SQL LABS REPORT

BY

ABASIOFON MOSES - 63922

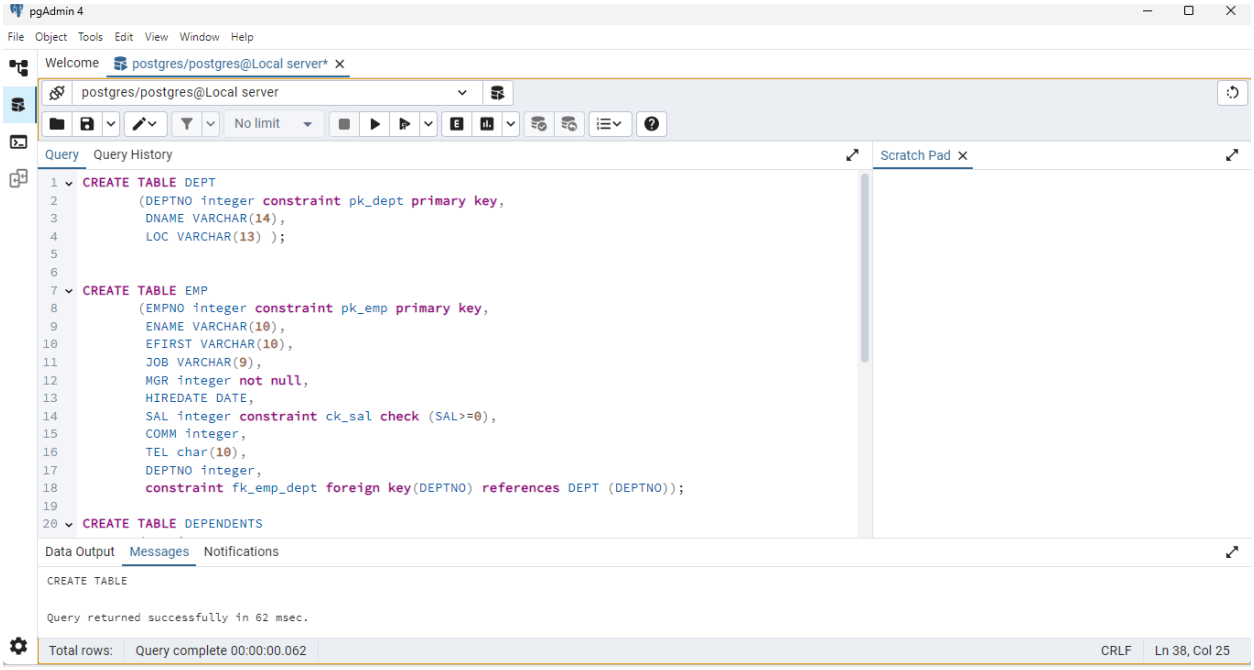
Github: https://github.com/DatagirlX/SQL-LAB-ISEP

Advanced Database - TP 1

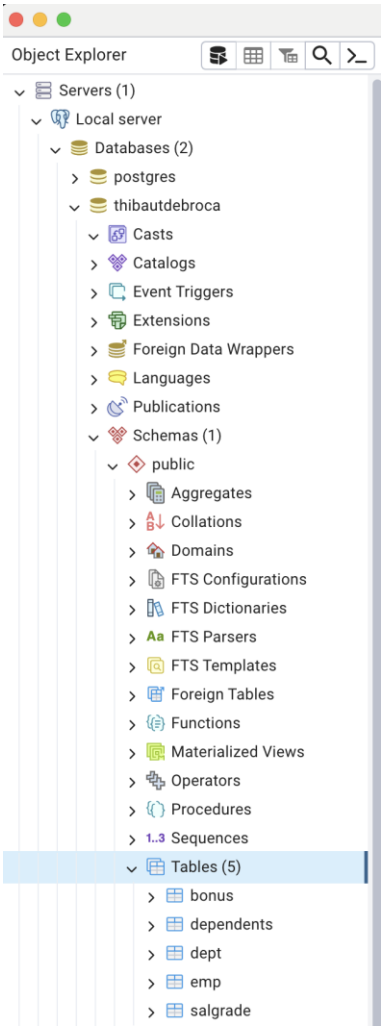
SQL

Exercice 1: DDL queries

1. **Copy** the content of the script creation.sql and paste in a Sql editor in Postgresql.



Check if the tables appear in the list of tables:



2. **E/R diagram:** Identify the different tables created and integrity constraints to ensure data consistency. Draw the entity / relational model from the given database.

Note: Dependents mean a person who depends on another person to be affiliated to the social Security System. In that case a dependent can be a child whose father is an employee (defined by empno link).

Tables Created:

DEPT (Department)
EMP (Employee)
DEPENDENTS (Employee's Dependents)
BONUS (Bonus for Employees)
SALGRADE (Salary Grade)

INTEGRITY CONSTRAINTS:

DEPT:

Primary Key (Ensures each DEPTNO is unique)

DEPTNO integer constraint pk_dept primary key

EMP:
EMPNO integer constraint pk_emp primary key - Primary Key (Ensures each EMPNO is unique)

Check (Ensures SAL >= 0 (salary cannot be negative))

SAL integer constraint ck_sal check (SAL>=0)

Null (Makes sure that the column is not null)

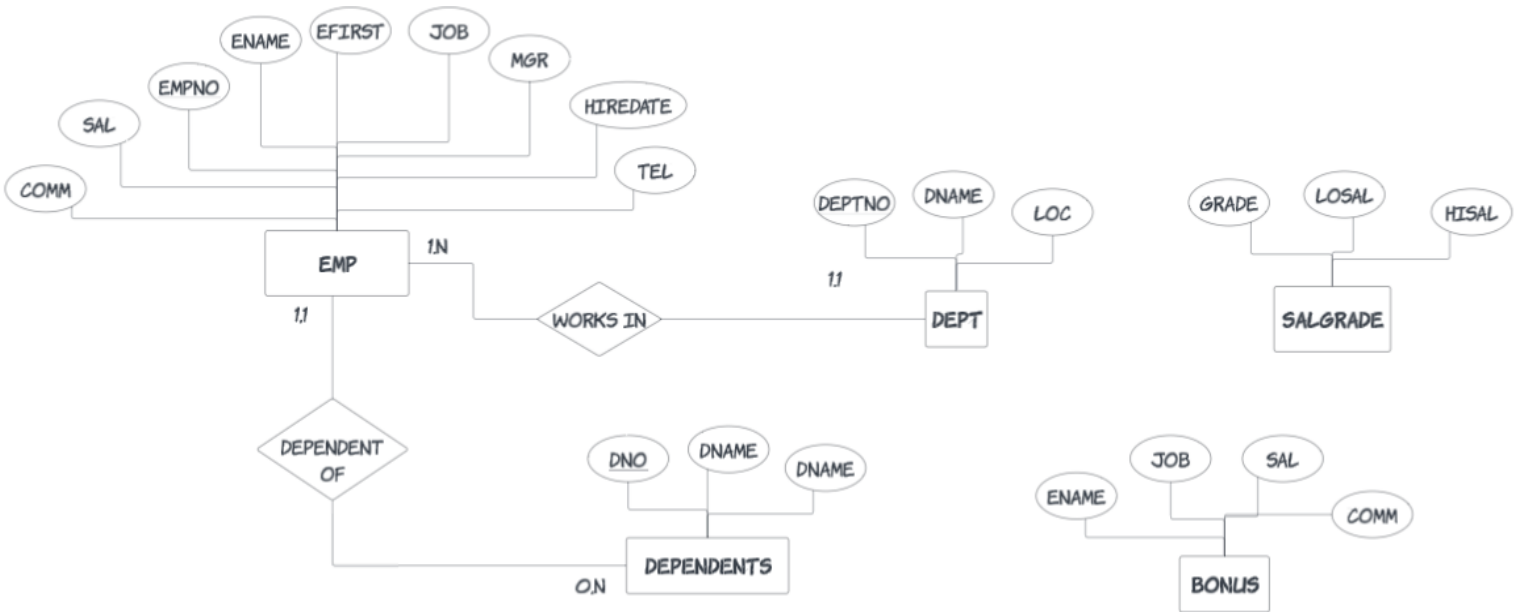
MGR integer not null,

constraint fk_emp_dept foreign key(DEPTNO) references DEPT (DEPTNO)

DEPENDENTS:

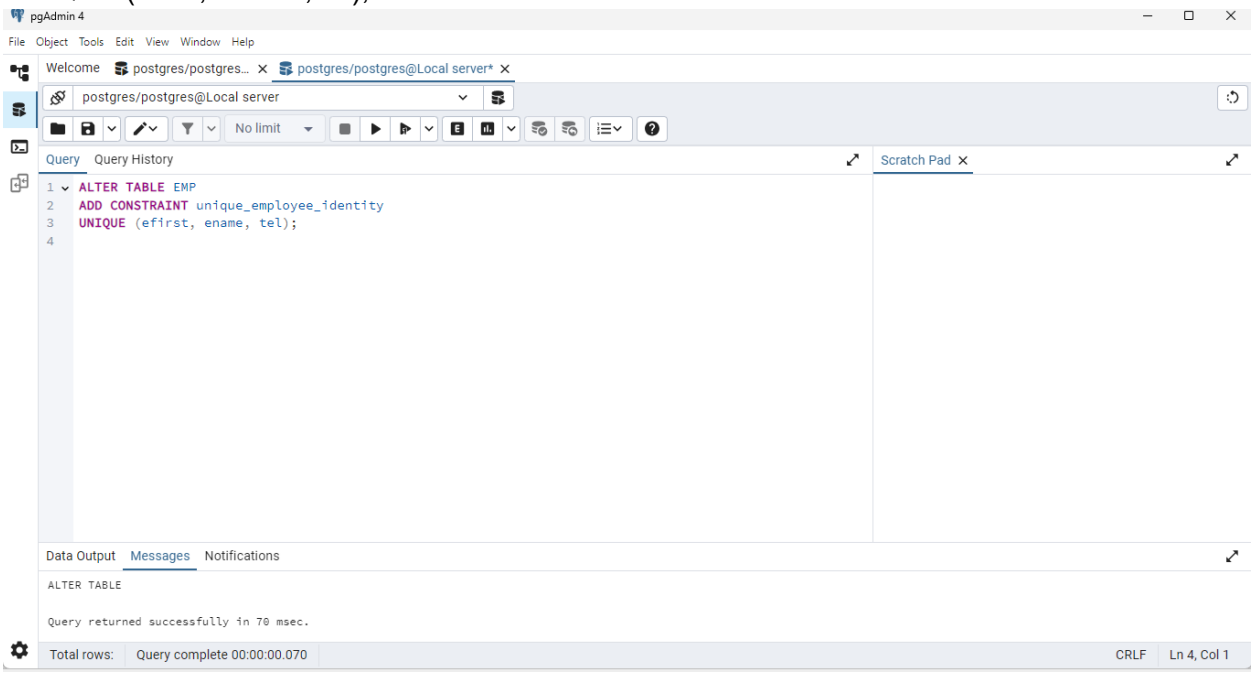
constraint pk_dependent primary key (DNO, EMPNO)
constraint fk_dependent_emp foreign key(EMPNO) references EMP (EMPNO)

THE ENTITY / RELATIONAL MODEL



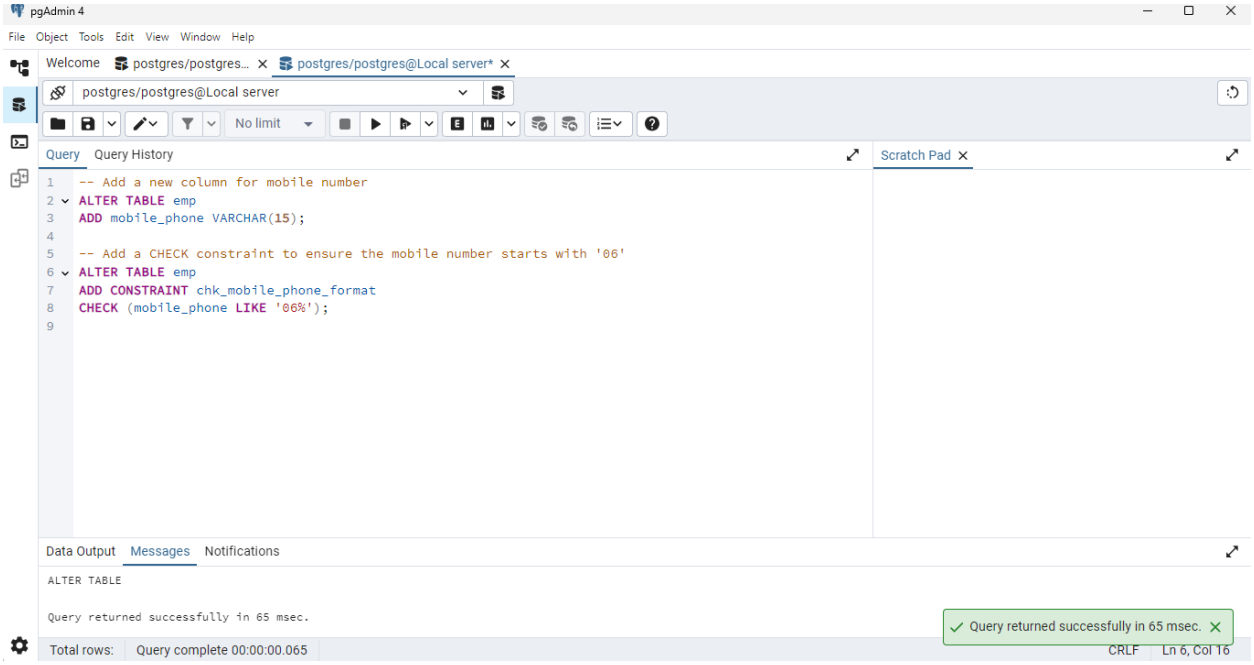
3. **First integrity Constraint:** Define an integrity constraint to prohibit the possibility of two employees to own the same firstname, lastname and phone number. Use the alter table statement.

```
ALTER TABLE EMP
ADD CONSTRAINT unique_employee_identity
UNIQUE (efirst, ename, tel);
```



4. **Second integrity Constraint:** More and more employees have two phone numbers: a fixed and a mobile phone. Unfortunately, the database allows us to store one phone number. Bring the necessary modifications to the database. Set up an integrity constraint in order to make sure that the entered mobile number begins with 06.

```
-- Add a new column for mobile number
ALTER TABLE emp
ADD mobile_phone VARCHAR(15);

-- Add a CHECK constraint to ensure the mobile number starts with '06'
ALTER TABLE emp
ADD CONSTRAINT chk_mobile_phone_format
CHECK (mobile_phone LIKE '06%');
```



5. **Third integrity Constraint On Delete:** In practice, we realize that when one wants to delete an employee, it is necessary to remove all his dependents. How is it possible to make such removal dynamic? Do not forget to first remove the referential constraint fk_ depenent_emp that already exists.

-- Drop the existing foreign key constraint

ALTER TABLE DEPENDENTS

DROP CONSTRAINT fk_dependent_emp;

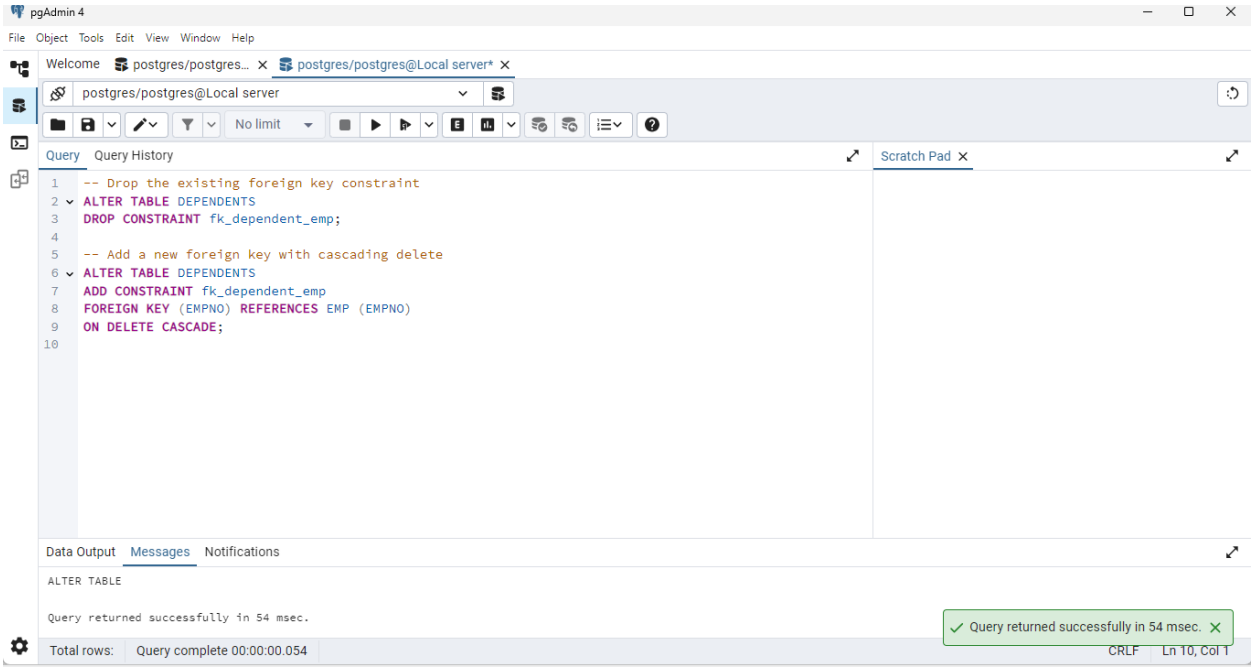
-- Add a new foreign key with cascading delete

ALTER TABLE DEPENDENTS

ADD CONSTRAINT fk_dependent_emp

FOREIGN KEY (EMPNO) REFERENCES EMP (EMPNO)

ON DELETE CASCADE;



6. **Explain Errors for Integrity Constraint:** Fill the tables with the insert.sql script. Explain the errors that you could get and correct them.

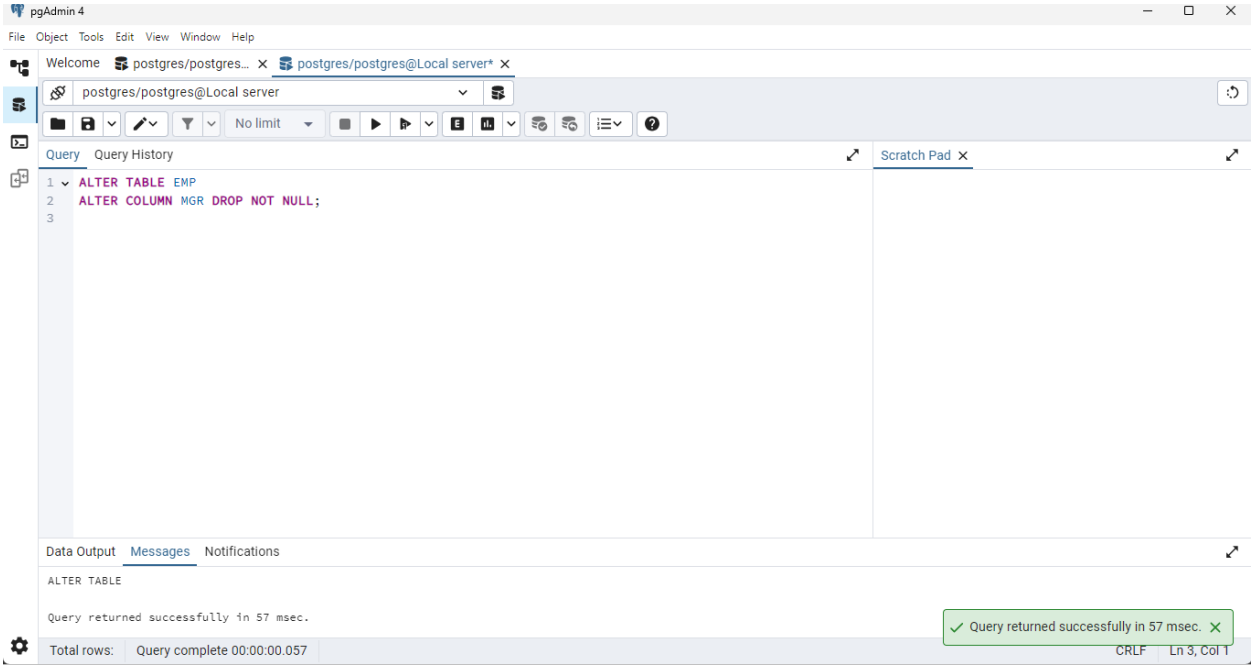
Errors that can be gotten:

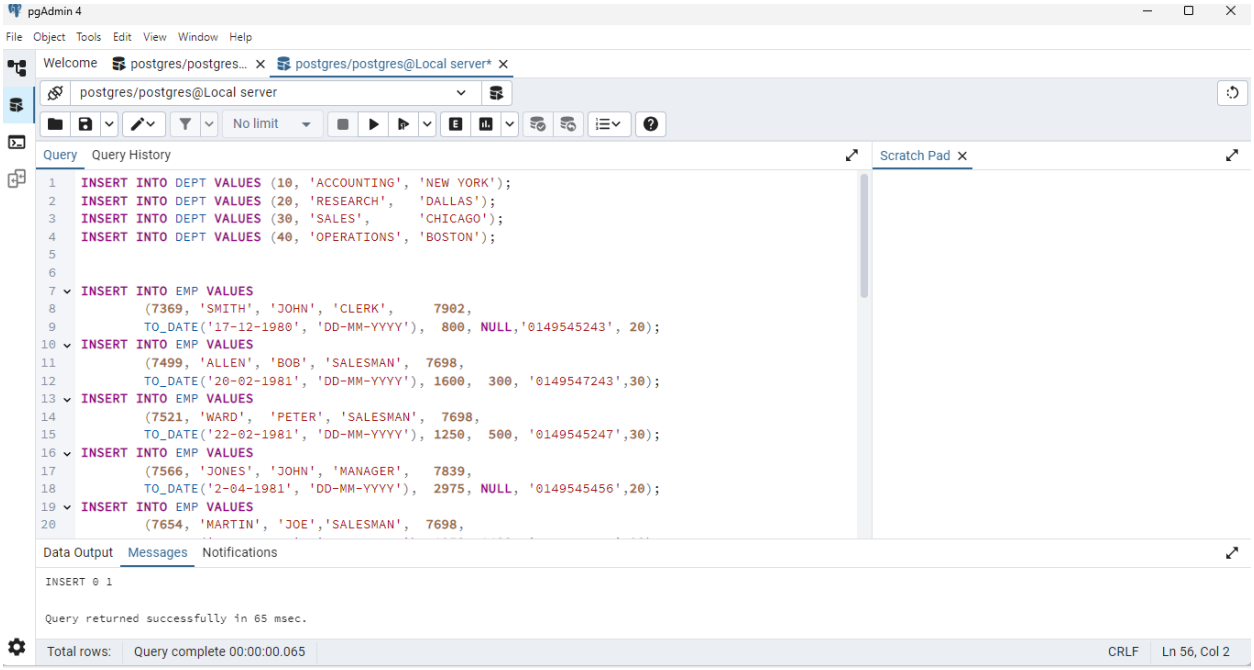
The MGR column in the EMP table has a NOT NULL constraint, but there's an entry for the PRESIDENT (EMPNO 7839) with NULL as the manager.

Correction:

ALTER TABLE EMP

ALTER COLUMN MGR DROP NOT NULL;



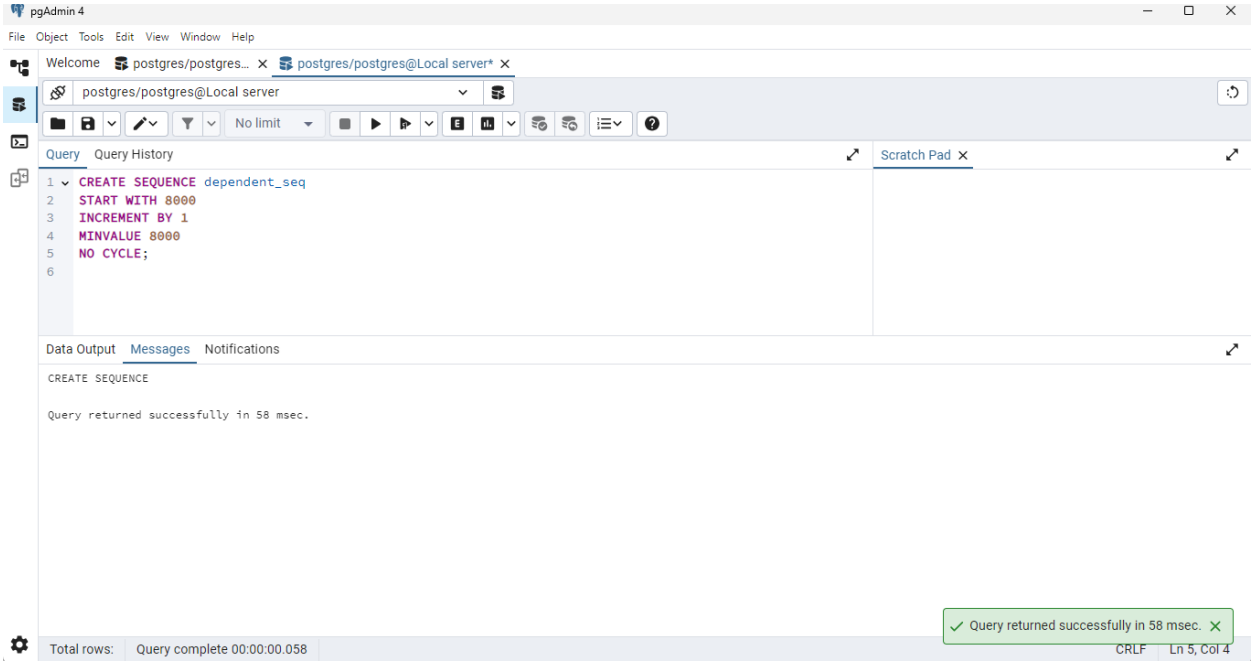


7. **Define a sequence** to make easy creation of a new dependent. This sequence has to start with the value 8000 and an increment gap of 1.

The instruction to create a sequence:

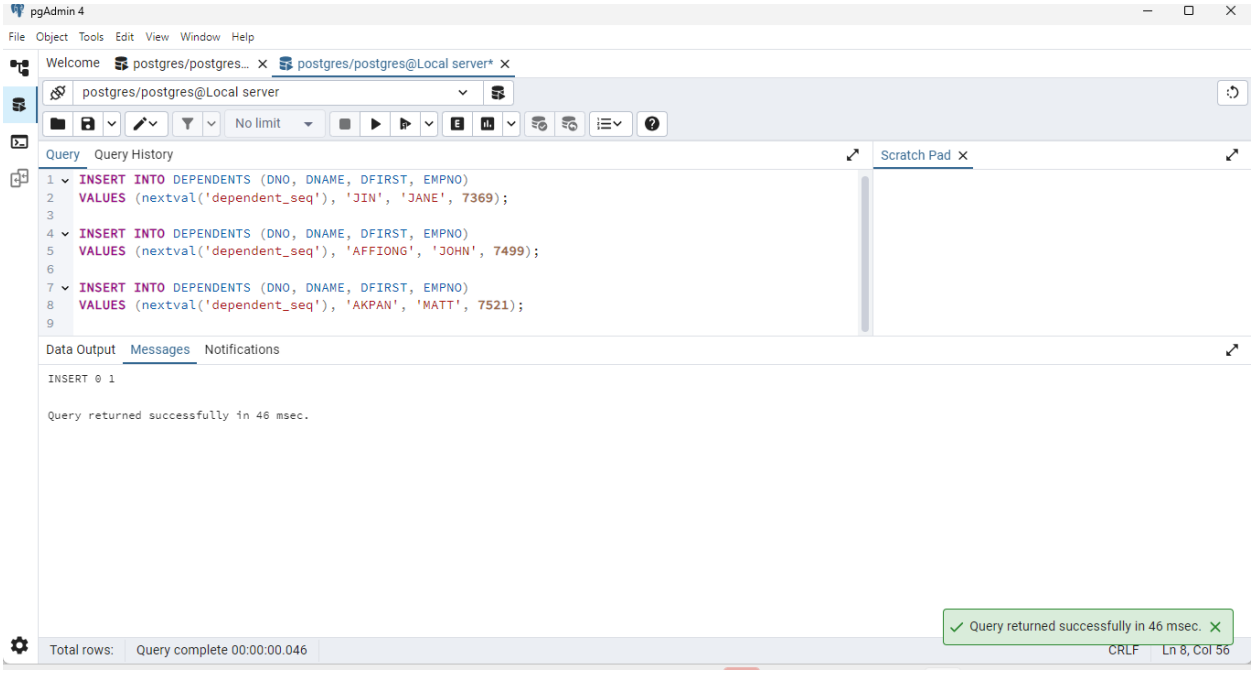
CREATE SEQUENCE ma_sequence [START WITH] ... [INCREMENT BY]... [MINVALUE]...[MAXVALUE]...[CYCLE|NOCYCLE];

```
CREATE SEQUENCE dependent_seq
START WITH 8000
INCREMENT BY 1
MINVALUE 8000
NO CYCLE;
```



8. **Use Sequence:** Add some tuples to the DEPENDENT table using the previous sequence Use nextval('ma_sequence') to get the values from the sequence.

```
INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
VALUES (nextval('dependent_seq'), 'JIN', 'JANE', 7369);

INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
VALUES (nextval('dependent_seq'), 'AFFIONG', 'JOHN', 7499);

INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
VALUES (nextval('dependent_seq'), 'AKPAN', 'MATT', 7521);
```



9. **Discuss** on the best way to do an auto-increment with postgres. Apart from Sequence, there is also the “serial” data object. But there is also the “identity” since postgres 10 (2017):

The best practice for implementing auto-incrementing fields in PostgreSQL depends on the version being used and the specific requirements of the project:

1. **For PostgreSQL 10 and later**, the recommended approach is to use the **`IDENTITY`** column. It is the SQL-standard method for defining auto-incrementing columns, providing a clean and declarative solution. The `IDENTITY` column is easier to manage, less error-prone, and aligns with modern database standards. It is also more readable compared to using `SERIAL` or manually created sequences.

2. **For flexibility and control**, using **`SEQUENCE`** is a good choice. A sequence provides full control over the auto-increment process and allows for advanced features like resetting or sharing sequences across multiple tables. However, it requires more setup and manual management.

3. **Using `SERIAL`** for new projects should be avoided. While `SERIAL` is easy to implement, it hides the sequence behind the scenes and offers limited control. This can lead to difficulties if the sequence needs to be altered or if more advanced sequence management is required.

For modern PostgreSQL projects, **`IDENTITY`** is the preferred method, offering simplicity and standard compliance, while **`SEQUENCE`** remains a good choice when more control is needed. **`SERIAL`** should be avoided in favor of the more robust and flexible solutions provided by `IDENTITY` and `SEQUENCE`.

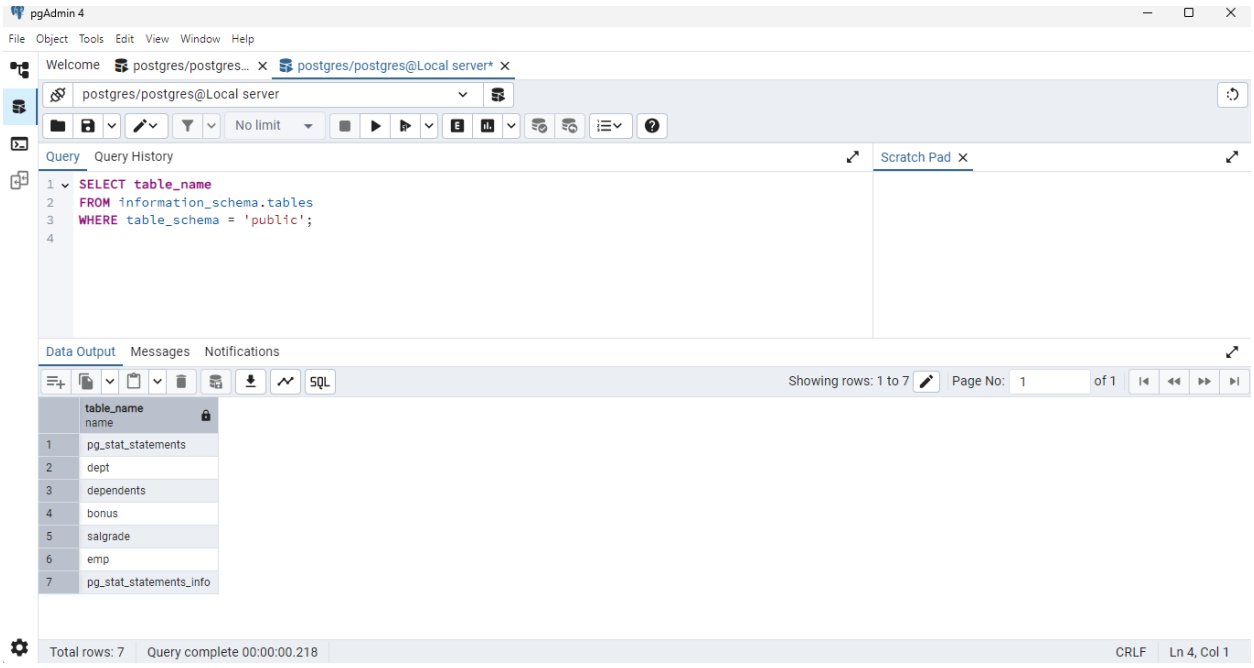
Exercise 2: DML queries Answer the following queries using SQL.

1. List the content of all tables to see the attributes names

SELECT table_name

FROM information_schema.tables

WHERE table_schema = 'public';

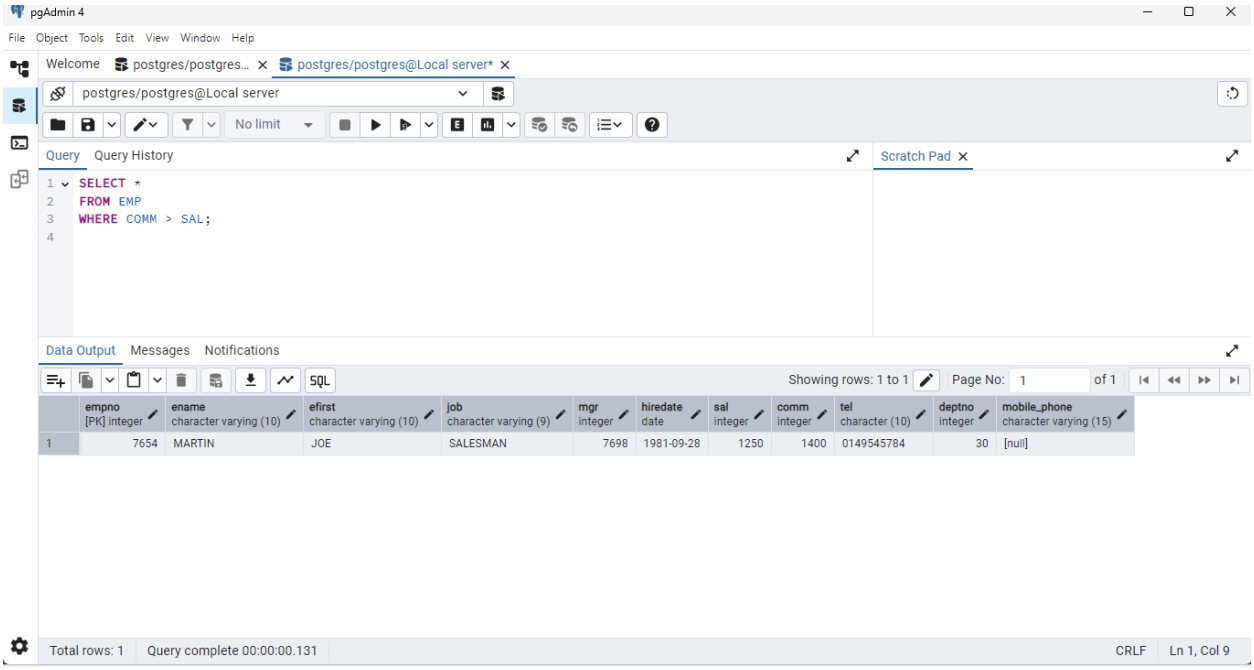


2. Select employees whose commission is higher than their salary

SELECT *

FROM EMP

WHERE COMM > SAL;

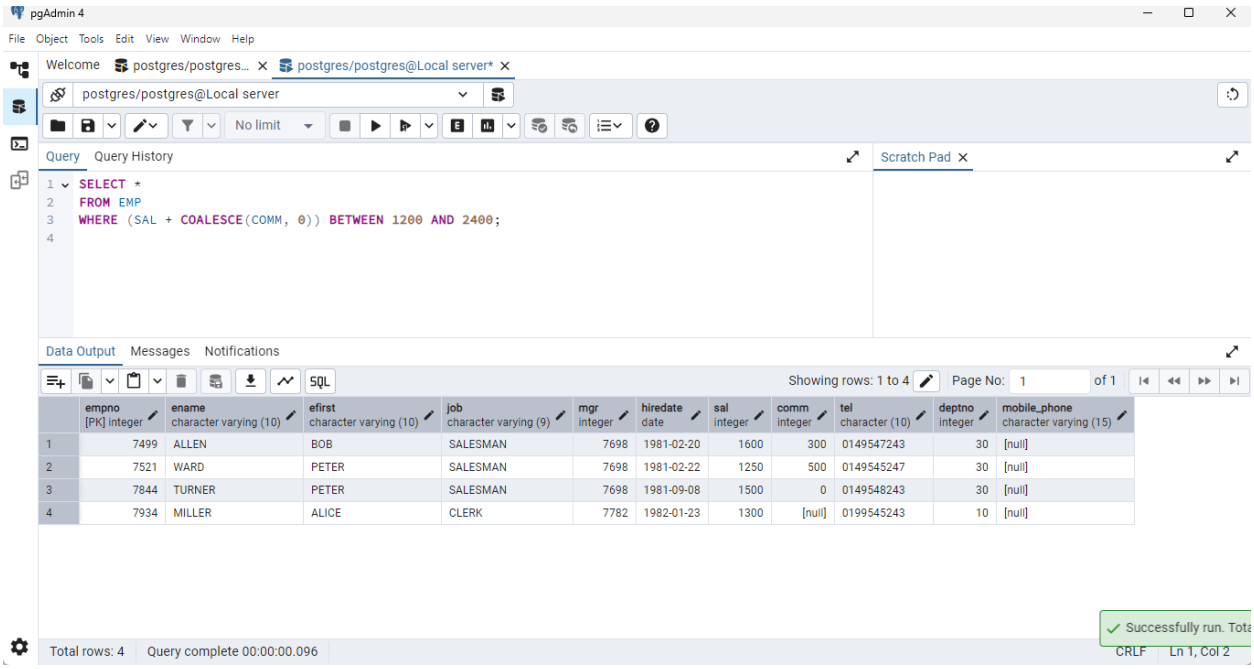


3. Select employees earning between 1200 and 2400 (earning is sal + commision)

SELECT *

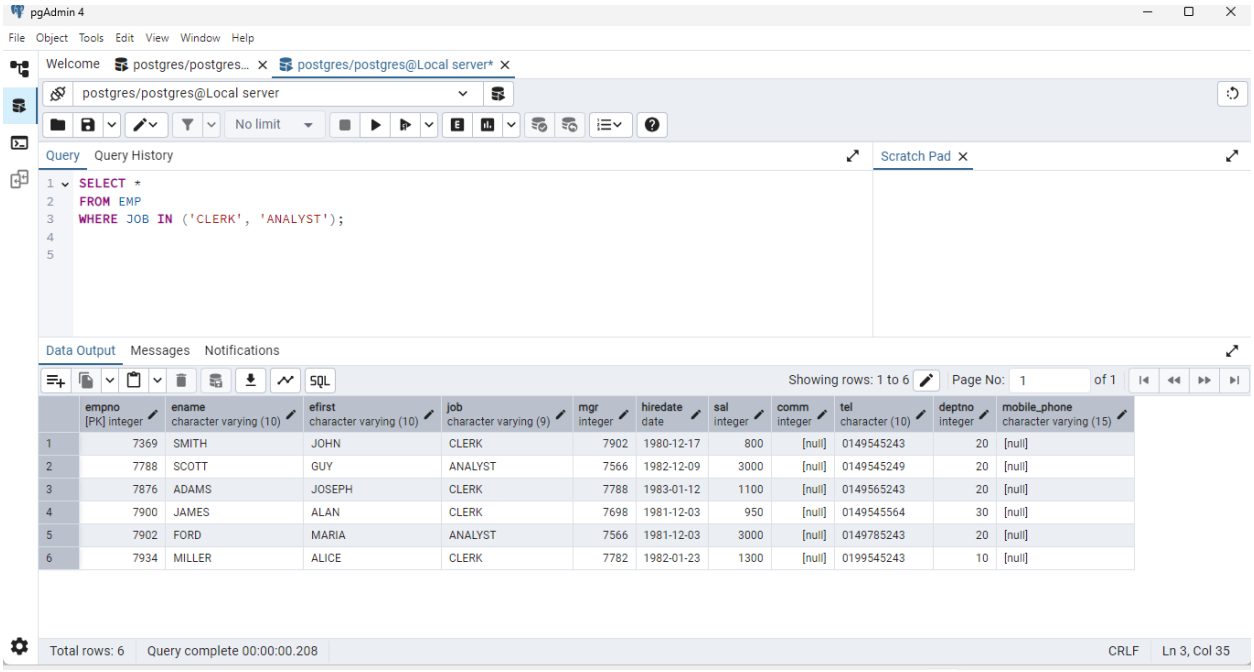
FROM EMP

WHERE (SAL + COALESCE(COMM, 0)) BETWEEN 1200 AND 2400;



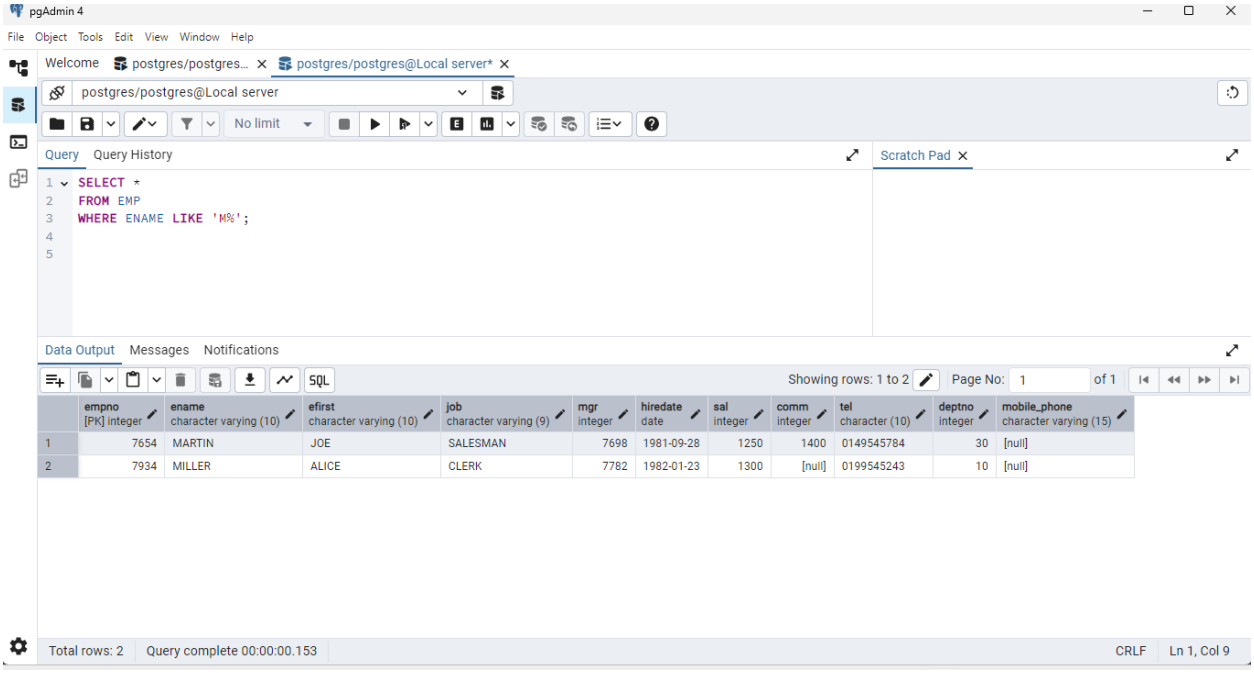
4. Select employees who are CLERK or ANALYST

```
SELECT *
FROM EMP
WHERE JOB IN ('CLERK', 'ANALYST');
```



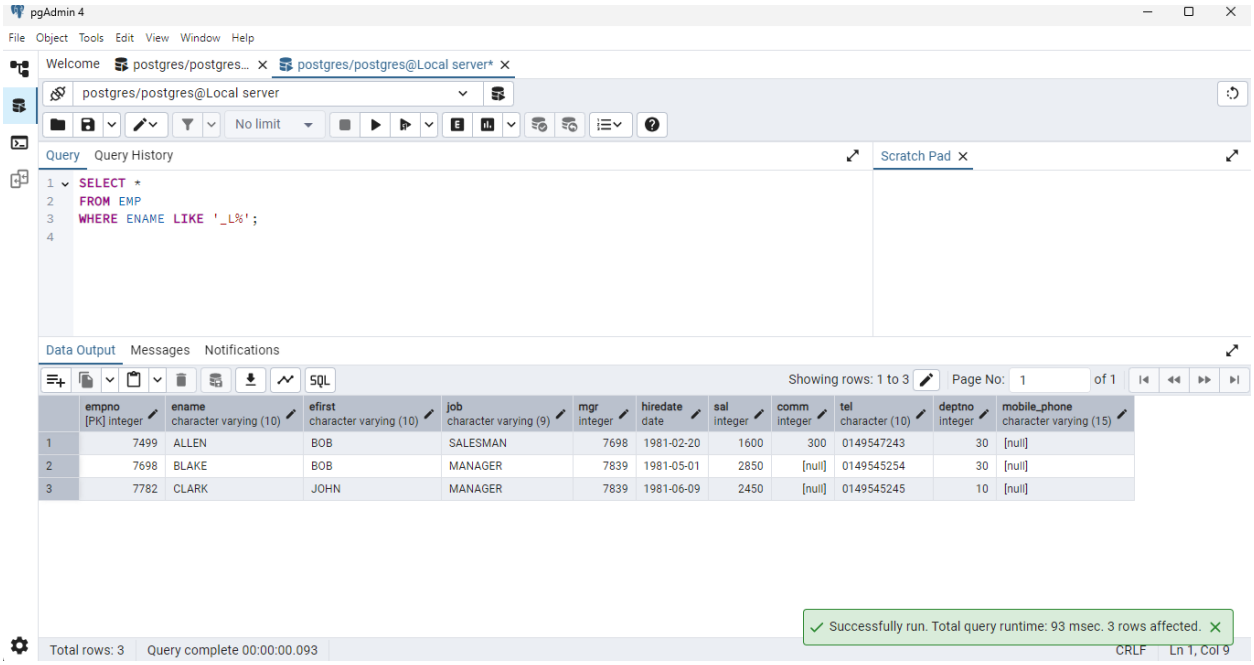
5. Select employees whose name begins by M

```
SELECT *
FROM EMP
WHERE ENAME LIKE 'M%';
```



6. Select employees whose name includes a L in second position

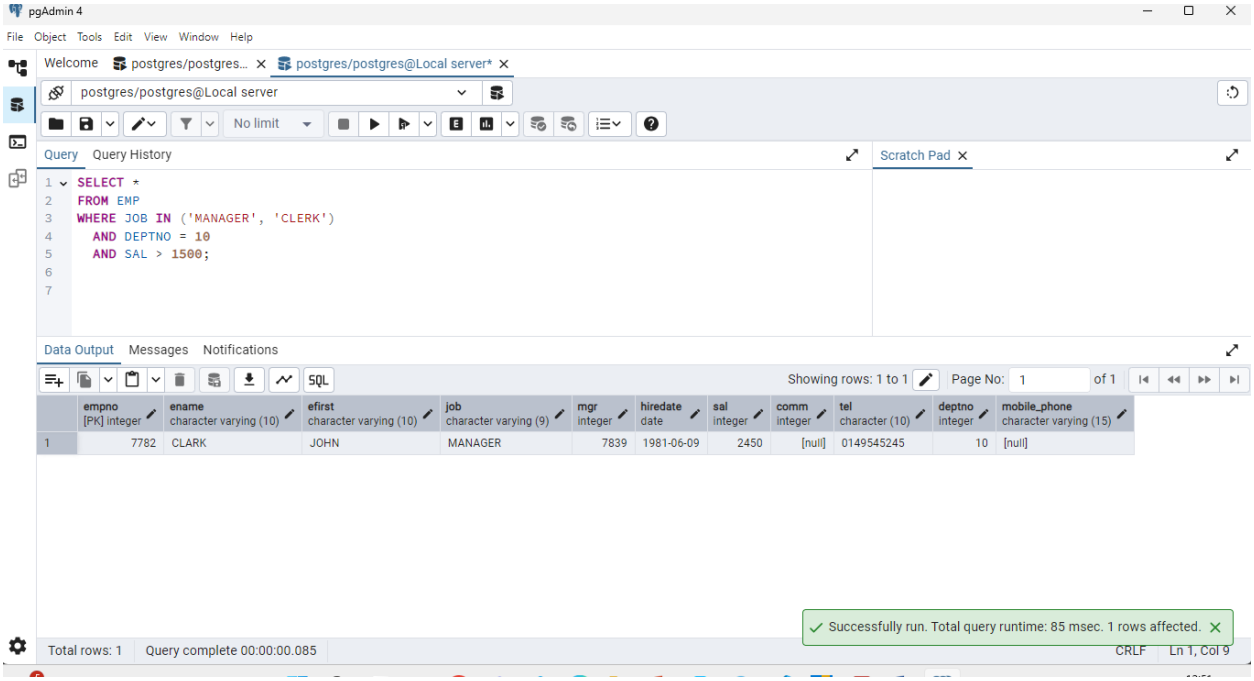
```
SELECT *
FROM EMP
WHERE ENAME LIKE '_L%';
```



7. Select employees who are MANAGER or CLERK in the department 10 and whose

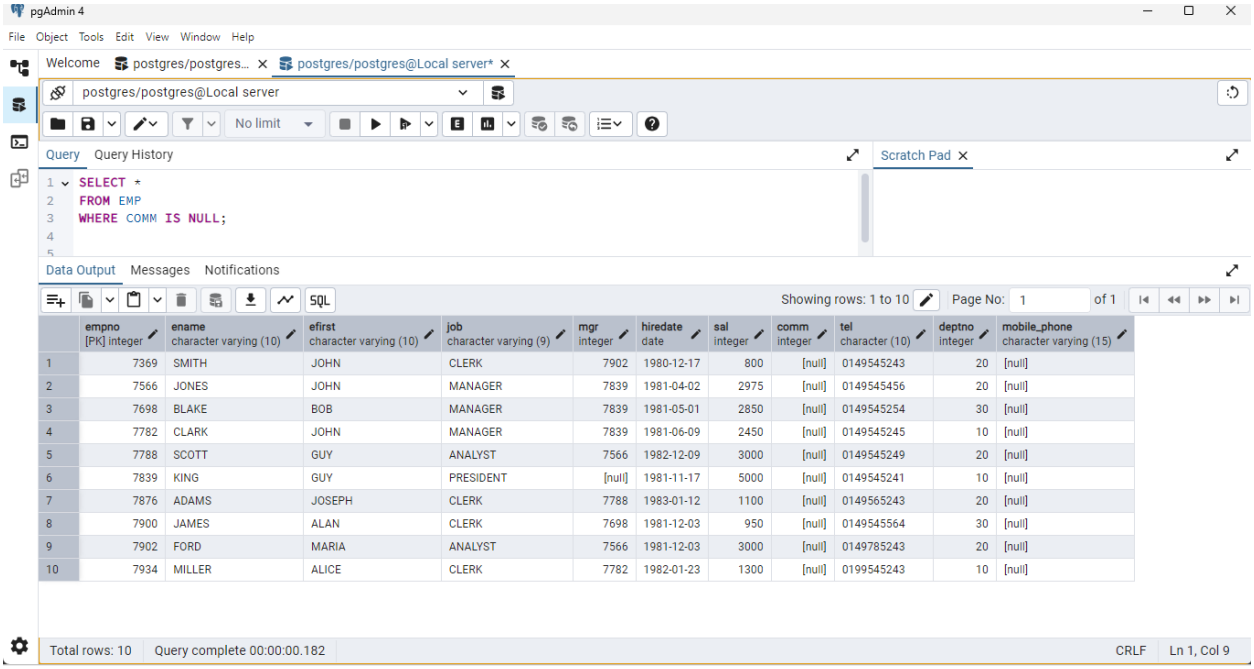
salary is greater than 1500

```
SELECT *
FROM EMP
WHERE JOB IN ('MANAGER', 'CLERK')
  AND DEPTNO = 10
  AND SAL > 1500;
```



