500, TO\_DATE('2023-09-01', 'YYYY-MM-DD'), 'Active'),  
(105, 'David Brown', 'HR', 50000, NULL, TO\_DATE('2019-11-12', 'YYYY-MM-DD'), 'Resigned');  
  
-- This insert relies on the DEFAULT values for 'commission' and 'status'.  
INSERT INTO employees (employee\_id, full\_name, department, salary, hire\_date)  
VALUES  
(106, 'Susan Davis', 'IT', 82000, TO\_DATE('2024-02-28', 'YYYY-MM-DD'));

#### **Explanation**

This query populates the employees table with six records, including varied data such as different departments, a NULL commission, and a record that relies on DEFAULT values.

#### **Screenshot of Output**



## **Part 2: Write and Execute SQL Queries**

### **2.1. Select All Columns and Rows**

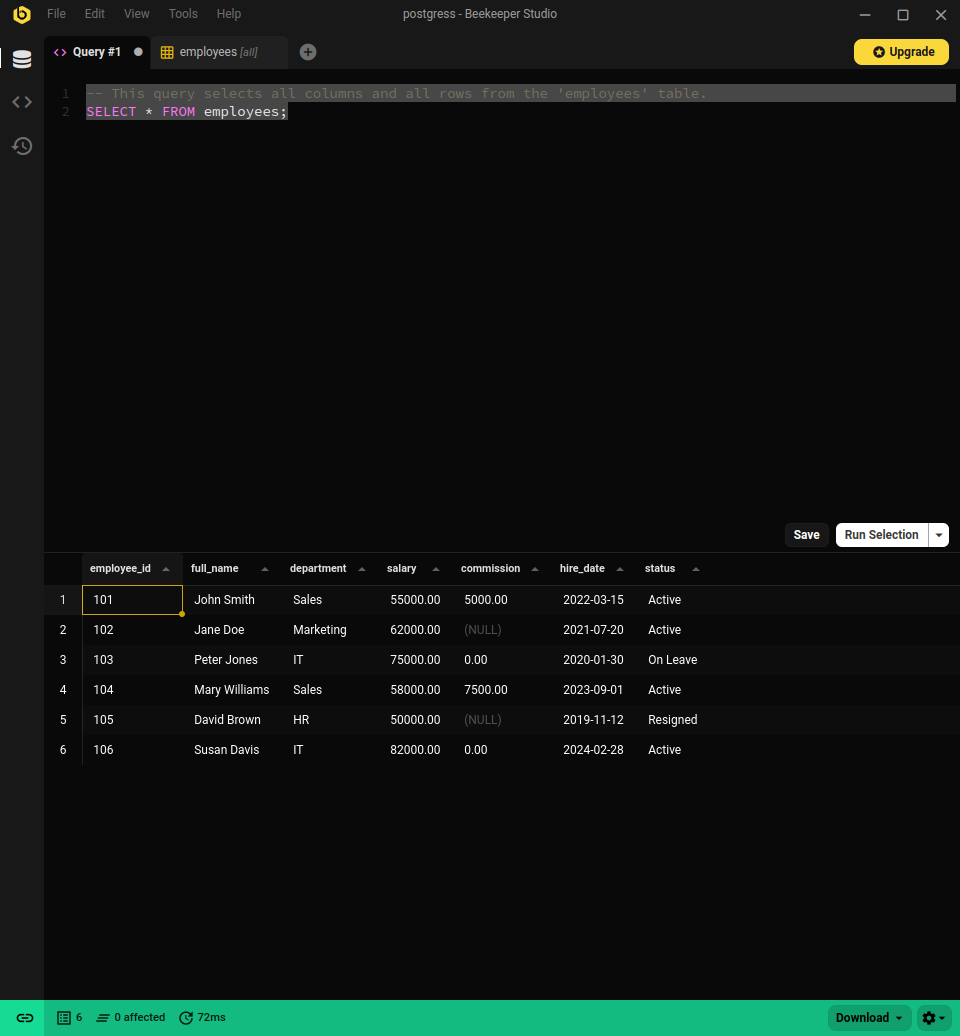
#### **SQL Query**

-- This query selects all columns and all rows from the 'employees' table.  
SELECT \* FROM employees;

#### **Explanation**

This query uses SELECT \* to retrieve and display every column and every row from the employees table, allowing for a full view of the current data.