8. Select employees whose commission is NULL

```
SELECT *
FROM EMP
WHERE COMM IS NULL;
```

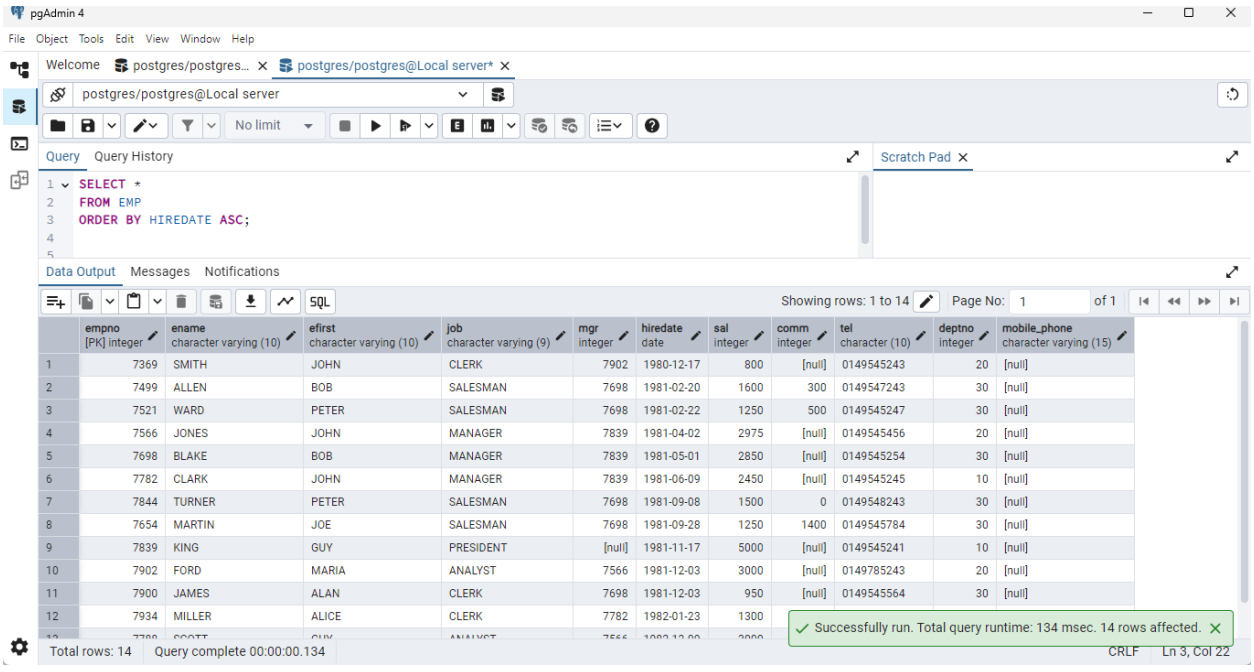


9. Select employees by ascending order (by hiredate)

SELECT *

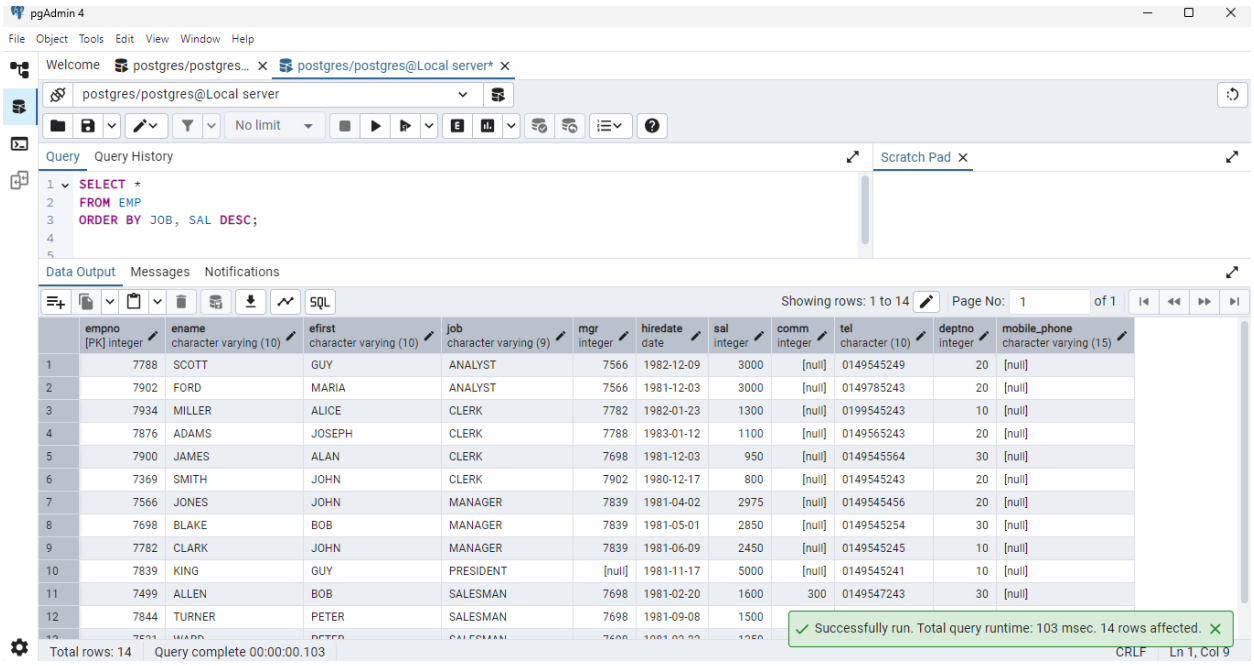
FROM EMP

ORDER BY HIREDATE ASC;



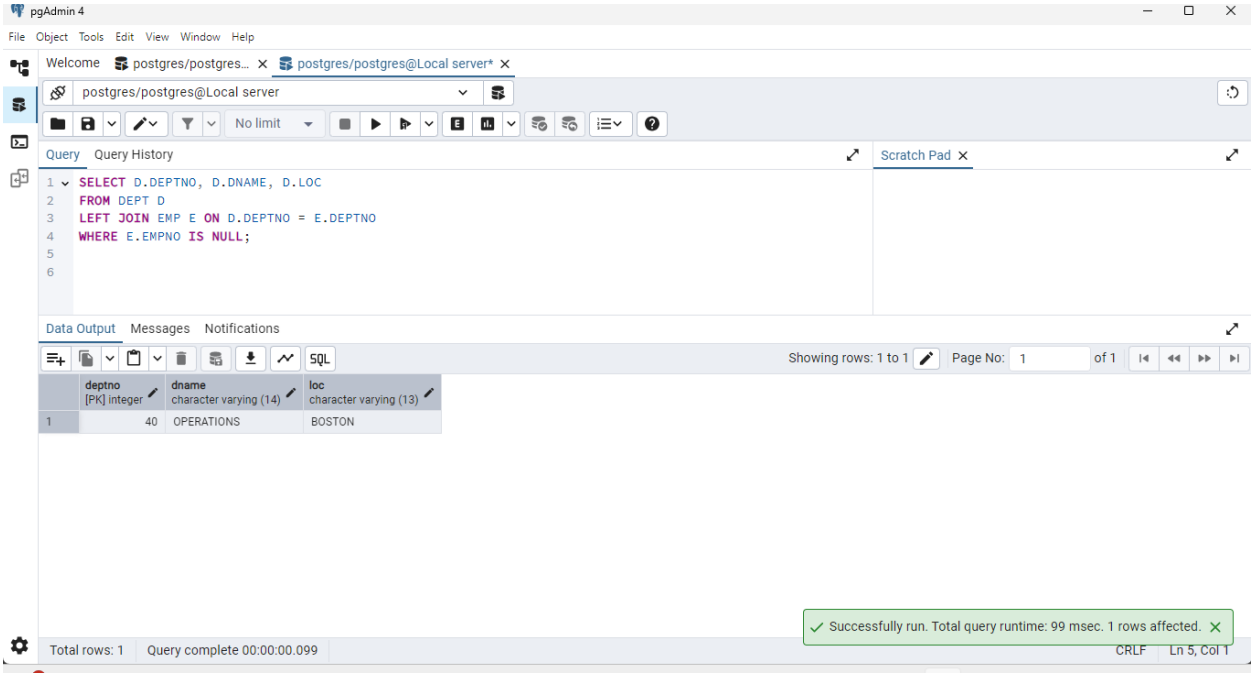
10. Select employees ordered by job, and for each job, by decreasing salary

```
SELECT *
FROM EMP
ORDER BY JOB, SAL DESC;
```



11. Select departments without employees

```
SELECT D.DEPTNO, D.DNAME, D.LOC
FROM DEPT D
LEFT JOIN EMP E ON D.DEPTNO = E.DEPTNO
WHERE E.EMPNO IS NULL;
```

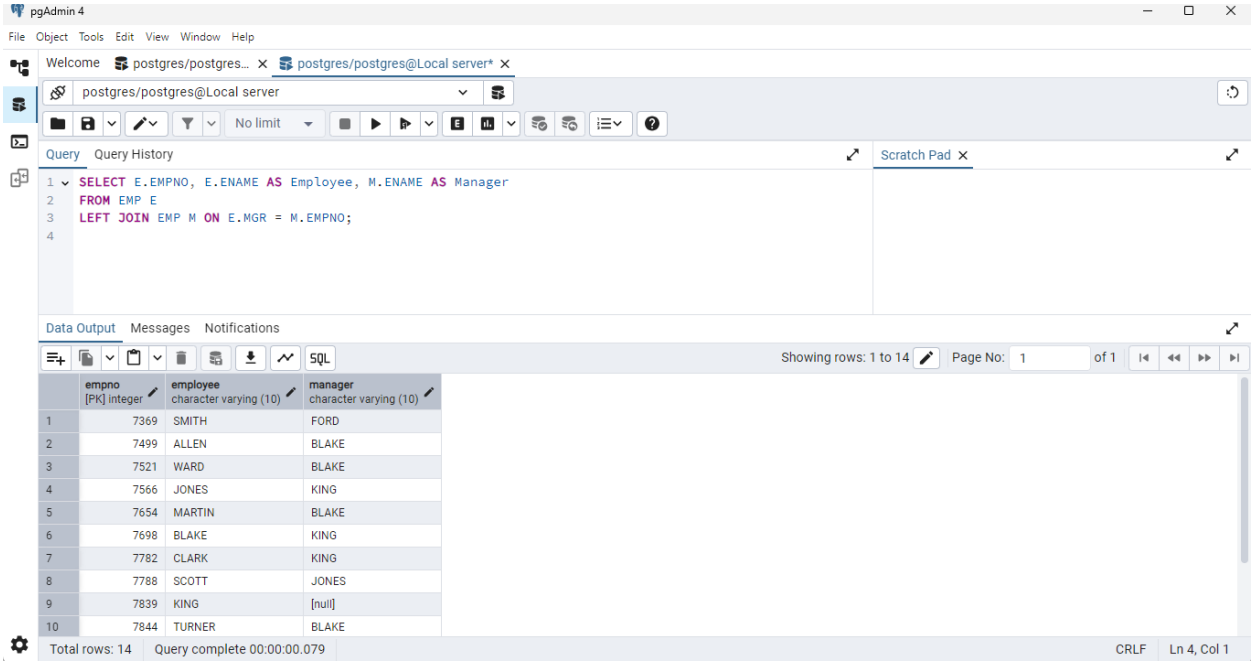


12. List employees indicating for each the name of his/her manager

SELECT E.EMPNO, E.ENAME AS Employee, M.ENAME AS Manager

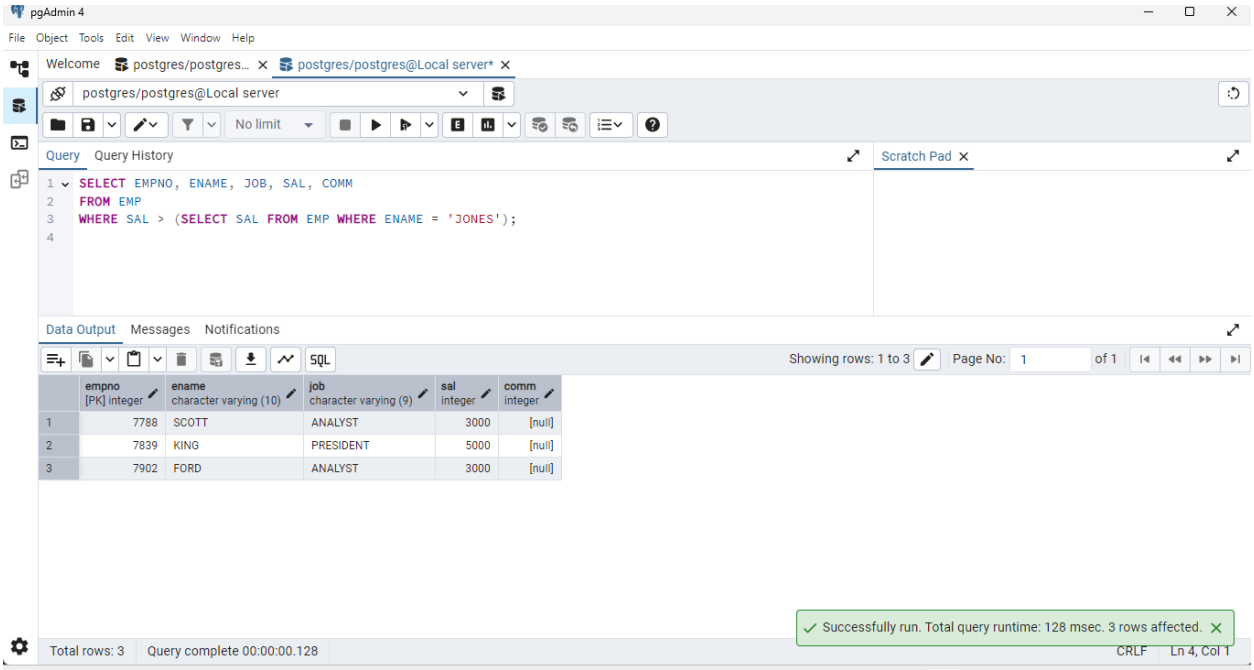
FROM EMP E

LEFT JOIN EMP M ON E.MGR = M.EMPNO;



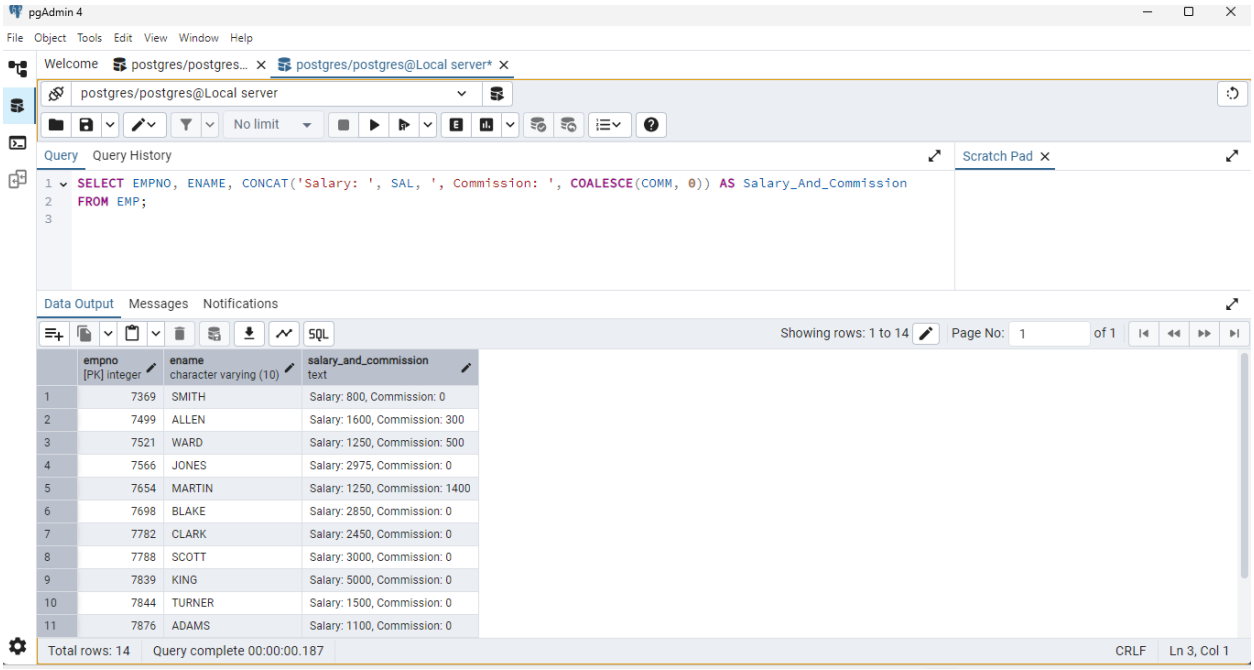
13. List employees earning more than JONES

```
SELECT EMPNO, ENAME, JOB, SAL, COMM
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME = 'JONES');
```



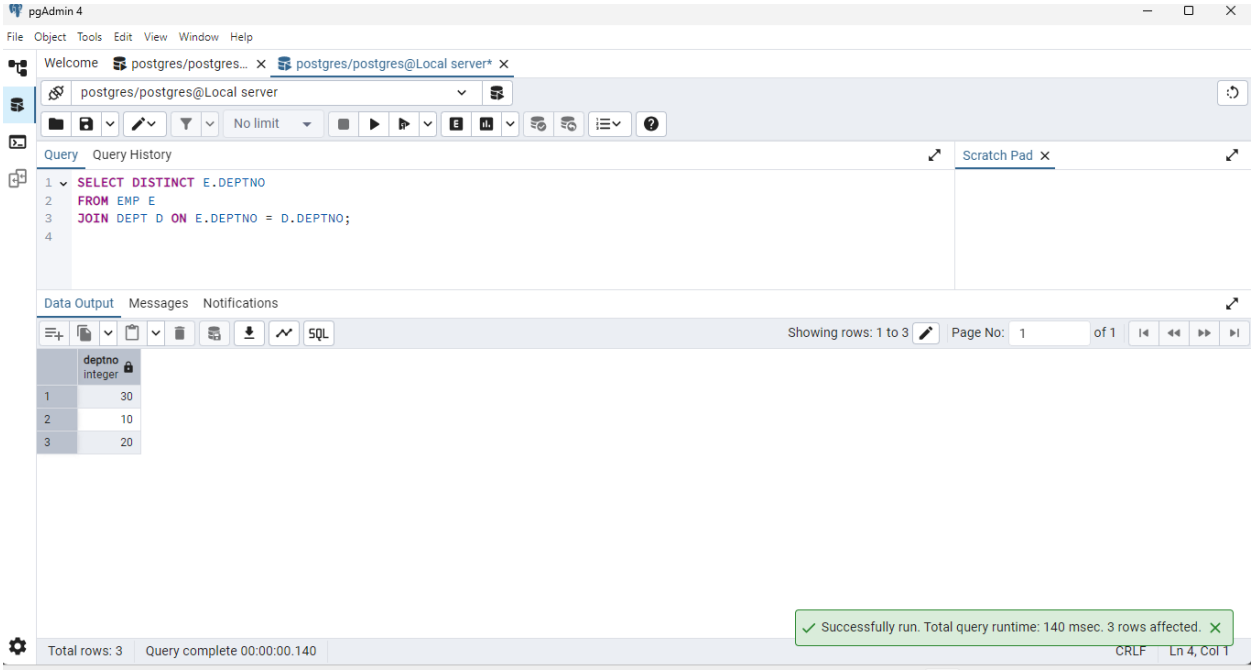
14. List employees displaying in the same column salary and commission

```
SELECT EMPNO, ENAME, CONCAT('Salary: ', SAL, ', Commission: ', COALESCE(COMM, 0))
AS Salary_And_Commission
FROM EMP;
```



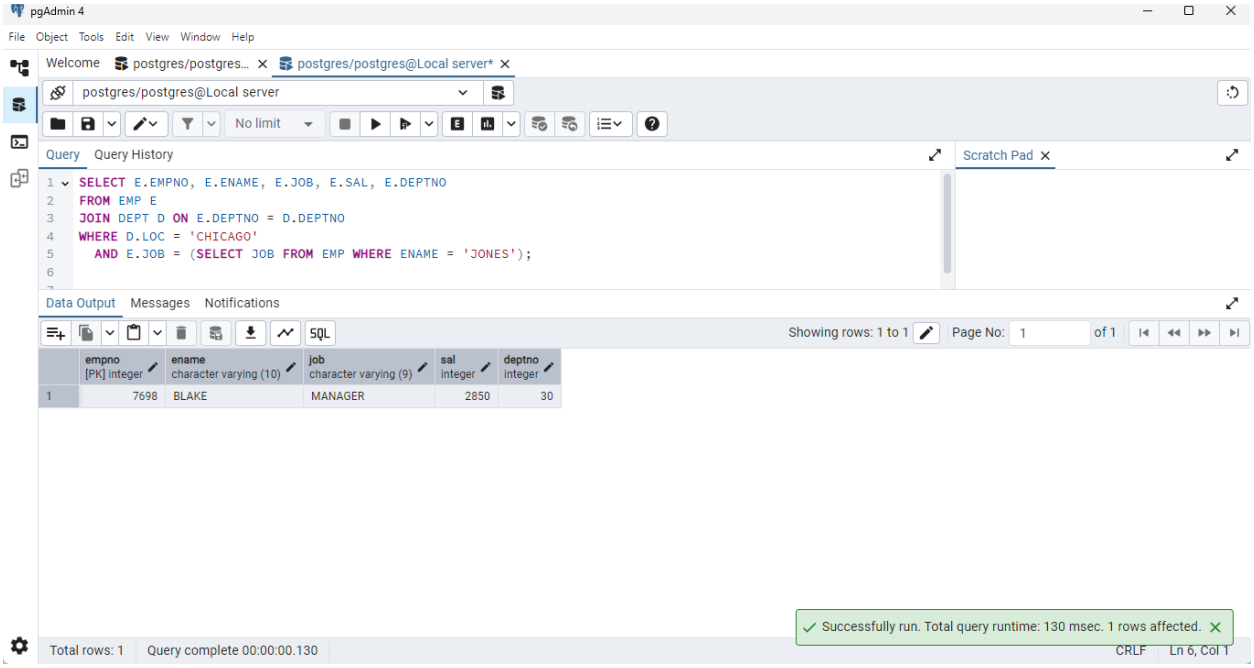
15. List department numbers which are both in table EMP and in table DEPT

```
SELECT DISTINCT E.DEPTNO
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO;
```



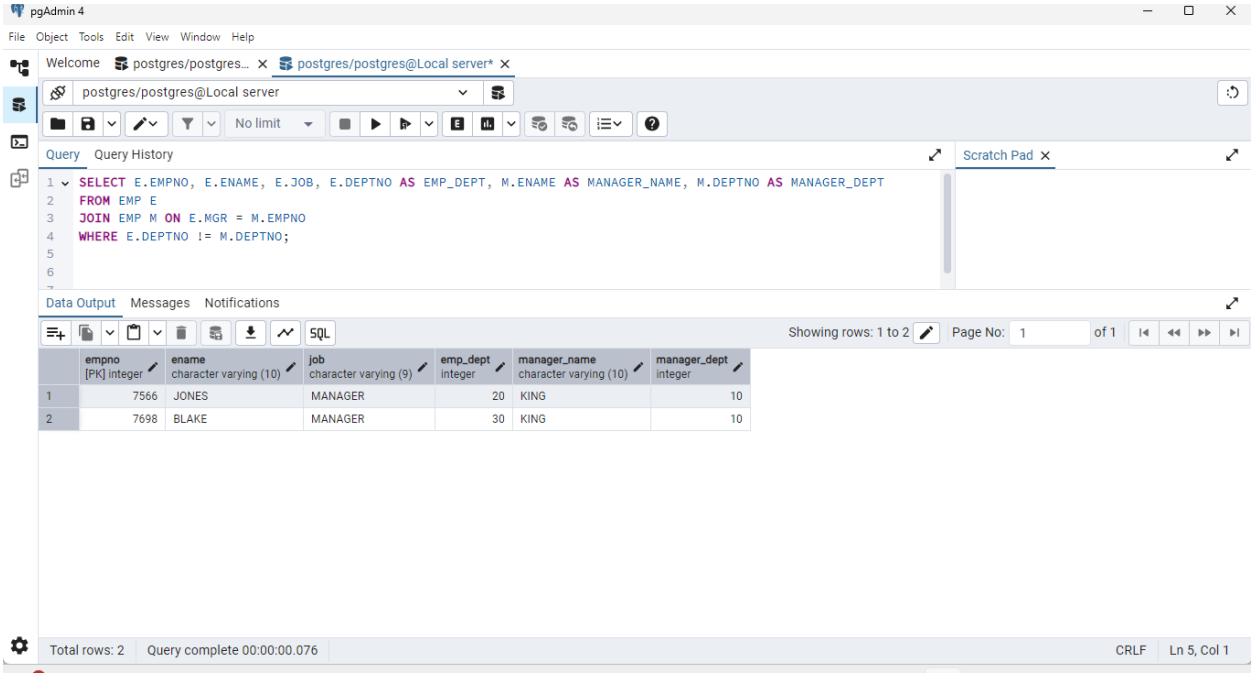
16. List employees working in CHICAGO and having the same job than JONES

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL, E.DEPTNO
FROM EMP E
JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.LOC = 'CHICAGO'
  AND E.JOB = (SELECT JOB FROM EMP WHERE ENAME = 'JONES');
```



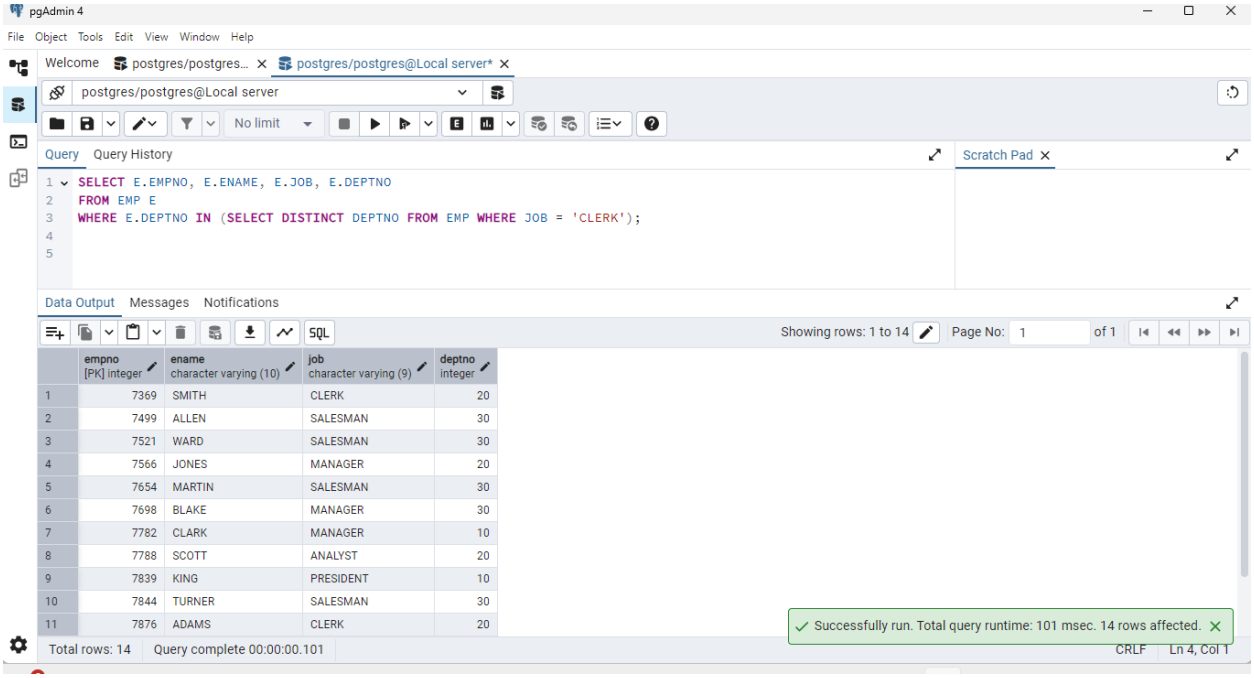
17. List employees who don't work in the same department than their manager

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO AS EMP_DEPT, M.ENAME AS
MANAGER_NAME, M.DEPTNO AS MANAGER_DEPT
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.DEPTNO != M.DEPTNO;
```



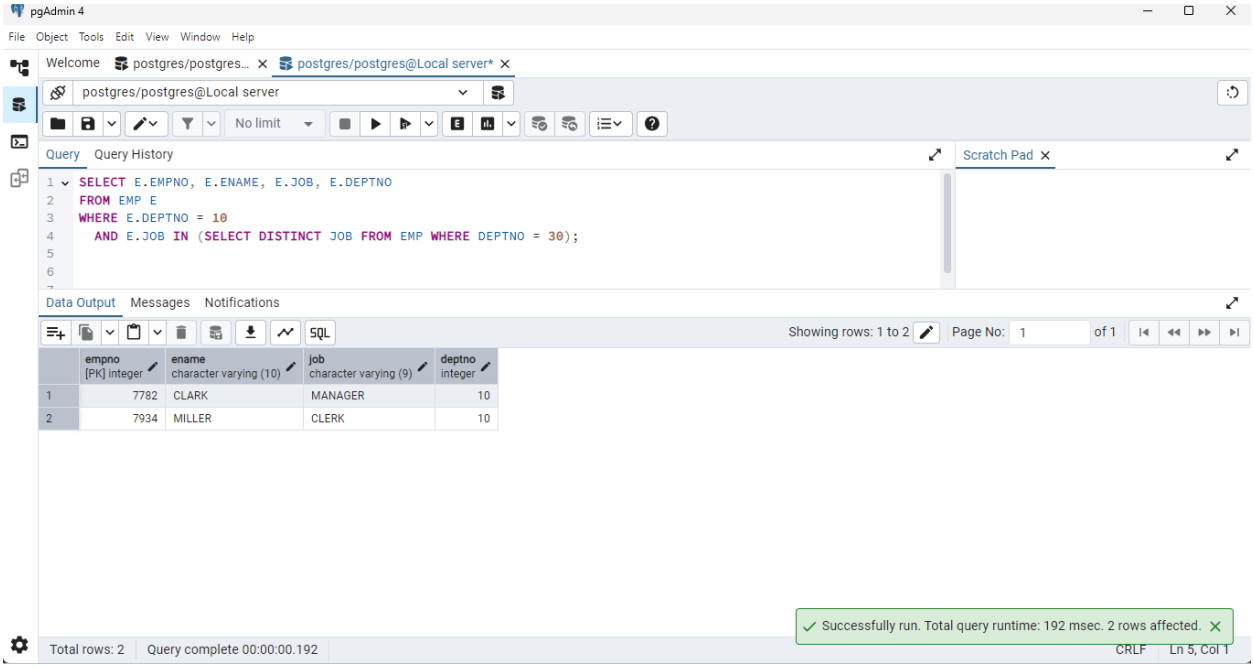
18. List employees working in a department having at least one CLERK

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO
FROM EMP E
WHERE E.DEPTNO IN (SELECT DISTINCT DEPTNO FROM EMP WHERE JOB = 'CLERK');
```



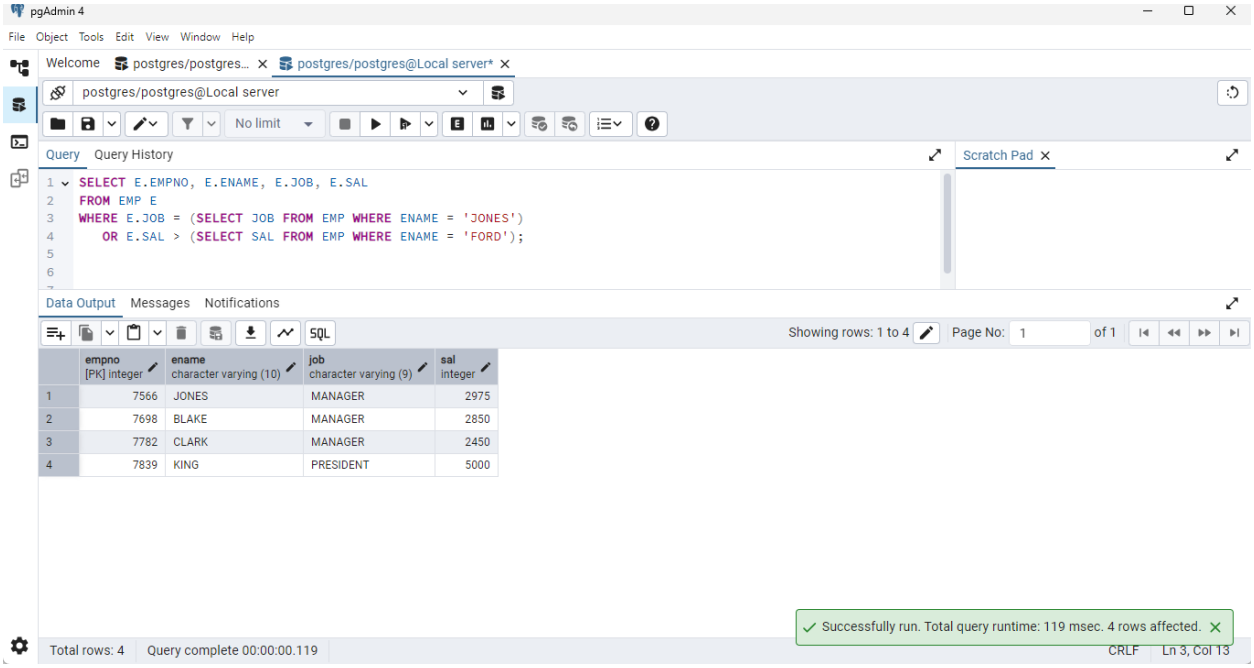
19. List employees of department 10 having the same job than someone from the department SALES

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO
FROM EMP E
WHERE E.DEPTNO = 10
  AND E.JOB IN (SELECT DISTINCT JOB FROM EMP WHERE DEPTNO = 30);
```



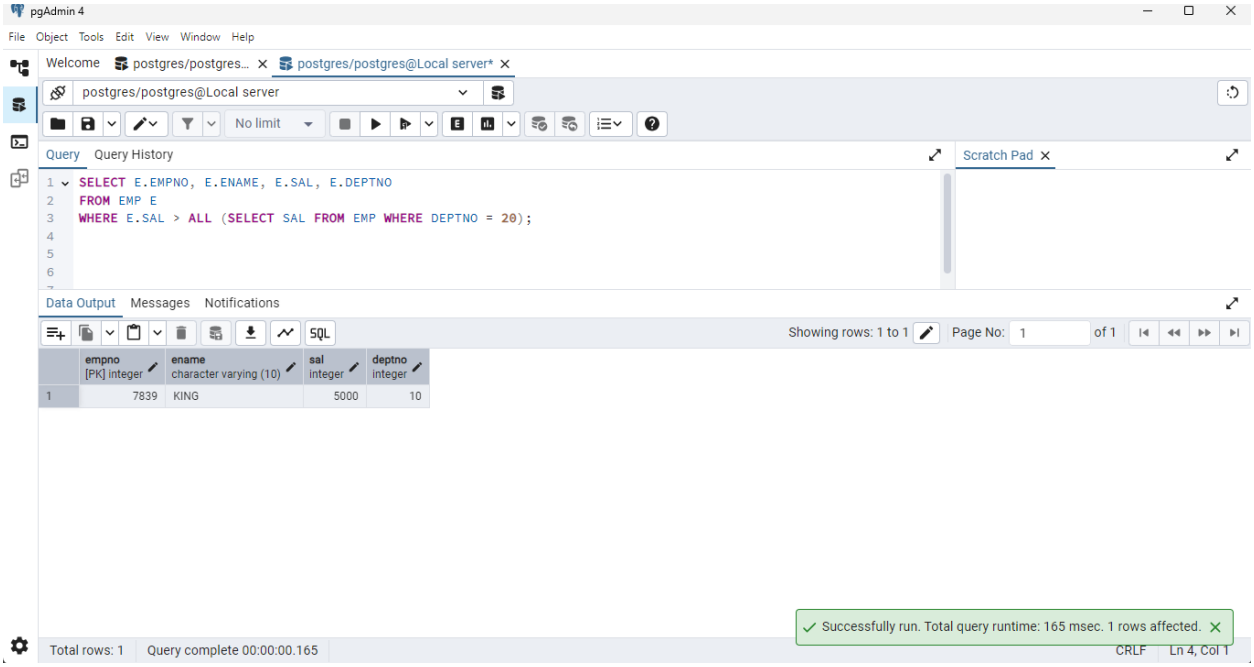
20. List employees having the same job than JONES or a salary greater than FORD's salary

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL
FROM EMP E
WHERE E.JOB = (SELECT JOB FROM EMP WHERE ENAME = 'JONES')
   OR E.SAL > (SELECT SAL FROM EMP WHERE ENAME = 'FORD');
```



21. List employees having a salary greater than all employees of department 20

```
SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO

FROM EMP E

WHERE E.SAL > ALL (SELECT SAL FROM EMP WHERE DEPTNO = 20);
```



Exercise 3: Join Table.

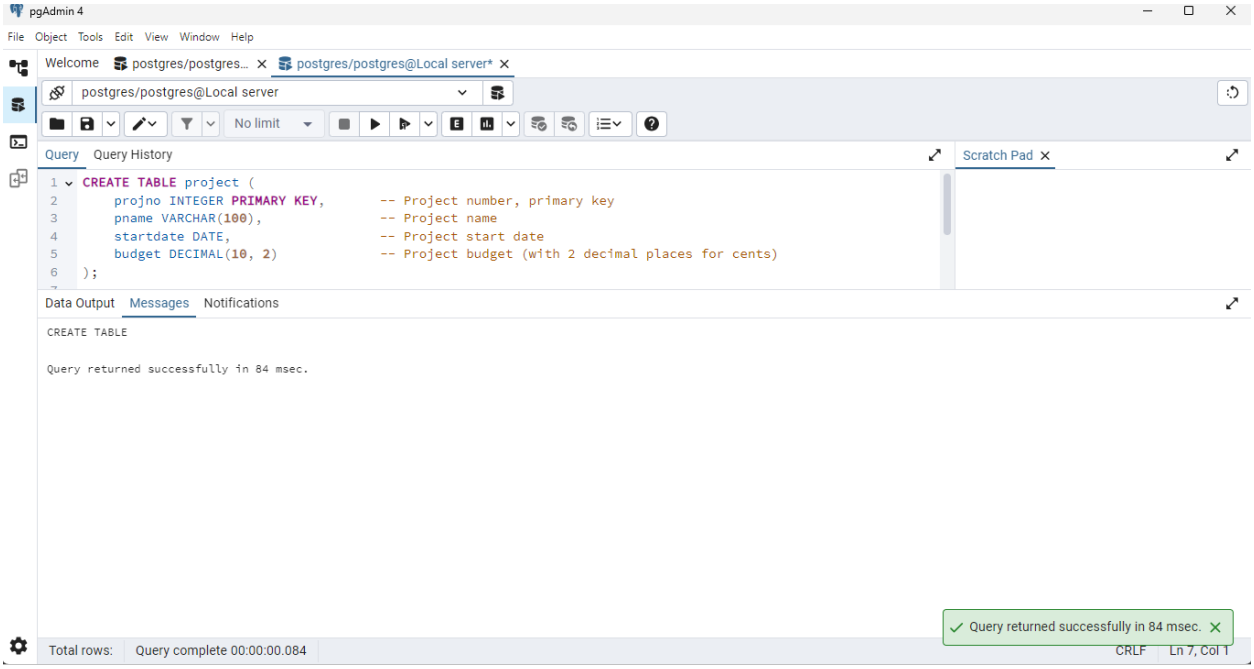
1. Create a Table for employee's Projects: Each employee is working on one or several projects. Create a table "project" containing the number, name, starting date and budget of each project.

Write the corresponding SQL Request

Table name is: "project"

Attributes are: "projno", "pname", "startdate", "budget"

```
CREATE TABLE project (
    projno INTEGER PRIMARY KEY,       -- Project number, primary key
    pname VARCHAR(100),               -- Project name
    startdate DATE,                   -- Project start date
    budget DECIMAL(10, 2)             -- Project budget (with 2 decimal places for cents)
);
```



2. Create the Join Table: An Employee can work on many projects and a project can be affected by many employees. Create the necessary tools for that. Insert some elements in the tables.

Write the SQL query to create the corresponding table (name of the table is : "project_emp"). Tip: don't forget foreign keys

Write some requests to insert elements in the table:

- You should have 4 projects in the project table
- In the join table ‘project_emp’ you should have at least 20 lines.
    - One of the employee should be assigned to all the projects

CREATE TABLE project_emp (

empno INTEGER, -- Employee number (foreign key)

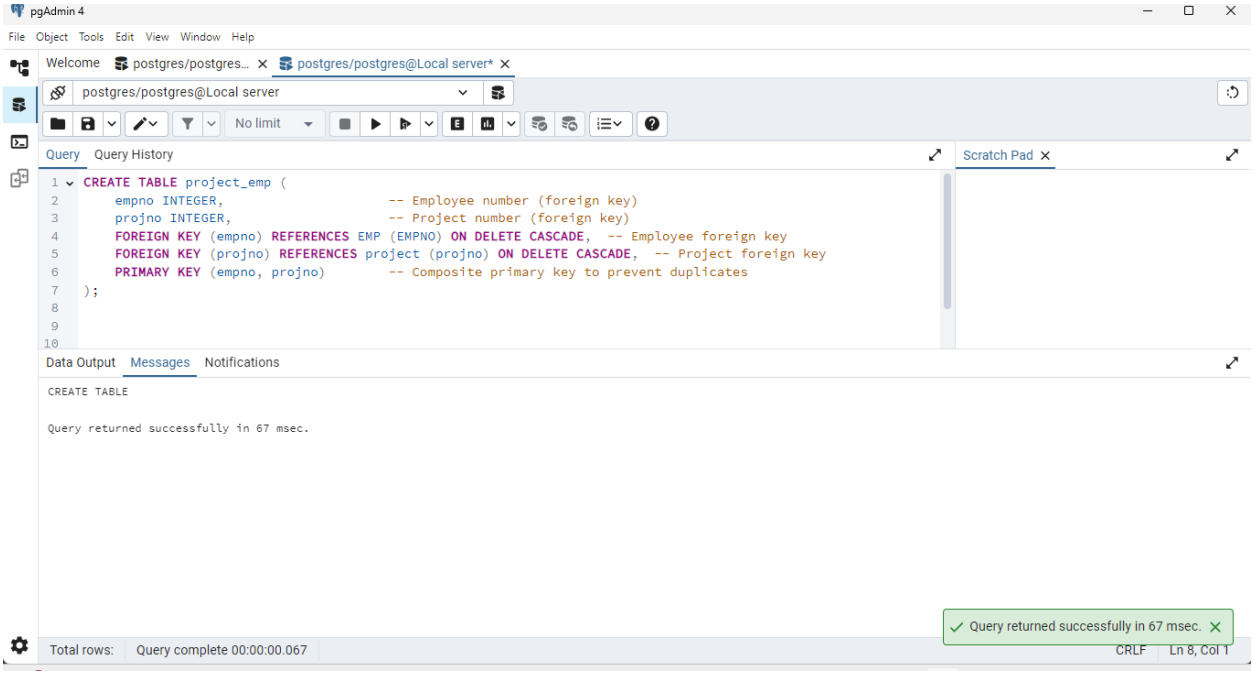
projno INTEGER, -- Project number (foreign key)

FOREIGN KEY (empno) REFERENCES EMP (EMPNO) ON DELETE CASCADE, -- Employee foreign key

FOREIGN KEY (projno) REFERENCES project (projno) ON DELETE CASCADE, -- Project foreign key

PRIMARY KEY (empno, projno) -- Composite primary key to prevent duplicates

);



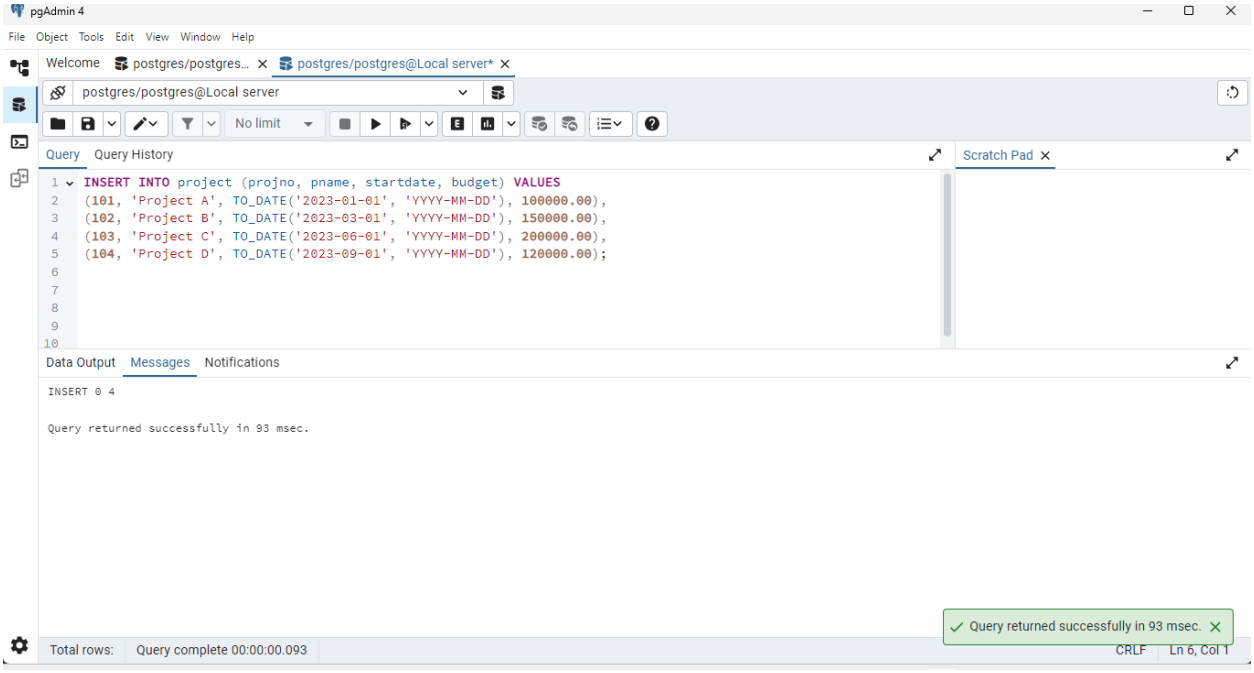
INSERT INTO project (projno, pname, startdate, budget) VALUES

(101, 'Project A', TO_DATE('2023-01-01', 'YYYY-MM-DD'), 100000.00),

(102, 'Project B', TO_DATE('2023-03-01', 'YYYY-MM-DD'), 150000.00),

(103, 'Project C', TO_DATE('2023-06-01', 'YYYY-MM-DD'), 200000.00),

(104, 'Project D', TO_DATE('2023-09-01', 'YYYY-MM-DD'), 120000.00);



INSERT INTO project_emp (empno, projno) VALUES

(7369, 101),

(7499, 101),

(7521, 101),

(7566, 102),

(7654, 102),

(7698, 102),

(7782, 103),

(7788, 103),

(7839, 103),

(7844, 104),

(7876, 104),

(7900, 104),

(7369, 102),

(7499, 103),

(7521, 104),

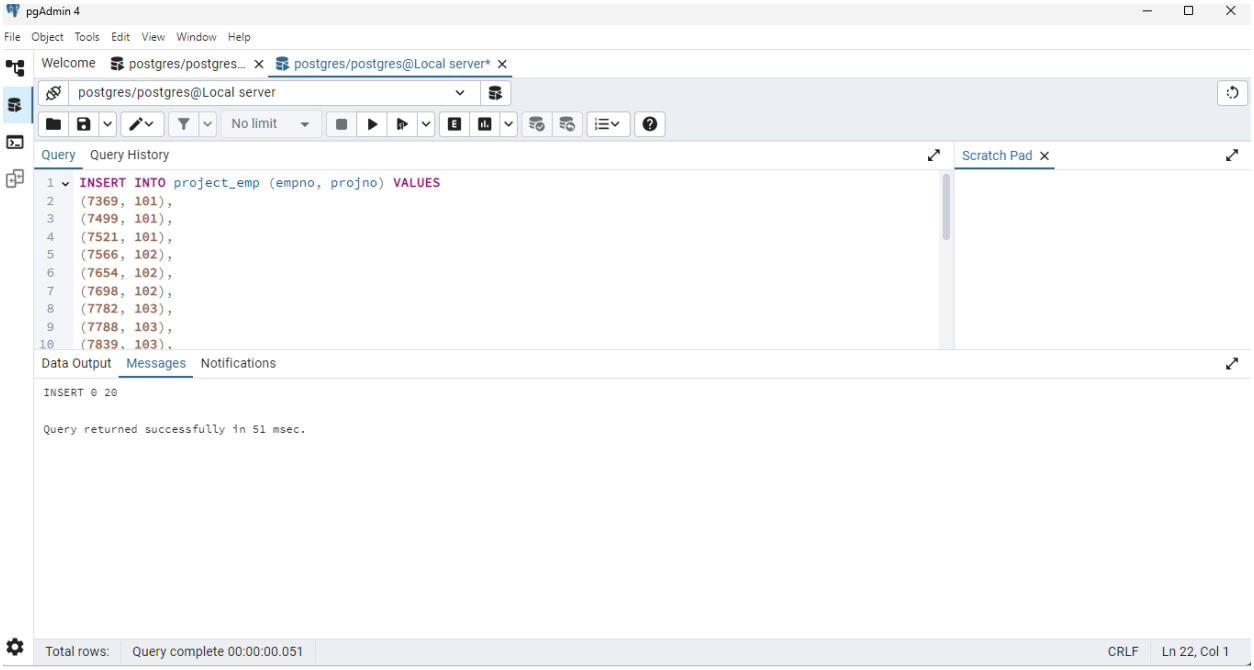
(7566, 103),

(7654, 101),

(7698, 104),

(7782, 102),

(7902, 103);



3. List all employees (by empno) who work on all projects ?

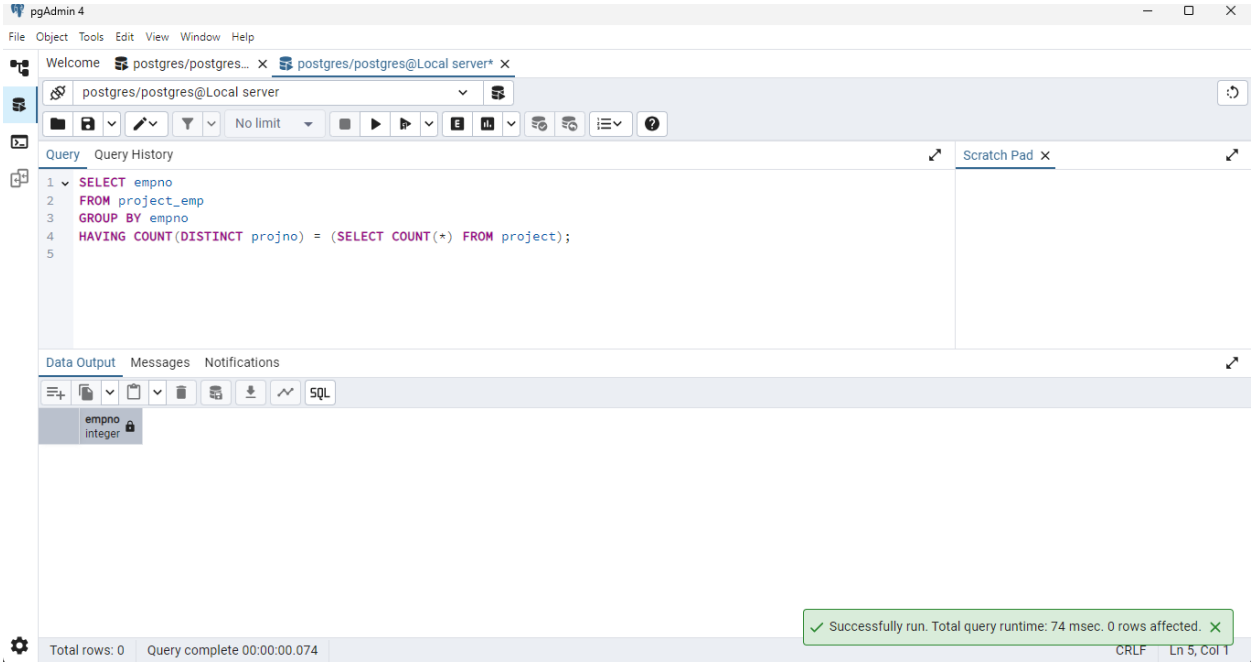
Write 1 SQL query to answer this question, return only the employee number (empno)

SELECT empno

FROM project_emp

GROUP BY empno

HAVING COUNT(DISTINCT projno) = (SELECT COUNT(*) FROM project);



4. Options on View creation: Explain the following instruction

```
CREATE VIEW sales_staff AS

SELECT empno, ename, deptno

    FROM emp

    WHERE deptno = 10 WITH CHECK OPTION
```

The SQL statement provided creates a **view** named sales_staff:

1. **CREATE VIEW sales_staff AS**:

   This part of the command defines a new view called sales_staff. A view in SQL is essentially a saved query that acts like a table, but it does not store data. Instead, it dynamically retrieves data from underlying tables based on the query defined in the view.

In this case, the view is meant to represent employees in a specific department (department 10, which is assumed to be the Sales department).

2. **SELECT empno, ename, deptno FROM emp WHERE deptno = 10**:

   This is the query that defines the data the view will show. It selects the employee number (empno), employee name (ename), and department number (deptno) from the emp table. The WHERE deptno = 10 condition filters the results to only include employees who work in department 10.

3. **WITH CHECK OPTION**:

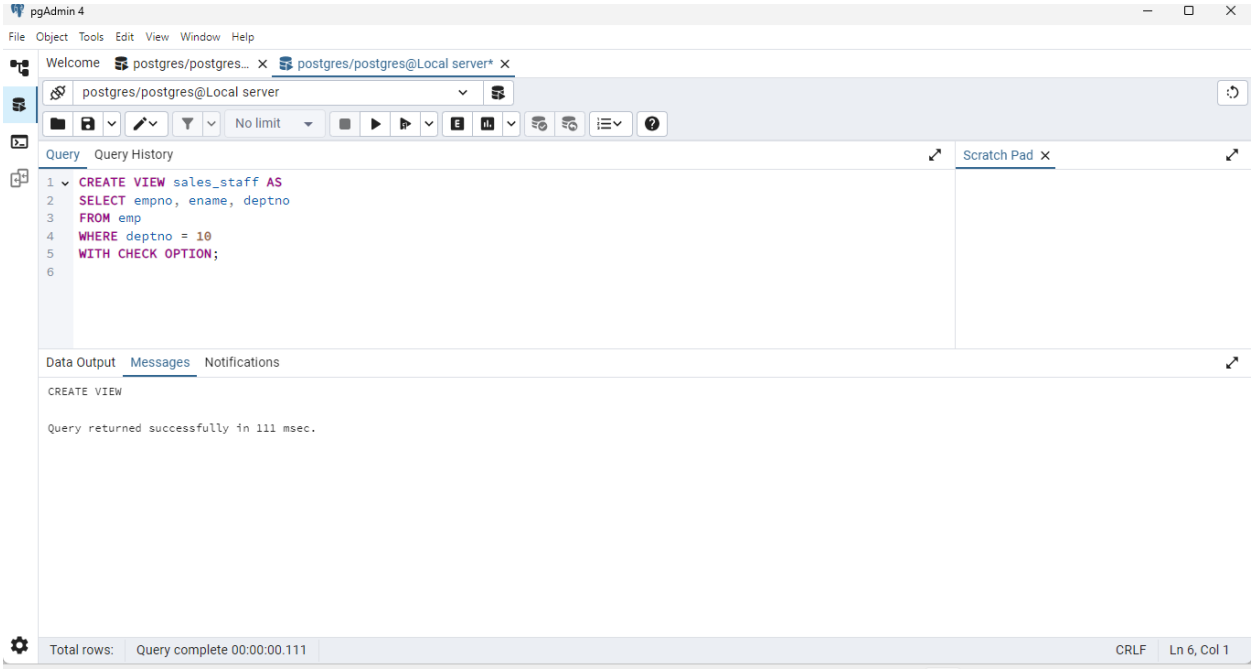
   This is a critical part of the command. The WITH CHECK OPTION ensures that any changes made to the data via this view must meet the criteria defined in the view's SELECT query. In other words, it enforces the condition that only employees from department 10 can be inserted or updated through the sales_staff view. If an insert or update violates this condition, the operation will fail. For example, if a user tries to insert an employee into the view with a deptno different from 10, the database will reject the operation.

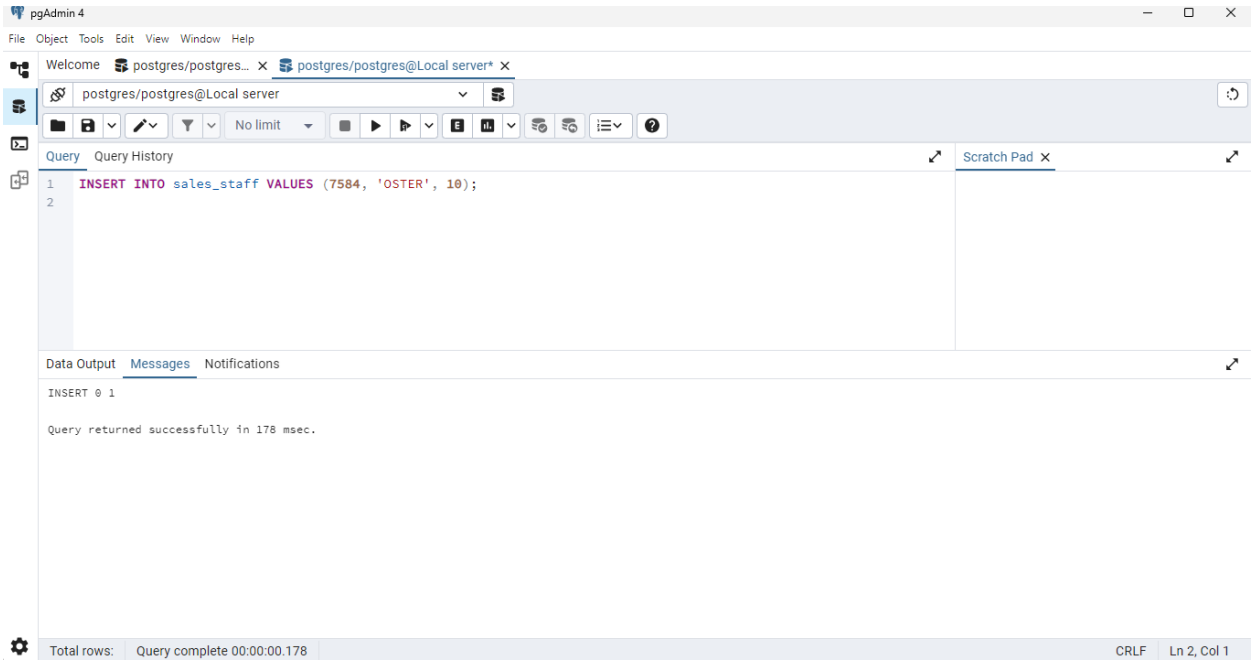
   - This ensures that the view always contains employees from department 10, preserving the integrity of the data in the view.

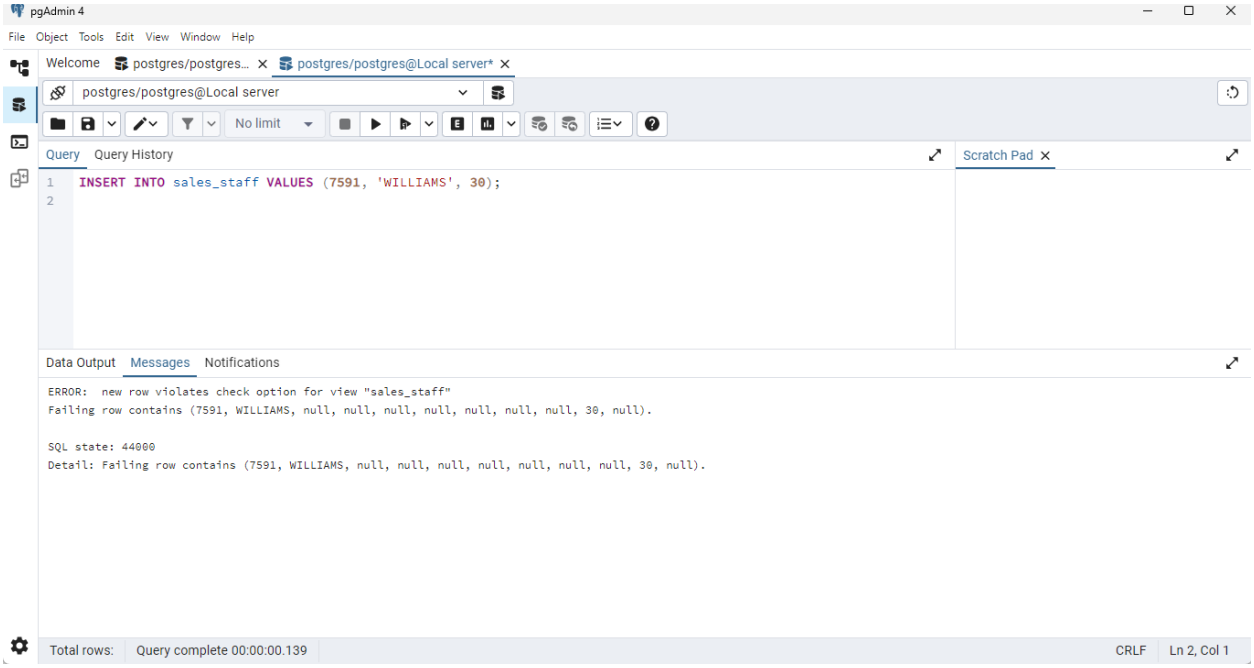
4. **View Creation:** Create the view and try the following queries, explain the result

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);

INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

```
CREATE VIEW sales_staff AS

SELECT empno, ename, deptno

FROM emp

WHERE deptno = 10

WITH CHECK OPTION;
```


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
postgres/postgres...
postgres/postgres@Local server*
postgres/postgres@Local server
No limit
Query
Query History
Scratch Pad
1 CREATE VIEW sales_staff AS
2 SELECT empno, ename, deptno
3 FROM emp
4 WHERE deptno = 10
5 WITH CHECK OPTION;
6
Data Output
Messages
Notifications
CREATE VIEW
Query returned successfully in 111 msec.
Total rows:
Query complete 00:00:00.111
CRLF
Ln 6, Col 1

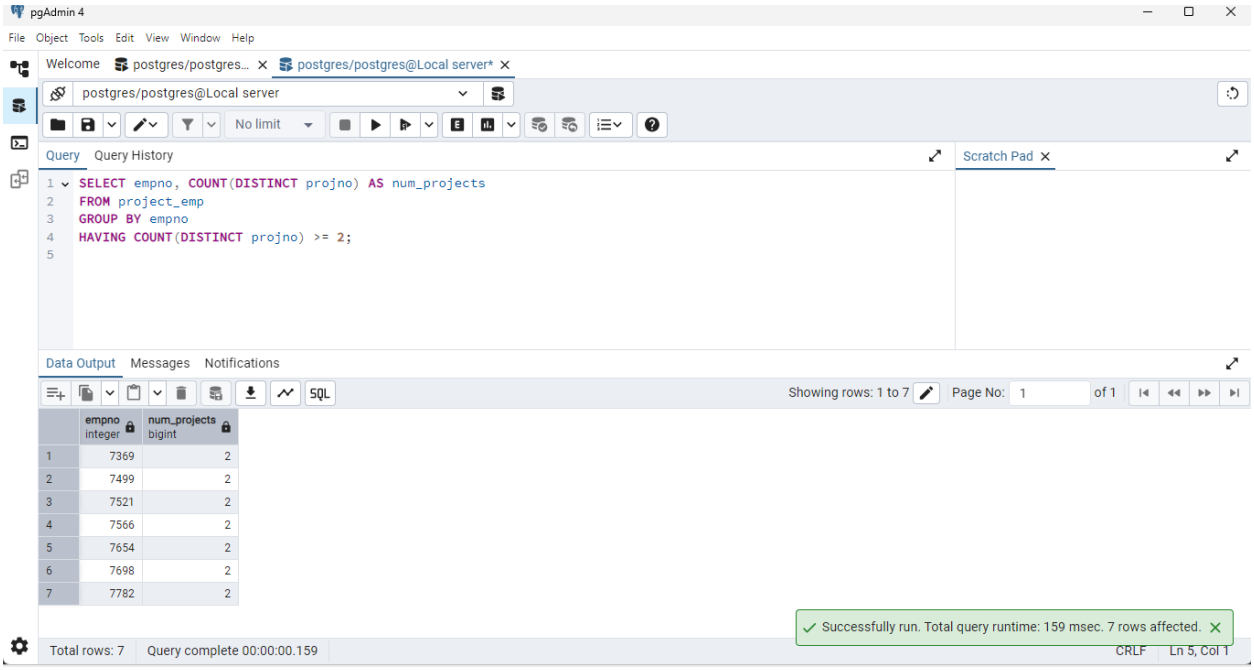

pgAdmin 4
File Object Tools Edit View Window Help
Welcome
postgres/postgres...
postgres/postgres@Local server*
postgres/postgres@Local server
No limit
Query
Query History
Scratch Pad
1 INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
2
Data Output
Messages
Notifications
INSERT 0 1
Query returned successfully in 178 msec.
Total rows:
Query complete 00:00:00.178
CRLF
Ln 2, Col 1



The error message indicates that the new row being inserted (for employee `WILLIAMS` with `empno = 7591` and `deptno = 30`) **violates the `CHECK OPTION`** applied to the `sales_staff` view. The `WITH CHECK OPTION` constraint restricts insertions into the view to rows where `deptno = 10`. Since `deptno = 30` in this case, the insert operation fails, and PostgreSQL generates the error message.

6. Analyze: Give the number of projects per employee and display only employees assigned to at least 2 projects.

```
SELECT empno, COUNT(DISTINCT projno) AS num_projects

FROM project_emp

GROUP BY empno

HAVING COUNT(DISTINCT projno) >= 2;
```



7. Set Queries:

- Find employees who have worked on project number 1 and also on project number 2
- Find employees who have worked on project number 3 but never on project number 4

SELECT empno

FROM project_emp

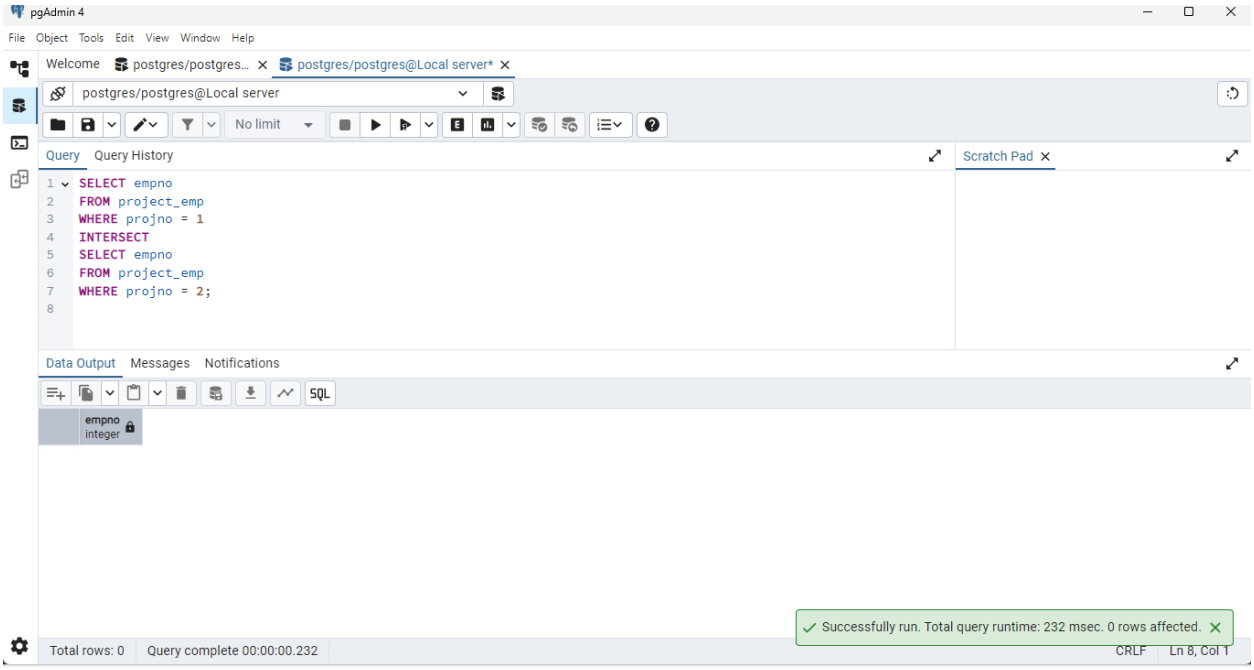
WHERE projno = 1

INTERSECT

SELECT empno

FROM project_emp

WHERE projno = 2;



SELECT empno

FROM project_emp

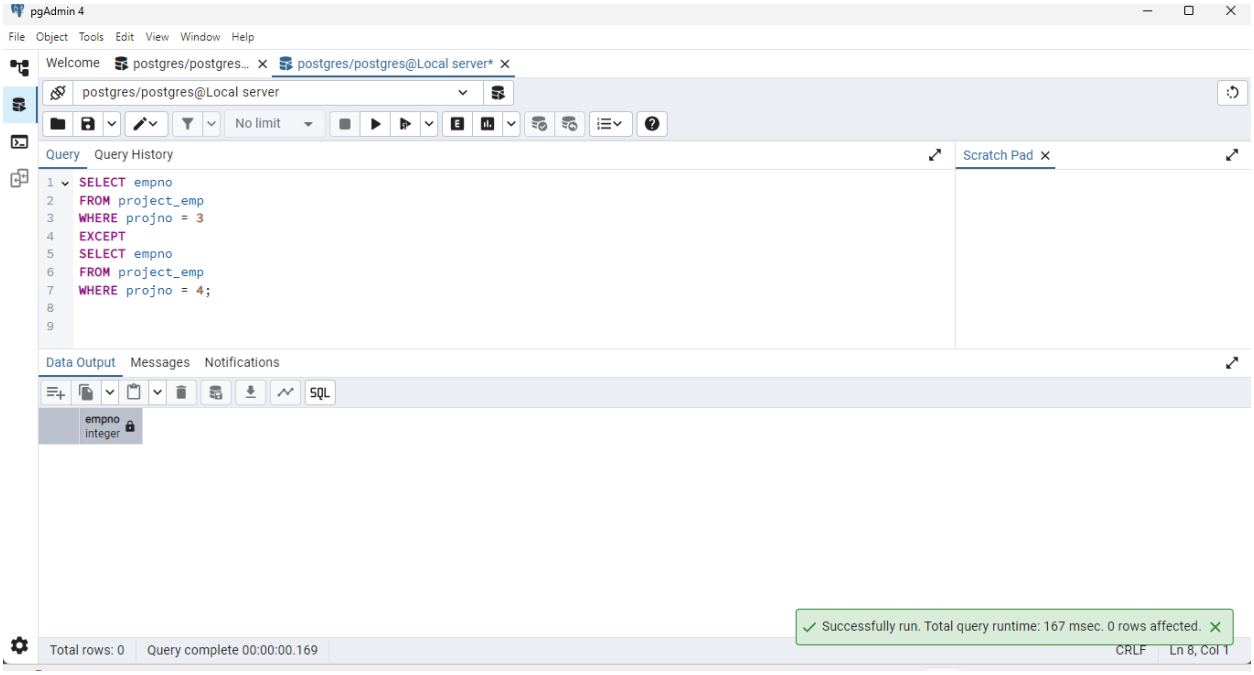
WHERE projno = 3

EXCEPT

SELECT empno

FROM project_emp

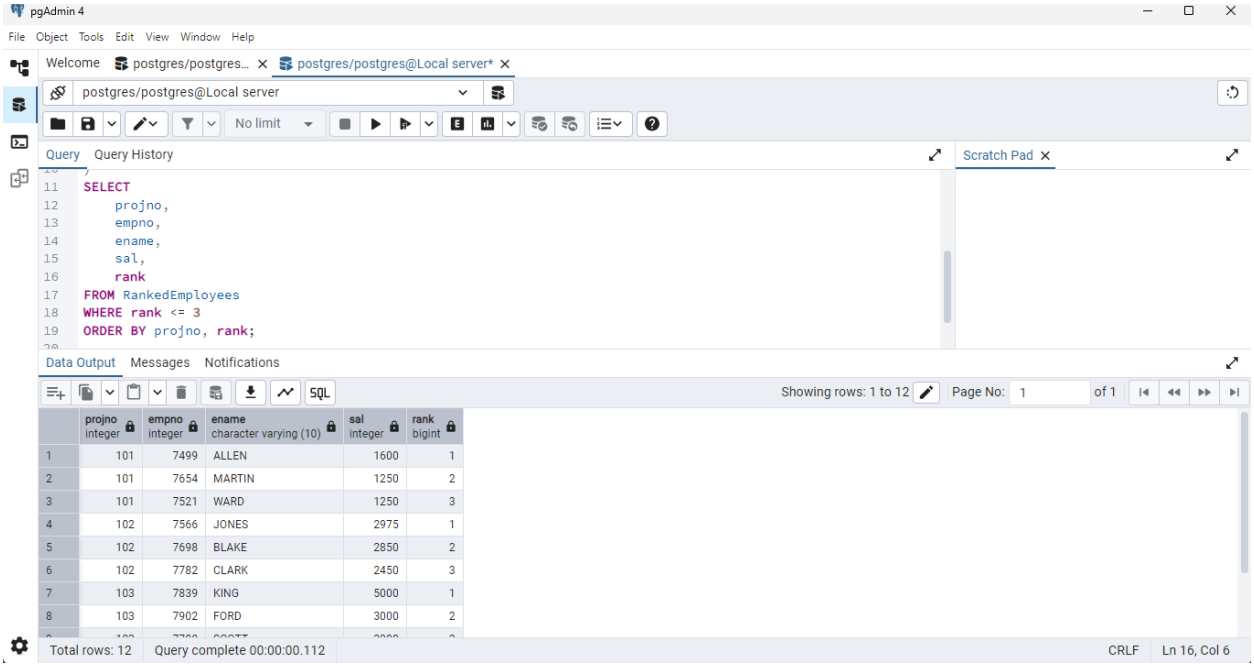
WHERE projno = 4;



8. Find Top 3 employees per project by their salary.

```
WITH RankedEmployees AS (
    SELECT
        pe.projno,
        pe.empno,
        e.ename,
        e.sal,
        ROW_NUMBER() OVER (PARTITION BY pe.projno ORDER BY e.sal DESC) AS rank
    FROM project_emp pe
    JOIN emp e ON pe.empno = e.empno
)
SELECT
    projno,
```

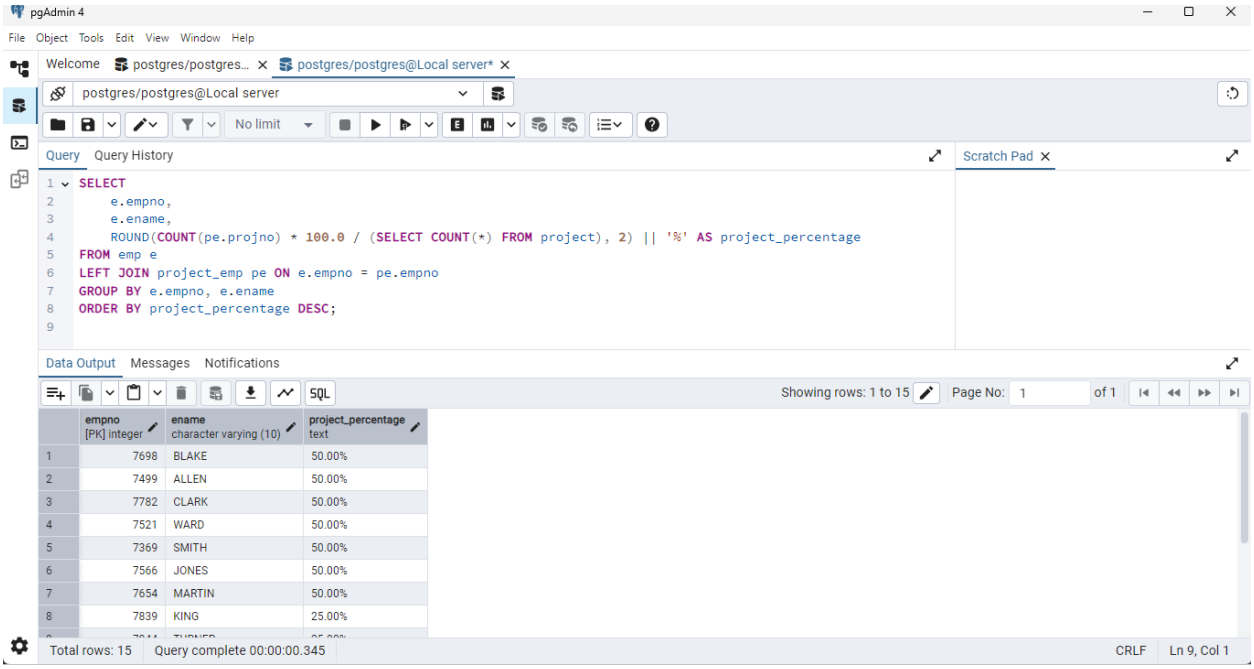
```
    empno,
    ename,
    sal,
        rank
FROM RankedEmployees
WHERE rank <= 3
ORDER BY projno, rank;
```



9.1 For each employee, display the percentage of projects he has been working on (for example, if there are 10 projects in the project table, and the employee is working on 3 projects, you should display 30%).

```
SELECT
    e.empno,
```

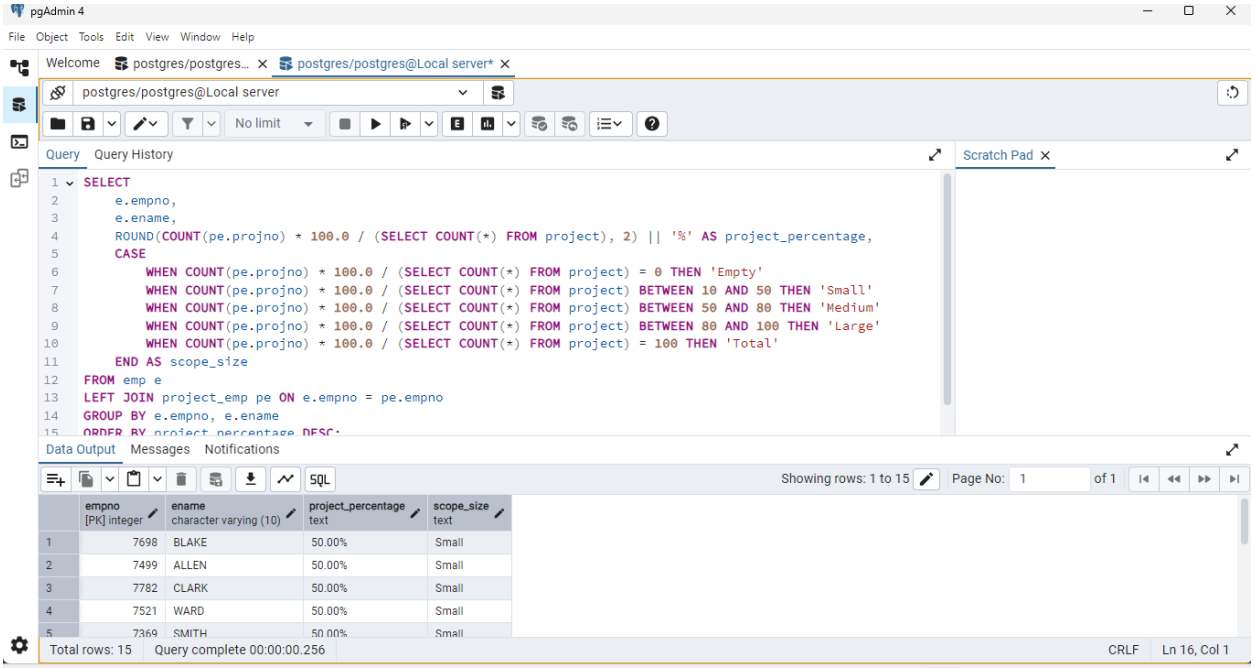
```
    e.ename,
    ROUND(COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project), 2) || '%' AS
project_percentage
FROM emp e
LEFT JOIN project_emp pe ON e.empno = pe.empno
GROUP BY e.empno, e.ename
ORDER BY project_percentage DESC;
```



9.2 Same query than before, but create another column ‘scope_size’:

- If the percentage of projects is 0, indicate value ‘Empty’
- If the percentage of projects is between 10 and 50, indicate value ‘Small’
- If the percentage of projects is between 50 and 80, indicate value ‘Medium’
- If the percentage of projects is between 80 and 100, indicate value ‘Large’
- If the percentage of projects is 100, indicate value ‘Total’

```
SELECT
    e.empno,
    e.ename,
    ROUND(COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project), 2) || '%' AS
project_percentage,
    CASE
        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 0 THEN 'Empty'
        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 10 AND
50 THEN 'Small'
        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 50 AND
80 THEN 'Medium'
        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 80 AND
100 THEN 'Large'
        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 100 THEN 'Total'
    END AS scope_size
FROM emp e
LEFT JOIN project_emp pe ON e.empno = pe.empno
GROUP BY e.empno, e.ename
ORDER BY project_percentage DESC;
```


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
postgres/postgres...
postgres/postgres@Local server*
postgres/postgres@Local server
No limit
Query
Query History
Scratch Pad
SELECT
e.empno,
e.ename,
ROUND(COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project), 2) || '%' AS project_percentage,
CASE
WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 0 THEN 'Empty'
WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 10 AND 50 THEN 'Small'
WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 50 AND 80 THEN 'Medium'
WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 80 AND 100 THEN 'Large'
WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 100 THEN 'Total'
END AS scope_size
FROM emp e
LEFT JOIN project_emp pe ON e.empno = pe.empno
GROUP BY e.empno, e.ename
ORDER BY project_percentage DESC;
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 15
Page No: 1
of 1
empno [PK] integer
ename character varying (10)
project_percentage text
scope_size text
1 7698 BLAKE 50.00% Small
2 7499 ALLEN 50.00% Small
3 7782 CLARK 50.00% Small
4 7521 WARD 50.00% Small
5 7369 SMITH 50.00% Small
Total rows: 15
Query complete 00:00:00.256
CRLF
Ln 16, Col 1

TP 2 - A Scene Of Information And Flexibility

Java and DataBase Connectivity (JDBC)

Part I : A Graphic Of Table: communication with database

1. Load Driver

Open your favorite IDE (I recommend IntelliJ, The Free Community Edition is great). Create a Java Project with a class Main.java.

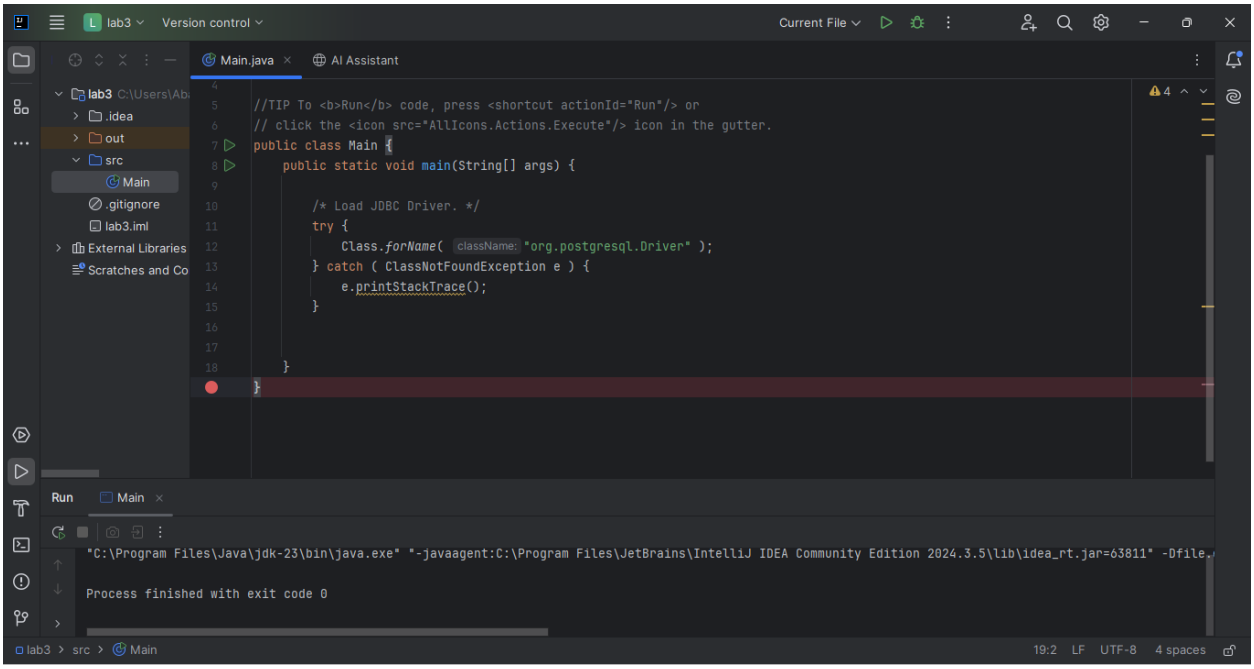
Copy the connector "**postgresql-42.7.3.jar**" in the project.
Add the jar to the Java Build Path, for that:

- On IntelliJ: File => Project Structure => Libraries => Add Library
- On Eclipse: Project => Properties => Java Build Path => Libraries => Add Jars

You will find it on Moodle (https://moodle-ovh.isep.fr/moodle/mod/resource/view.php?id=24671) or here: https://jdbc.postgresql.org/download/

Add At the top of the Main.java:

```
/* Load JDBC Driver. */
try {
    Class.forName( "org.postgresql.Driver" );
} catch ( ClassNotFoundException e ) {
    e.printStackTrace();
}
...
```



2. Connect to bdd

Note: If you are on Windows Family and use Local/legs connection or Bequeath, there is a struggle to find the right JDBC URL connection. If you succeed to connect with Windows Family on your local database with JDBC, please indicate tips here in the comment.

First you need to get the url specific to jdbc in order to connect to your database.

With Postgres the URL is like this:

jdbc:postgresql://host/database

https://jdbc.postgresql.org/documentation/use/

Connection is made through the DriverManager. You will call its method getConnection() to have an object of type Connection.

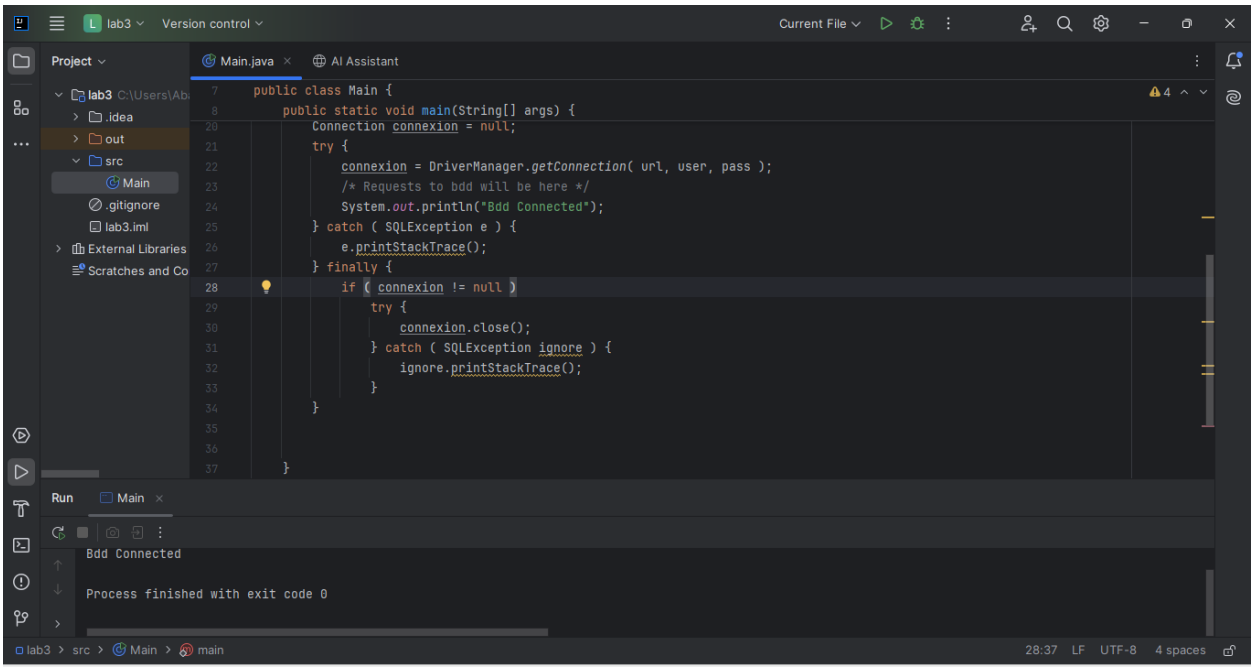
It will look like something like this:

jdbc:postgresql://localhost/thibautdebroca

Don’t forget to close the request in the block “Finally”. If you keep too many opened connections under a small amount of time, the server will get saturated. If you don’t need a connection anymore, close it !

```
String url = "jdbc:postgresql://localhost/postgres";
String user = "postgres";
String pass = "";
Connection connexion = null;
try {
    connexion = DriverManager.getConnection( url, user, pass );

    /* Requests to bdd will be here */
    System.out.println("Bdd Connected");


} catch ( SQLException e ) {
       e.printStackTrace();
} finally {
    if ( connexion != null )
        try {
            connexion.close();
        } catch ( SQLException ignore ) {
              ignore.printStackTrace();
        }
}
```

Don’t forget to add imports on the top of your Java Class:



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

3. Make a request

In order to make a request, you need a Statement object.
How to get it ? You just have to call createStatement() from the Connection object.

```
Statement statement = connexion.createStatement();
```

Statement will provide you several methods especially:

- executeQuery() : dedicated to SELECT request. It returns a ResultSet containing all rows asked in the query,
- executeUpdate() : for queries that have effect on the bdd'state: UPDATE, DELETE, INSERT... It returns an Integer :
    - INSERT returns 0 if case of failure and 1 when succes
    - UPDATE or DELETE returns the number of rows affected.
    - CREATE returns 0.

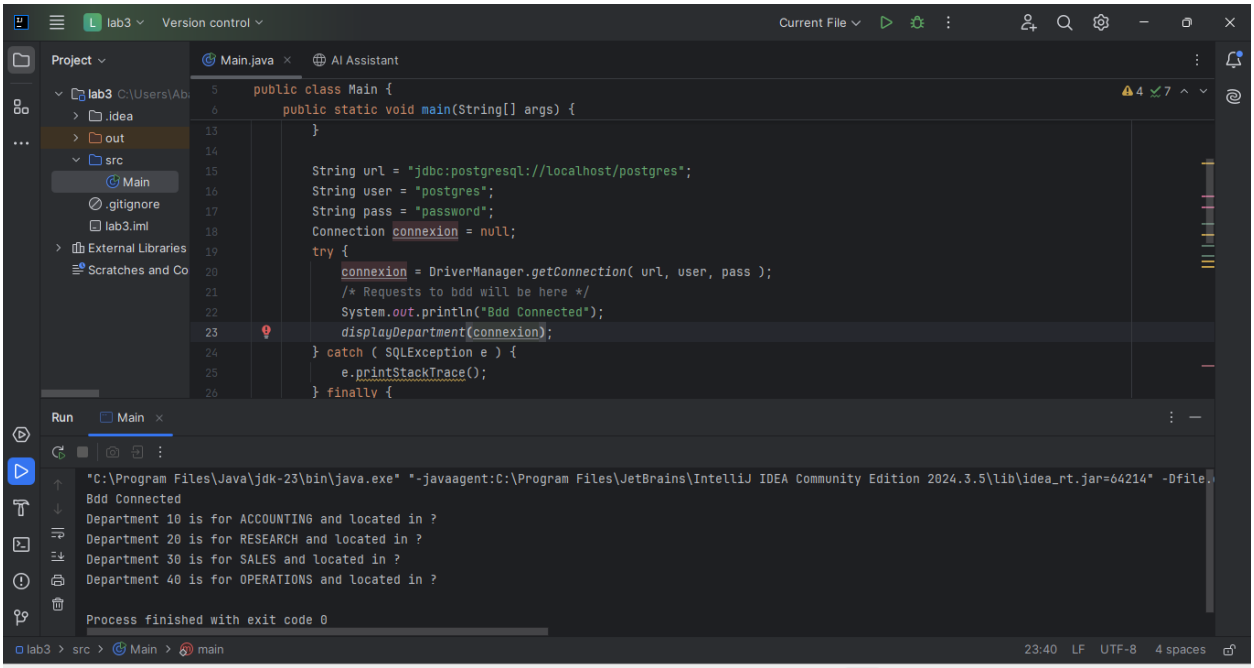
Don't forget to close Statement when you have finished to use it:

```
statement.close();
```

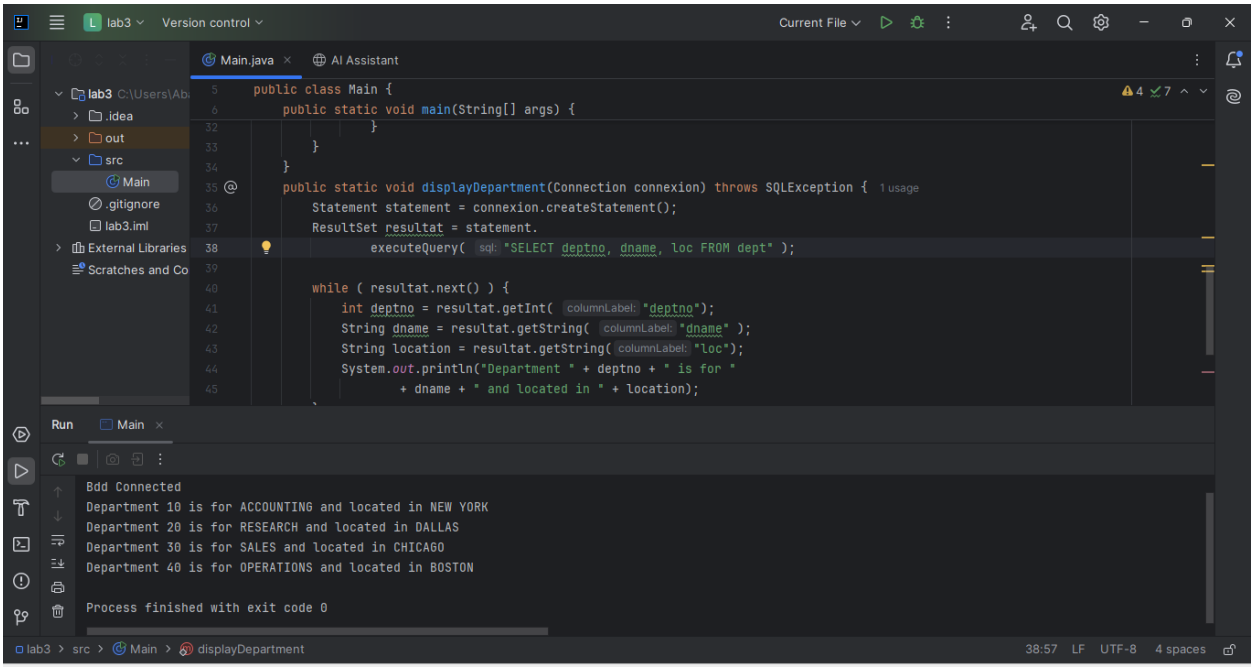
Create a method displayDepartment(Connection connexion) that will display all the departments.