#### **Screenshot of Output**



### **2.2. DML Operations**

#### **Update an Employee's Salary (with COMMIT)**

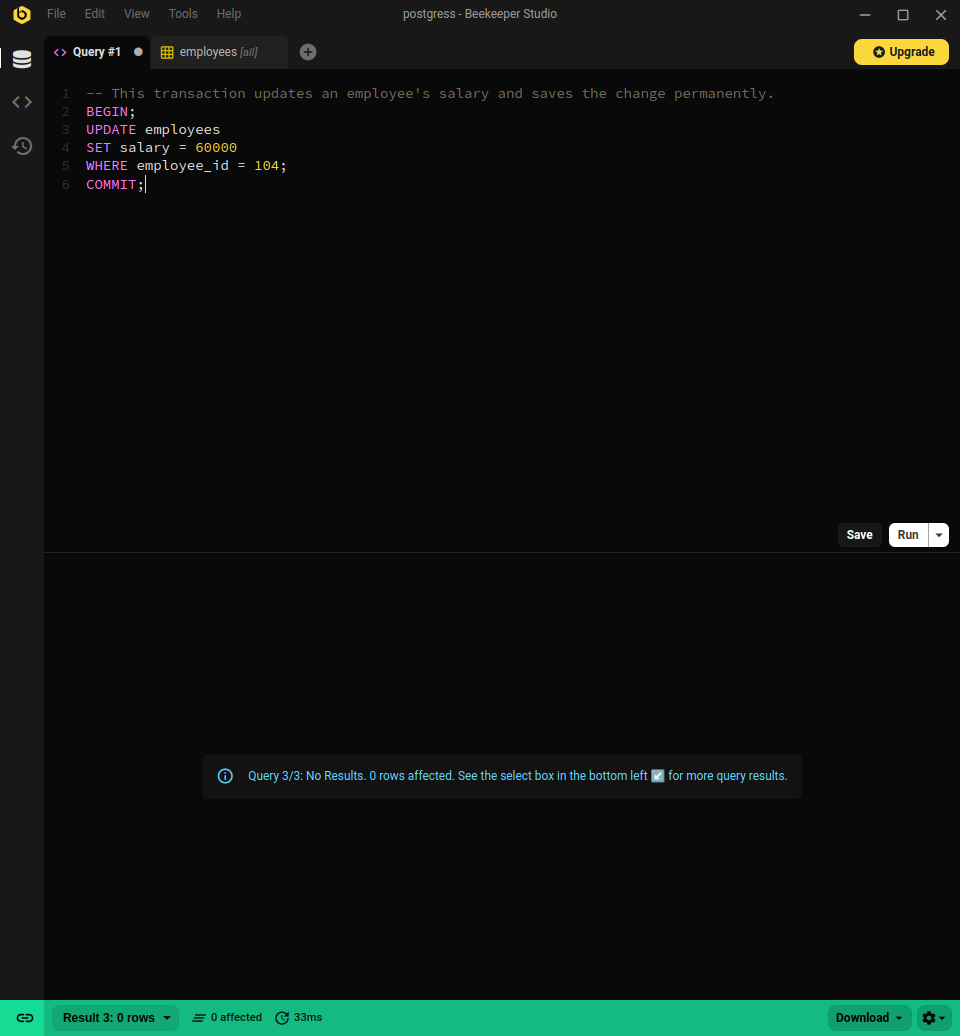
#### **SQL Query**

-- This transaction updates an employee's salary and saves the change permanently.  
BEGIN;  
UPDATE employees  
SET salary = 60000  
WHERE employee\_id = 104;  
COMMIT;

#### **Explanation**

This transaction permanently increases the salary for employee 104 to 60,000. The COMMIT command saves the change to the database.

#### **Screenshot of Output**



#### **Delete and ROLLBACK an Employee**

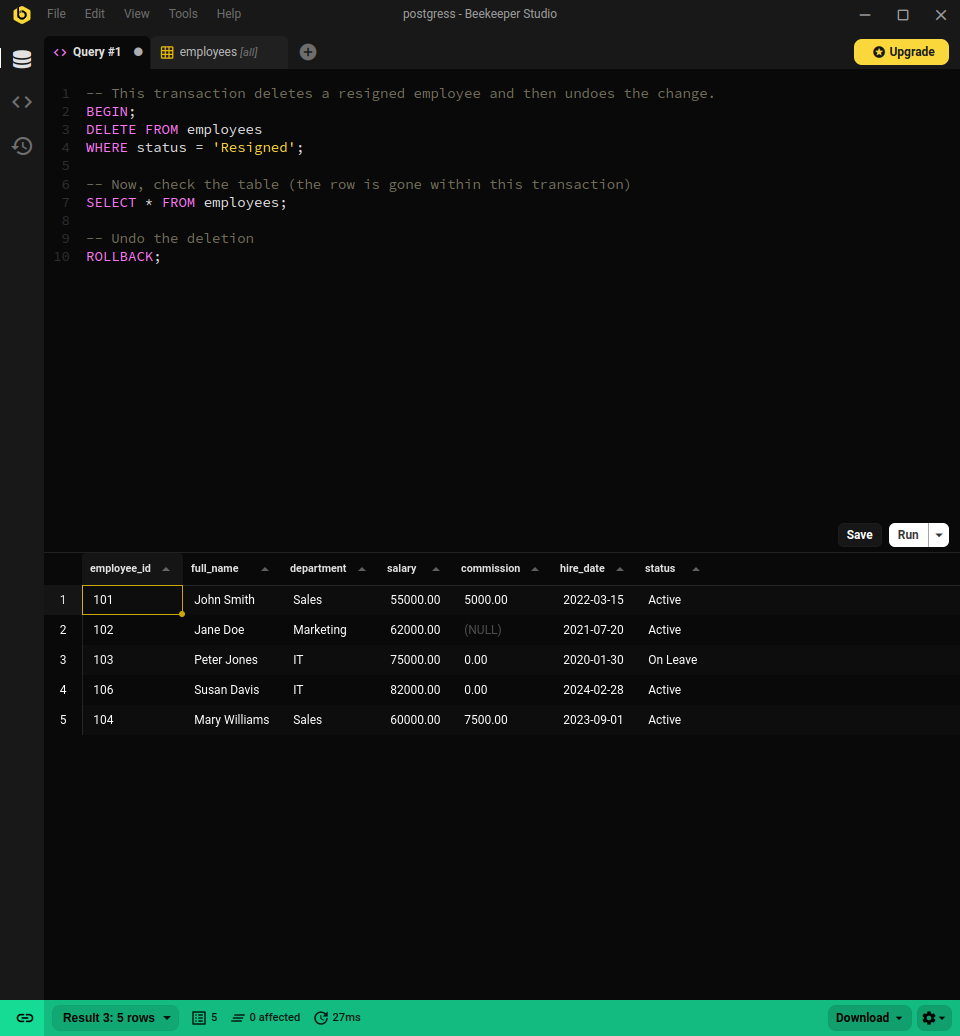
#### **SQL Query**

-- This transaction deletes a resigned employee and then undoes the change.  
BEGIN;  
DELETE FROM employees  
WHERE status = 'Resigned';  
ROLLBACK;

#### **Explanation**

This transaction begins by deleting the employee with the 'Resigned' status. However, the ROLLBACK command is then executed, which undoes the deletion and restores the data to its previous state.