```
public static void displayDepartment(Connection connexion) throws SQLException {
        Statement statement = connexion.createStatement();
        ResultSet resultat = statement.
                        executeQuery( "SELECT deptno, dname FROM dept" );

        while ( resultat.next() ) {
                int deptno = resultat.getInt( "deptno");
                String dname = resultat.getString( "dname" );

                System.out.println("Department " + deptno + " is for "
                                + dname + " and located in ? ");
```

```
            }
            resultat.close();
        }
```

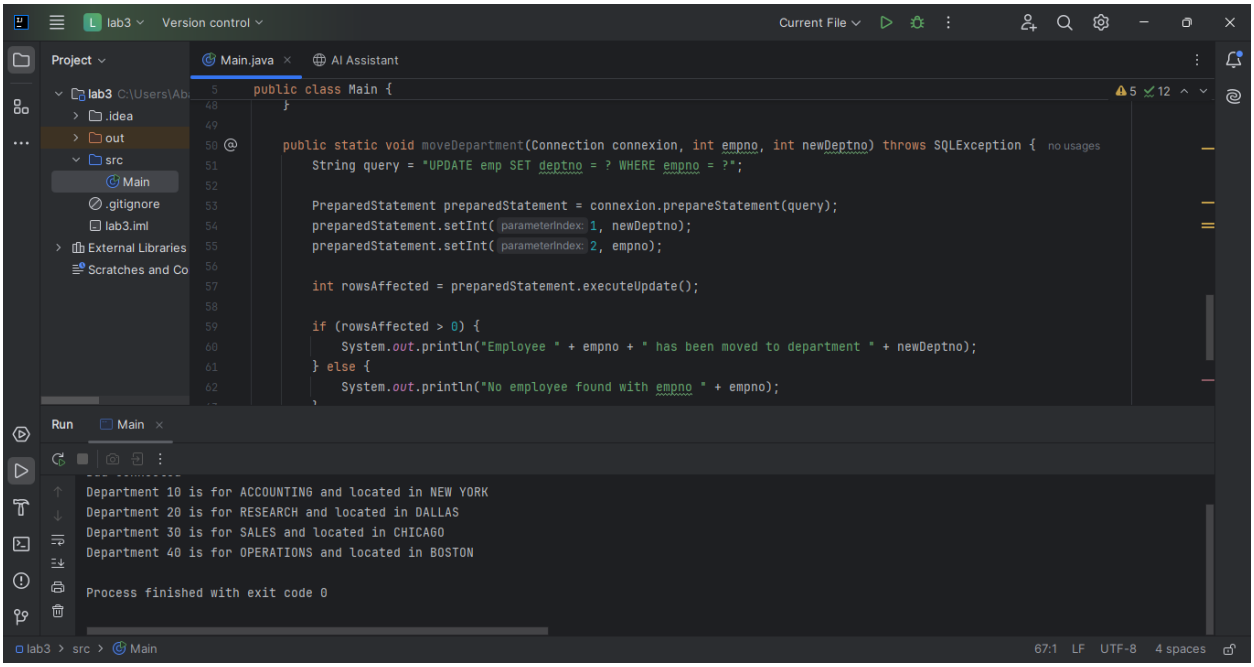


Exercise 1: Modify the query above to add the location of the department.



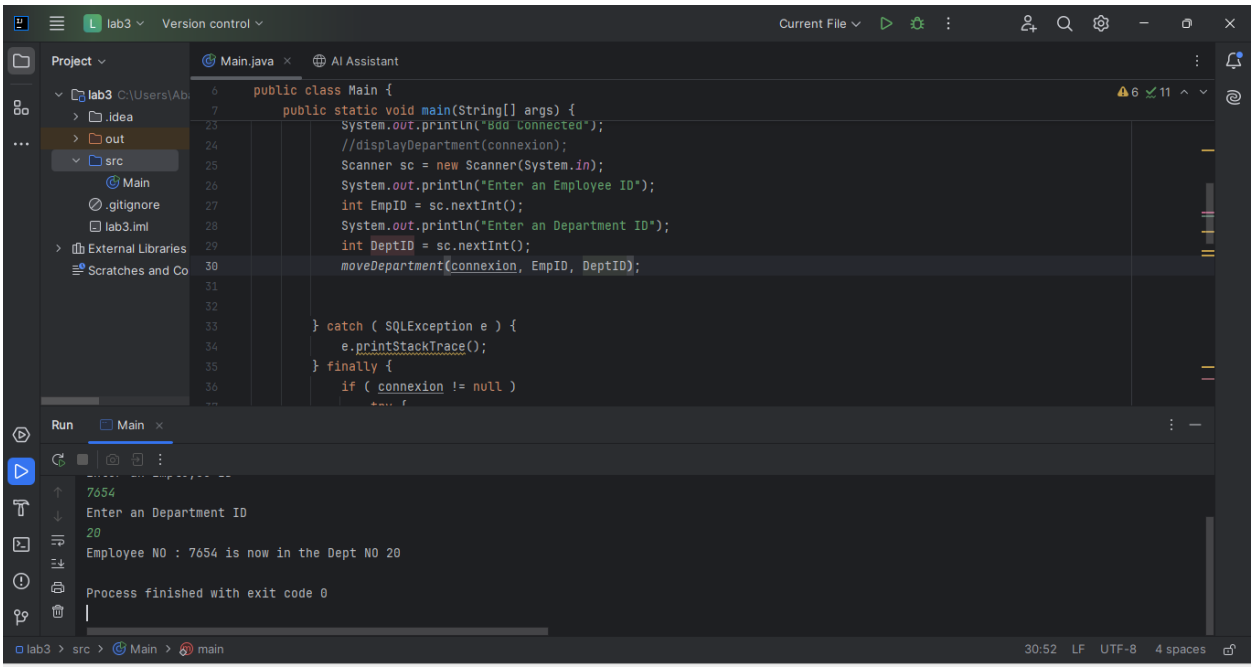
Exercice 2: Move Department

Create a method moveDepartment(int empno, int newDeptno) that will move an employee from a department to another.



Note: If we want to display all employees with all information about employees, we'll need to extract 10 columns' value inside the `while ( resultat.next() )` *. We would use a method that displays content of a table generically without having to know how many rows and columns there are inside. That leads to exercise 3.*

(1) Tips Exercice 2



Exercice 3: Generic display of Tables:

Create a method

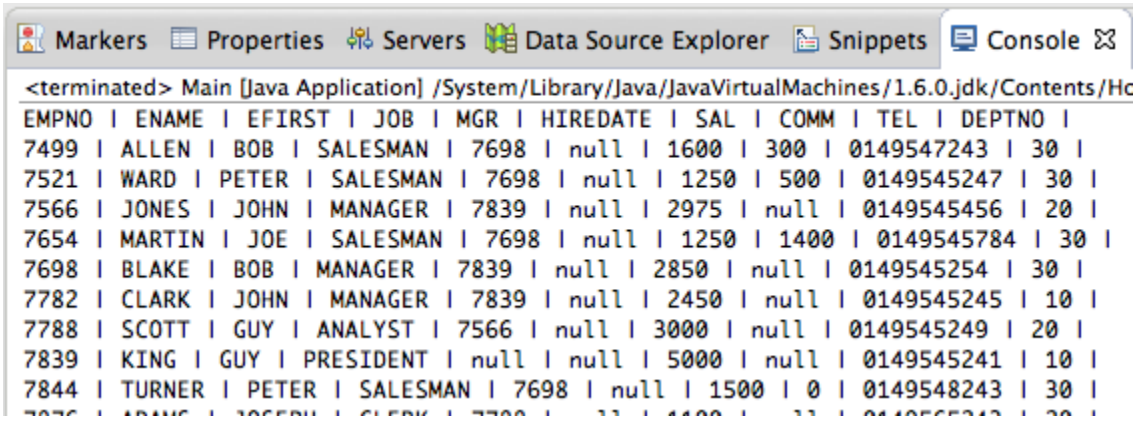
public static void displayTable(String tableName)

It takes as a parameter the name of a table and it displays all the content of a table, with as a first line the names of the table columns.

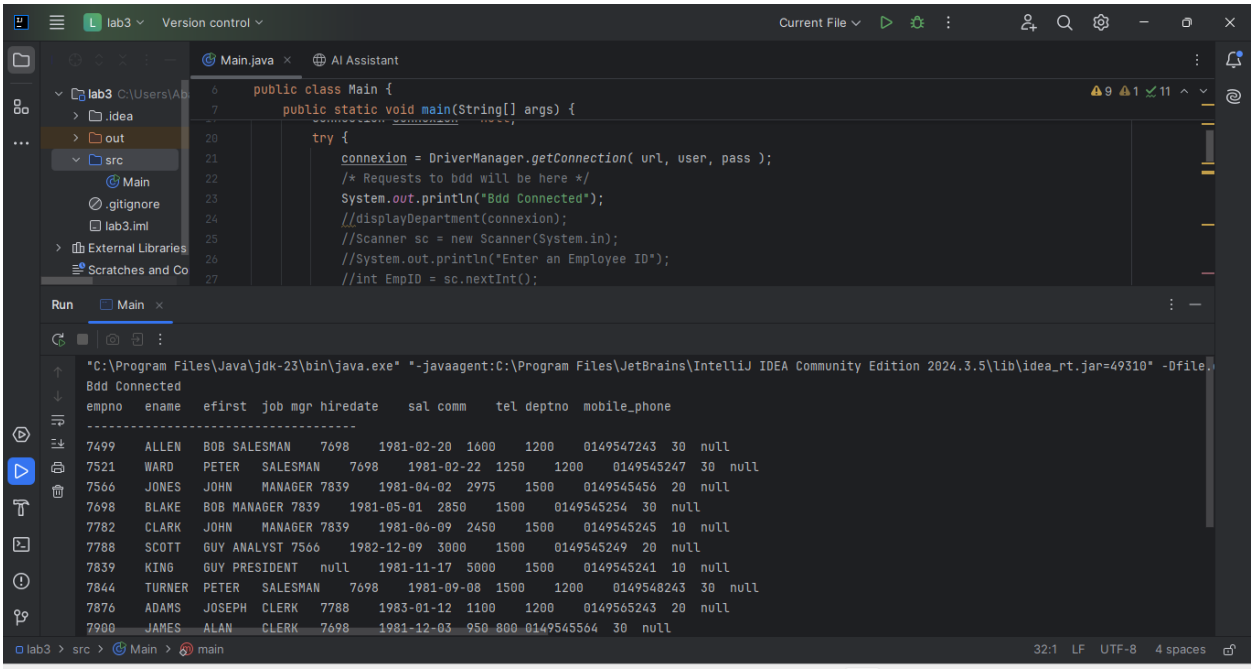
Example:
calling:

```
displayTable("emp");
```

will display:



(2) Tips Exercice 3



Exercice 4: Security

What is the Security flow to the method above ? What are others Cons from this basic native method ?

Security Flow to the Method Above

The security flow refers to how data and interactions with the database are managed to ensure that there are no vulnerabilities or opportunities for malicious exploitation, particularly **SQL injection**.

In the context of the Main.java code, here's the security flow:

1. **Loading JDBC Driver:**

   The driver is loaded using Class.forName("org.postgresql.Driver");. While this is a standard and safe way to load the JDBC driver, it's important to make sure that this is done with a trusted driver and that no untrusted libraries are loaded.

   Security Concern: In general, loading JDBC drivers from untrusted sources can lead to potential malicious behavior.

2. **Establishing Database Connection:**

The connection to the database is established via DriverManager.getConnection(url, user, pass);.

Security Concern: Storing database credentials (username, password) directly in the code is insecure. This is a **hardcoded sensitive information** issue and exposes the application to potential data breaches if the source code is compromised.

3. **SQL Query Execution (Display Table):**

   The code directly constructs the SQL query and executes it. The method displayTable uses Statement to execute queries, and the query is written as "SELECT * FROM " + tableName;, where tableName is a user input.

   Security Concern: The use of Statement and concatenation of user-supplied data into the query makes the code **vulnerable to SQL Injection**. For example, if a user enters a malicious value like emp; DROP TABLE emp; --, this could cause a destructive query to be executed.

4. **Using Prepared Statements:**

   For methods like moveDepartment(), **PreparedStatement** is used, which helps prevent SQL injection by separating the query from the user input.

   This is a **good practice** and protects against injection vulnerabilities in places where user input is used directly in queries.

Other Cons (Downsides) of This Basic Native Method

While this approach works, it has several **security weaknesses** and **performance issues** that need to be addressed:

1. **SQL Injection Vulnerability:**

   As mentioned, the code is **vulnerable to SQL Injection** when concatenating user input directly into the SQL query. In displayTable(), the tableName input is concatenated into the query. An attacker could potentially inject arbitrary SQL, causing data leakage, deletion, or modification.

2. **Sensitive Data in Code:**

   Hardcoding sensitive information (like database credentials) within the code (String user = "postgres"; String pass = "password";) is **insecure**. If the source code is exposed, the credentials are vulnerable to misuse.

3. **No Access Control (Role-based Security):**

   The code does not implement any form of **access control** or user authorization to restrict who can run which queries.

4. **Database Connection Leak:**

   The code properly closes the connection in a finally block, but there's still a risk if this block is accidentally omitted or if the connection isn't closed properly due to an error.

5. **Error Handling (Stack Trace Exposure):**

   The current error handling prints stack traces (e.printStackTrace()) directly to the console. This can **leak sensitive information** such as database structure, SQL errors, or internal application logic to the client.

6. **No Input Validation:**

   The user input (tableName, empno, and deptno) is not validated or sanitized. This leaves the code open to unexpected input that could lead to errors or security vulnerabilities.

In any case:
In server-Side: Never Trust Entry from the Client !
Even if you apply a pre-treatment e.g.: like in Javascript

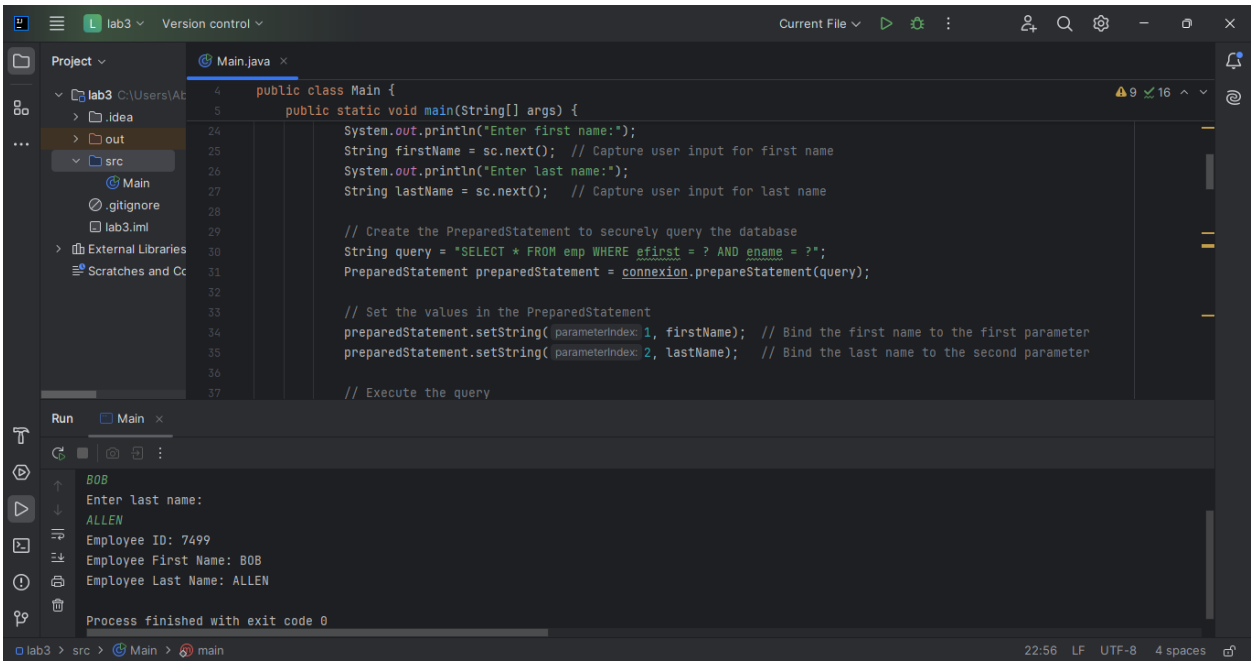
4. Prepared Request

Following your remarks in Exercise 4, we'll discover a new Object that helps to filter arguments passed to a query. It prepares the query, it's called PreparedStatement:

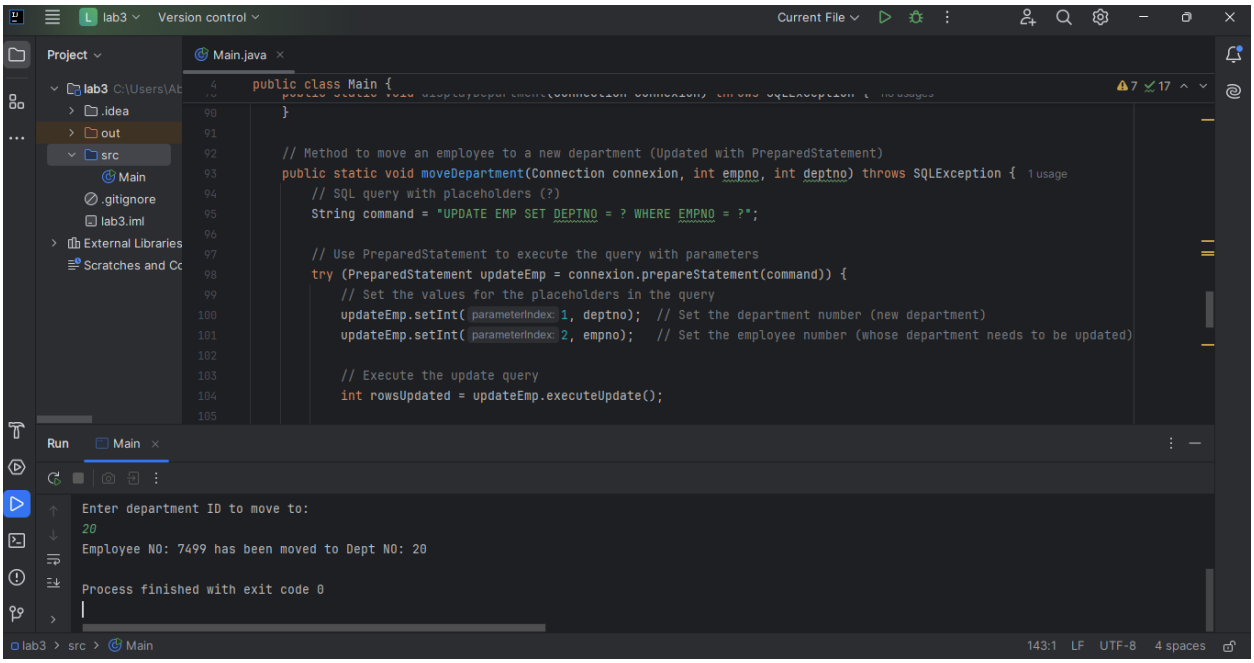
- This can pre-compilate SQL performing optimizations.
- Prepared statements are resilient against SQL injection and potential others wrong arguments passed in the query,

How to write it :

```
PreparedStatement preparedStatement = connexion.prepareStatement(
       "SELECT * FROM emp WHERE efirst = ? AND ename = ?" );

Scanner sc = new Scanner(System.in);
String firstName = sc.next();
preparedStatement.setString( 1, firstName );
String lastName = sc.next();
preparedStatement.setString( 2, lastName );