#### **Screenshot of Output**



### **2.3. Simple View**

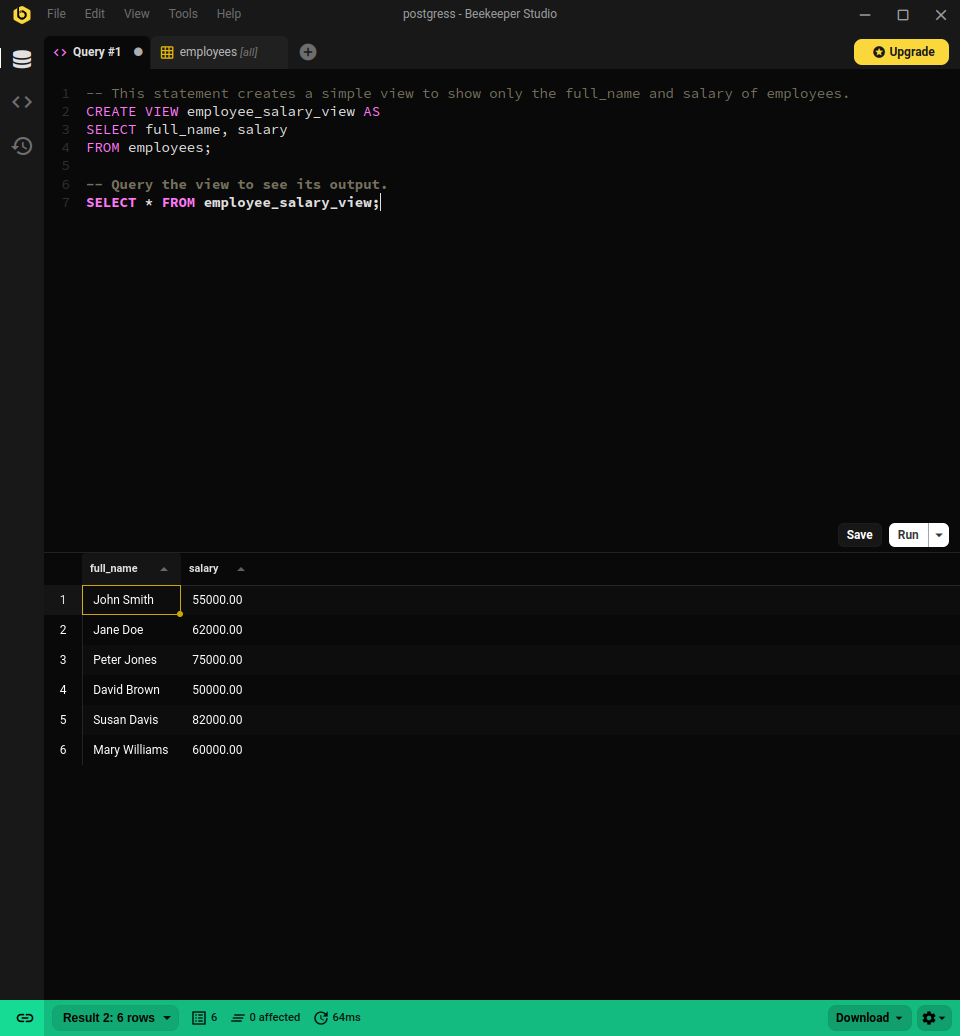
#### **SQL Query**

-- This statement creates a simple view to show only the full\_name and salary of employees.  
CREATE VIEW employee\_salary\_view AS  
SELECT full\_name, salary  
FROM employees;  
  
-- Query the view to see its output.  
SELECT \* FROM employee\_salary\_view;

#### **Explanation**

This CREATE VIEW statement creates a virtual table named employee\_salary\_view that simplifies the employees table to only show the full\_name and salary columns.