ResultSet results = preparedStatement.executeQuery();
```



Exercice 5: Modify moveDepartement() to use PrepareStatement



The codes can be found here: https://github.com/DatagirlX/JDBCPart1

Exercice 6: Try with displayTable

Try to modify your method displayTable by using PrepareStatement with tableName as a parameter of PrepareStatement.
Why doesn’t it work ?

The issue lies in the fact that **PreparedStatement** does not allow for the parameterization of table names or column names. In SQL queries, placeholders (?) can only be used for values (e.g., strings, numbers) but **not** for structural elements like table names, column names, or SQL keywords.

Here's a breakdown of the problem:

Why doesn't it work?

- **PreparedStatement and table names**:

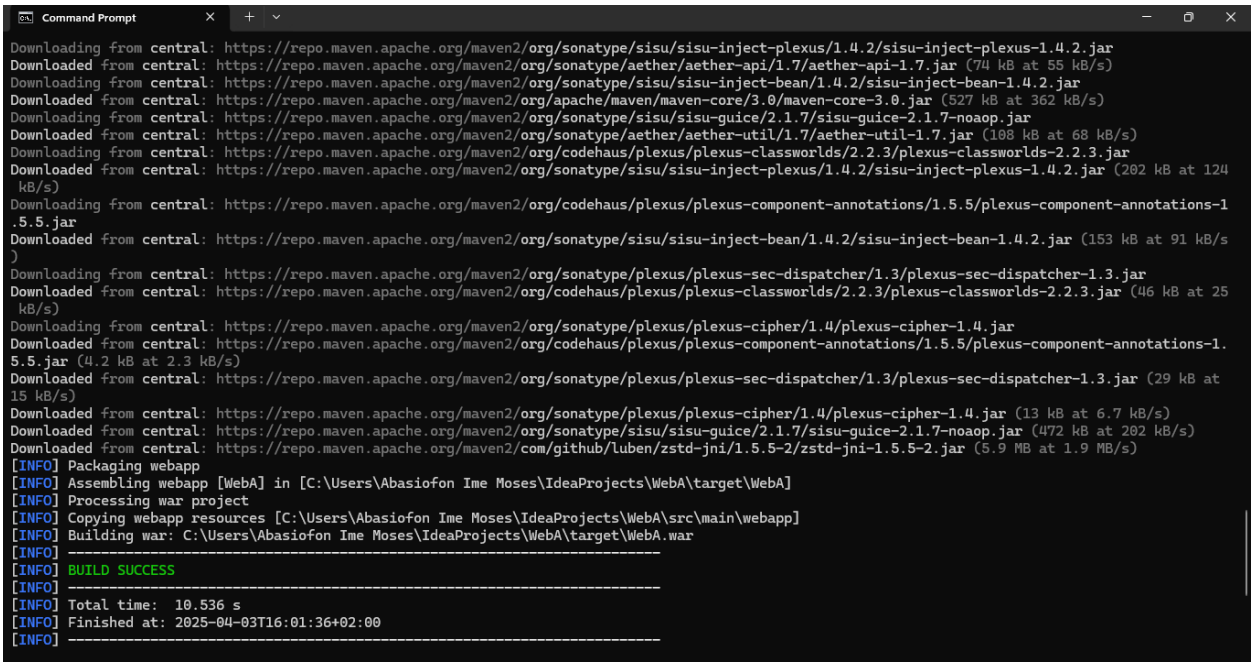
  PreparedStatement can only handle **values**, not **database schema** elements like table names or column names.

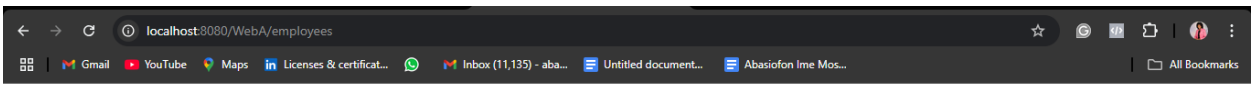
- “?” cannot be used for the table name because PreparedStatement is designed to safely insert values, but not structural components of the query.

- **Dynamic Table Name**:
  In the code, the table name is dynamic, meaning that it is provided by the user, so it's part of the query structure. This is why PreparedStatement is not suitable in this case. SQL statements can be parameterized, but the table name itself cannot.

Exercice 7 (Bonus to do at home because Long): With J2EE

Nowadays, Database and Java are more used together in J2EE, for Web Application Server. You'll use them together very often in that case. Your server in J2EE receives http requests from your client and has to communicate with your BDD to persist data.
Transform your Java application to build a J2EE server and display answers in a web browser like Firefox.





Employee List

Employee ID: 7521, Name: WARD

Employee ID: 7566, Name: JONES

Employee ID: 7698, Name: BLAKE

Employee ID: 7782, Name: CLARK

Employee ID: 7788, Name: SCOTT

Employee ID: 7839, Name: KING

Employee ID: 7844, Name: TURNER

Employee ID: 7876, Name: ADAMS

Employee ID: 7900, Name: JAMES

Employee ID: 7902, Name: FORD

Employee ID: 7934, Name: MILLER

Employee ID: 7584, Name: OSTER

Employee ID: 7369, Name: SMITH

Employee ID: 7654, Name: MARTIN

Employee ID: 7499, Name: ALLEN

The code can be found here: https://github.com/DatagirlX/J2EE

Part II : DAO.

Exercise 8 : What are cons of JDBC as used above ?

Cons of JDBC as Used Above (without DAO Pattern)

1. **Tight Coupling Between Application and Database**: the database connection and SQL queries are directly embedded within the servlet (or controller), making the code tightly coupled to the database. This can lead to difficulties in maintenance and makes the application harder to test, especially when changes to the database schema or the database itself occur.

2. **Repetitive Code**: Without using a Data Access Object (DAO) pattern, the database connection, query execution, and result processing code may be duplicated across multiple classes or methods. This makes the code harder to maintain, as changes to the database interaction logic need to be replicated across the entire codebase.

3. **Error Handling**: error handling is done directly in the servlet. While exceptions are caught, there isn't a clear separation of concerns between error handling logic and the business logic. This can lead to cluttered, difficult-to-maintain code.

4. **Lack of Abstraction**: Using JDBC directly in the servlet exposes the details of database interaction to the rest of the application. It does not provide any abstraction, meaning other parts of the application need to know about the database structure and how to interact with it. This violates the **Single Responsibility Principle** and makes the code harder to modify or extend.

5. **Scalability Issues**: As the application grows, directly using JDBC for database interactions can lead to scalability problems. The servlet might be overwhelmed with database interaction code, leading to reduced performance and more complex database operations being handled directly in the controller.

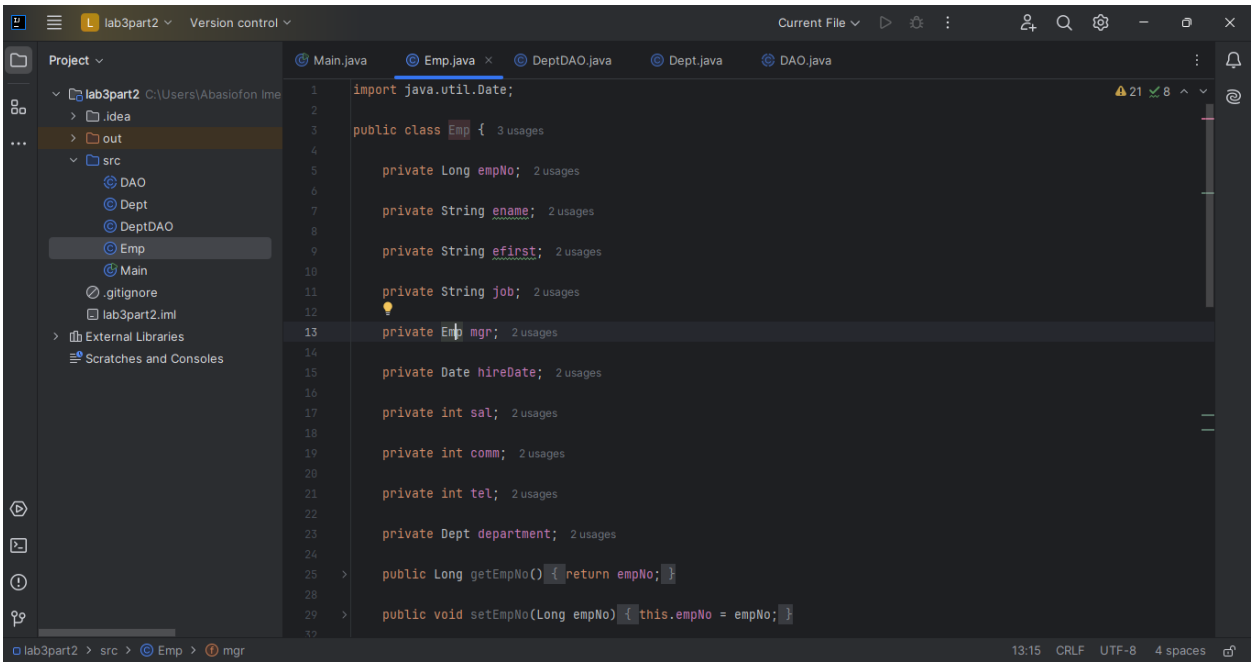
6. **Transaction Management**: In JDBC, manual transaction management is required (begin, commit, rollback). When using JDBC directly, managing transactions across multiple

queries can be error-prone. The DAO pattern centralizes this logic, reducing the risk of errors and improving transaction consistency.

7. **No Separation of Concerns**: The business logic, database logic, and presentation logic are tightly coupled in the servlet. By using a DAO pattern, the database logic is separated from business logic, making the code easier to maintain, understand, and extend. This separation of concerns also makes unit testing much easier.

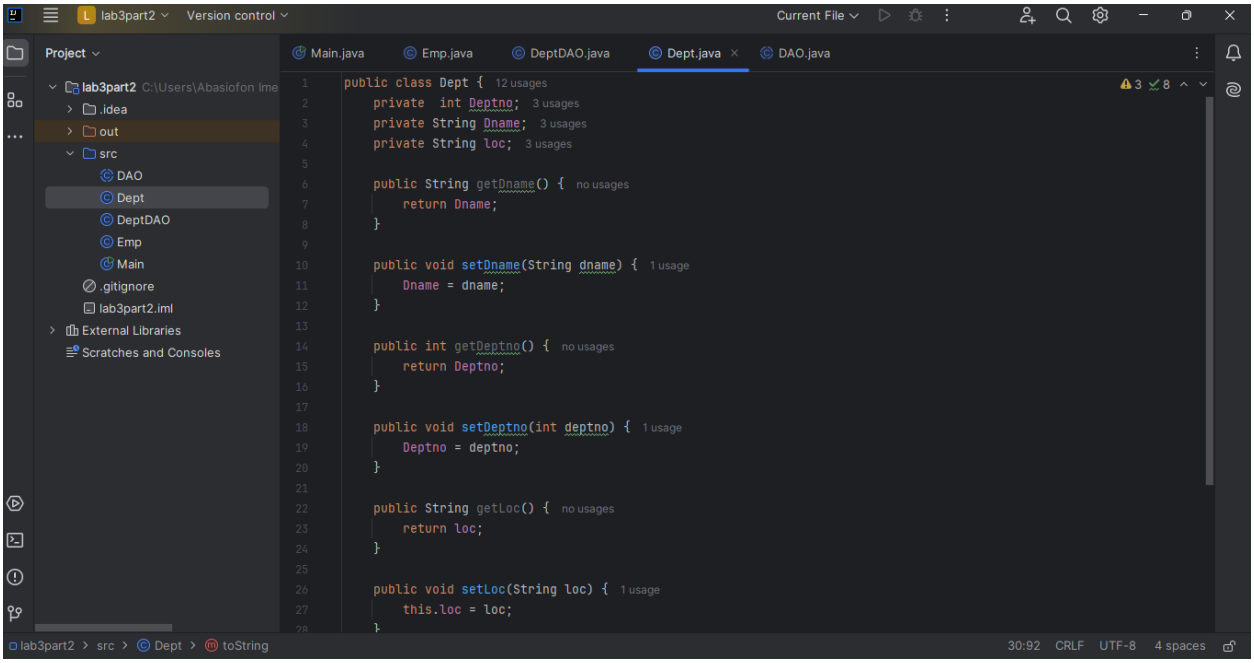
1. Bean - POJO Creation

Each Table of your Data Model needs to have its equivalent as a Java Class.
This class is called a [POJO (Plain Old Java Object)] or a Bean.
The name of Each column of your table must map an attribute of your class with the same type.



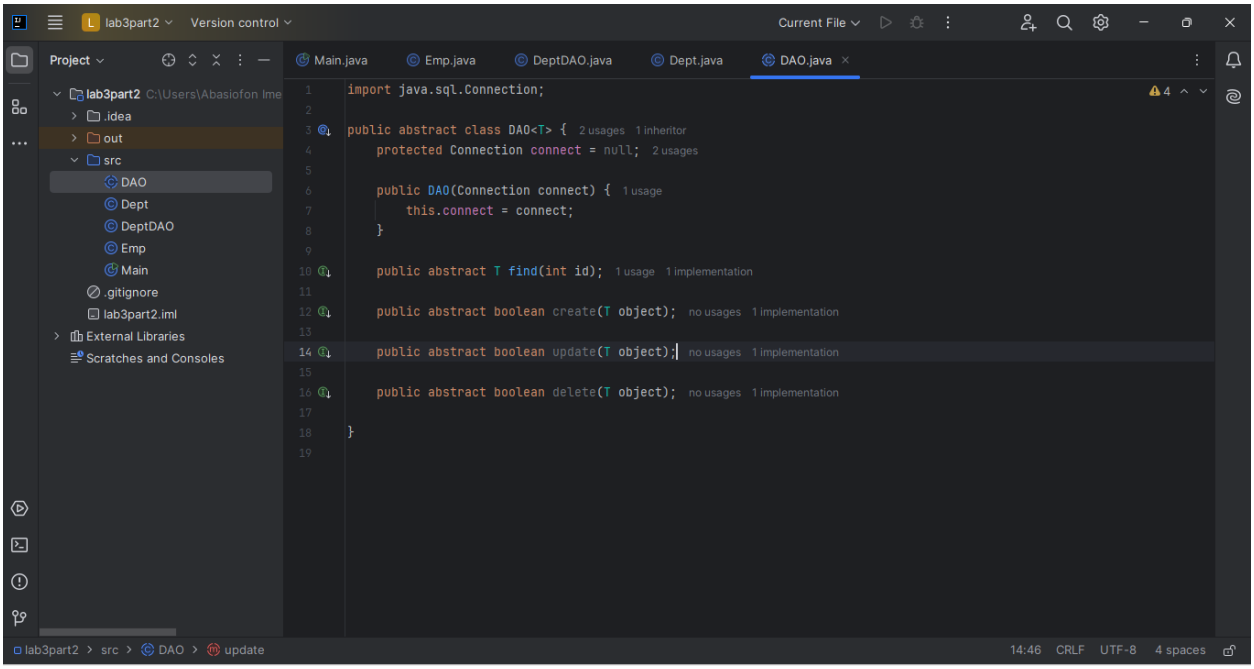
Exercice 9 : Write the Bean for Department

Following what we have written for the Employee, write the POJO class for the Department.



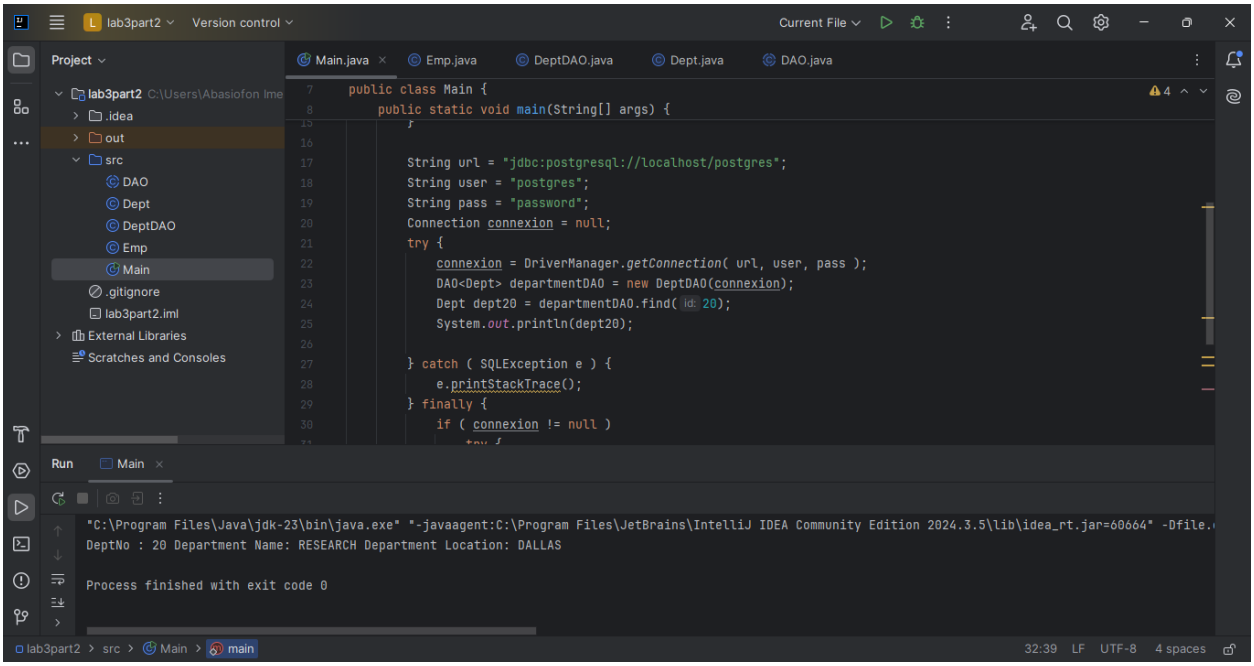
2. DAO Implementation

In our application, our objects will perform some [CRUD (Create, Read, Update, Delete)] operations.



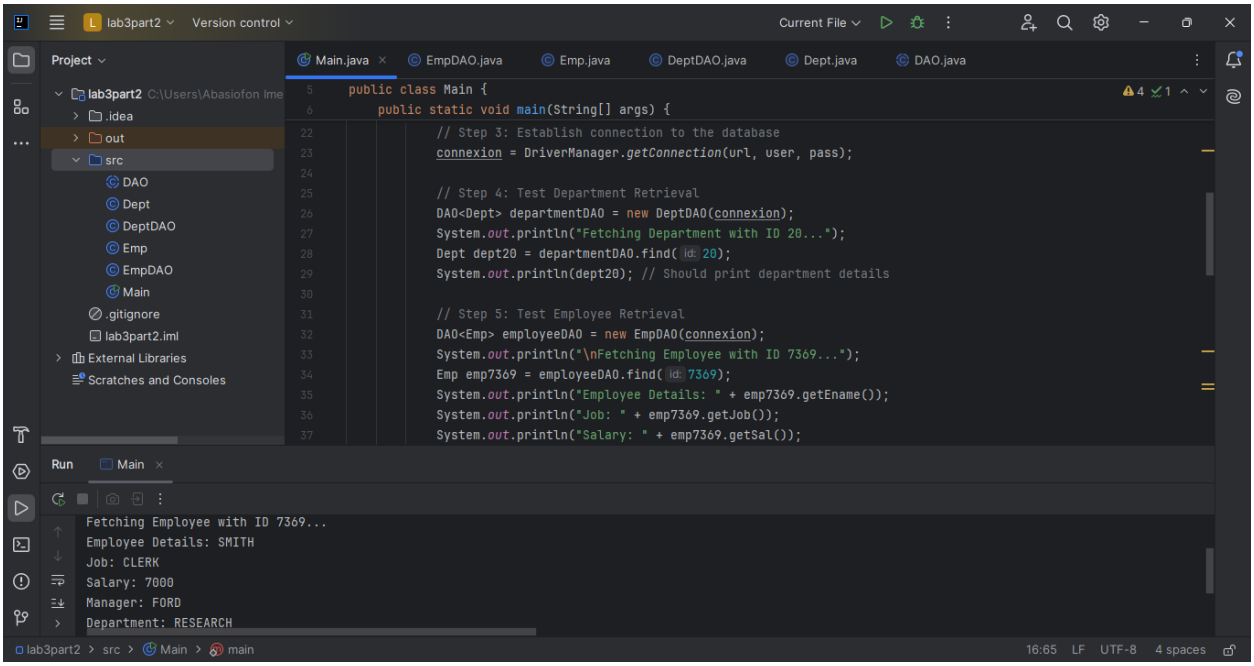
Exercice 10 : Write Implementation for method find for departement DAO.

In DeptDAO.java, implement find(int id) method by calling database, using the Connection object.
Then test it:



Exercice 11 : Write Implementation for method find for employee DAO.

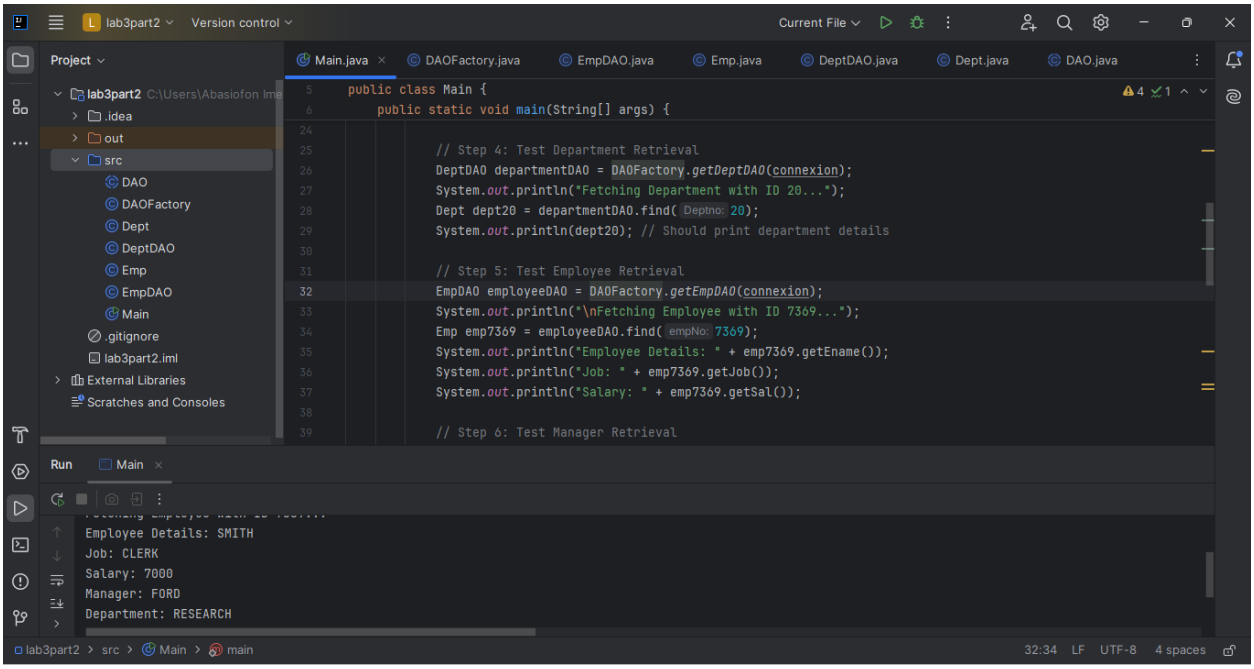
This one will be tricker as an Employee has a manager represented by an integer in the database but in the class Employee, the attribute manager, should be a type Employee.
So for each employee, you need to call find(idManager) to get its manager. So you will need to build recursively the Employee



3. Factory Pattern

In the DAO pattern we can add the Factory Pattern which is a class that takes care of object instantiation.
The goal is that all object instantiation is done in the same place.

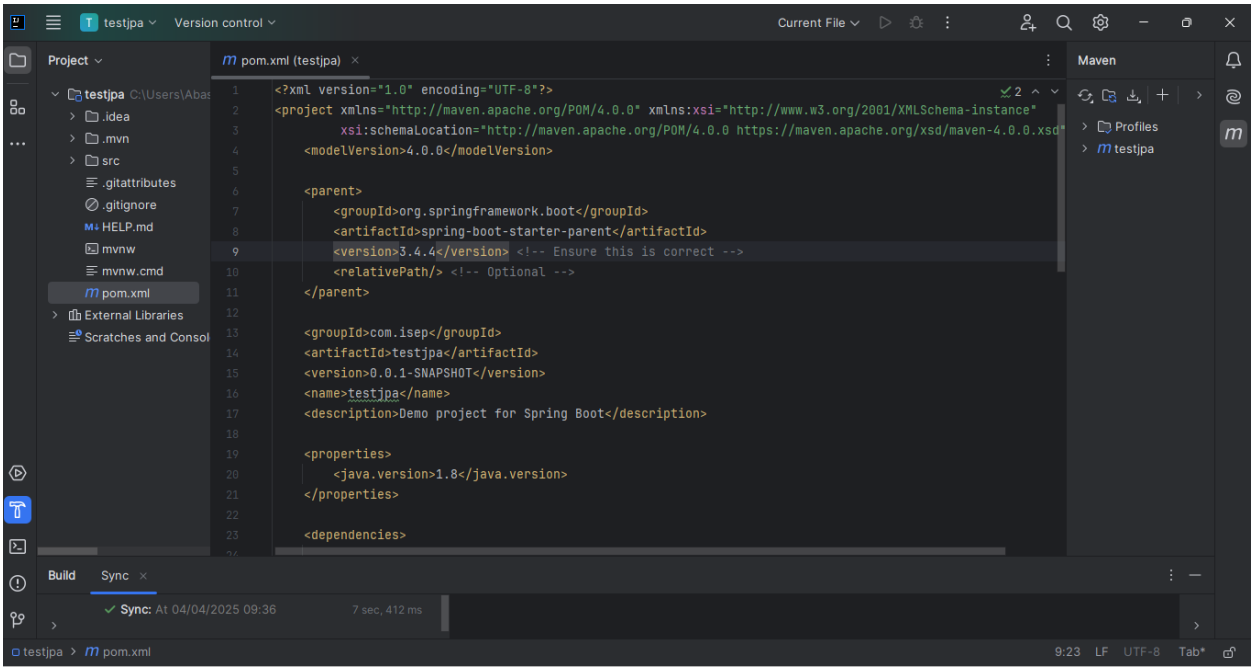
Exercice 12 : Create DAOFactory and use it in previous example

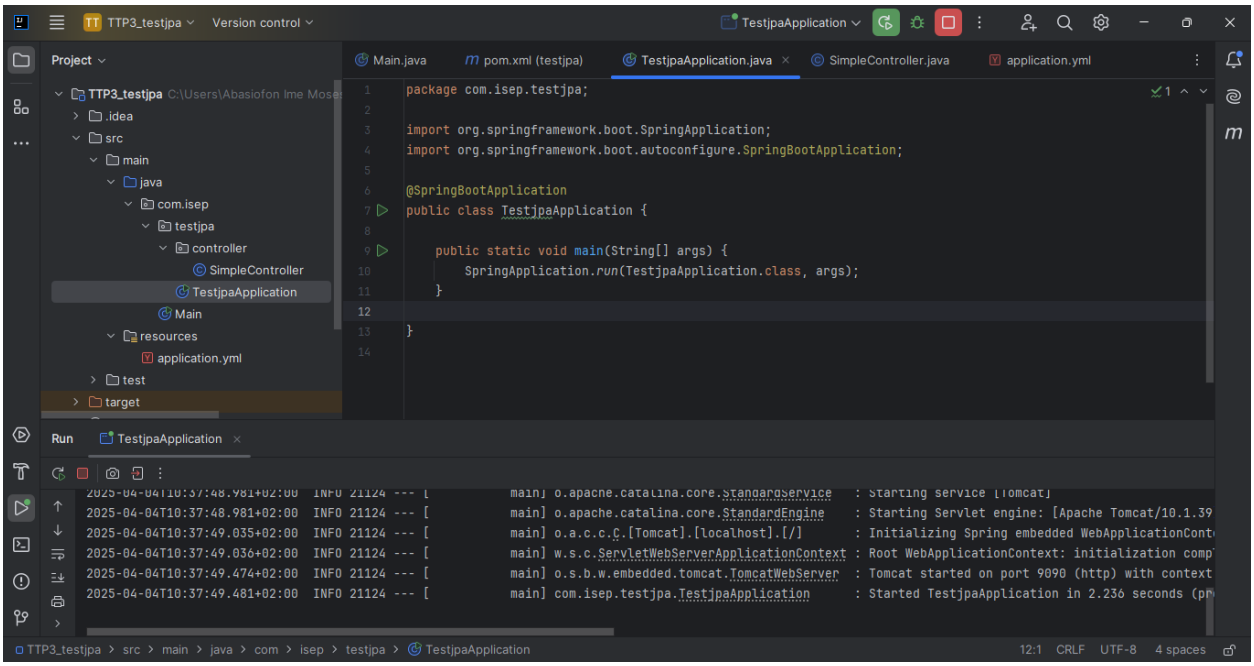


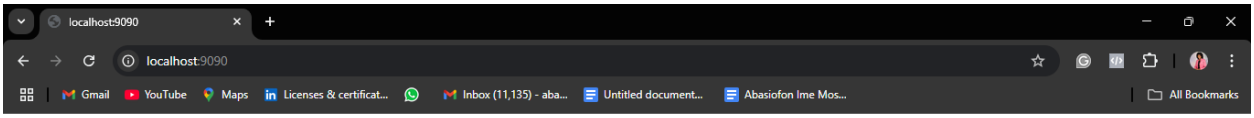
The codes can be found here: https://github.com/DatagirlX/JDBCPart2

Part III : Spring Boot & JPA.

Exercice III.1 : Init your Spring project on IntelliJ

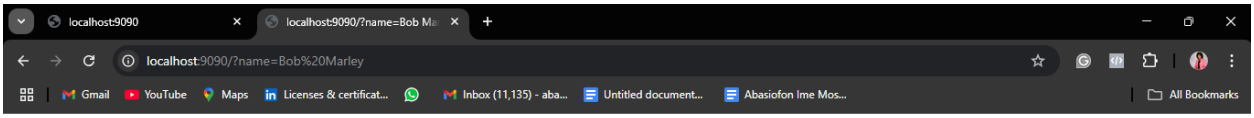

testjpa
Version control
Current File
Project
pom.xml (testjpa)
Maven
Profiles
testjpa
testjpa C:\Users\Abas
.idea
.mvn
src
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consol
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.4</version> <!-- Ensure this is correct -->
<relativePath/> <!-- Optional -->
</parent>
<groupId>com.isep</groupId>
<artifactId>testjpa</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>testjpa</name>
<description>Demo project for Spring Boot</description>
<properties>
<java.version>1.8</java.version>
</properties>
<dependencies>
Build
Sync
Sync: At 04/04/2025 09:36
7 sec, 412 ms
testjpa > pom.xml
9:23 LF UTF-8 Tab*


TTP3_testjpa
Version control
TestjpaApplication
Project
Main.java
pom.xml (testjpa)
TestjpaApplication.java
SimpleController.java
application.yml
TTP3_testjpa C:\Users\Abasiofon Ime Mose
.idea
src
main
java
com.isep
testjpa
controller
SimpleController
TestjpaApplication
Main
resources
application.yml
test
target
package com.isep.testjpa;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class TestjpaApplication {
public static void main(String[] args) {
SpringApplication.run(TestjpaApplication.class, args);
}
}
Run
TestjpaApplication
2025-04-04T10:37:48.981+02:00 INFO 21124 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-04-04T10:37:48.981+02:00 INFO 21124 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.39
2025-04-04T10:37:49.035+02:00 INFO 21124 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationCont
2025-04-04T10:37:49.036+02:00 INFO 21124 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization comp
2025-04-04T10:37:49.474+02:00 INFO 21124 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9090 (http) with context
2025-04-04T10:37:49.481+02:00 INFO 21124 --- [main] com.isep.testjpa.TestjpaApplication : Started TestjpaApplication in 2.236 seconds (pr
TTP3_testjpa > src > main > java > com > isep > testjpa > TestjpaApplication
12:1 CRLF UTF-8 4 spaces



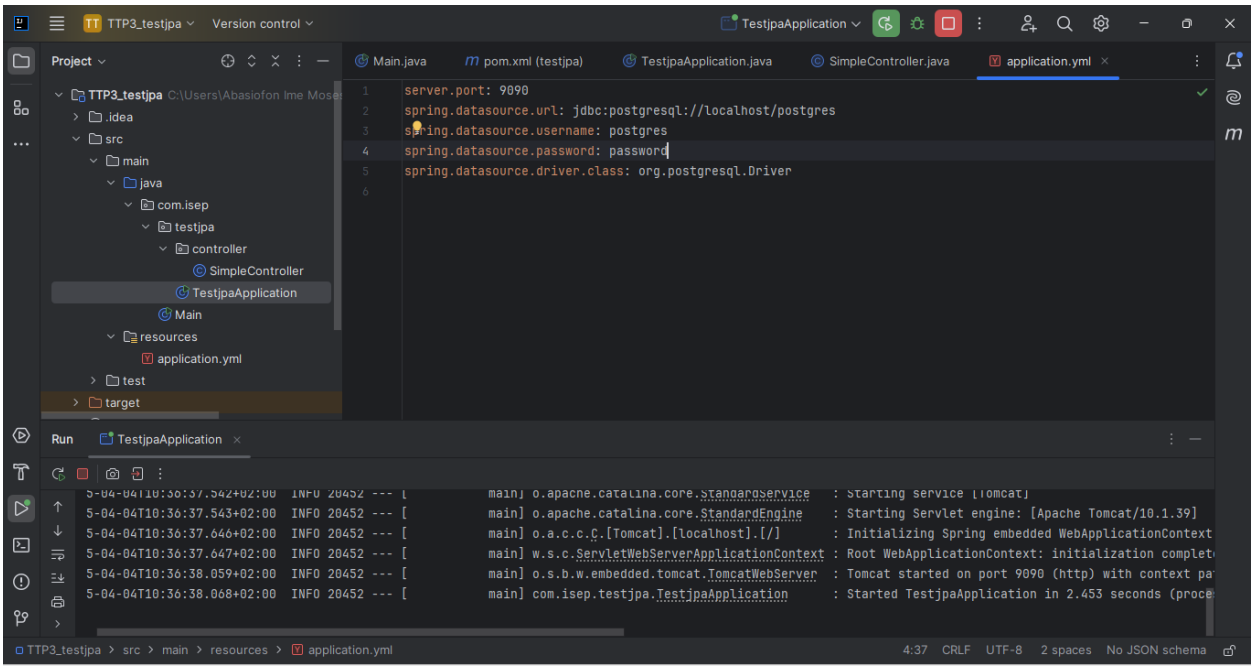
Hello null

You can also add a GET parameter like this : http://localhost:9090/?name=Bob%20Marley

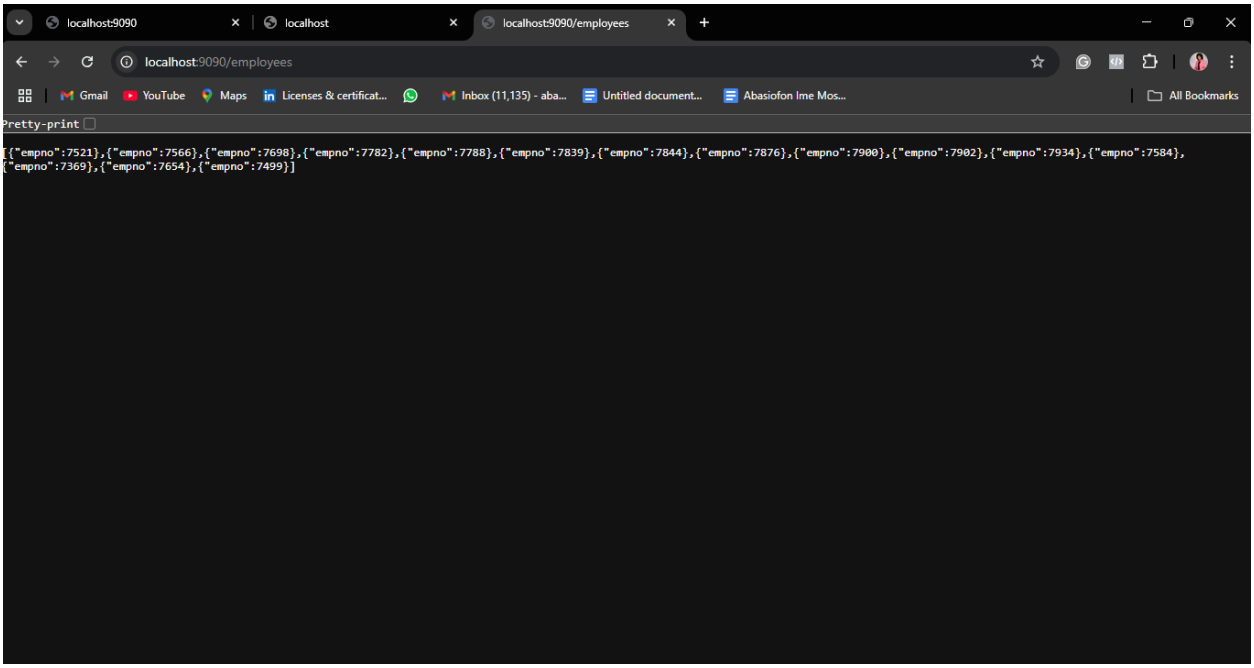


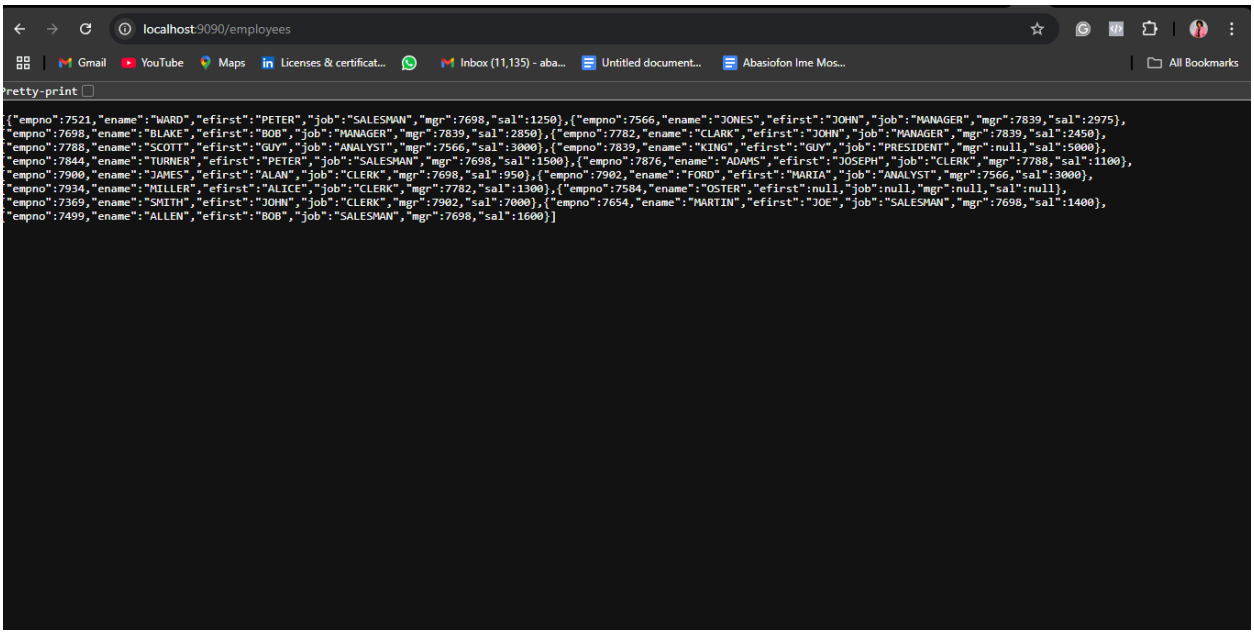
Hello Bob Marley

Exercice III.2 : Activate JPA and connect to your Database

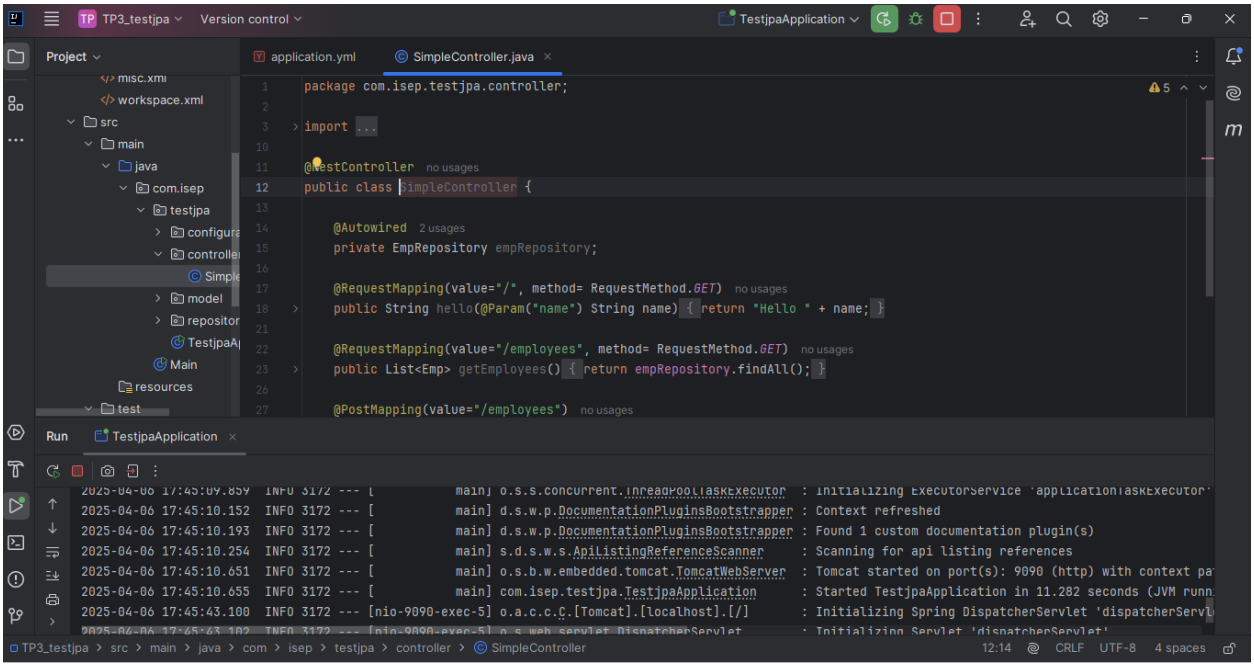


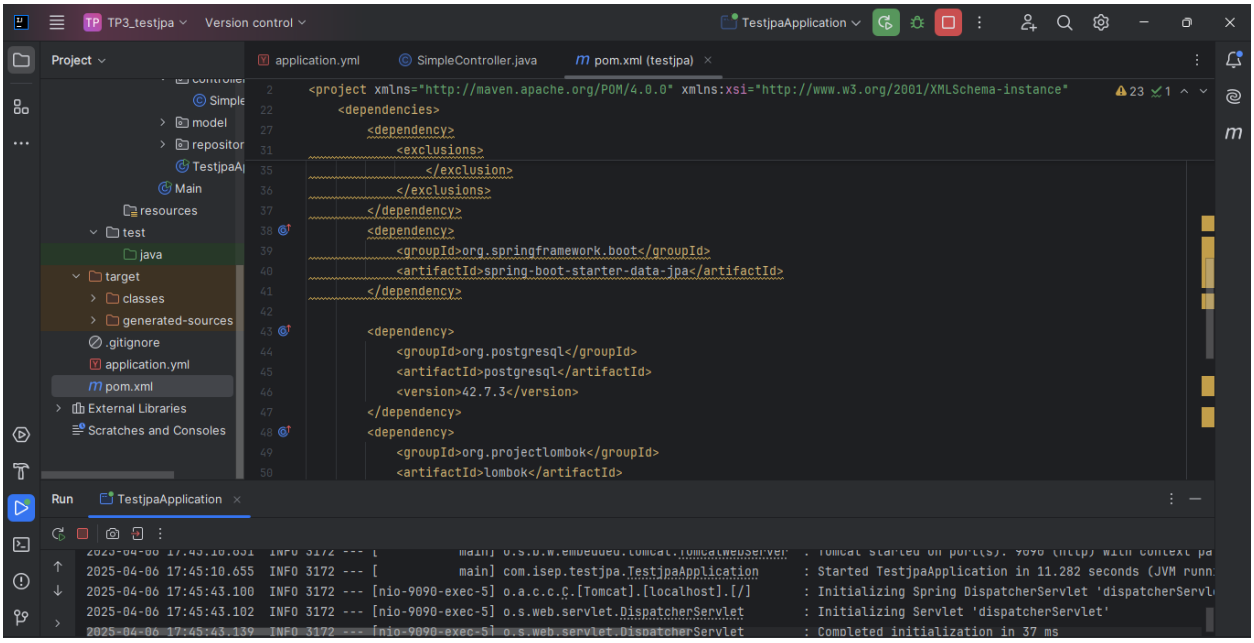
Exercise III.3 : Read Data in your Database

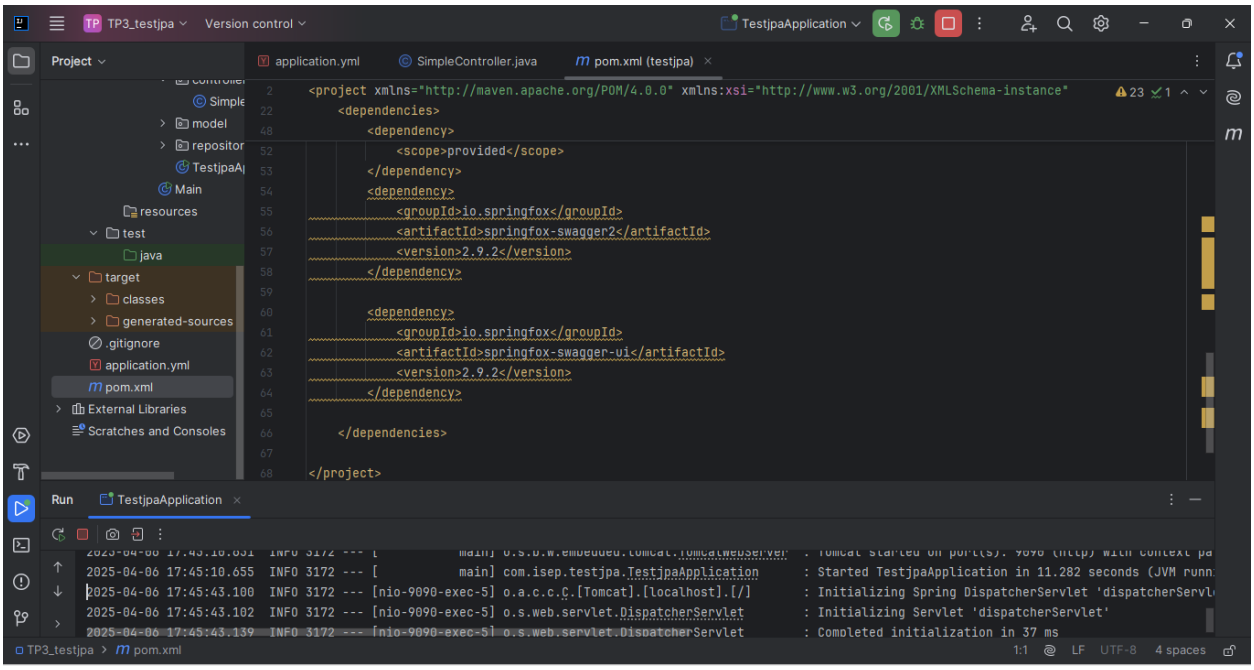


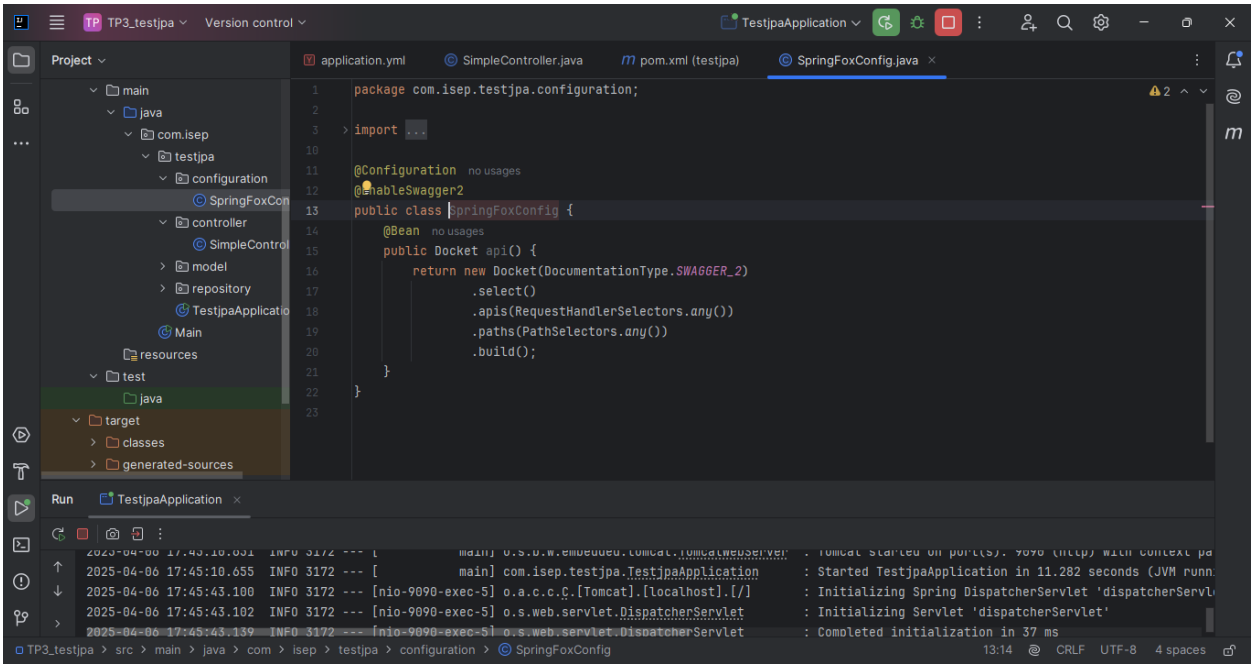


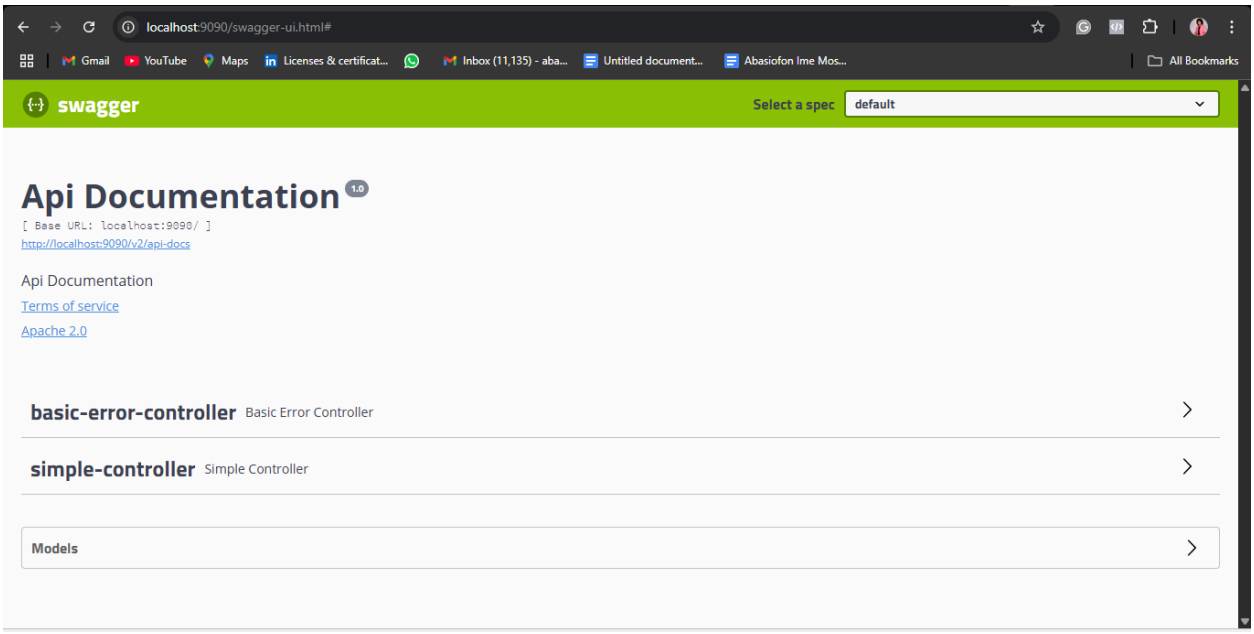
Exercise III.4 : Create an Object via JPA and expose it on a REST endpoint



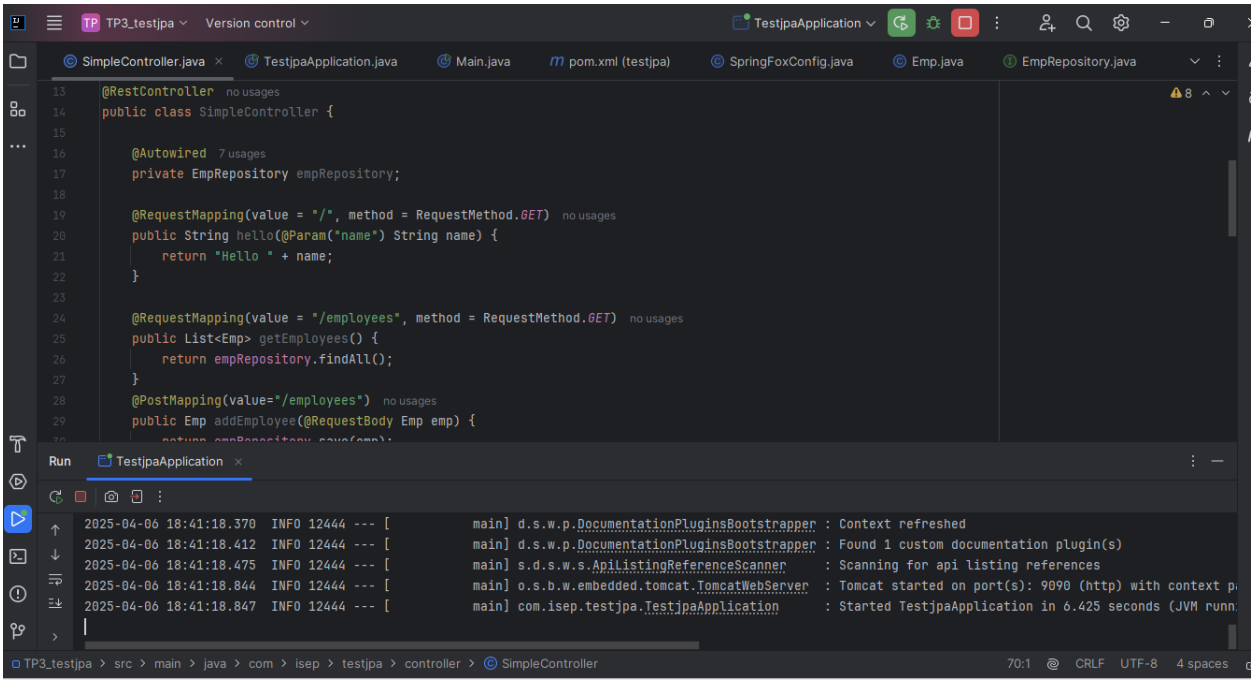

TP3_testjpa
Version control
TestjpaApplication
Project
application.yml
SimpleController.java
pom.xml (testjpa)
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<dependencies>
<dependency>
<exclusions>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.postgresql</groupId>
<artifactId>postgresql</artifactId>
<version>42.7.3</version>
</dependency>
<dependency>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
Run
TestjpaApplication
2025-04-06 17:45:10.655 INFO 3172 --- [main] com.isep.testjpa.TestjpaApplication : Started TestjpaApplication in 11.282 seconds (JVM runn
2025-04-06 17:45:43.100 INFO 3172 --- [nio-9090-exec-5] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServl
2025-04-06 17:45:43.102 INFO 3172 --- [nio-9090-exec-5] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-04-06 17:45:43.139 INFO 3172 --- [nio-9090-exec-5] o.s.web.servlet.DispatcherServlet : Completed initialization in 37 ms


TP3_testjpa
Version control
TestjpaApplication
Project
application.yml
SimpleController.java
pom.xml (testjpa)
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<dependencies>
<dependency>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>io.springfox</groupId>
<artifactId>springfox-swagger2</artifactId>
<version>2.9.2</version>
</dependency>
<dependency>
<groupId>io.springfox</groupId>
<artifactId>springfox-swagger-ui</artifactId>
<version>2.9.2</version>
</dependency>
</dependencies>
</project>
Run
TestjpaApplication
2025-04-06 17:45:10.655 INFO 3172 --- [main] com.isep.testjpa.TestjpaApplication : Started TestjpaApplication in 11.282 seconds (JVM runn
2025-04-06 17:45:43.100 INFO 3172 --- [nio-9090-exec-5] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServl
2025-04-06 17:45:43.102 INFO 3172 --- [nio-9090-exec-5] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-04-06 17:45:43.139 INFO 3172 --- [nio-9090-exec-5] o.s.web.servlet.DispatcherServlet : Completed initialization in 37 ms
TP3_testjpa > pom.xml
1:1 LF UTF-8 4 spaces





Exercise III.5 : GET by ID / Update by id / Delete by ID / Get All, via JPA / REST



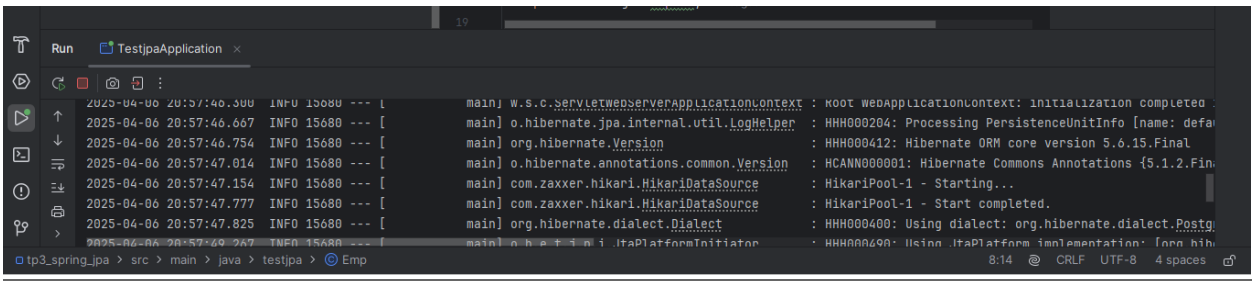
Part IV : Connections Pool (Bonus)

Update accepted : "Cela me donne envie de partir en vacances !!"

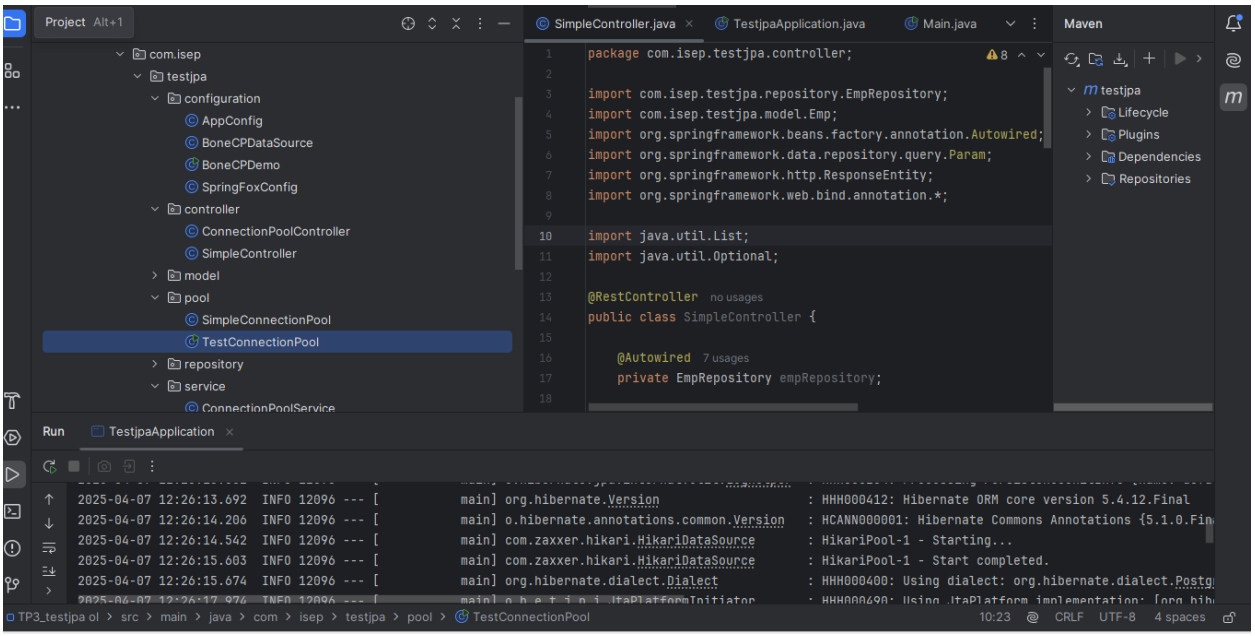
1. A Connection Originally Kept

Last but not least, we'll investigate the performance of what we've done. In a normal J2EE application, your application will serve many clients. Each call to the bdd will be done in Servlet (Object which treats an Http Request). Several clients can call at the same time via different threads. Let's analyse what will consume the most time: The Connection to BDD. Each connection to the bdd can cost several hundred a milliseconds which would be enormous for too many calls to the bdd or with too many clients at the same time.
That's why, you will want to open several Connections to the BDD and be able to re-use them from client to client.
At the start of your server, you'll open a certain number of Connections and store them in a pool (a list as a stack for example). A client c1 will take a Connection do its request and once he has finished with it (at the moment of the close() statement) , he will replace it in the pool for the use of another client.

Exercice IV.1 : (Bonus): Manually Create a List which store connection and use it in your code to make request



Exercice IV.2 (Bonus): use BoneCP to create your Connection Pool



Here is the link to the codes: https://github.com/DatagirlX/JDBCPart3

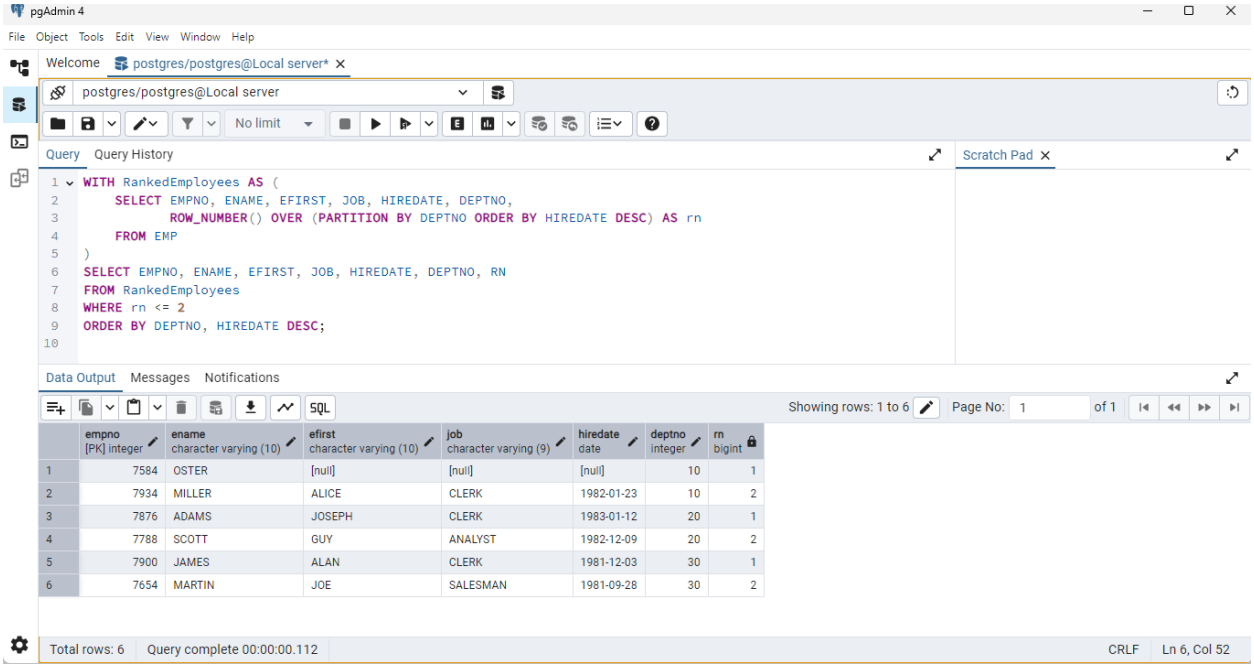
Advanced Database - TP

Advanced SQL

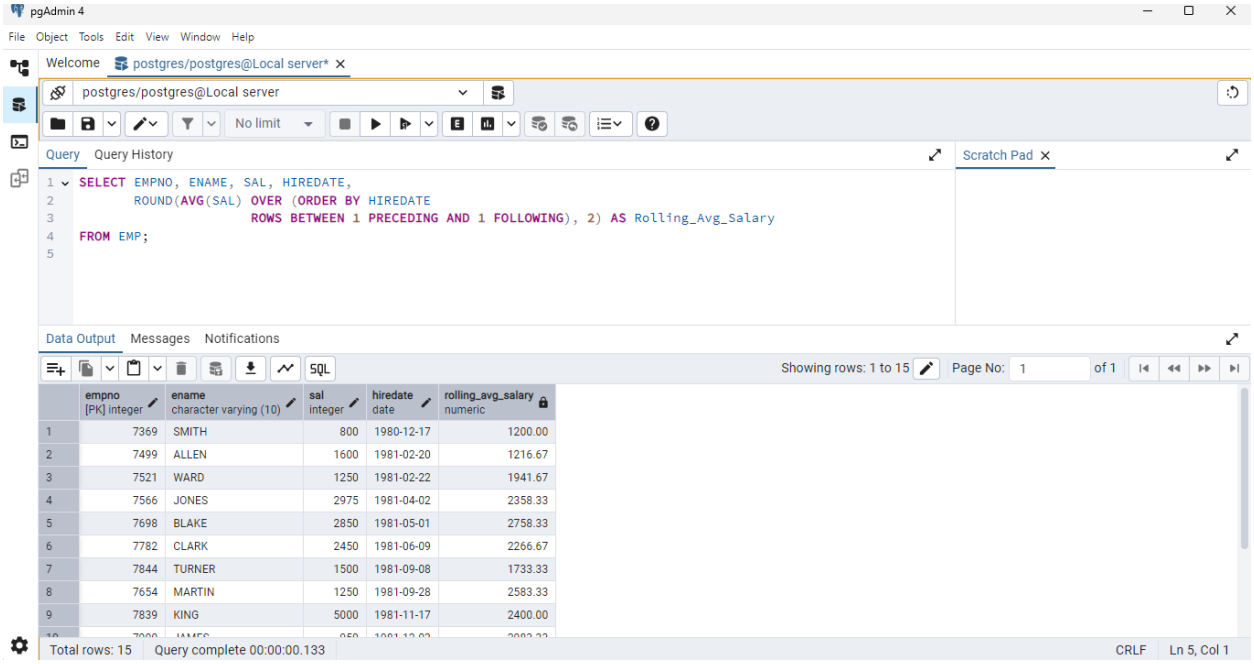
Exercice 1. Analytics Queries . Window Queries

If you don’t remember about Analytical / Windows Queries (very important!), please re-read this chapter.

1. Gets the 2 persons per department, who have arrived the latest in the company.

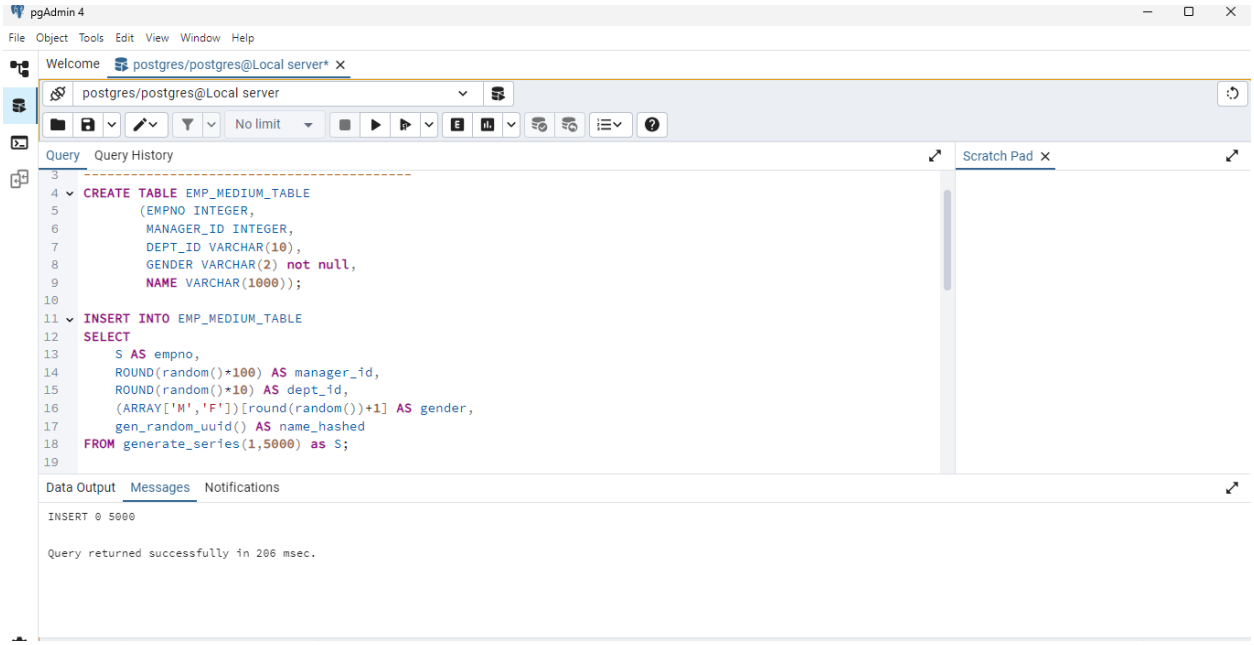


2. Show your analytical Skill and Invents an interesting query using Windows Functions (i.e.: a SELECT query on EMP table): The query should include the usage of “ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING”.



Exercice 2. Index & Explain Plan

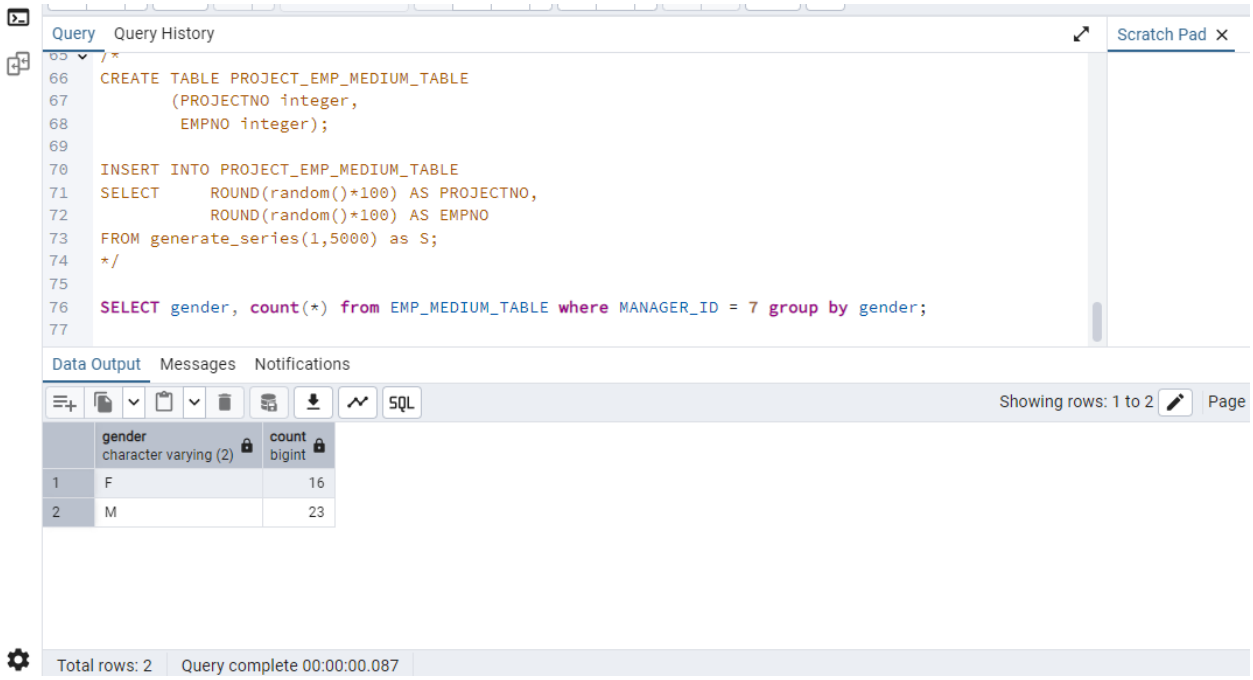
1. Execute this script (create 3 tables and load random data inside):



2. The goal of all exercice will be to tune the following query:

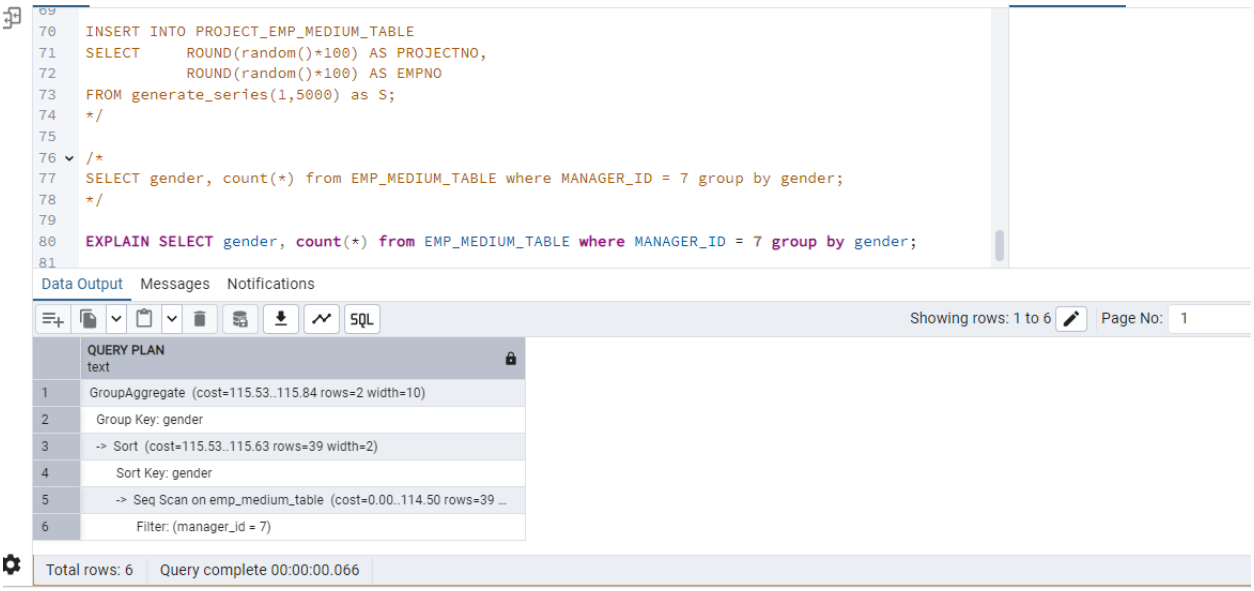
```
SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by
gender;
```

Execute it and Note the response time.

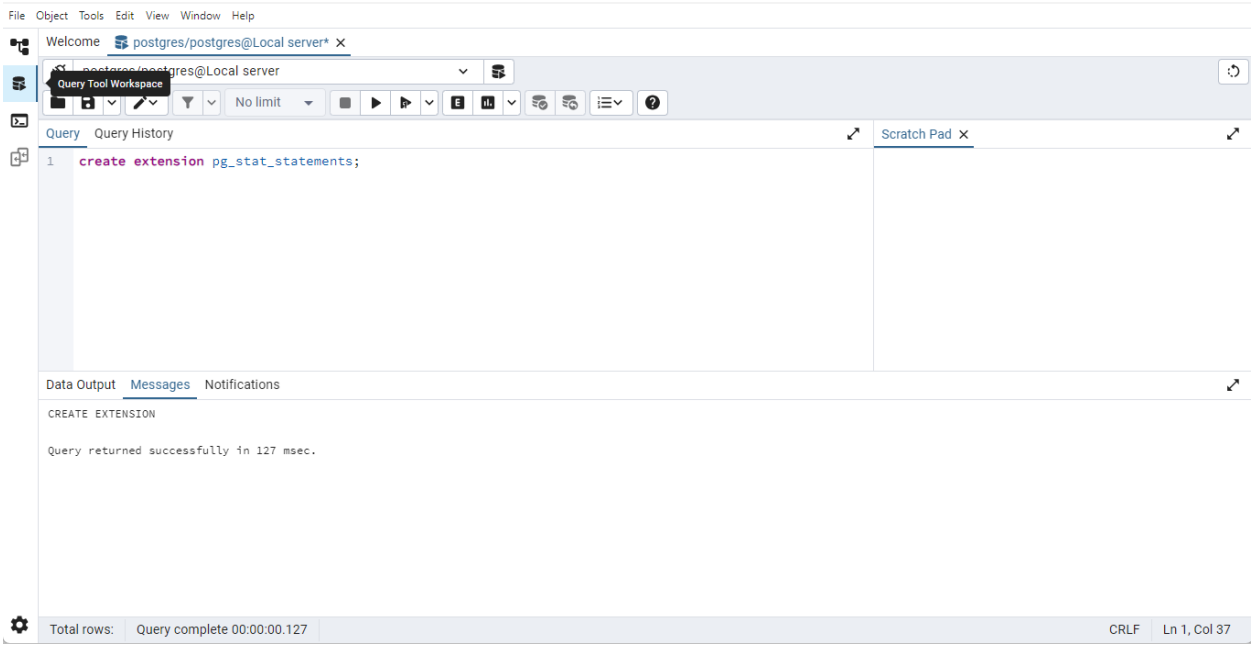


3.1. Use EXPLAIN plan to analyze the query:

Keep the Execution Plan for this query



3.2. **Activate stats**



Now you can run:

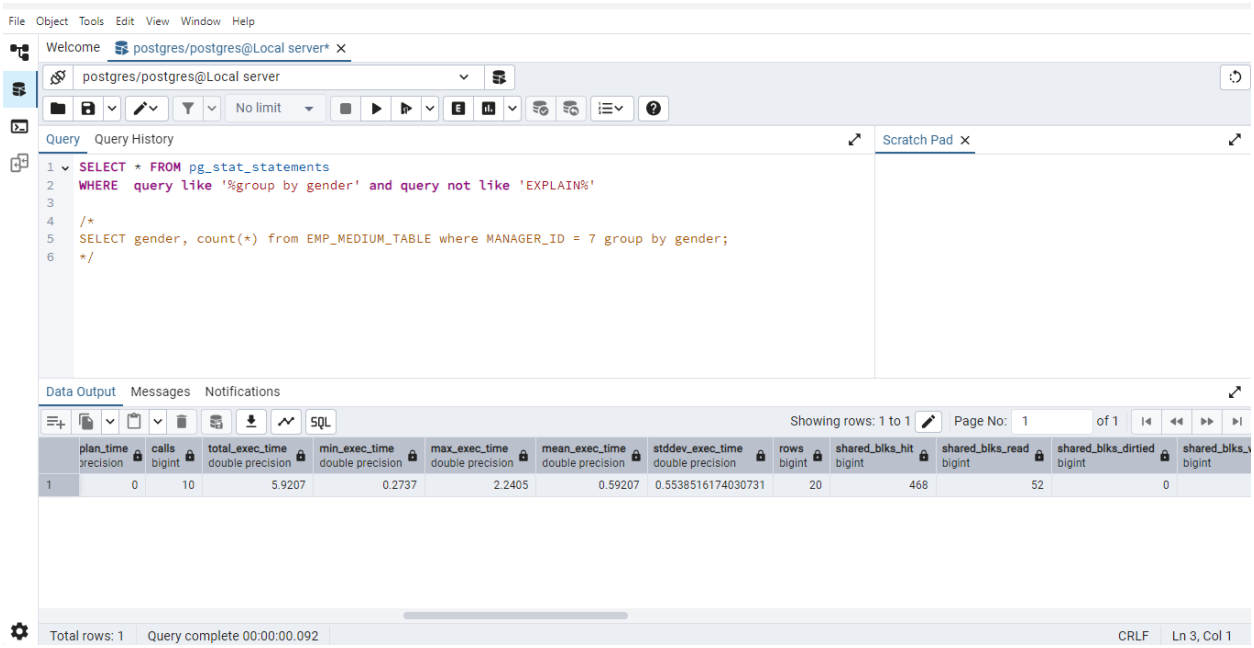
```
SELECT * FROM pg_stat_statements

    WHERE  query like '%group by gender' and query not like 'EXPLAIN%'
```

Notice the values of “calls” and “mean_exec_time” and explain them.

Re-run 10 times:

```
SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
```

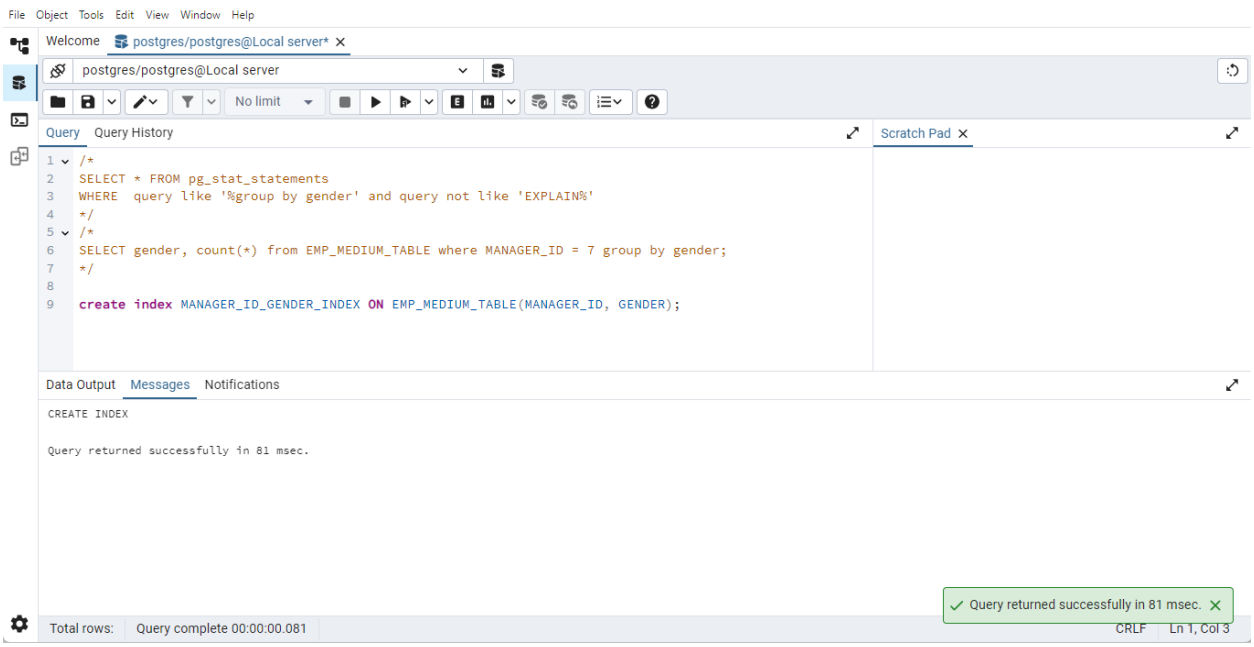
Keep the values of “calls” and “mean_exec_time”.



4. Add a **covering index** on both columns fetched:

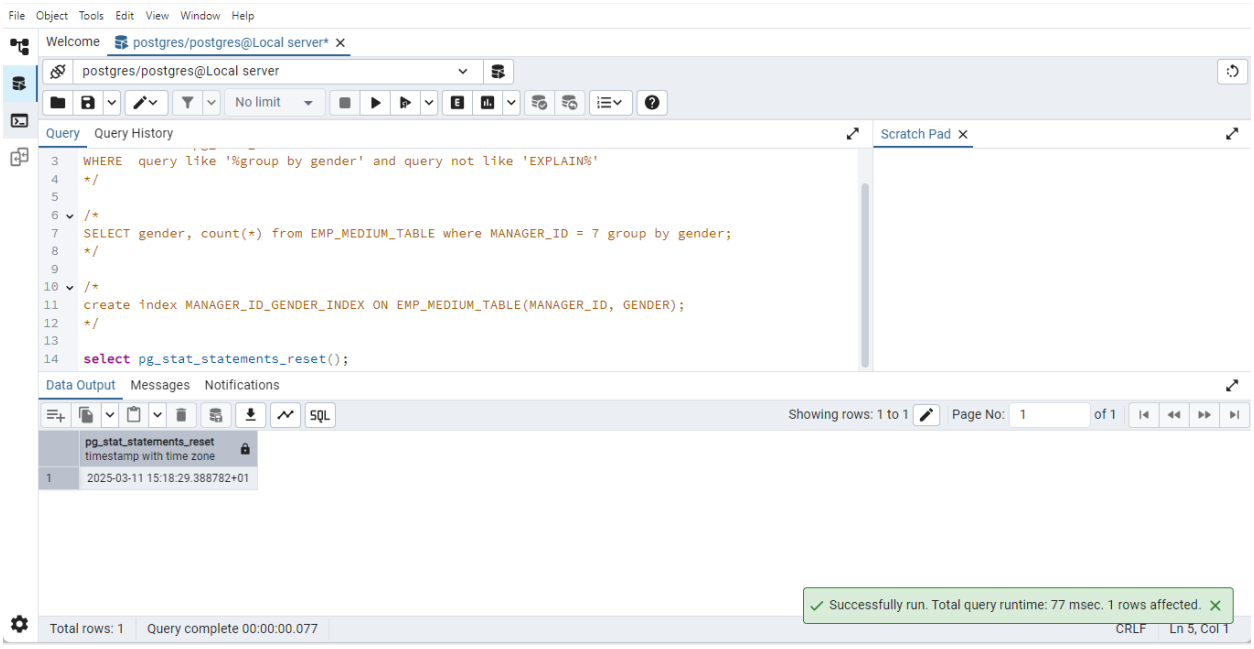
```
create index MANAGER_ID_GENDER_INDEX ON EMP_MEDIUM_TABLE(MANAGER_ID, GENDER);
```

This should accelerate further queries with clause on the 2 columns.



5. Reset the stats:

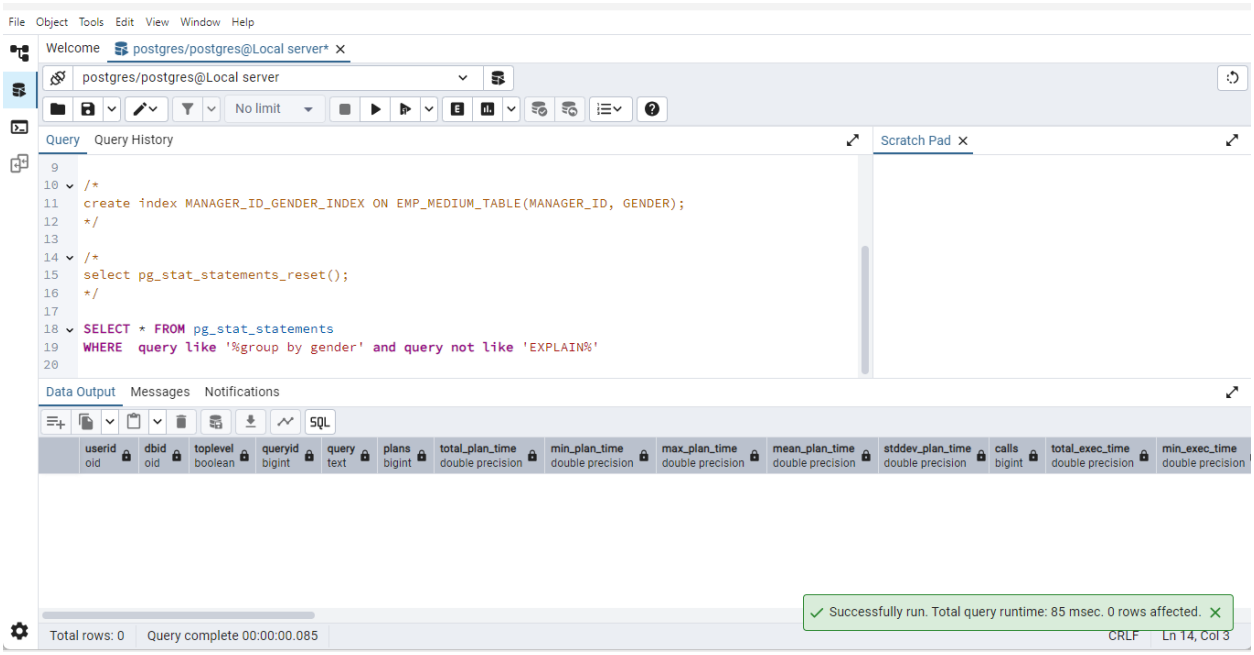
```
select pg_stat_statements_reset();
```



If you run:

```
SELECT * FROM pg_stat_statements

        WHERE  query like '%group by gender' and query not like 'EXPLAIN%'
```

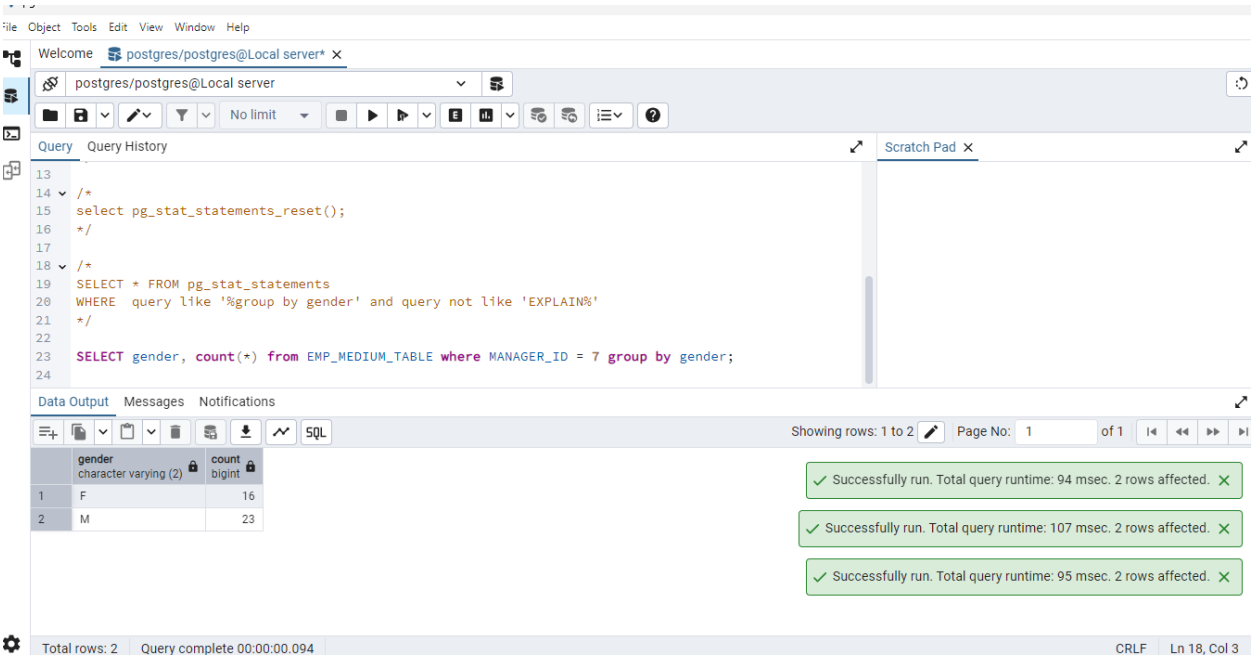
You should see nothing



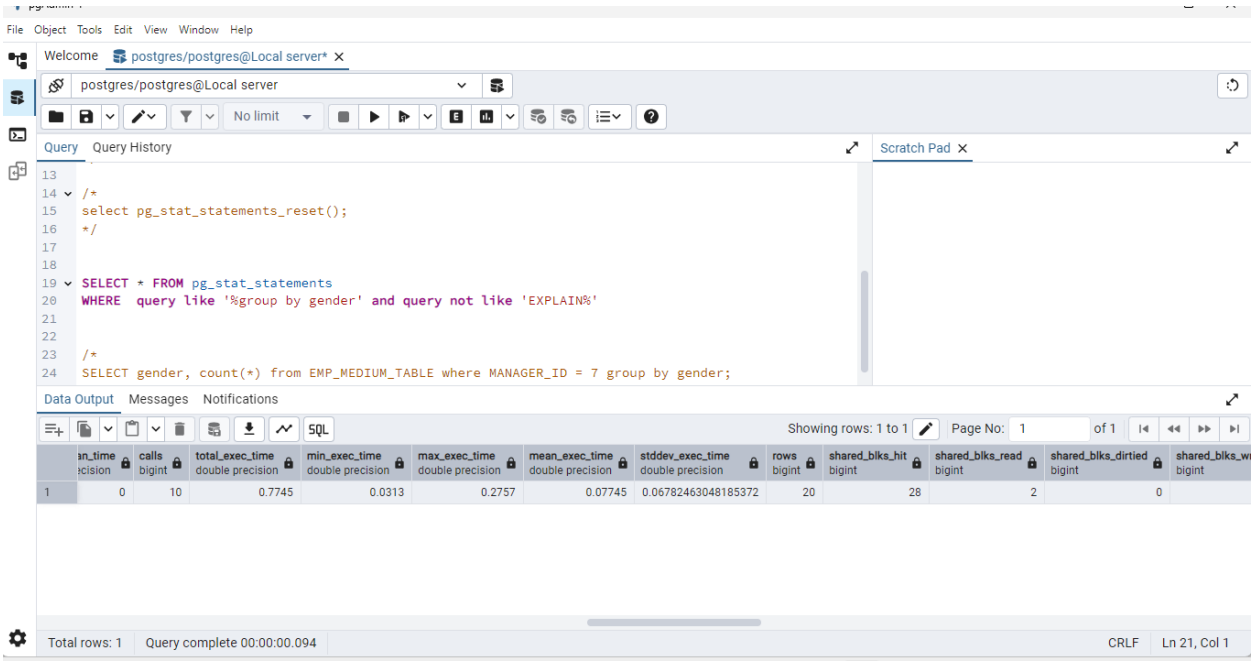
6. Re-run query the same query 10 times:

```
SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by
gender;
```

Note the mean_exec_time and compare with the one before you added the INDEX.



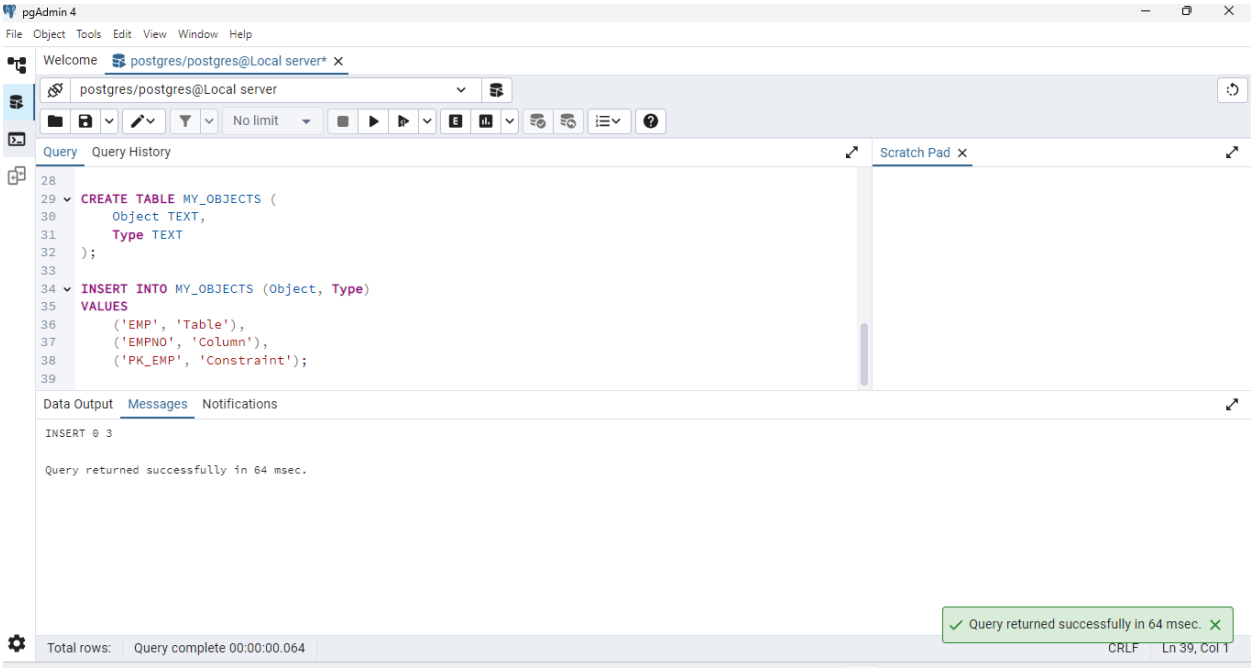
7. Use the EXPLAIN plan again and compare the plan before the INDEX (you should see a difference and explain it).



- **Before Index**: The query performed a **full table scan**, meaning it checked every row in the table. This is inefficient for large datasets.

- **After Index**: The query now uses the **index**, reducing the number of rows scanned and making retrieval much faster.

Exercice 3. Data Dictionary

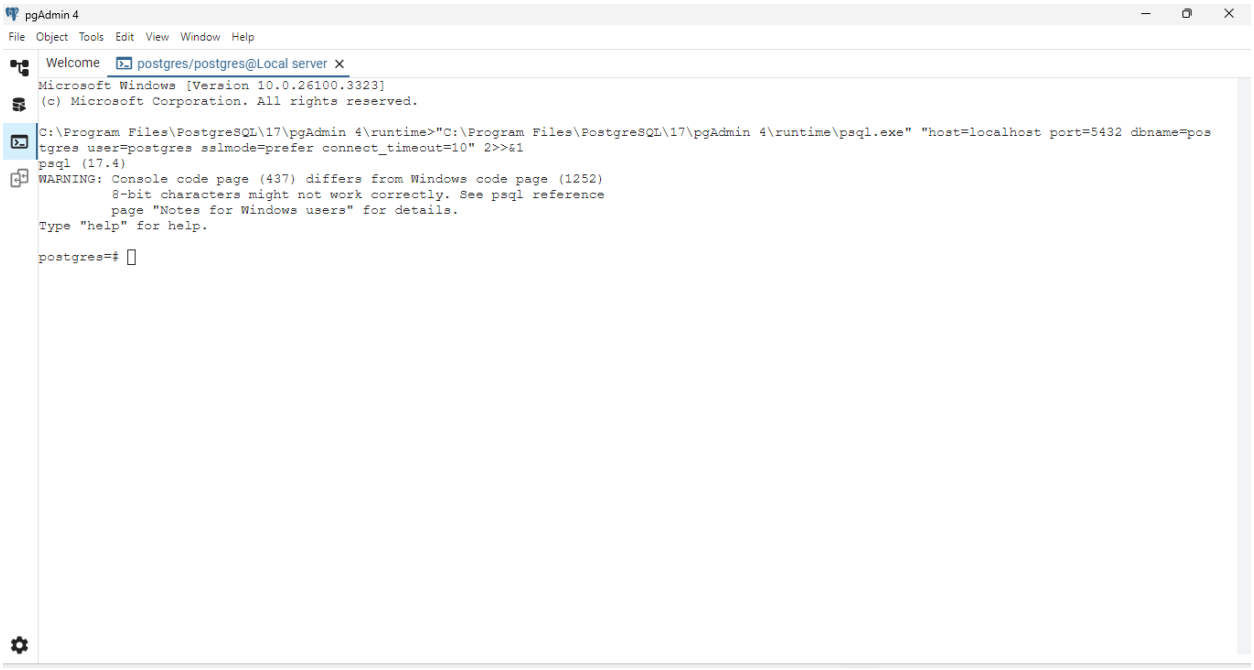


Exercice 4. Use postgres via CLI

Here you will see that you can manipulate the Database with a simple shell.

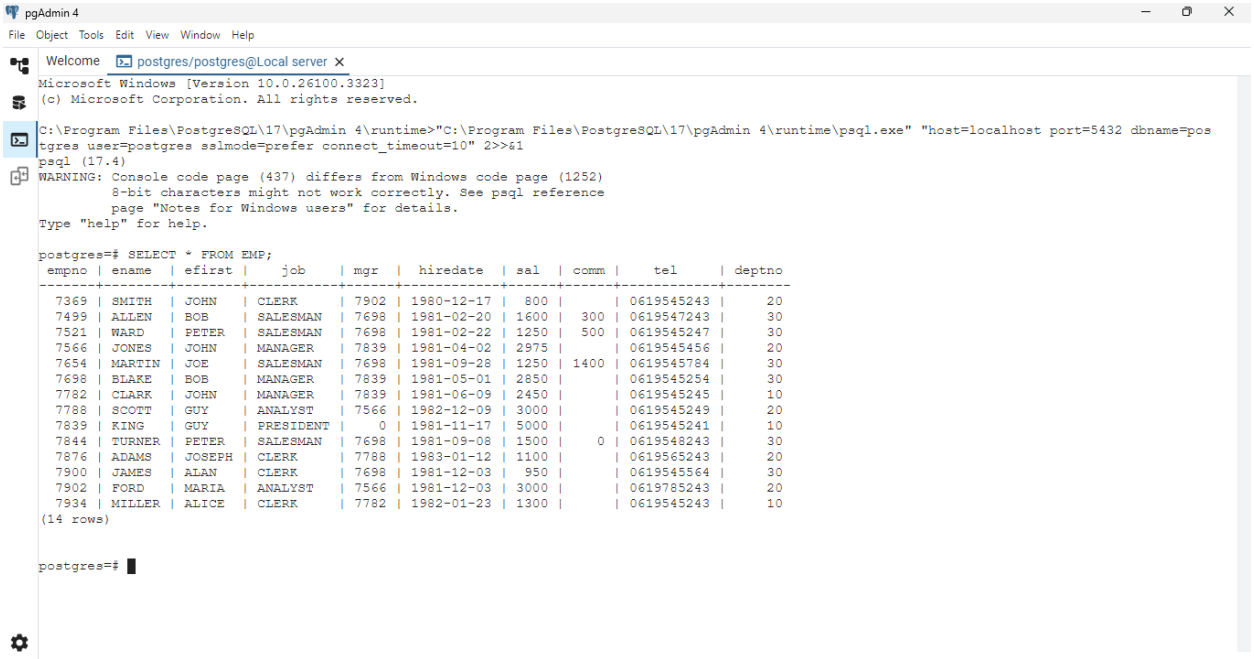
1. Launch CLI

Double click on your database, this should open a CLI connected to your database.



2. On the CLI: Launch a

```
SELECT * FROM EMP;
```



Exercice 5. Transaction Part 1 - Beginner

0. Open 2 sql clients on the same Database.

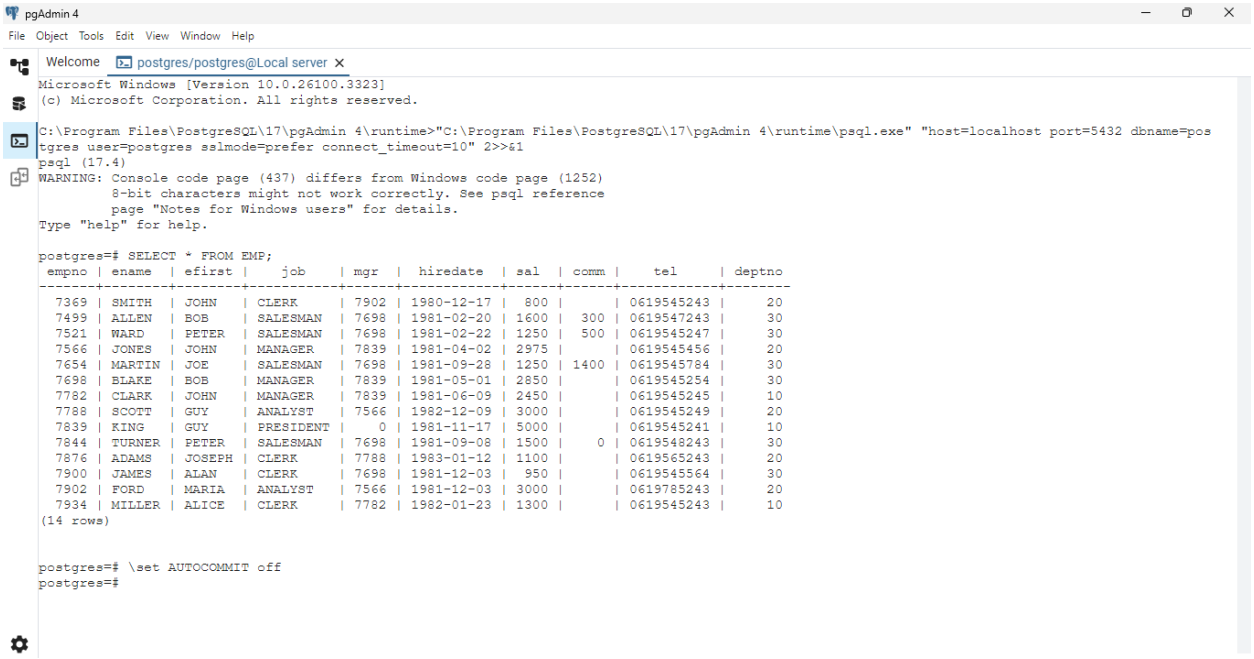
For this Exercise, you should have 2 clients either:

- Open 1 SQLDeveloper + 1 CLI Session.
- Open 2 CLI Sessions.

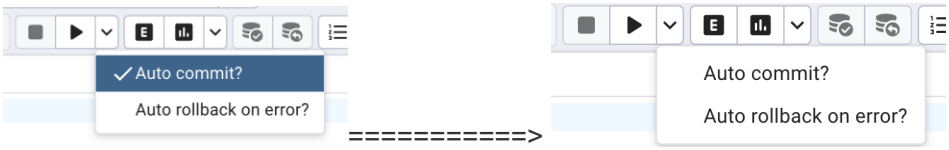
1. **Be sure to have autocommit disabled (default behaviour in SQL).**

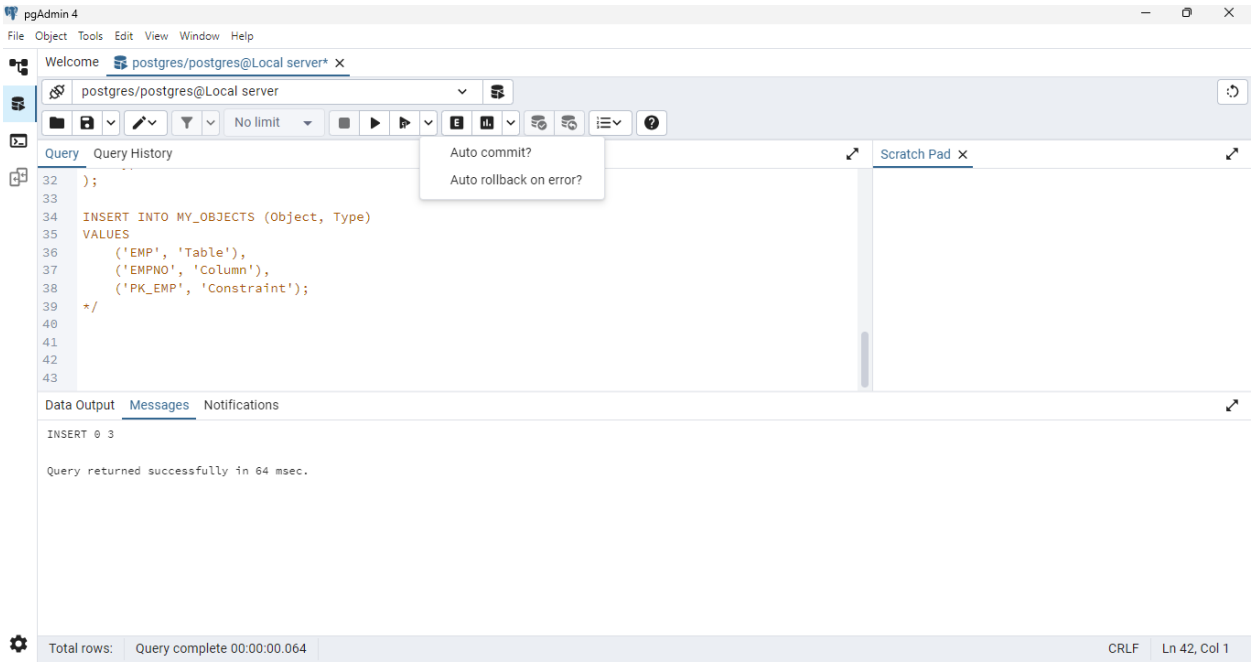
1.1 on the CLI, run:

```
\set AUTOCOMMIT off
```



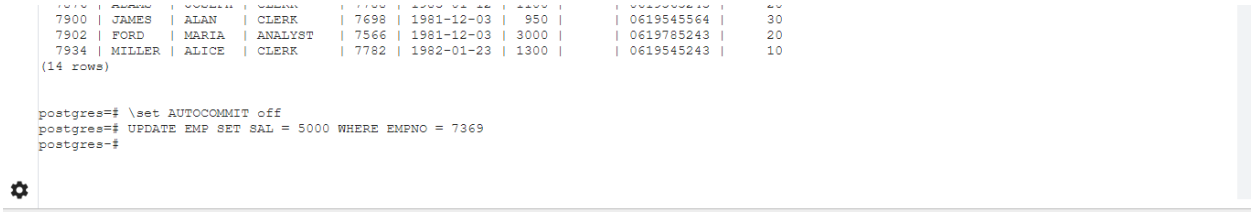
1.2 On the UI, untick that option:





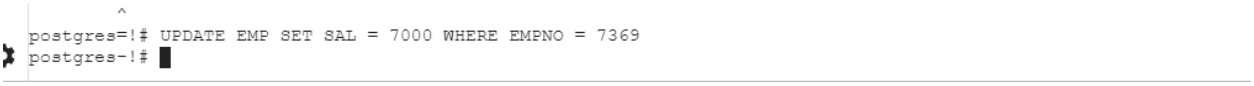
2. **On the first client**: Launch:

```
UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369
```



3. **On the first client**, launch:

```
UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
```



Can you see the UPDATE of the salary for the employee ?

Yes, the update is applied only in the session of client 1. As Autocommit is disabled , the update is not yet committed to the database, that means that other sessions like the client 2 cannot see this update.

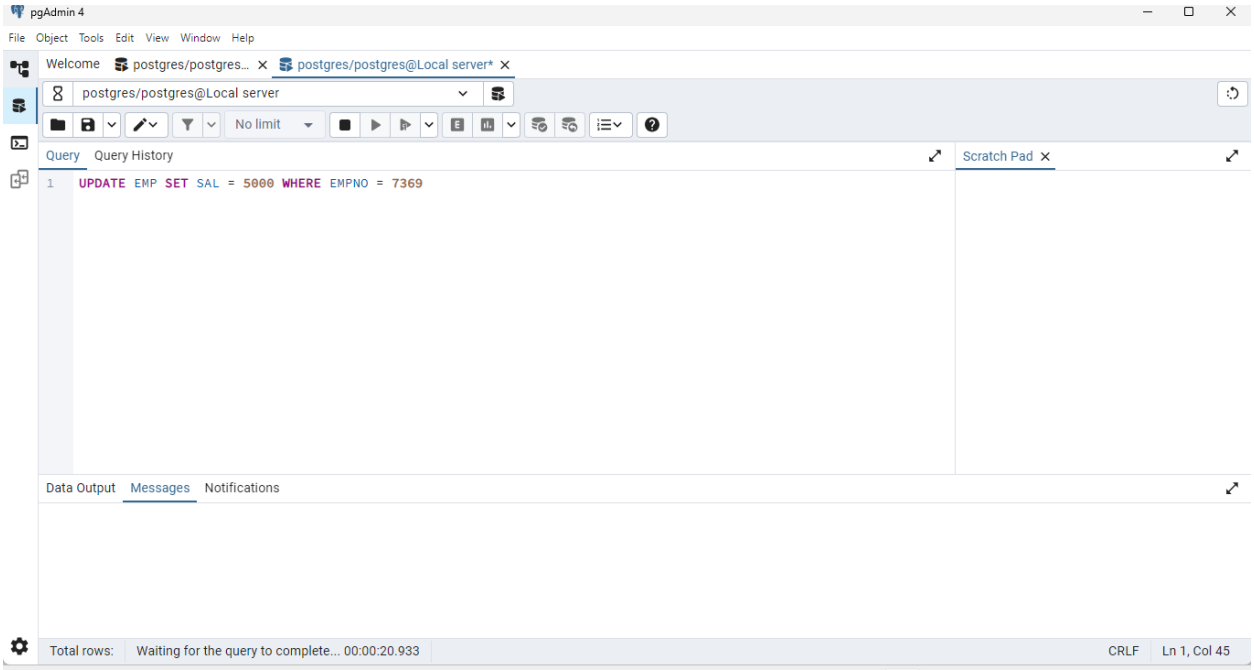
4. **On the second client**, launch:

```
UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
```

Can you see the UPDATE of the salary for the employee ? Why ? How could you make the update available for the second client ?

NO, This is because of transaction isolation. The autocommit is disabled on the CLI meaning that client 1 has not committed yet, client 2 gets blocked or has to wait until client 1 completes its transaction and commit it.

To make the update available for Client 2 to see the updated salary, **Client 1 must commit the transaction**:



5. **On the second client**, launch:

```
UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
```

What is happening and why ?

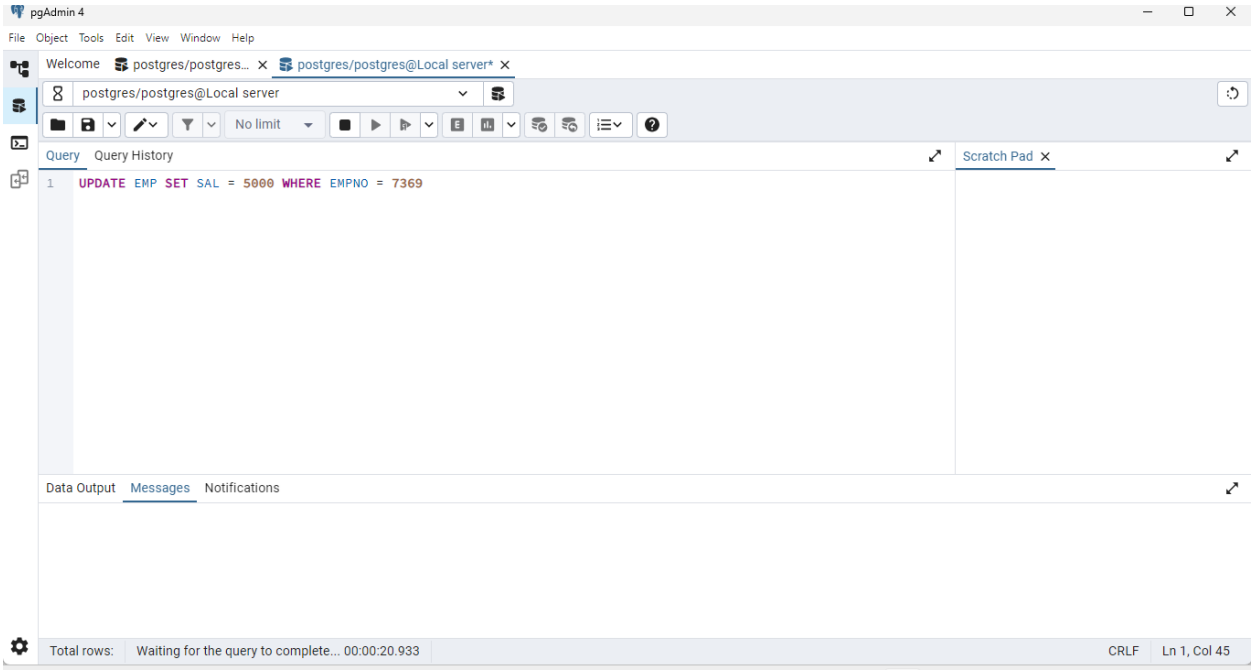
What is the name of this mechanism ? (hint: l**k)

The query is not executing immediately instead it hangs(waits).

This happens because the **first client** has already updated the same row but **has not committed** the transaction yet.

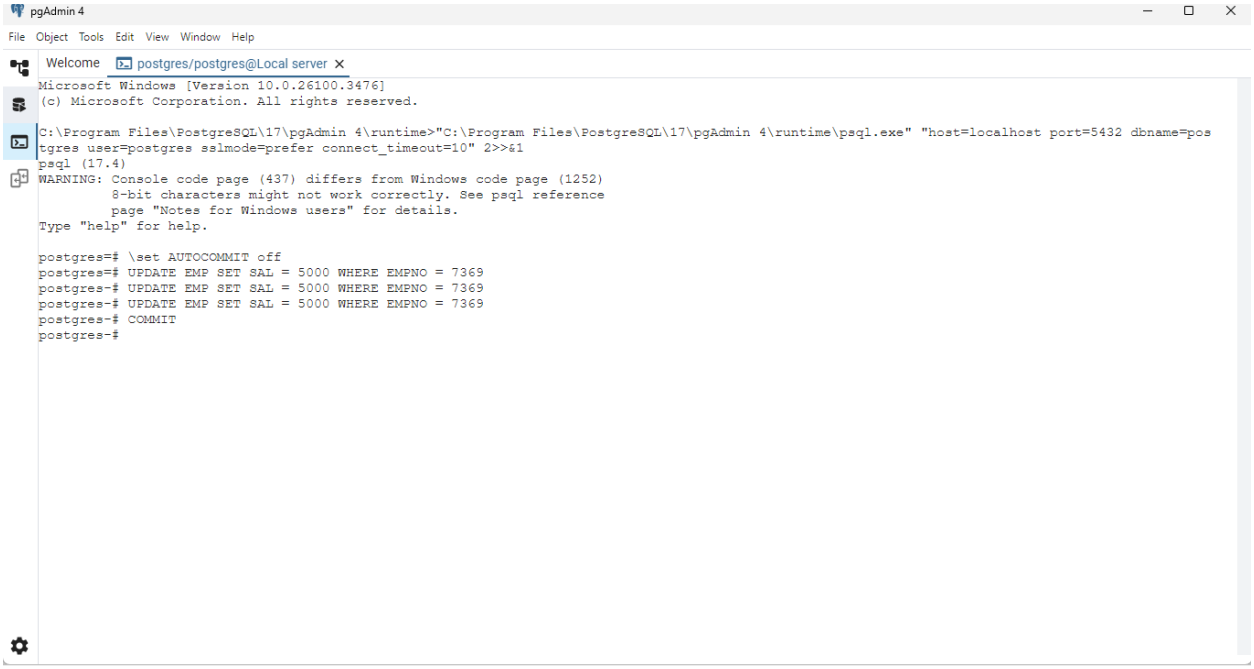
This mechanism is called **locking** (**lock**) - deadlock

Specifically, this is known as a **row-level lock** (or **exclusive lock**).



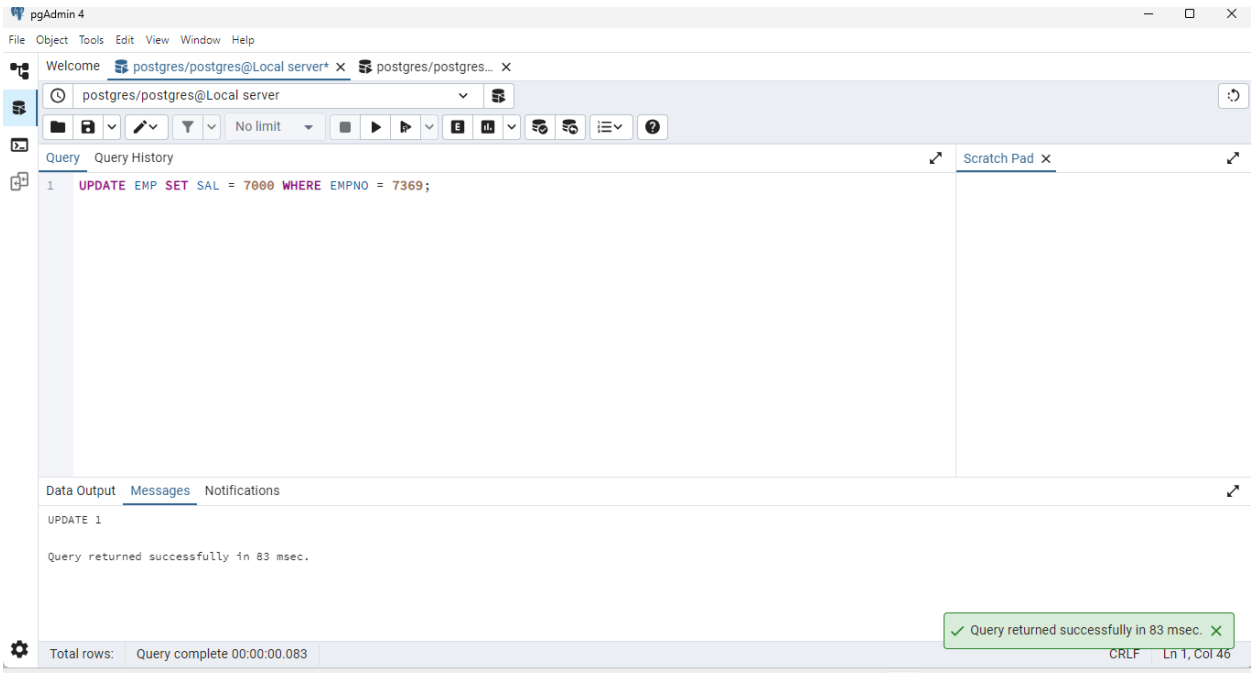
6. **On the first client**, launch:

```
COMMIT
```



7. **On the second client**, launch:

```
UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
```



What happened and why ?

The query by the second client was executed successfully.

This is because client 1 has committed its transaction and the lock is released, The second client's update **now executes successfully** without waiting.

Advanced Database - TP - Part 2

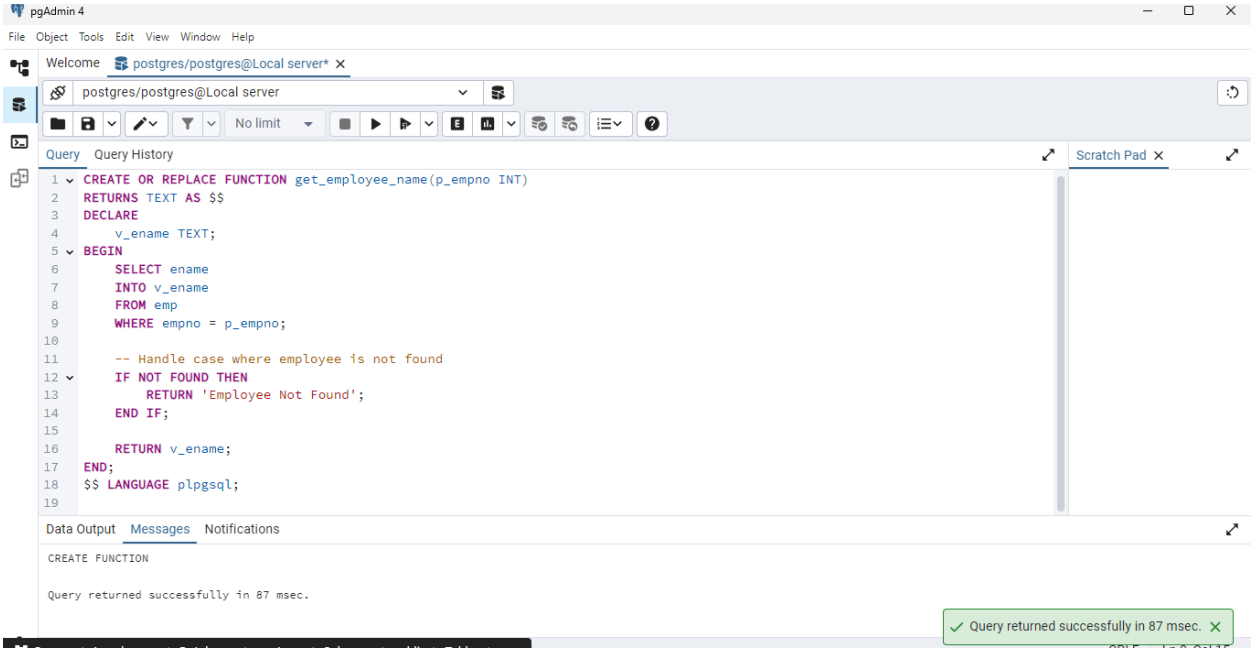
PL/SQL

Exercice 1. Functions

1. Create a function that gets an employee's name from it's empno.

```
CREATE OR REPLACE FUNCTION get_employee_name(p_empno INT)
RETURNS TEXT AS $$
DECLARE
    v_ename TEXT;
BEGIN
    SELECT ename
    INTO v_ename
    FROM emp
    WHERE empno = p_empno;

    -- Handle case where employee is not found
    IF NOT FOUND THEN
        RETURN 'Employee Not Found';
    END IF;

    RETURN v_ename;
END;
```

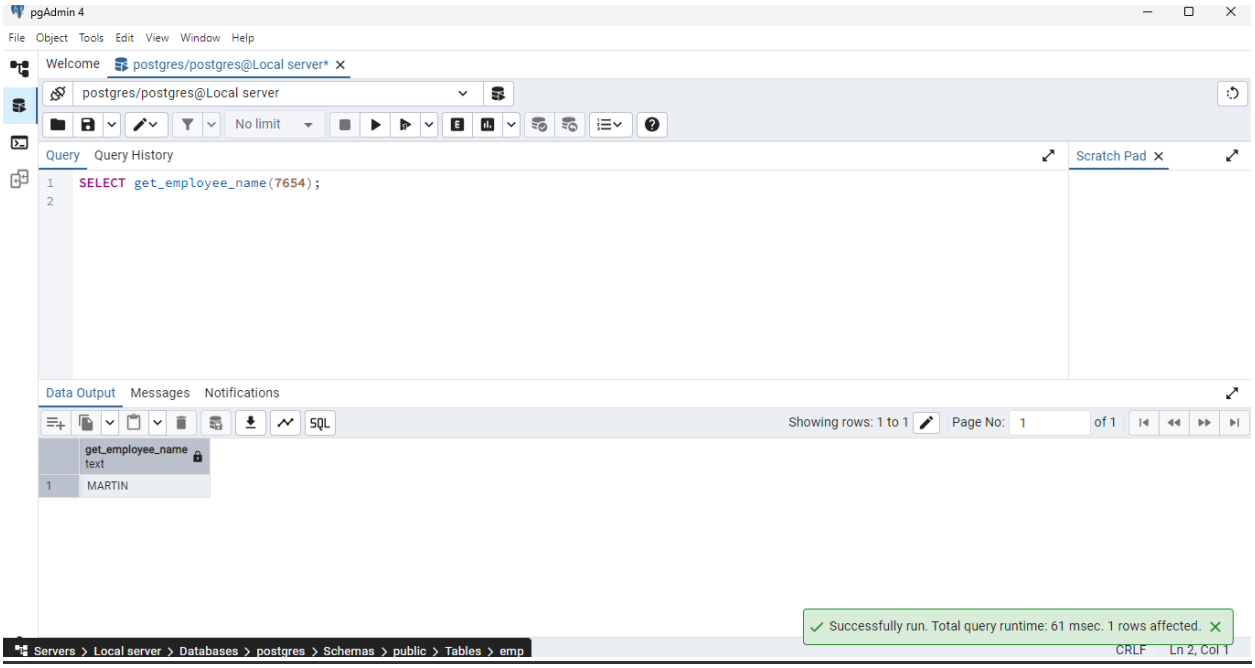
$$ LANGUAGE plpgsql;



2. Test the query with

```
SELECT myFunction(7654) FROM DUAL;
```

SELECT get_employee_name(7654);



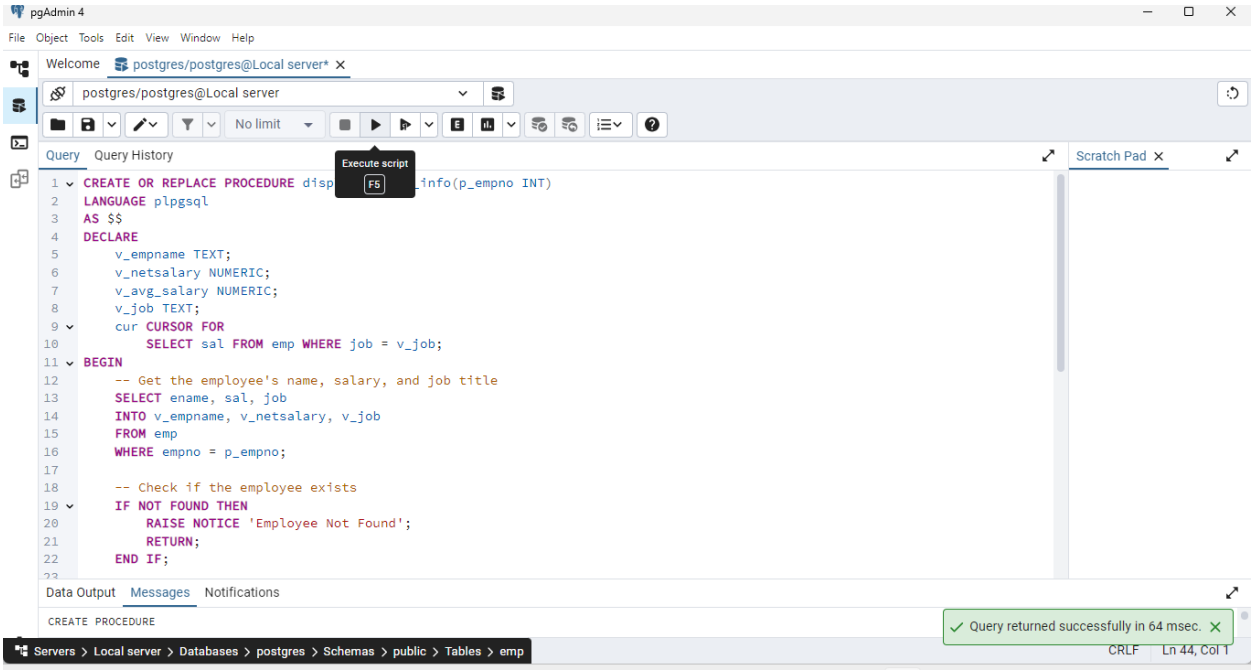
Exercice 2. Procedure & Display & Cursors

Create a procedure that displays the net salary of this employee and the average salary of employees doing the same job

```
CREATE OR REPLACE PROCEDURE display_salary_info(p_empno INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_empname TEXT;
    v_netsalary NUMERIC;
    v_avg_salary NUMERIC;
    v_job TEXT;
    cur CURSOR FOR
```

```
    SELECT sal FROM emp WHERE job = v_job;
BEGIN
    -- Get the employee's name, salary, and job title
    SELECT ename, sal, job
    INTO v_empname, v_netsalary, v_job
    FROM emp
    WHERE empno = p_empno;

    -- Check if the employee exists
    IF NOT FOUND THEN
        RAISE NOTICE 'Employee Not Found';
        RETURN;
    END IF;

    -- Calculate average salary for the same job
    SELECT AVG(sal)
    INTO v_avg_salary
    FROM emp
    WHERE job = v_job;

    -- Display results
    RAISE NOTICE 'Employee: %, Net Salary: %', v_empname, v_netsalary;
    RAISE NOTICE 'Average Salary for Job (%): %', v_job, v_avg_salary;

    -- Cursor to fetch all salaries for the same job
    OPEN cur;
```

```
    LOOP
        FETCH cur INTO v_netsalary;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Salary for %: %', v_job, v_netsalary;
    END LOOP;
    CLOSE cur;
END;
$$;
```



Exercice 3. Procedure & Update

Create a procedure that updates the employee's salary:

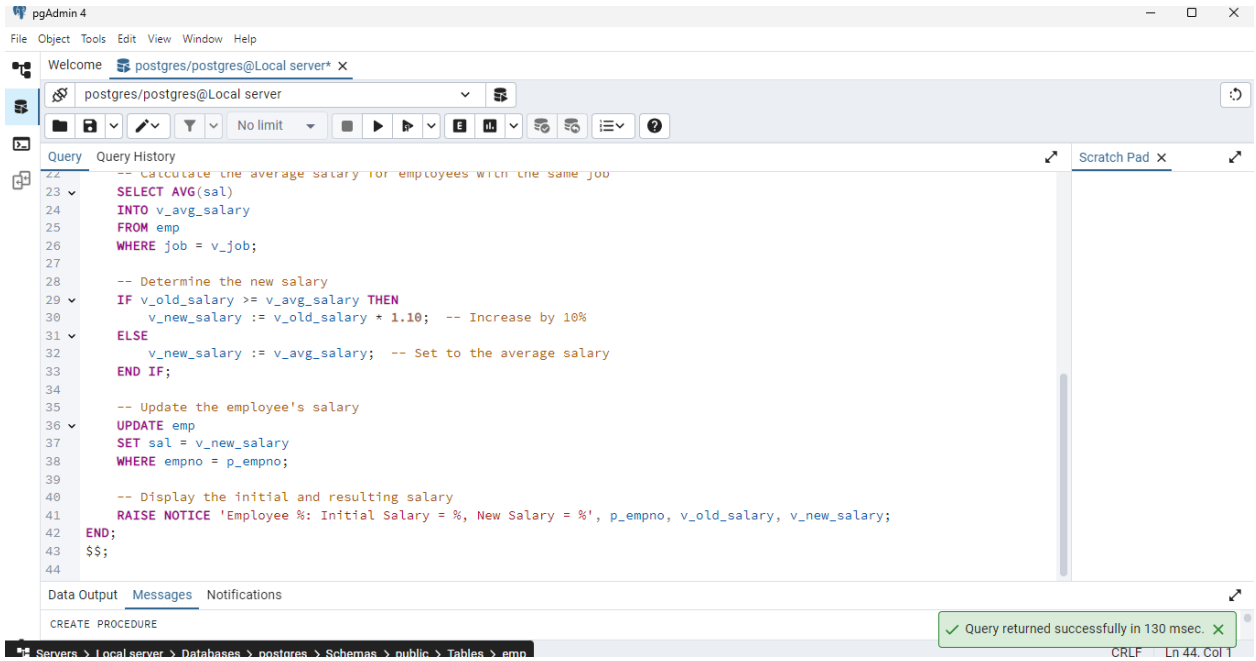
a. if his/her salary is greater or equal to the average one, wage increase of 10%

b. else his/her new salary becomes the average

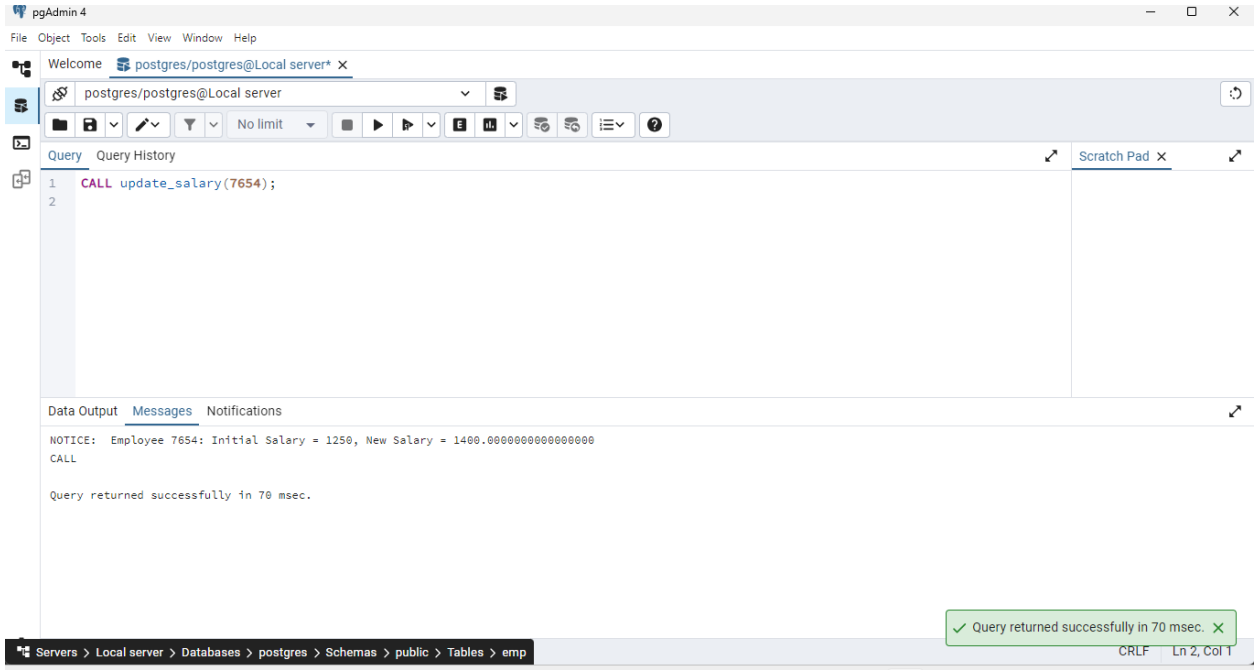
Note: Displays initial and resulting salary.

```
CREATE OR REPLACE PROCEDURE update_salary(p_empno INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_old_salary NUMERIC;
    v_new_salary NUMERIC;
    v_avg_salary NUMERIC;
    v_job TEXT;
BEGIN
    -- Fetch the employee's current salary and job
    SELECT sal, job
    INTO v_old_salary, v_job
    FROM emp
    WHERE empno = p_empno;

    -- Check if employee exists
    IF NOT FOUND THEN
        RAISE NOTICE 'Employee Not Found';
        RETURN;
    END IF;

    -- Calculate the average salary for employees with the same job
    SELECT AVG(sal)
    INTO v_avg_salary
```

```
    FROM emp
    WHERE job = v_job;

    -- Determine the new salary
    IF v_old_salary >= v_avg_salary THEN
        v_new_salary := v_old_salary * 1.10;  -- Increase by 10%
    ELSE
        v_new_salary := v_avg_salary;  -- Set to the average salary
    END IF;

    -- Update the employee's salary
    UPDATE emp
    SET sal = v_new_salary
    WHERE empno = p_empno;

    -- Display the initial and resulting salary
    RAISE NOTICE 'Employee %: Initial Salary = %, New Salary = %', p_empno, v_old_salary,
v_new_salary;
END;
$$;
```



CALL update_salary(7654);



Exercice 4. Procedure

Calculate a bonus for the employees of table EMP, as follows:

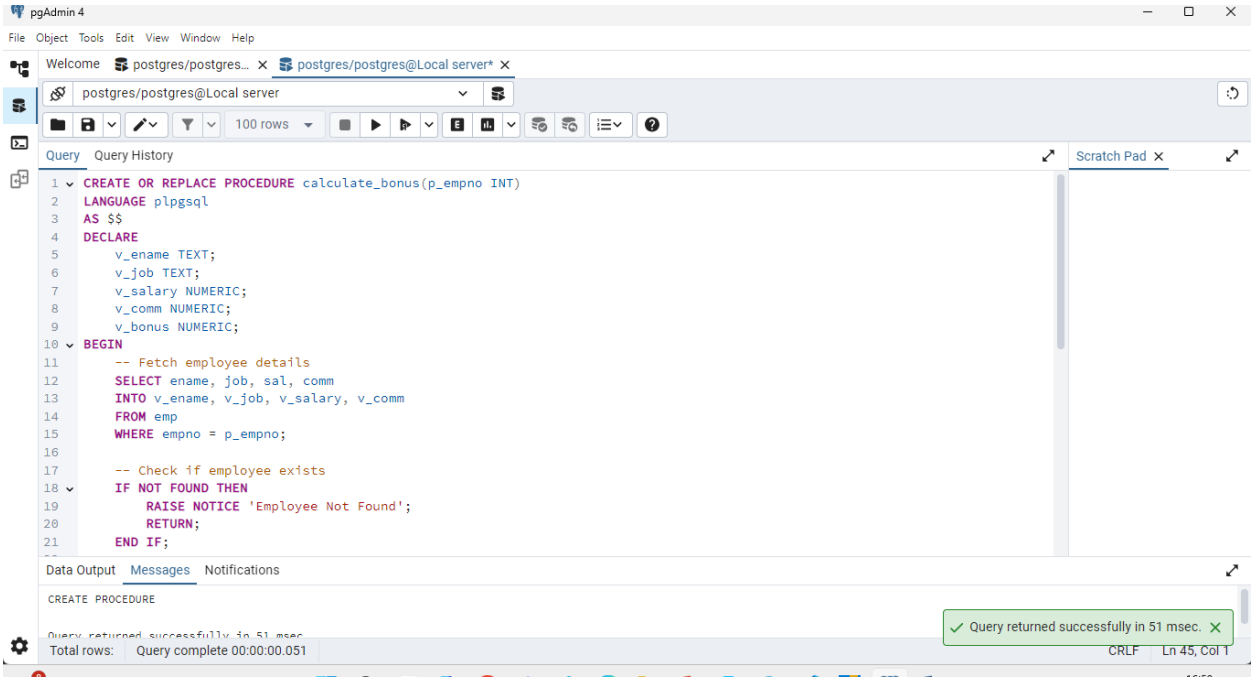
- SALESMAN: bonus = commission * 2

- CLERK: bonus = wage increase of 15%
- MANAGER: bonus = wage increase of 18%

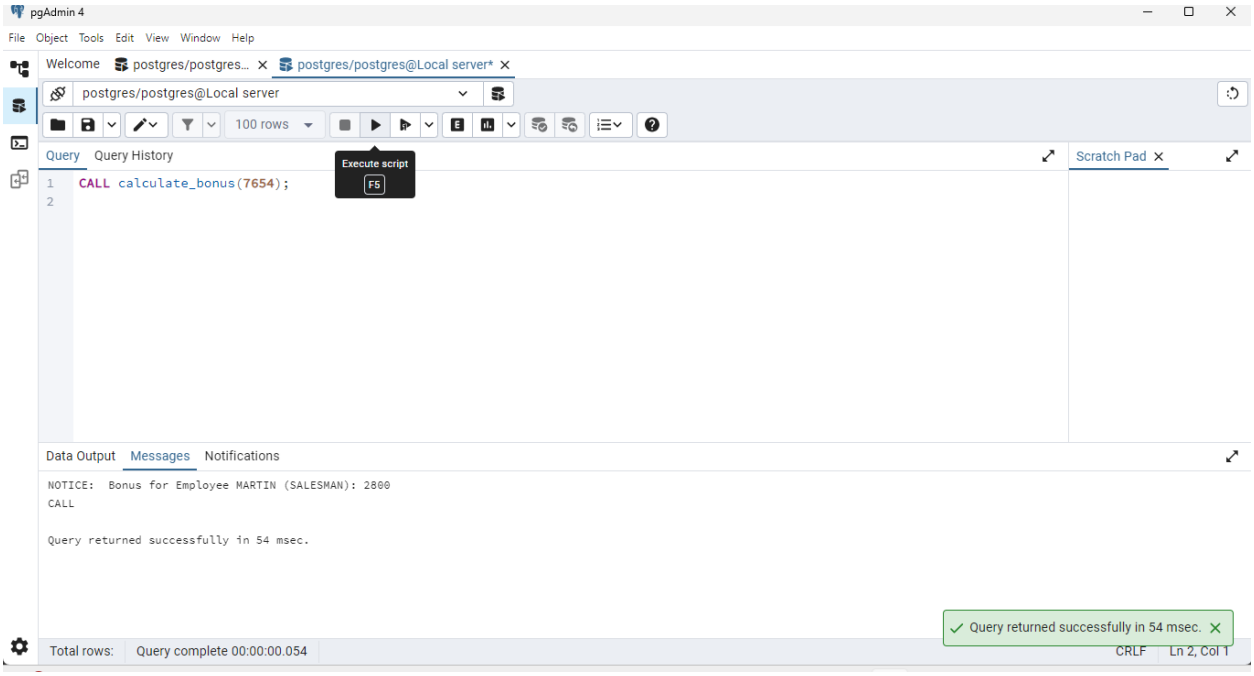
Insert the calculated bonuses in the table BONUS.