#### **Screenshot of Output**



### **2.4. Complex View**

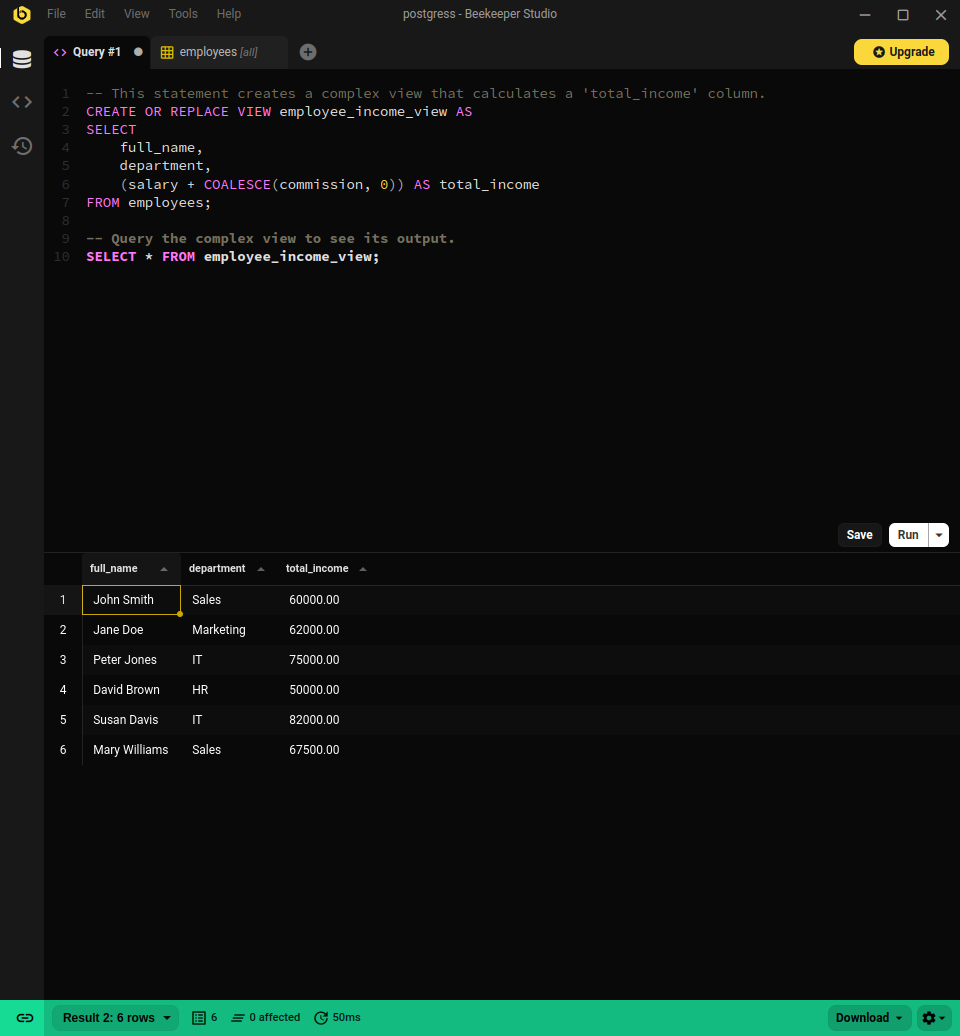
#### **SQL Query**

-- This statement creates a complex view that calculates a 'total\_income' column.  
CREATE OR REPLACE VIEW employee\_income\_view AS  
SELECT  
 full\_name,  
 department,  
 (salary + COALESCE(commission, 0)) AS total\_income  
FROM employees;  
  
-- Query the complex view to see its output.  
SELECT \* FROM employee\_income\_view;

#### **Explanation**

This query creates a more complex view named employee\_income\_view. It includes a calculated column called total\_income, which is the sum of salary and commission. The COALESCE function is used to treat any NULL commission values as 0.