```
CREATE OR REPLACE PROCEDURE calculate_bonus(p_empno INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_ename TEXT;
    v_job TEXT;
    v_salary NUMERIC;
    v_comm NUMERIC;
    v_bonus NUMERIC;
BEGIN
    -- Fetch employee details
    SELECT ename, job, sal, comm
    INTO v_ename, v_job, v_salary, v_comm
    FROM emp
    WHERE empno = p_empno;

    -- Check if employee exists
    IF NOT FOUND THEN
        RAISE NOTICE 'Employee Not Found';
        RETURN;
    END IF;
```

```
    -- Calculate bonus based on job role
    CASE v_job
        WHEN 'SALESMAN' THEN
            v_bonus := COALESCE(v_comm, 0) * 2;  -- Commission * 2 (Handle NULL commission)
        WHEN 'CLERK' THEN
            v_bonus := v_salary * 0.15;  -- 15% increase
        WHEN 'MANAGER' THEN
            v_bonus := v_salary * 0.18;  -- 18% increase
        ELSE
            v_bonus := 0;  -- No bonus for other roles
    END CASE;

    -- Insert or update the bonus in the BONUS table
    INSERT INTO bonus (empno, ename, job, sal, comm)
    VALUES (p_empno, v_ename, v_job, v_bonus, v_comm)
    ON CONFLICT (empno)
    DO UPDATE SET sal = EXCLUDED.sal, comm = EXCLUDED.comm;

    -- Display the result
    RAISE NOTICE 'Bonus for Employee % (%): %', v_ename, v_job, v_bonus;
END;
$$;
```



CALL calculate_bonus(7654);



Exercice 5. SELECT for UPDATE

Change the commission for employees by putting them all to 0

Write the procedure UpdateCommission for modifying the commission of employees based on their salary

- Sal<=1000 COMM=800
- Sal<=2000 COMM=1200
- Sal>2000 COMM=1500

Indications:

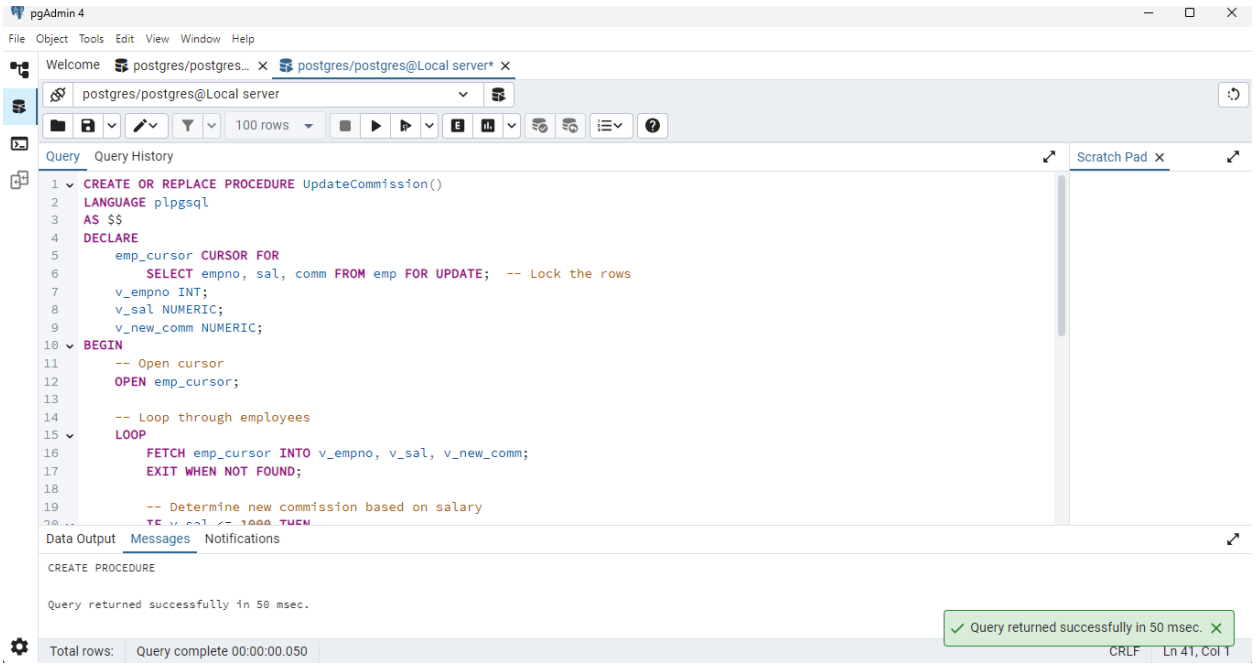
use the instruction “SELECT ... FROM table FOR UPDATE”

This will allow you to put a lock on the rows you want to update

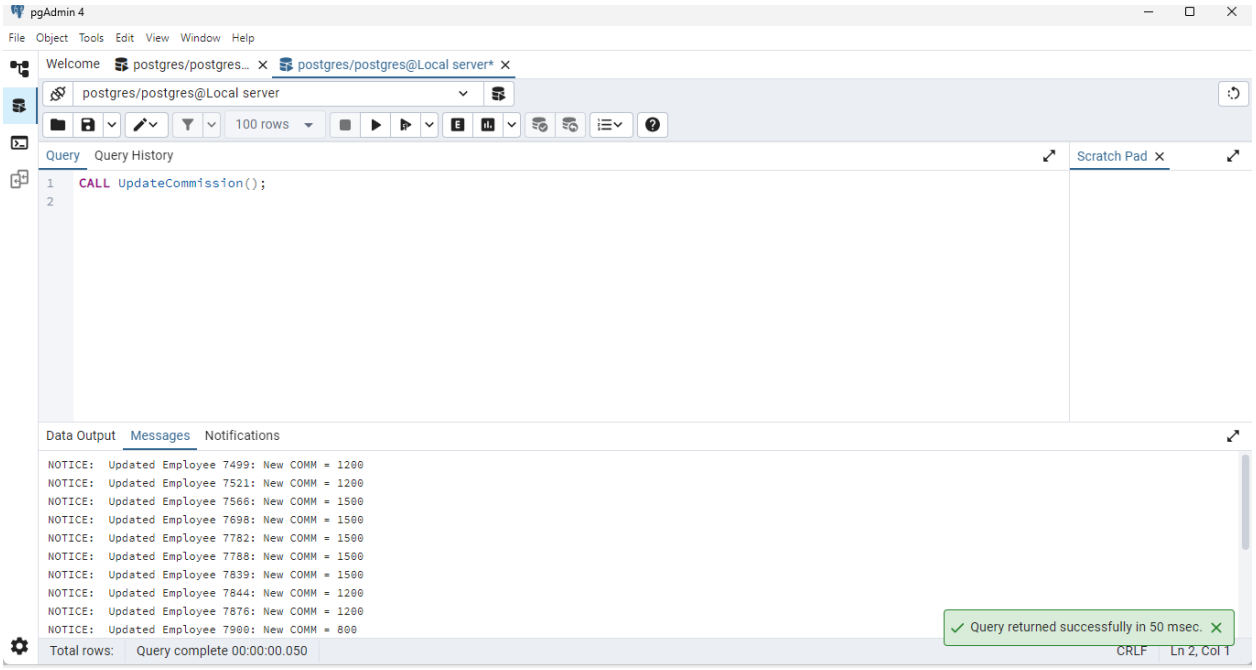
The following statement specifies the current tuple to modify with an UPDATE or DELETE WHERE CURRENT OF cursor

```
CREATE OR REPLACE PROCEDURE UpdateCommission()
LANGUAGE plpgsql
AS $$
DECLARE
    emp_cursor CURSOR FOR
        SELECT empno, sal, comm FROM emp FOR UPDATE;  -- Lock the rows
    v_empno INT;
    v_sal NUMERIC;
    v_new_comm NUMERIC;
BEGIN
    -- Open cursor
    OPEN emp_cursor;

    -- Loop through employees
```

```
    LOOP
        FETCH emp_cursor INTO v_empno, v_sal, v_new_comm;
        EXIT WHEN NOT FOUND;

        -- Determine new commission based on salary
        IF v_sal <= 1000 THEN
            v_new_comm := 800;
        ELSIF v_sal <= 2000 THEN
            v_new_comm := 1200;
        ELSE
            v_new_comm := 1500;
        END IF;

        -- Update the employee's commission
        UPDATE emp
        SET comm = v_new_comm
        WHERE CURRENT OF emp_cursor;

        -- Display updated result
        RAISE NOTICE 'Updated Employee %: New COMM = %', v_empno, v_new_comm;
    END LOOP;

    -- Close cursor
    CLOSE emp_cursor;
END;
$$;
```



CALL UpdateCommission();

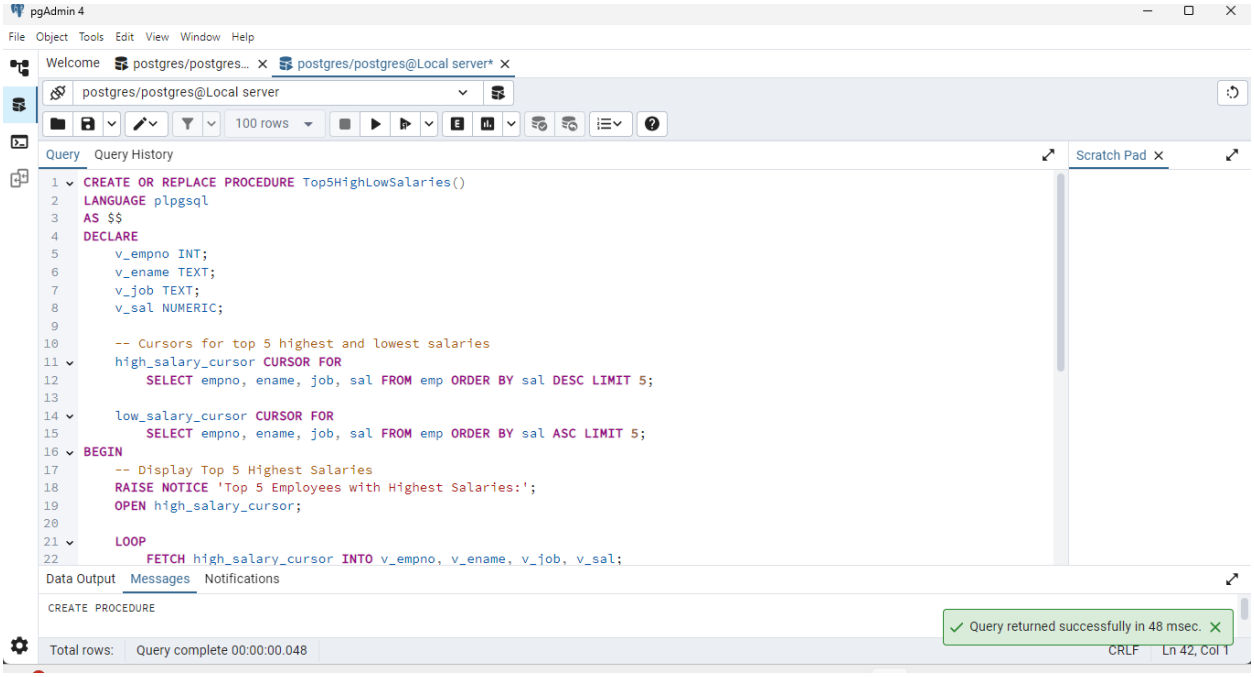


Exercice 6. Condition on EXIT loop

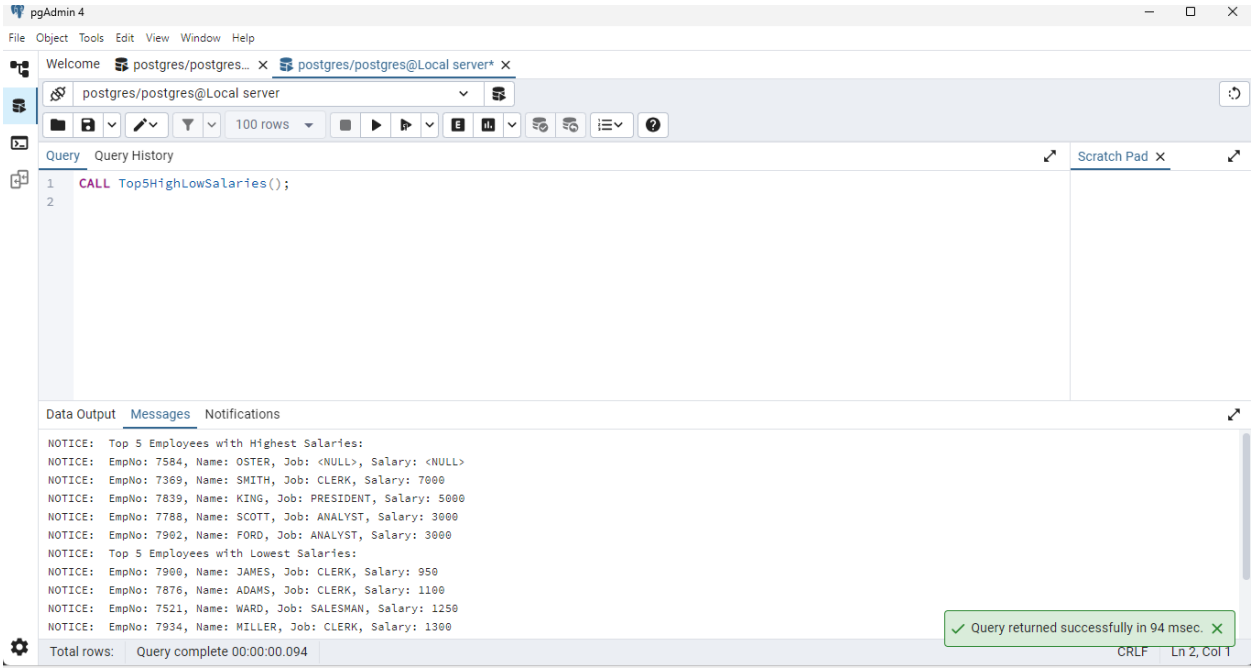
Write a procedure that displays the five employees with the highest salaries and the five employees with the lowest salaries. This procedure should use a cursor, a loop and a condition to exit the loop.

```
CREATE OR REPLACE PROCEDURE Top5HighLowSalaries()
LANGUAGE plpgsql
AS $$
DECLARE
    v_empno INT;
    v_ename TEXT;
    v_job TEXT;
    v_sal NUMERIC;

    -- Cursors for top 5 highest and lowest salaries
    high_salary_cursor CURSOR FOR
        SELECT empno, ename, job, sal FROM emp ORDER BY sal DESC LIMIT 5;

    low_salary_cursor CURSOR FOR
        SELECT empno, ename, job, sal FROM emp ORDER BY sal ASC LIMIT 5;
BEGIN
    -- Display Top 5 Highest Salaries
    RAISE NOTICE 'Top 5 Employees with Highest Salaries:';
    OPEN high_salary_cursor;

    LOOP
```

```
        FETCH high_salary_cursor INTO v_empno, v_ename, v_job, v_sal;
        EXIT WHEN NOT FOUND;  -- Exit loop if no more records
        RAISE NOTICE 'EmpNo: %, Name: %, Job: %, Salary: %', v_empno, v_ename, v_job, v_sal;
    END LOOP;

    CLOSE high_salary_cursor;

    -- Display Top 5 Lowest Salaries
    RAISE NOTICE 'Top 5 Employees with Lowest Salaries:';
    OPEN low_salary_cursor;

    LOOP
        FETCH low_salary_cursor INTO v_empno, v_ename, v_job, v_sal;
        EXIT WHEN NOT FOUND;  -- Exit loop if no more records
        RAISE NOTICE 'EmpNo: %, Name: %, Job: %, Salary: %', v_empno, v_ename, v_job, v_sal;
    END LOOP;

    CLOSE low_salary_cursor;
END;
$$;
```



CALL Top5HighLowSalaries();

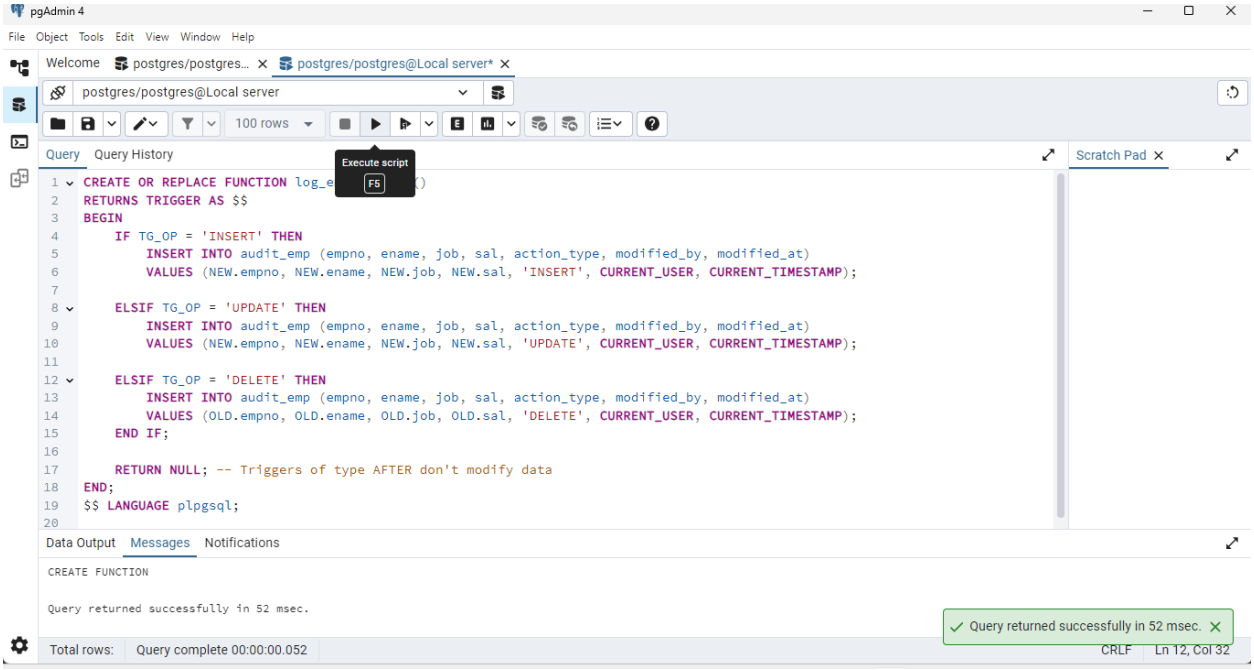


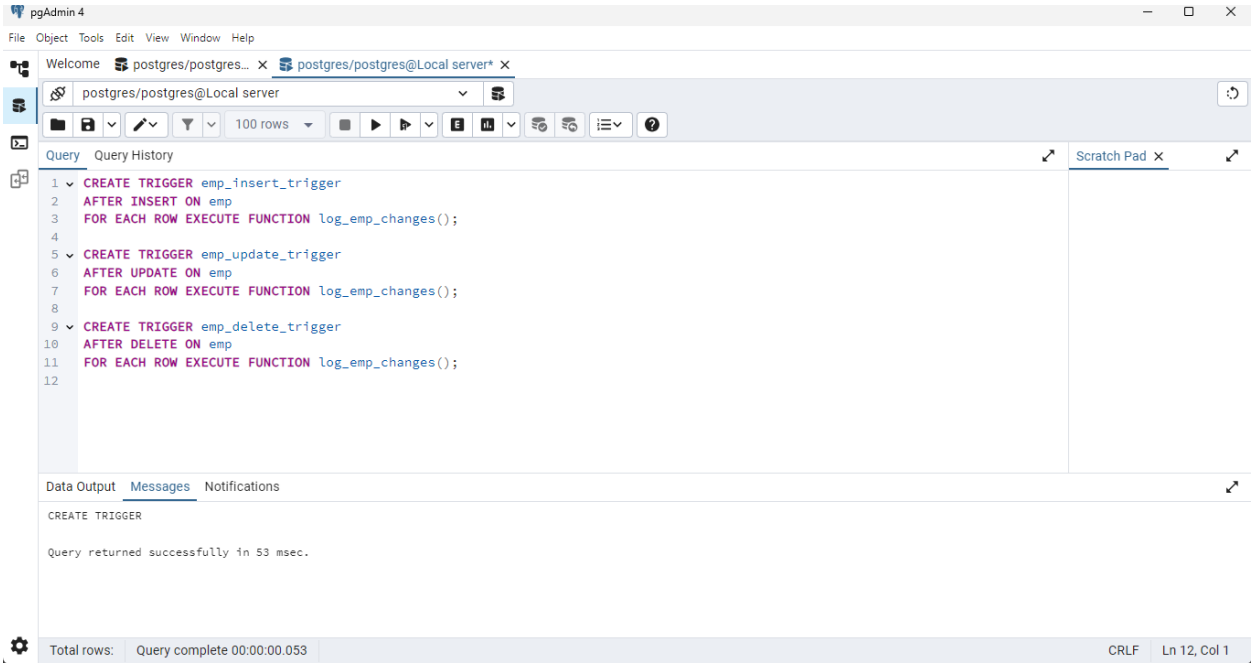
Exercice 7. Triggers

The database administrator (DBA) wants to store the user who modifies (insert, delete, or update) the EMP table and also the time of modification.

1. **Create The triggers:** Set one or many triggers that can take into account these requirements. Use the functions user() and sysdate() that can retrieve the name of the Oracle user and current date and time respectively.

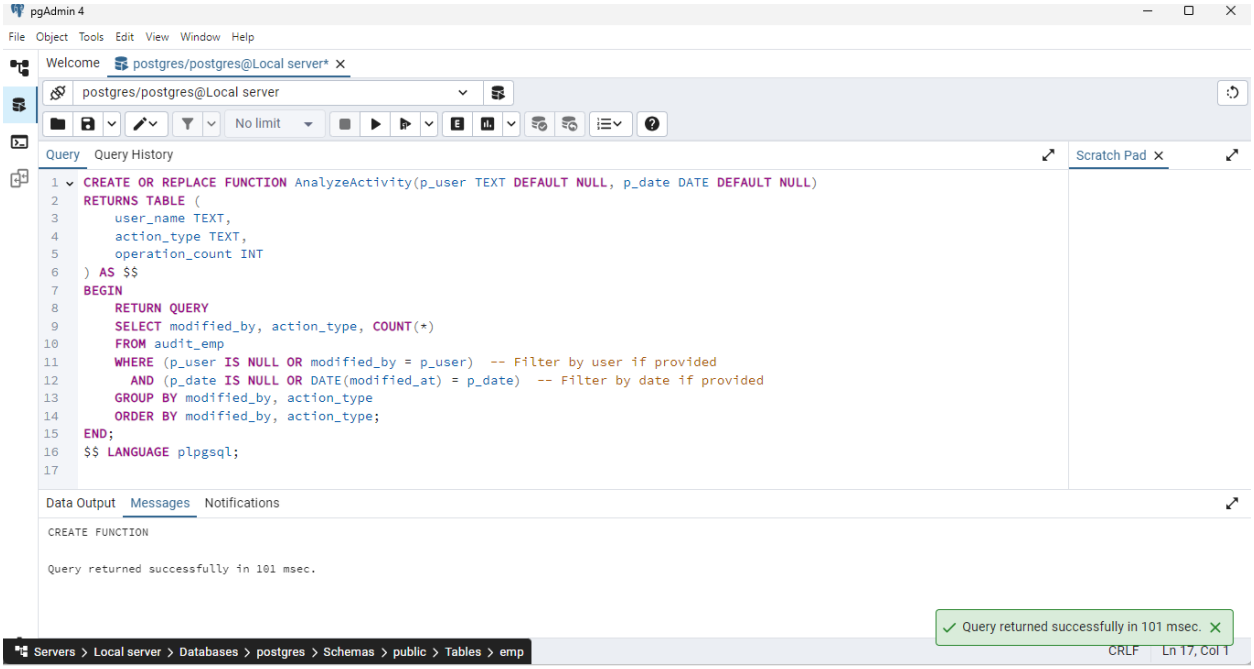
```
CREATE OR REPLACE FUNCTION log_emp_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
        VALUES (NEW.empno, NEW.ename, NEW.job, NEW.sal, 'INSERT', CURRENT_USER, CURRENT_TIMESTAMP);

    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
        VALUES (NEW.empno, NEW.ename, NEW.job, NEW.sal, 'UPDATE', CURRENT_USER, CURRENT_TIMESTAMP);

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
        VALUES (OLD.empno, OLD.ename, OLD.job, OLD.sal, 'DELETE', CURRENT_USER, CURRENT_TIMESTAMP);
    END IF;

    RETURN NULL; -- Triggers of type AFTER don't modify data
END;
$$ LANGUAGE plpgsql;
```





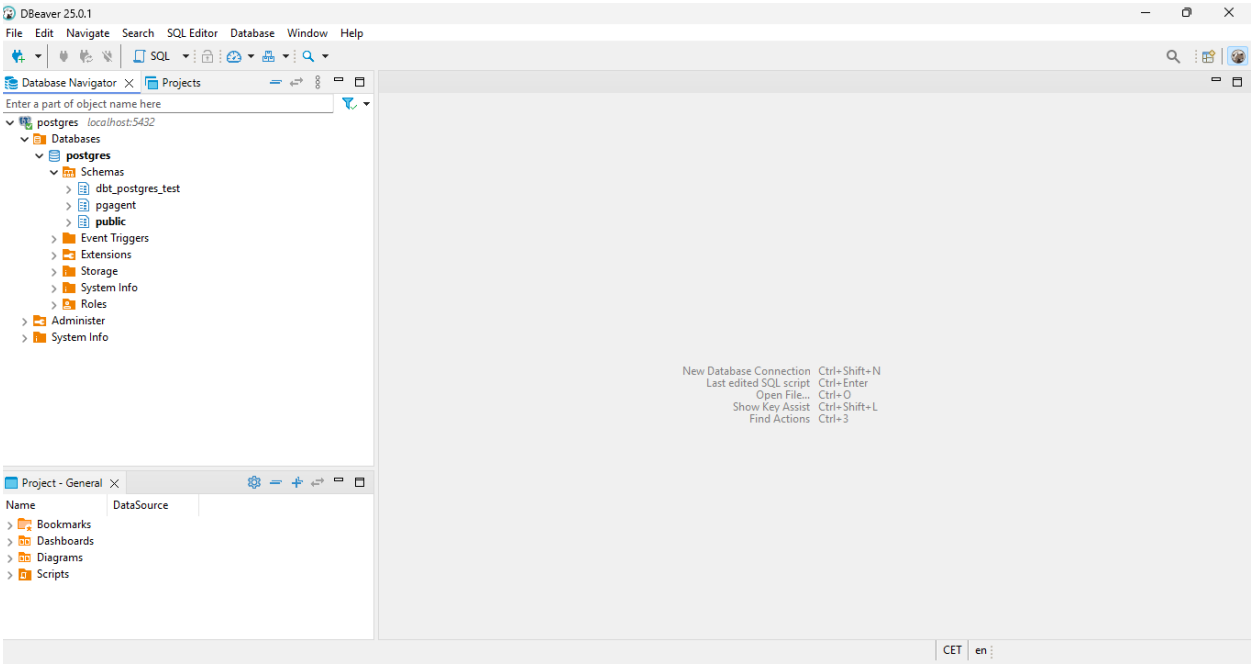
2. **Function to analyze results:** Write a function AnalyzeActivity that accepts a Oracle user name and/or a date and calculate the number of operations performed by the user, whether for all the users during the specific day, or for the specific user on the specific day, or for a specific user since the table creation.