#### **Screenshot of Output**



### **2.5. Sequence for employee\_id**

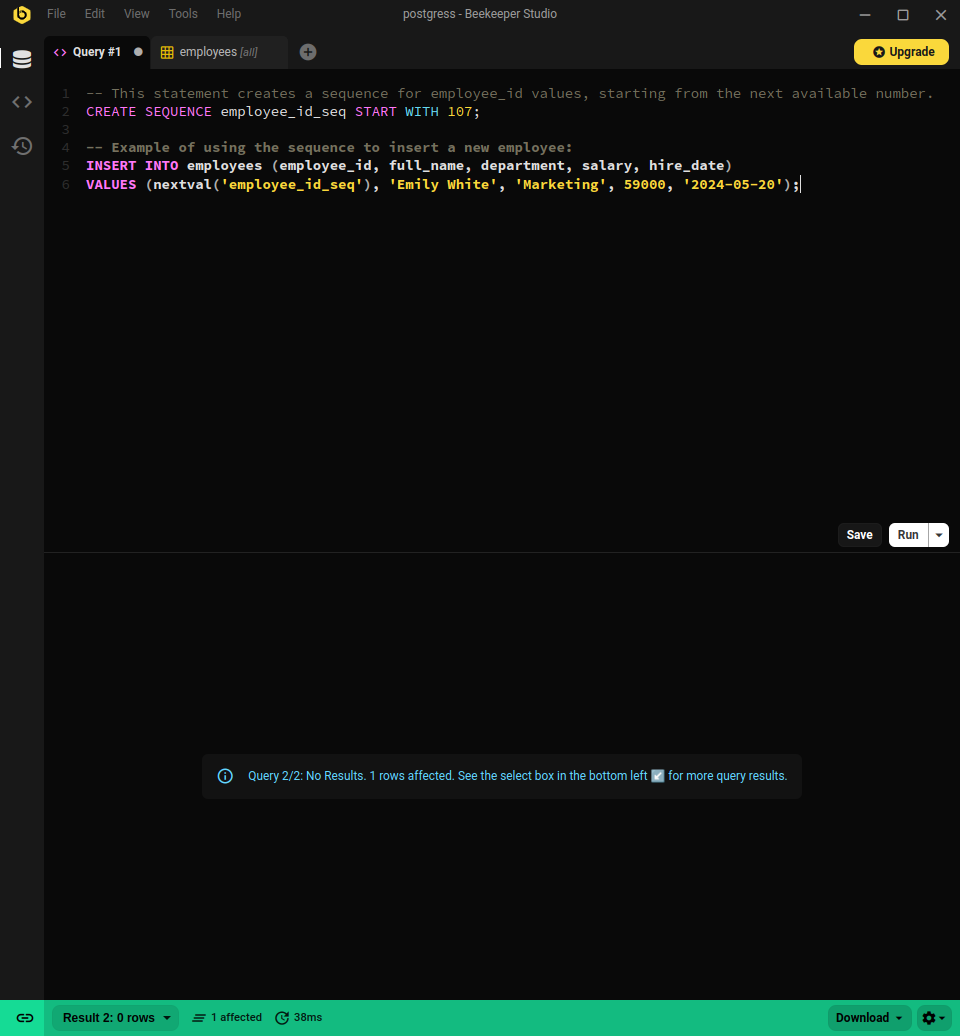
#### **SQL Query**

-- This statement creates a sequence for employee\_id values, starting from the next available number.  
CREATE SEQUENCE employee\_id\_seq START WITH 107;

#### **Explanation**

This query creates a sequence object named employee\_id\_seq. This object can be used to automatically generate unique numbers for the employee\_id column when inserting new records, starting with 107.

#### **Screenshot of Output**



### **2.6. Index for Performance**

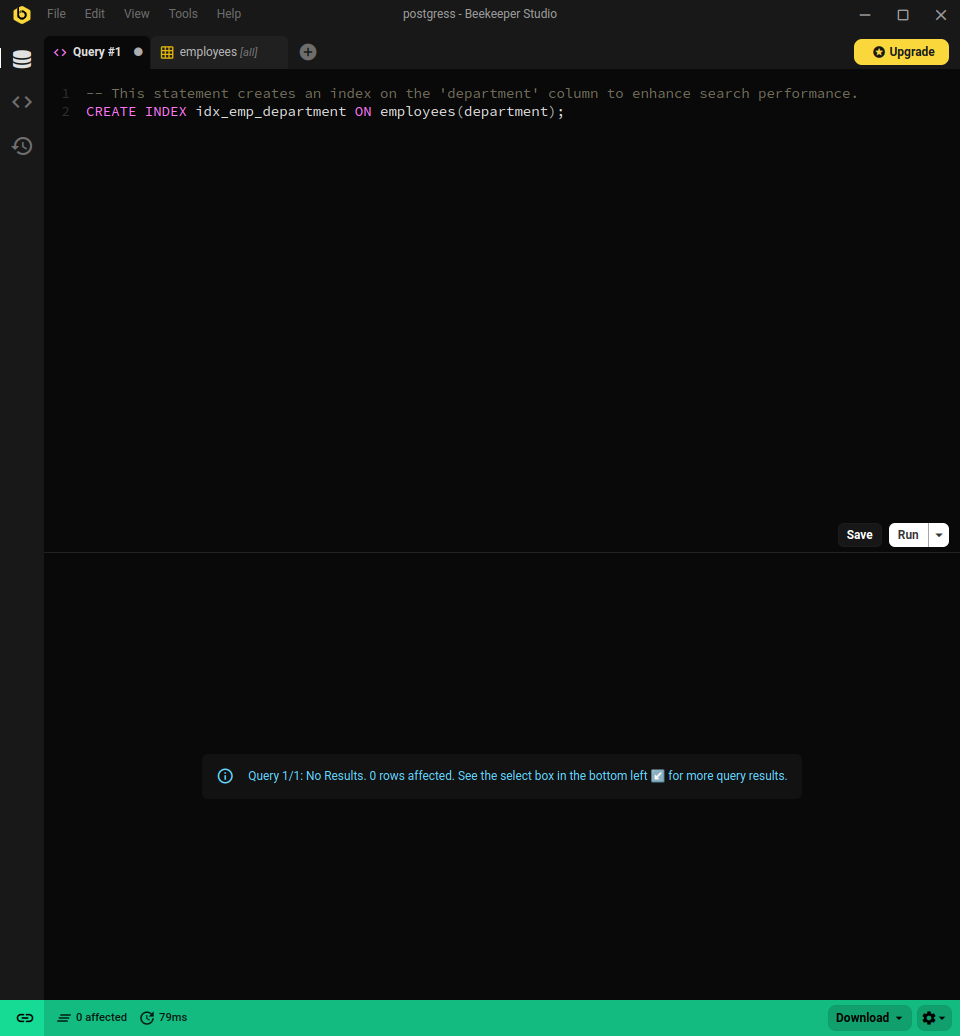
#### **SQL Query**

-- This statement creates an index on the 'department' column to enhance search performance.  
CREATE INDEX idx\_emp\_department ON employees(department);

#### **Explanation**

This query creates an index named idx\_emp\_department on the department column of the employees table. This index will speed up queries that filter or sort by department.

#### **Screenshot of Output**



### **2.7. Synonyms (PostgreSQL Workaround)**

#### **Public Synonym (Workaround using a View)**

#### **SQL Query**

-- This creates a view in the 'public' schema, making it accessible to all users.  
-- This acts as a public synonym.  
CREATE VIEW public.all\_employees AS  
SELECT \* FROM public.employees;  
  
-- Now any user can query it directly.  
SELECT \* FROM all\_employees;

#### **Explanation**

Since PostgreSQL does not have a CREATE SYNONYM command, a view in the public schema is created as a workaround. This all\_employees view can be accessed by any user, effectively serving as a public synonym for the employees table.

#### **Screenshot of Output**