```
CREATE OR REPLACE FUNCTION AnalyzeActivity(p_user TEXT DEFAULT NULL, p_date DATE DEFAULT NULL)
RETURNS TABLE (
    user_name TEXT,
    action_type TEXT,
    operation_count INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT modified_by, action_type, COUNT(*)
    FROM audit_emp
    WHERE (p_user IS NULL OR modified_by = p_user)  -- Filter by user if provided
      AND (p_date IS NULL OR DATE(modified_at) = p_date)  -- Filter by date if provided
    GROUP BY modified_by, action_type
    ORDER BY modified_by, action_type;
END;
$$ LANGUAGE plpgsql;
```

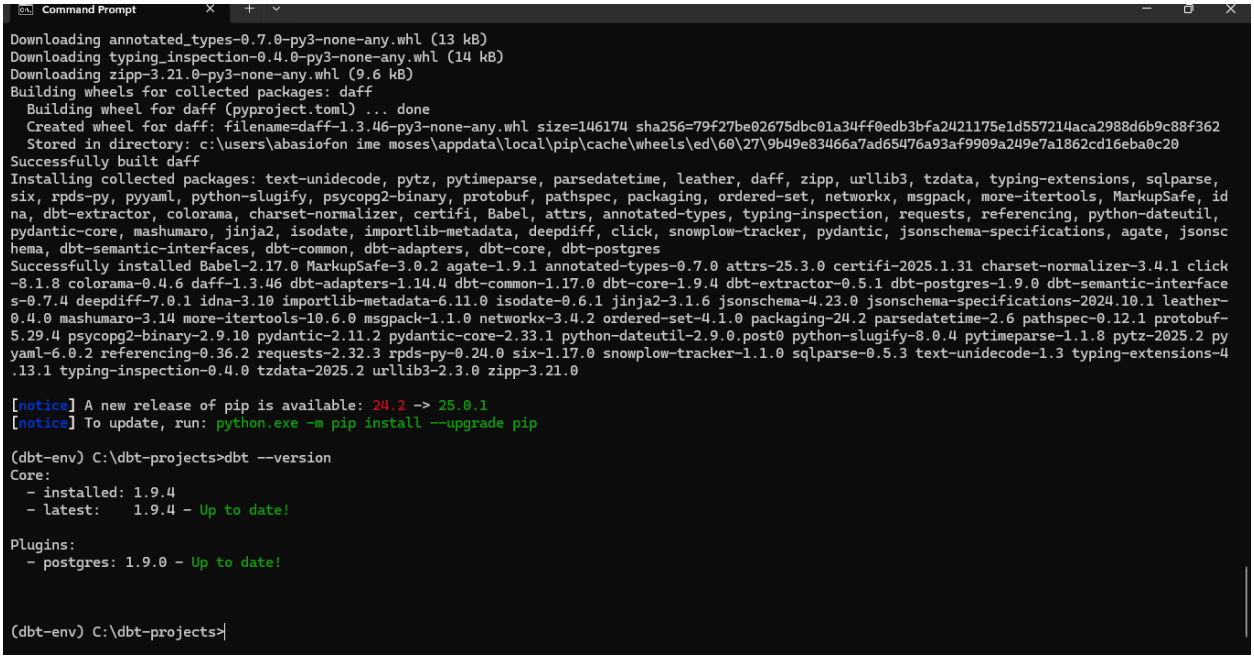

pgAdmin 4
File Object Tools Edit View Window Help
Welcome
postgres/postgres@Local server*
postgres/postgres@Local server
No limit
Query
Query History
Scratch Pad
CREATE OR REPLACE FUNCTION AnalyzeActivity(p_user TEXT DEFAULT NULL, p_date DATE DEFAULT NULL)
RETURNS TABLE (
user_name TEXT,
action_type TEXT,
operation_count INT
) AS $$
BEGIN
RETURN QUERY
SELECT modified_by, action_type, COUNT(*)
FROM audit_emp
WHERE (p_user IS NULL OR modified_by = p_user) -- Filter by user if provided
AND (p_date IS NULL OR DATE(modified_at) = p_date) -- Filter by date if provided
GROUP BY modified_by, action_type
ORDER BY modified_by, action_type;
END;
$$ LANGUAGE plpgsql;
Data Output
Messages
Notifications
CREATE FUNCTION
Query returned successfully in 101 msec.
Query returned successfully in 101 msec.
Servers > Local server > Databases > postgres > Schemas > public > Tables > emp
CRLF
Ln 17, Col 1

By Thibaut de Broca

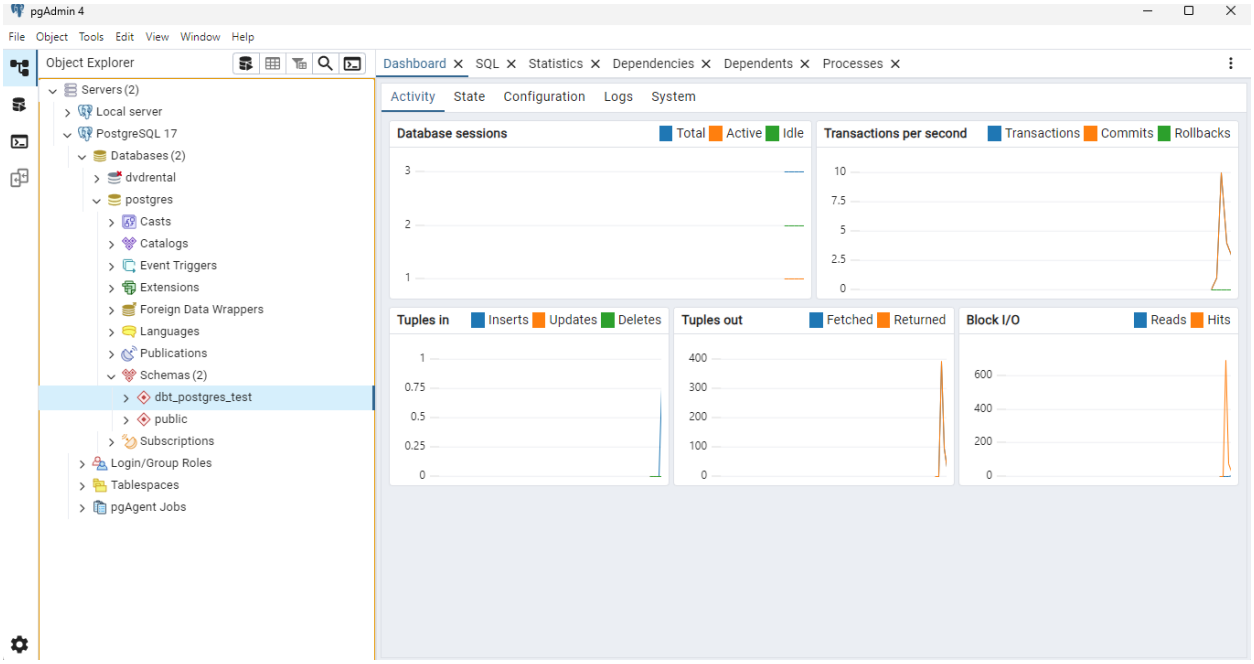
GROUPTOWN



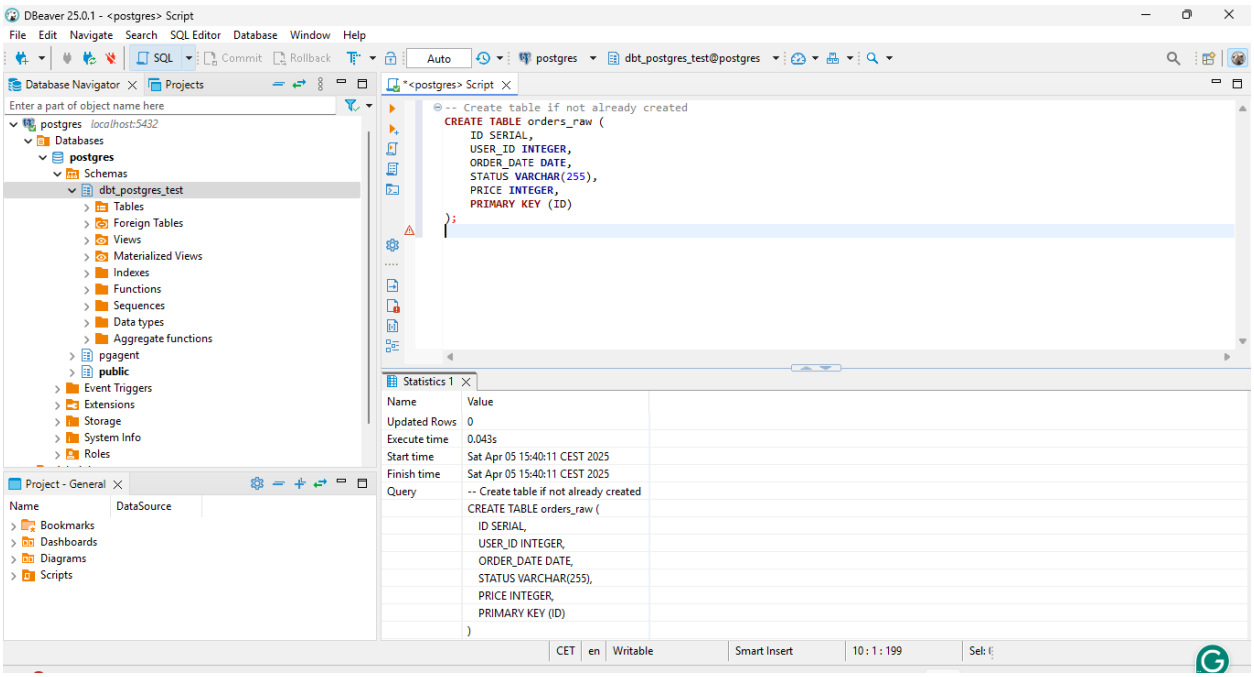
Install DBT

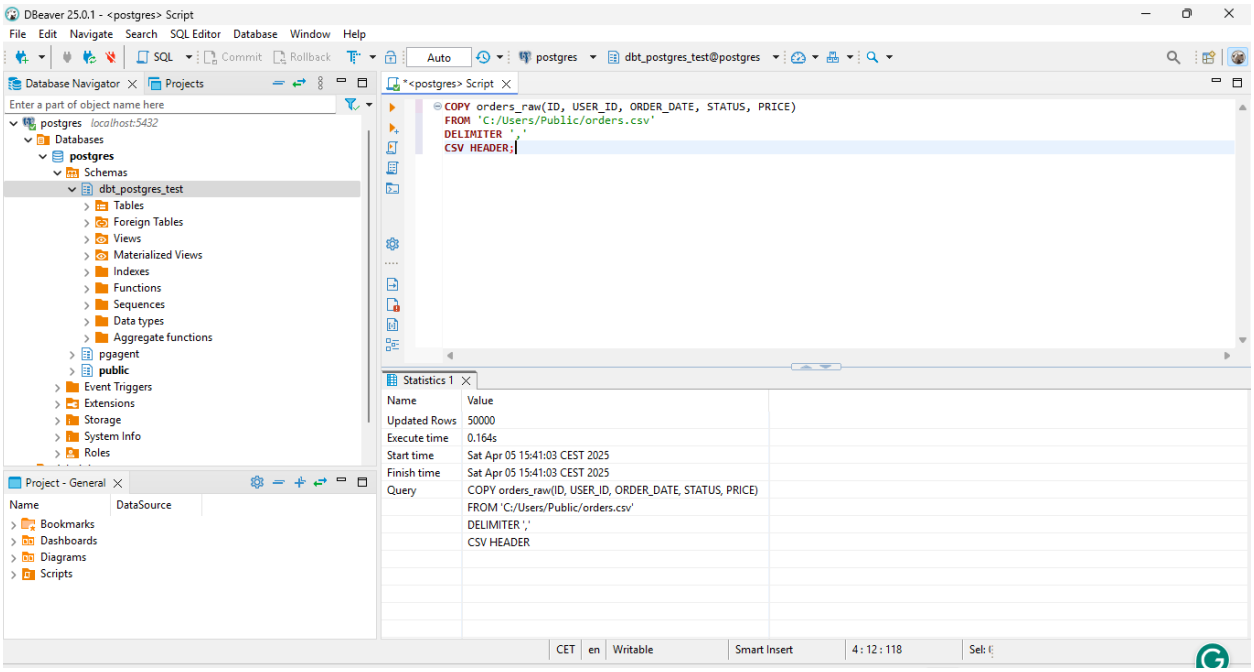


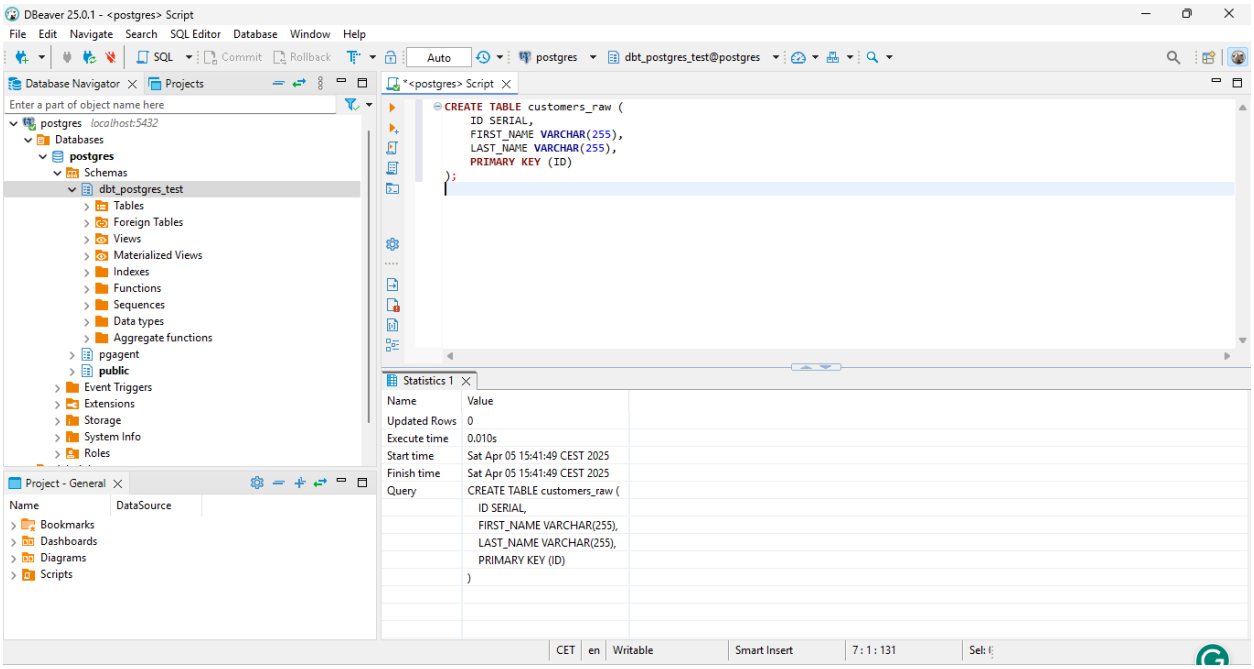
In Postgres, add a new schema

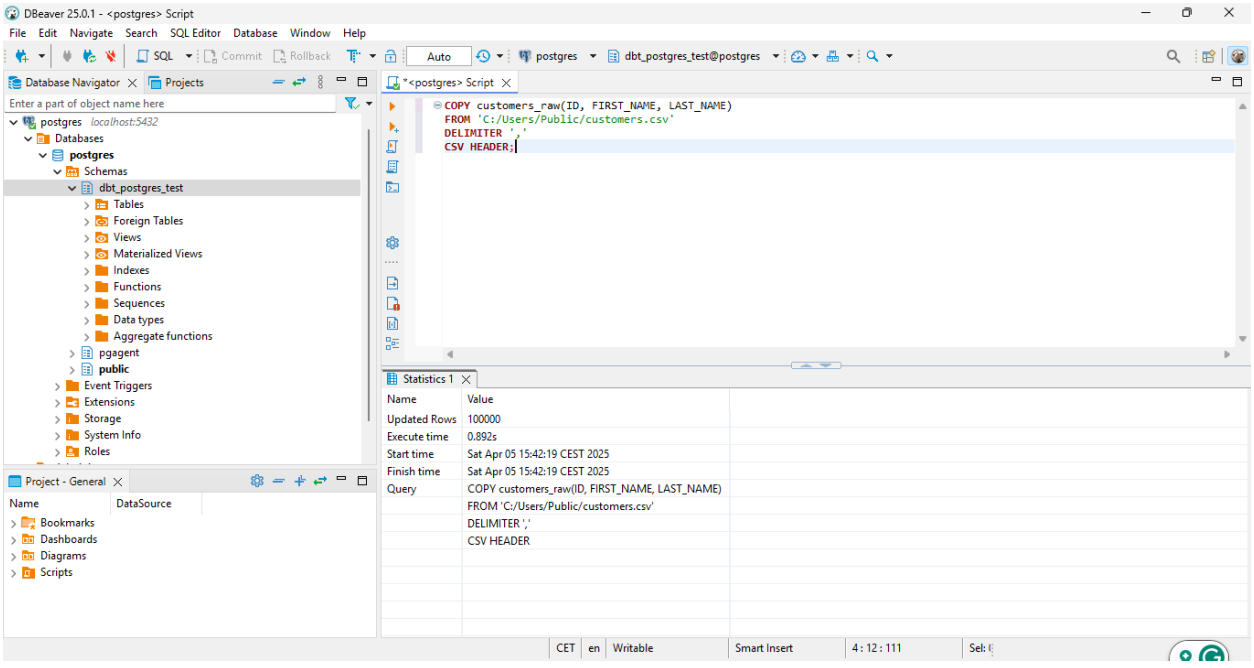


Episode 3: Load raw data in your DataWareHouse


DBeaver 25.0.1 - <postgres> Script
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
Enter a part of object name here
postgres localhost:5432
Databases
postgres
Schemas
dbt_postgres_test
Tables
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
pgagent
public
Event Triggers
Extensions
Storage
System Info
Roles
Project - General
Name
DataSource
Bookmarks
Dashboards
Diagrams
Scripts
*<postgres> Script
-- Create table if not already created
CREATE TABLE orders_raw (
ID SERIAL,
USER_ID INTEGER,
ORDER_DATE DATE,
STATUS VARCHAR(255),
PRICE INTEGER,
PRIMARY KEY (ID)
);
Statistics 1
Name
Value
Updated Rows
0
Execute time
0.043s
Start time
Sat Apr 05 15:40:11 CEST 2025
Finish time
Sat Apr 05 15:40:11 CEST 2025
Query
-- Create table if not already created
CREATE TABLE orders_raw (
ID SERIAL,
USER_ID INTEGER,
ORDER_DATE DATE,
STATUS VARCHAR(255),
PRICE INTEGER,
PRIMARY KEY (ID)
)
CET
en
Writable
Smart Insert
10 : 1 : 199

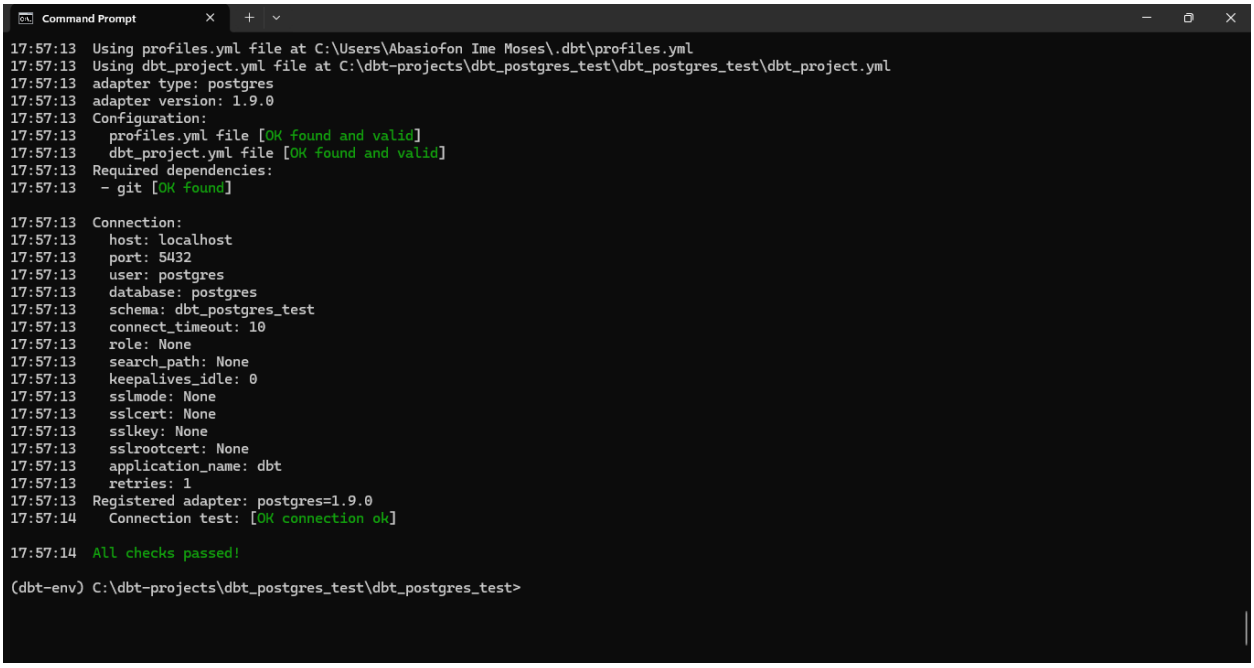

DBeaver 25.0.1 - <postgres> Script
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
Enter a part of object name here
postgres localhost:5432
Databases
postgres
Schemas
dbt_postgres_test
Tables
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
pgagent
public
Event Triggers
Extensions
Storage
System Info
Roles
Project - General
Name
DataSource
Bookmarks
Dashboards
Diagrams
Scripts
*<postgres> Script
COPY orders_raw(ID, USER_ID, ORDER_DATE, STATUS, PRICE)
FROM 'C:/Users/Public/orders.csv'
DELIMITER ','
CSV HEADER;
Statistics 1
Name
Value
Updated Rows
50000
Execute time
0.164s
Start time
Sat Apr 05 15:41:03 CEST 2025
Finish time
Sat Apr 05 15:41:03 CEST 2025
Query
COPY orders_raw(ID, USER_ID, ORDER_DATE, STATUS, PRICE)
FROM 'C:/Users/Public/orders.csv'
DELIMITER ','
CSV HEADER
CET
en
Writable
Smart Insert
4 : 12 : 118



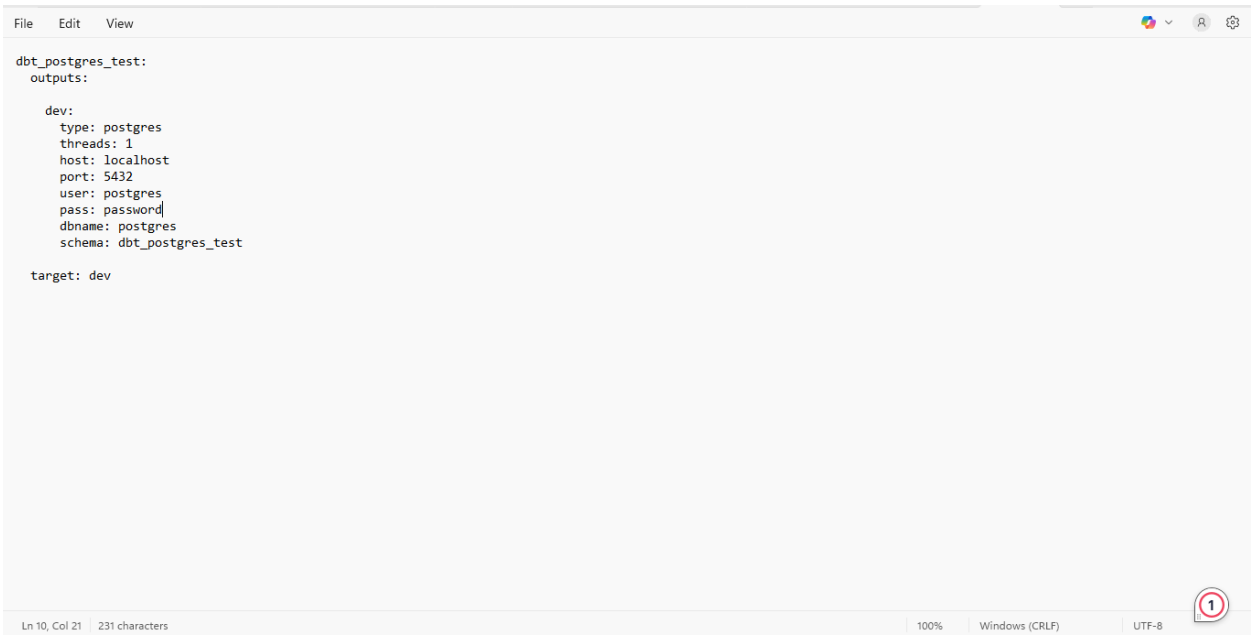


Episode 4: Dbt Init

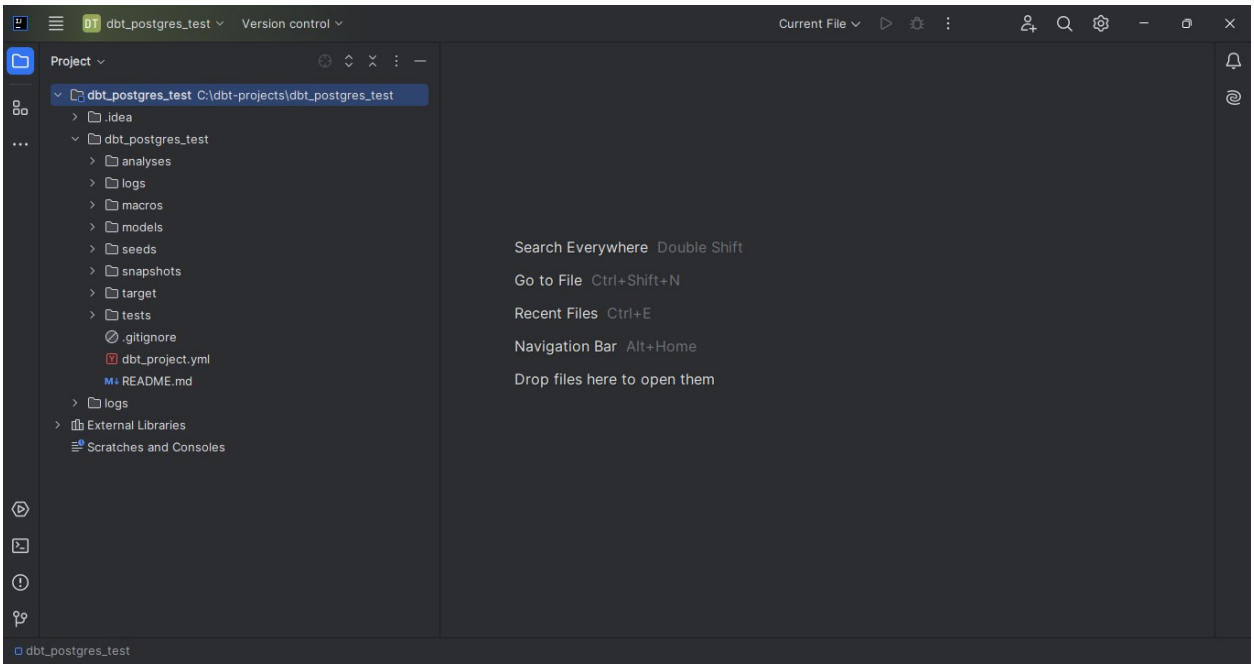
Init your project



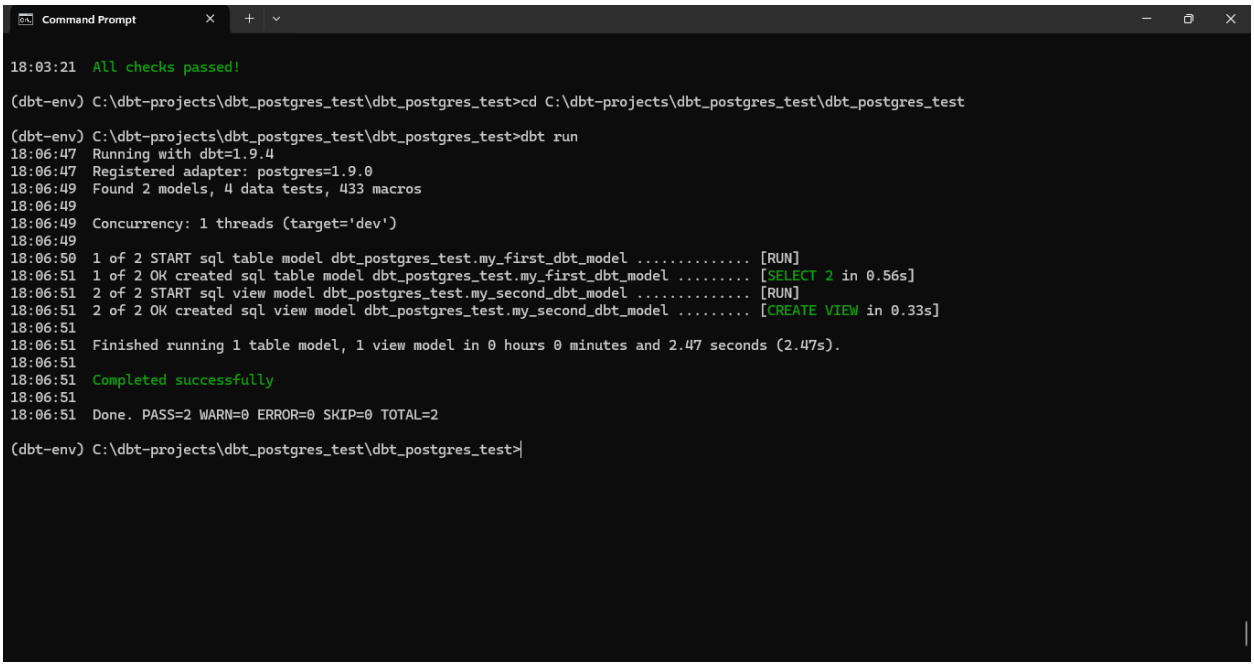
Configure your DBT profile



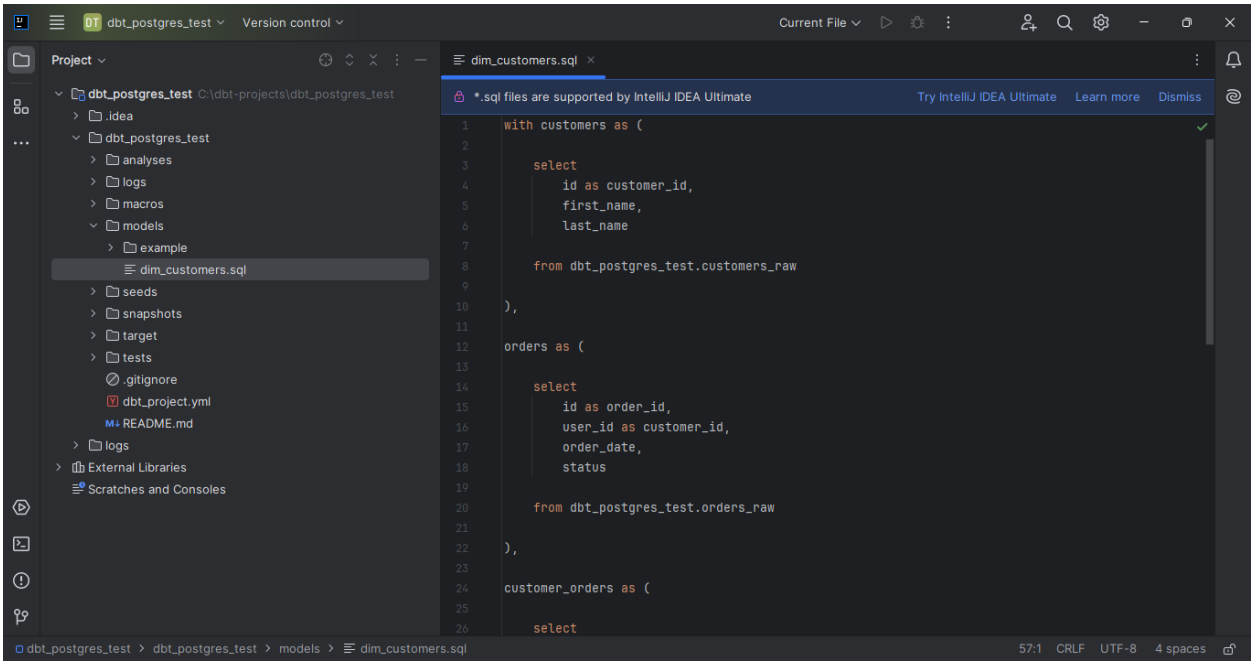
Open in a IDE

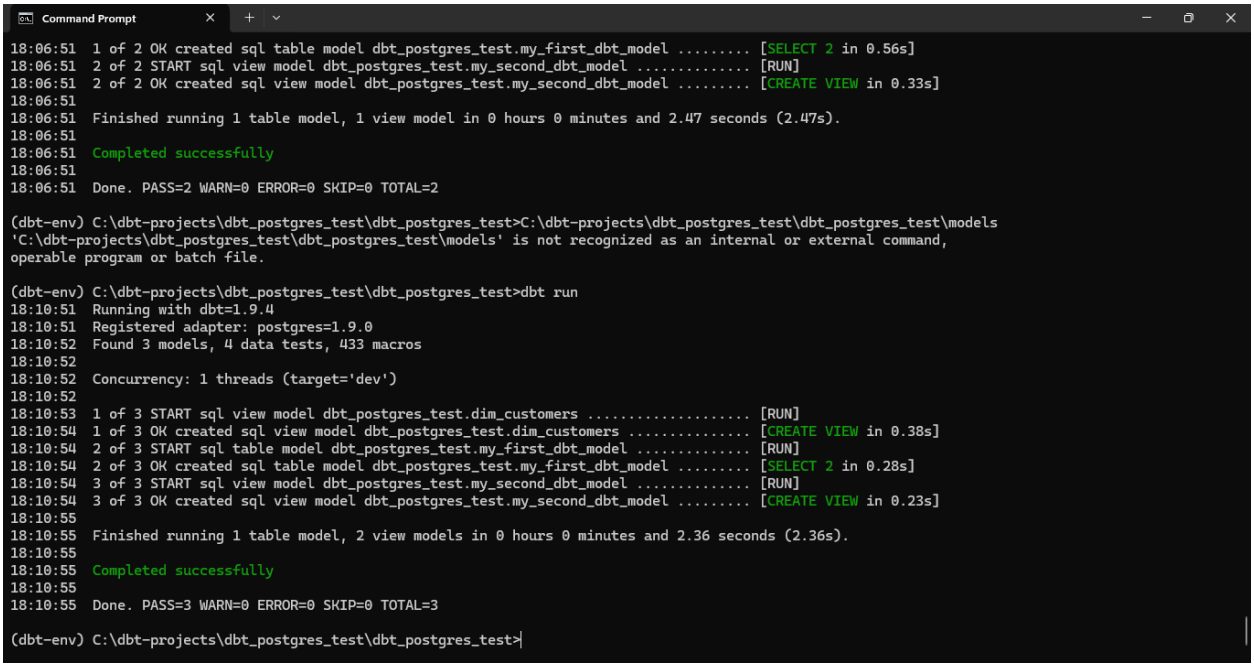


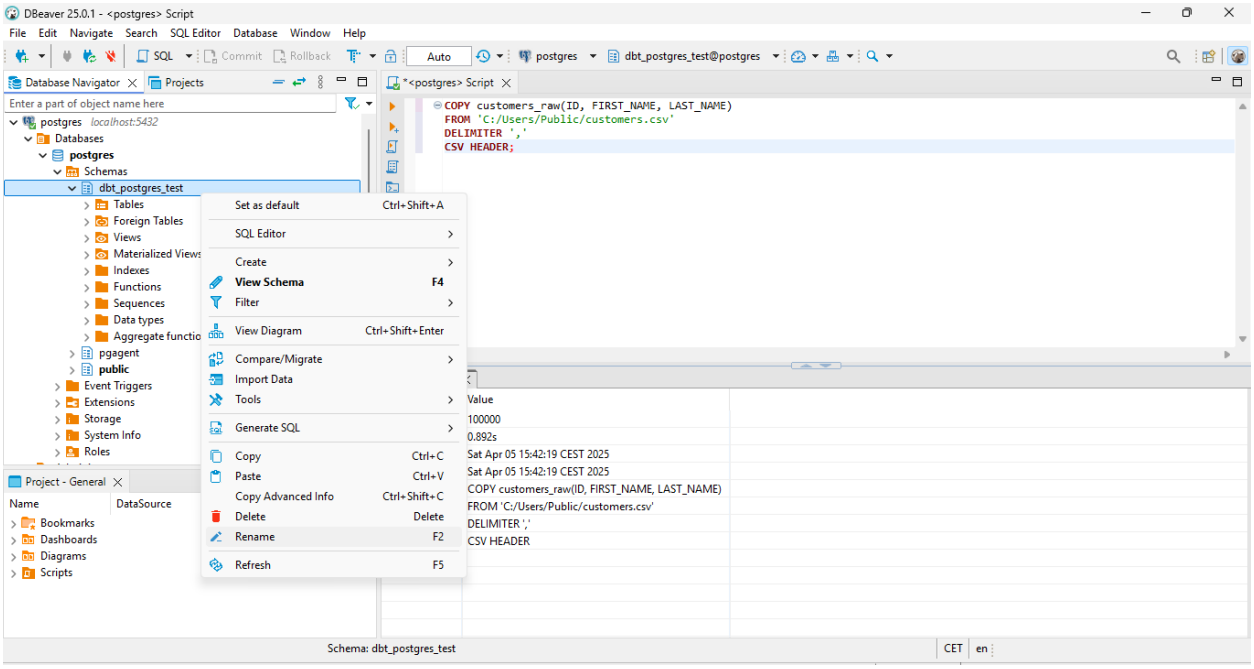
1st Launch

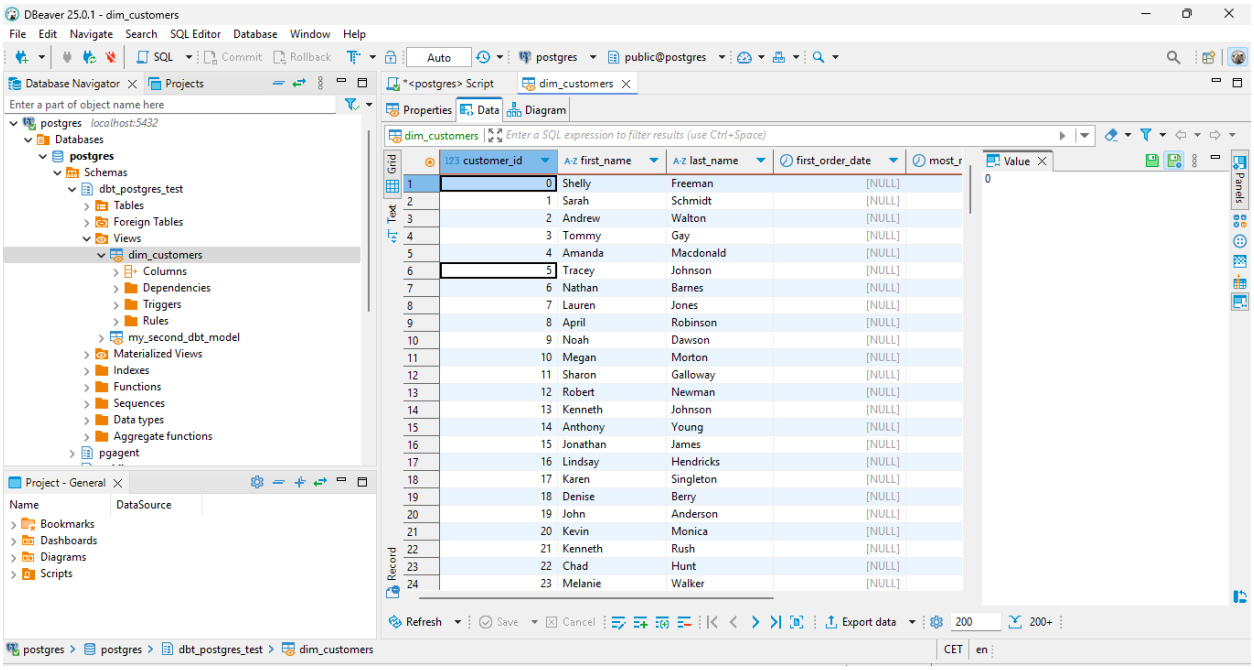


Episode 5: Create Models

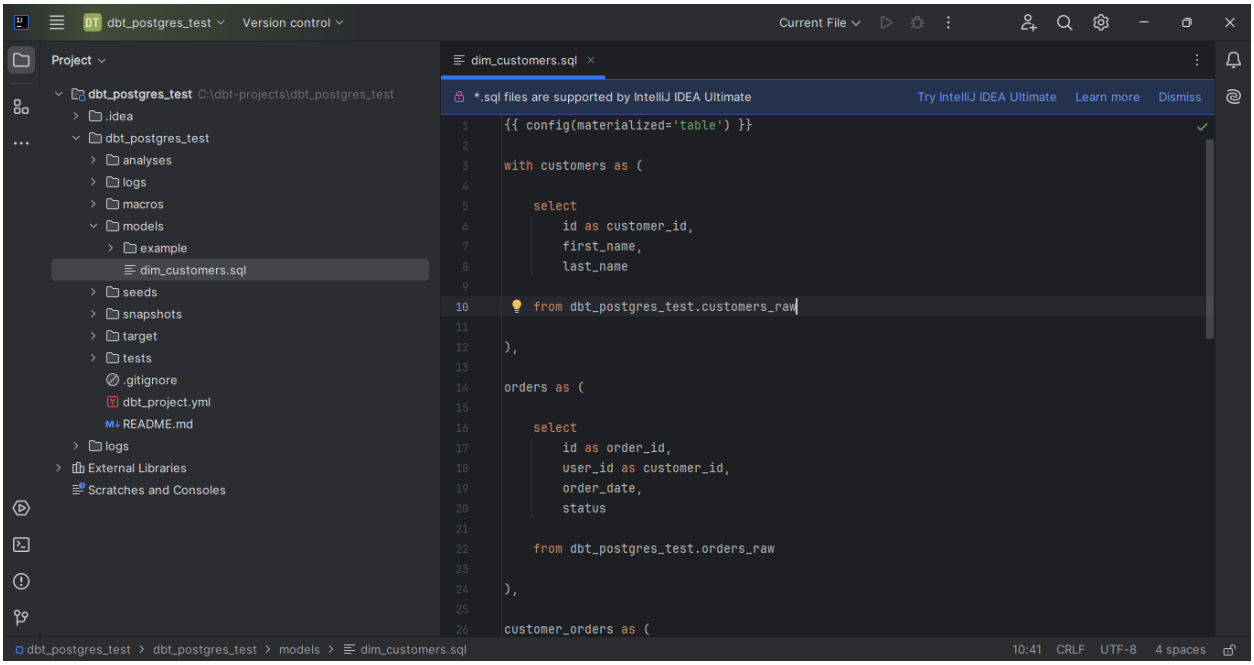


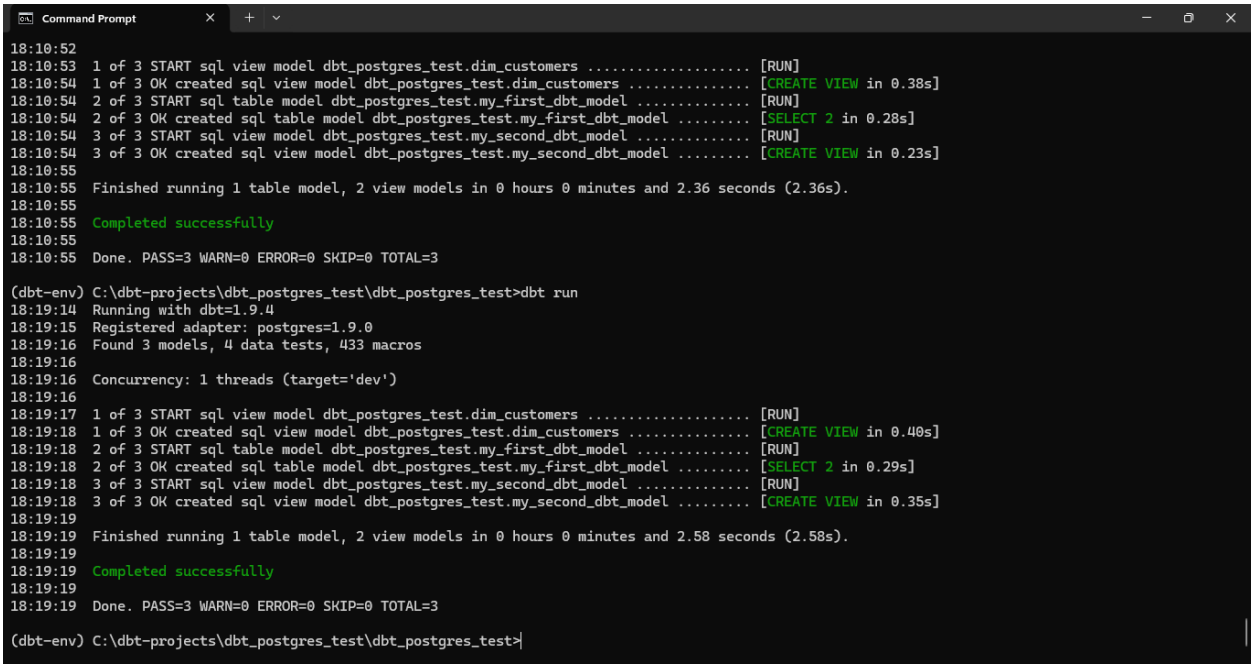


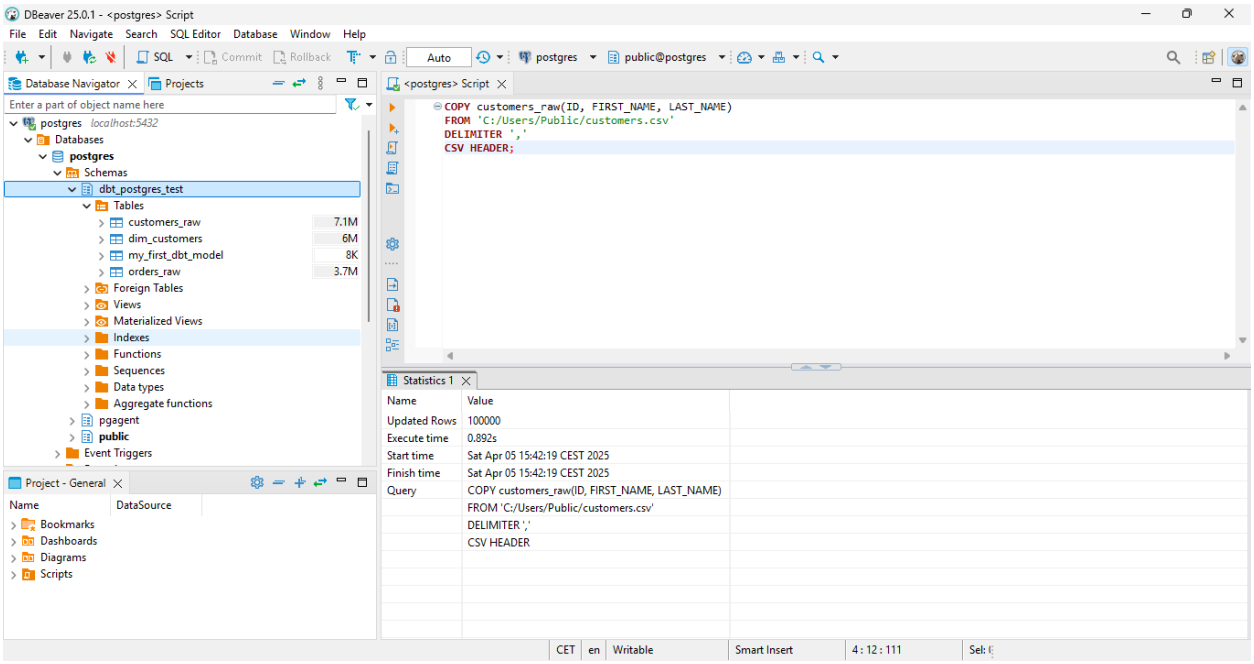




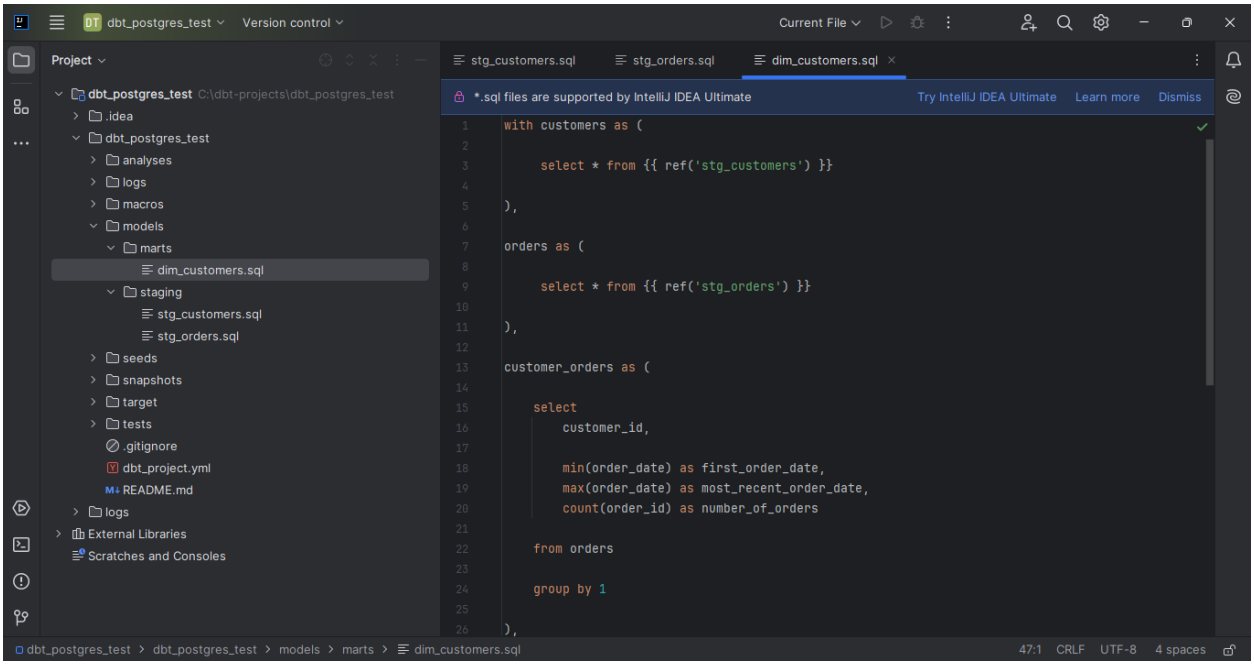
Materialize final Table

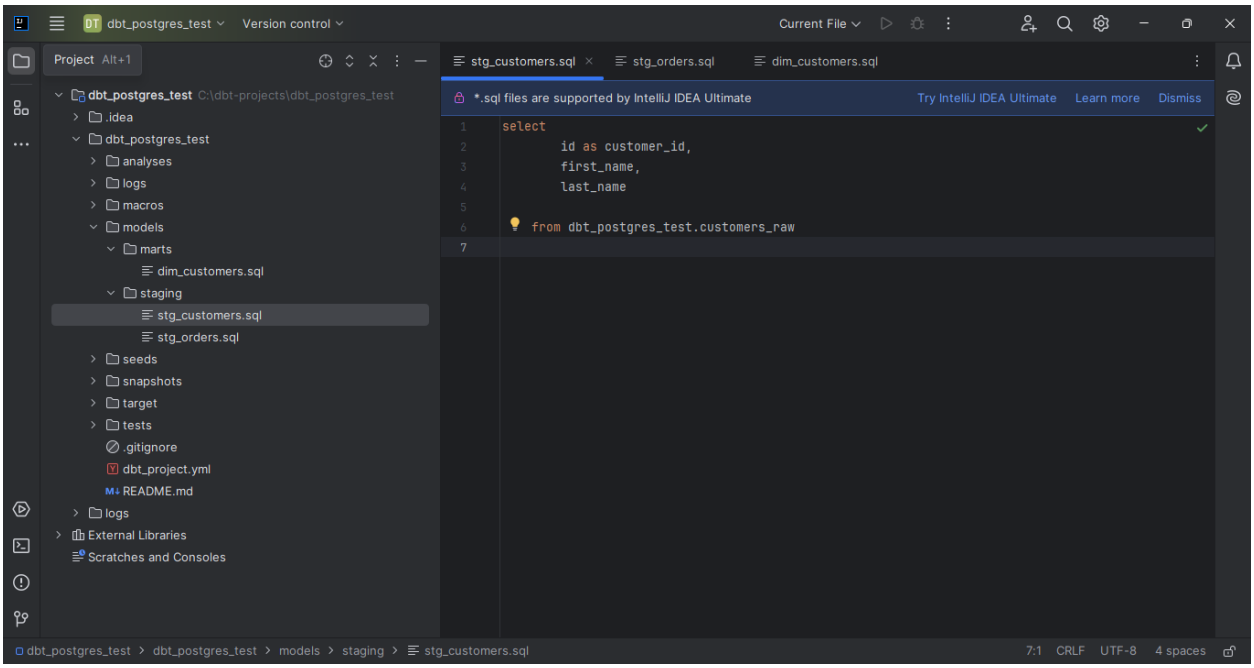

dbt_postgres_test
Version control
Current File
Project
dbt_postgres_test C:\dbt-projects\dbt_postgres_test
.idea
dbt_postgres_test
analyses
logs
macros
models
example
dim_customers.sql
seeds
snapshots
target
tests
.gitignore
dbt_project.yml
README.md
logs
External Libraries
Scratches and Consoles
dim_customers.sql
*.sql files are supported by IntelliJ IDEA Ultimate
Try IntelliJ IDEA Ultimate
Learn more
Dismiss
{{ config(materialized='table') }}
with customers as (
select
id as customer_id,
first_name,
last_name
from dbt_postgres_test.customers_raw
),
orders as (
select
id as order_id,
user_id as customer_id,
order_date,
status
from dbt_postgres_test.orders_raw
),
customer_orders as (
dbt_postgres_test > dbt_postgres_test > models > dim_customers.sql
10:41 CRLF UTF-8 4 spaces

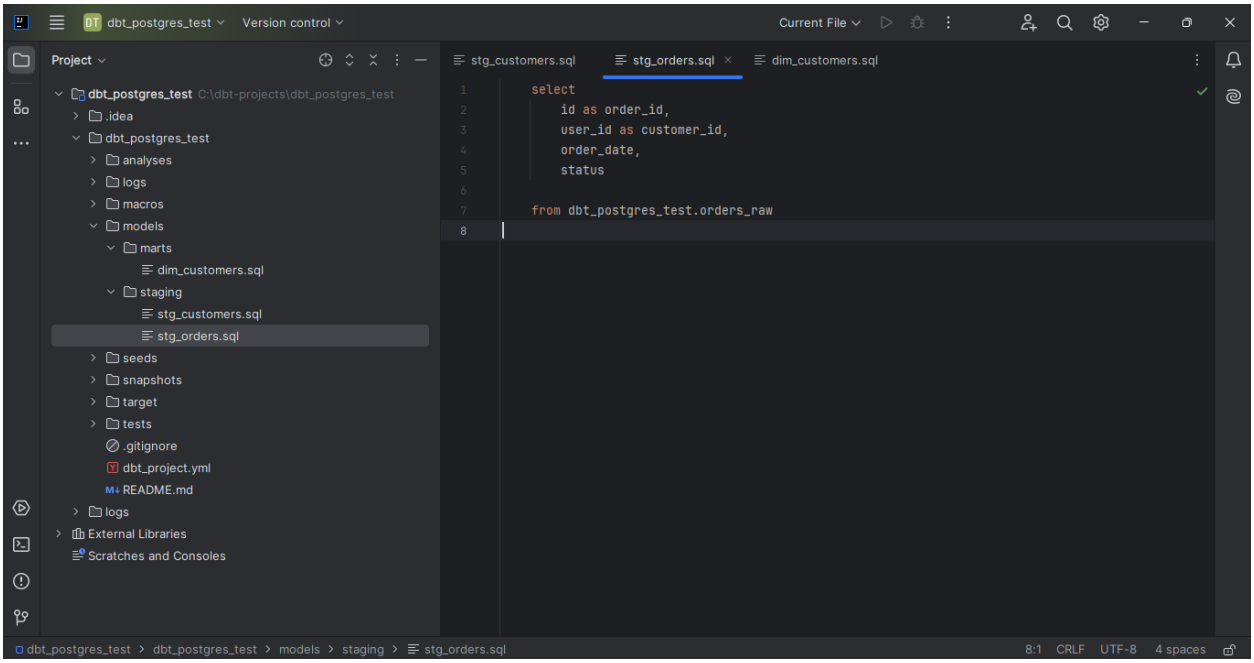

Command Prompt
18:10:52
18:10:53 1 of 3 START sql view model dbt_postgres_test.dim_customers ..... [RUN]
18:10:54 1 of 3 OK created sql view model dbt_postgres_test.dim_customers ..... [CREATE VIEW in 0.38s]
18:10:54 2 of 3 START sql table model dbt_postgres_test.my_first_dbt_model ..... [RUN]
18:10:54 2 of 3 OK created sql table model dbt_postgres_test.my_first_dbt_model ..... [SELECT 2 in 0.28s]
18:10:54 3 of 3 START sql view model dbt_postgres_test.my_second_dbt_model ..... [RUN]
18:10:54 3 of 3 OK created sql view model dbt_postgres_test.my_second_dbt_model ..... [CREATE VIEW in 0.23s]
18:10:55
18:10:55 Finished running 1 table model, 2 view models in 0 hours 0 minutes and 2.36 seconds (2.36s).
18:10:55
18:10:55 Completed successfully
18:10:55
18:10:55 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt run
18:19:14 Running with dbt=1.9.4
18:19:15 Registered adapter: postgres=1.9.0
18:19:16 Found 3 models, 4 data tests, 433 macros
18:19:16
18:19:16 Concurrency: 1 threads (target='dev')
18:19:16
18:19:17 1 of 3 START sql view model dbt_postgres_test.dim_customers ..... [RUN]
18:19:18 1 of 3 OK created sql view model dbt_postgres_test.dim_customers ..... [CREATE VIEW in 0.40s]
18:19:18 2 of 3 START sql table model dbt_postgres_test.my_first_dbt_model ..... [RUN]
18:19:18 2 of 3 OK created sql table model dbt_postgres_test.my_first_dbt_model ..... [SELECT 2 in 0.29s]
18:19:18 3 of 3 START sql view model dbt_postgres_test.my_second_dbt_model ..... [RUN]
18:19:18 3 of 3 OK created sql view model dbt_postgres_test.my_second_dbt_model ..... [CREATE VIEW in 0.35s]
18:19:19
18:19:19 Finished running 1 table model, 2 view models in 0 hours 0 minutes and 2.58 seconds (2.58s).
18:19:19
18:19:19 Completed successfully
18:19:19
18:19:19 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>



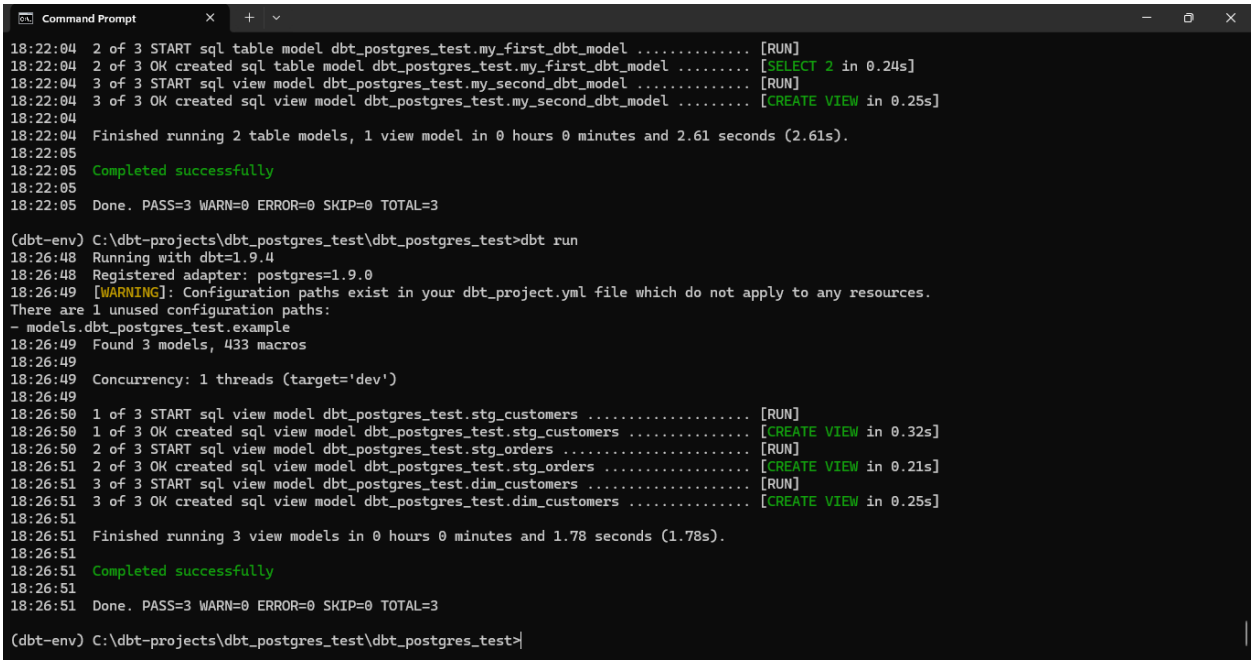
Reorganize models



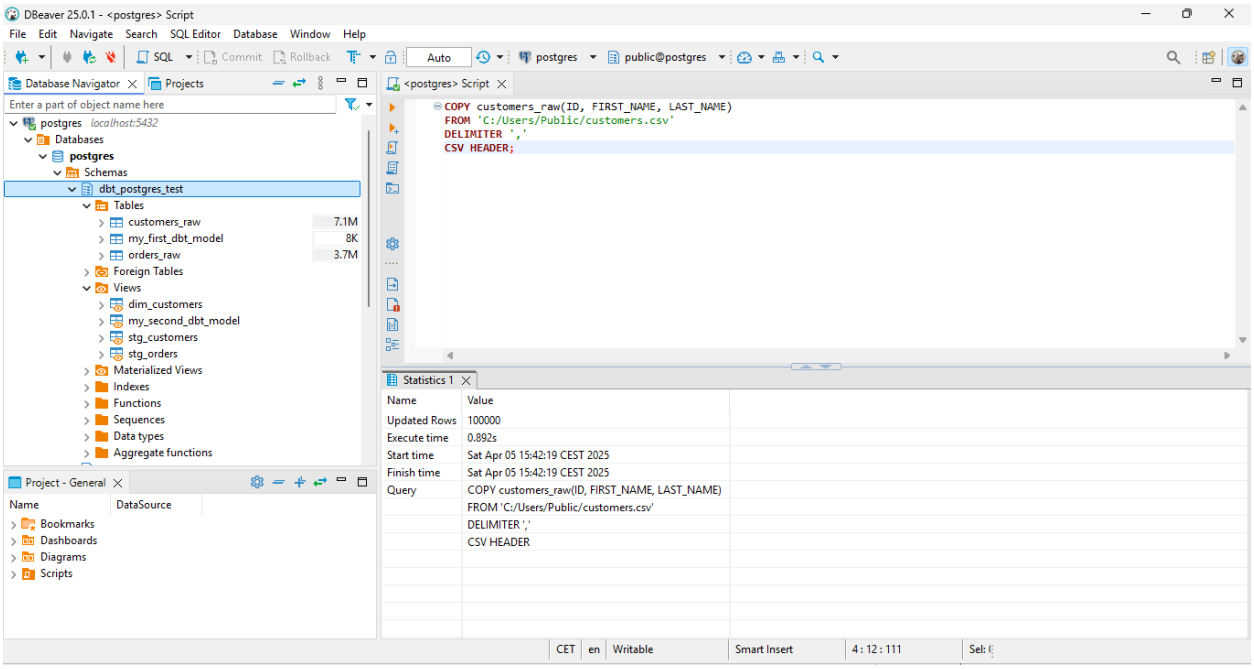

dbt_postgres_test
Version control
Current File
Project Alt+1
stg_customers.sql
stg_orders.sql
dim_customers.sql
*.sql files are supported by IntelliJ IDEA Ultimate
Try IntelliJ IDEA Ultimate
Learn more
Dismiss
dbt_postgres_test C:\dbt-projects\dbt_postgres_test
.idea
dbt_postgres_test
analyses
logs
macros
models
marts
dim_customers.sql
staging
stg_customers.sql
stg_orders.sql
seeds
snapshots
target
tests
.gitignore
dbt_project.yml
README.md
logs
External Libraries
Scratches and Consoles
select
id as customer_id,
first_name,
last_name
from dbt_postgres_test.customers_raw
dbt_postgres_test › dbt_postgres_test › models › staging › stg_customers.sql
7:1 CRLF UTF-8 4 spaces


dbt_postgres_test
Version control
Current File
Project
stg_customers.sql
stg_orders.sql
dim_customers.sql
dbt_postgres_test C:\dbt-projects\dbt_postgres_test
.idea
dbt_postgres_test
analyses
logs
macros
models
marts
dim_customers.sql
staging
stg_customers.sql
stg_orders.sql
seeds
snapshots
target
tests
.gitignore
dbt_project.yml
README.md
logs
External Libraries
Scratches and Consoles
select
id as order_id,
user_id as customer_id,
order_date,
status
from dbt_postgres_test.orders_raw
dbt_postgres_test › dbt_postgres_test › models › staging › stg_orders.sql
8:1 CRLF UTF-8 4 spaces

Re-run dbt:

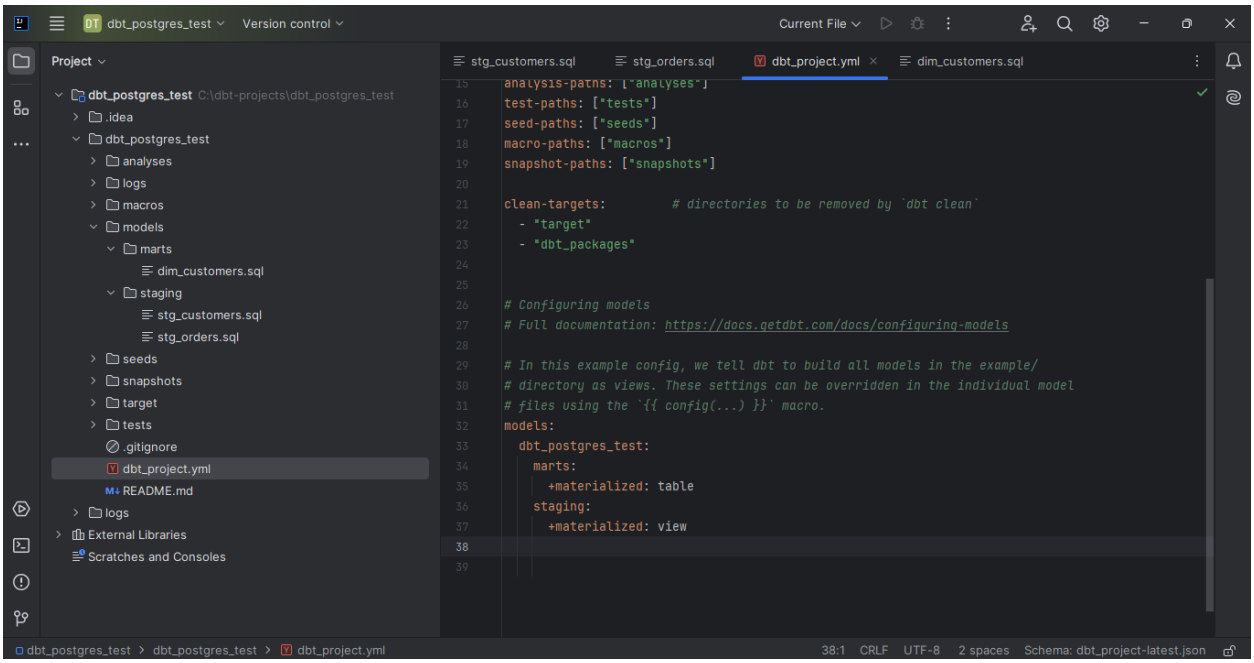


It has created 3 views, which is the default with DBT.



A nice thing with DBT, is to define default materialization (view or table) by folder.

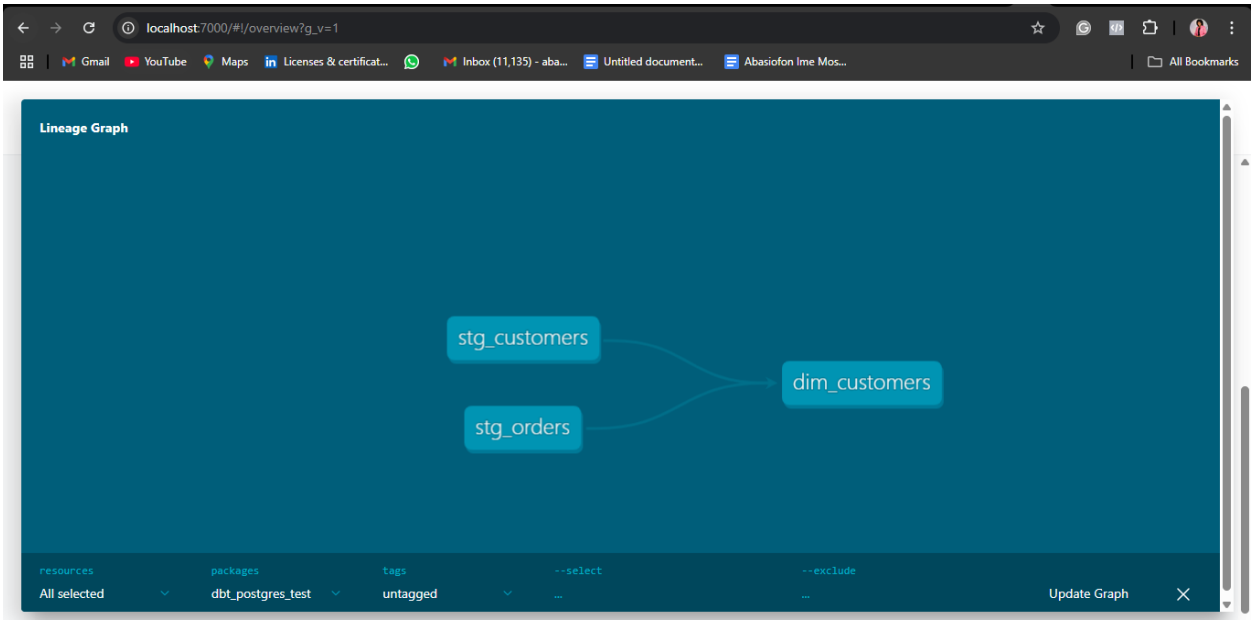
Define global default materialization per folder



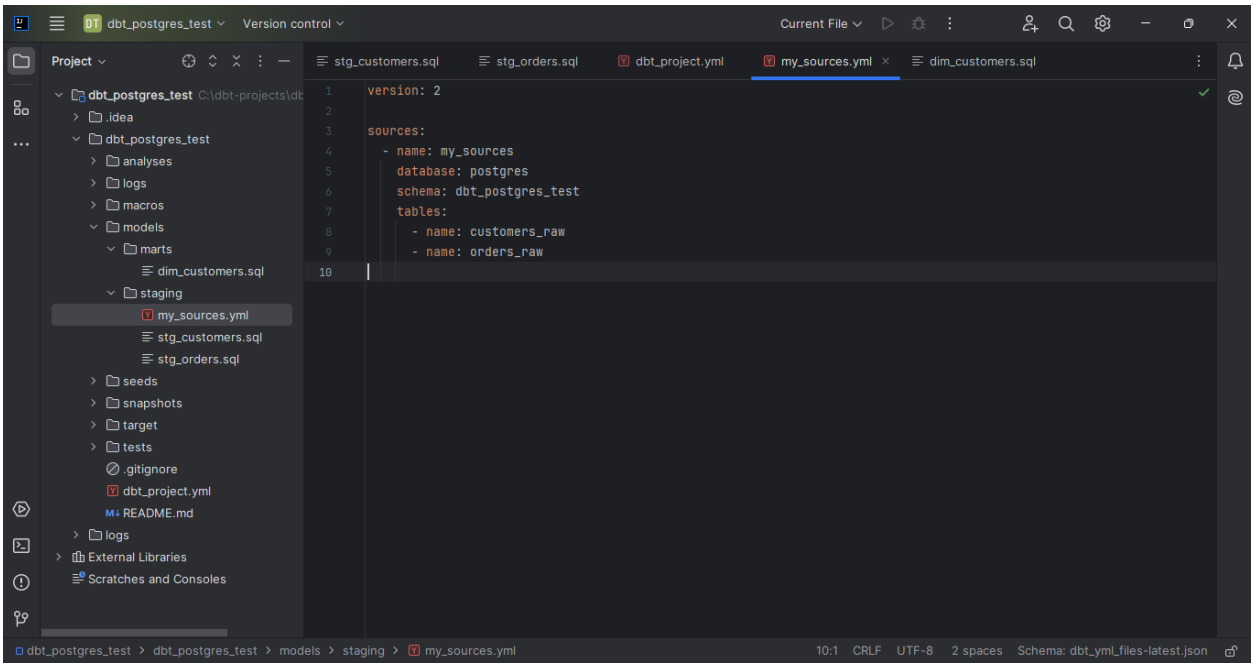
Re-run Dbt and see in output that indeed the datamart has been created as table and not a view:

```
18:30:00  Found 3 models, 433 macros
18:30:00
18:30:00  Concurrency: 1 threads (target='dev')
18:30:00
18:30:01  1 of 3 START sql view model dbt_postgres_test.stg_customers ..... [RUN]
18:30:02  1 of 3 OK created sql view model dbt_postgres_test.stg_customers ..... [CREATE VIEW in 0.30s]
18:30:02  2 of 3 START sql view model dbt_postgres_test.stg_orders ..... [RUN]
18:30:02  2 of 3 OK created sql view model dbt_postgres_test.stg_orders ..... [CREATE VIEW in 0.23s]
18:30:02  3 of 3 START sql table model dbt_postgres_test.dim_customers ..... [RUN]
18:30:02  3 of 3 OK created sql table model dbt_postgres_test.dim_customers ..... [SELECT 100000 in 0.47s]
18:30:03
18:30:03  Finished running 1 table model, 2 view models in 0 hours 0 minutes and 2.12 seconds (2.12s).
18:30:03
18:30:03  Completed successfully
18:30:03
18:30:03  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>
```

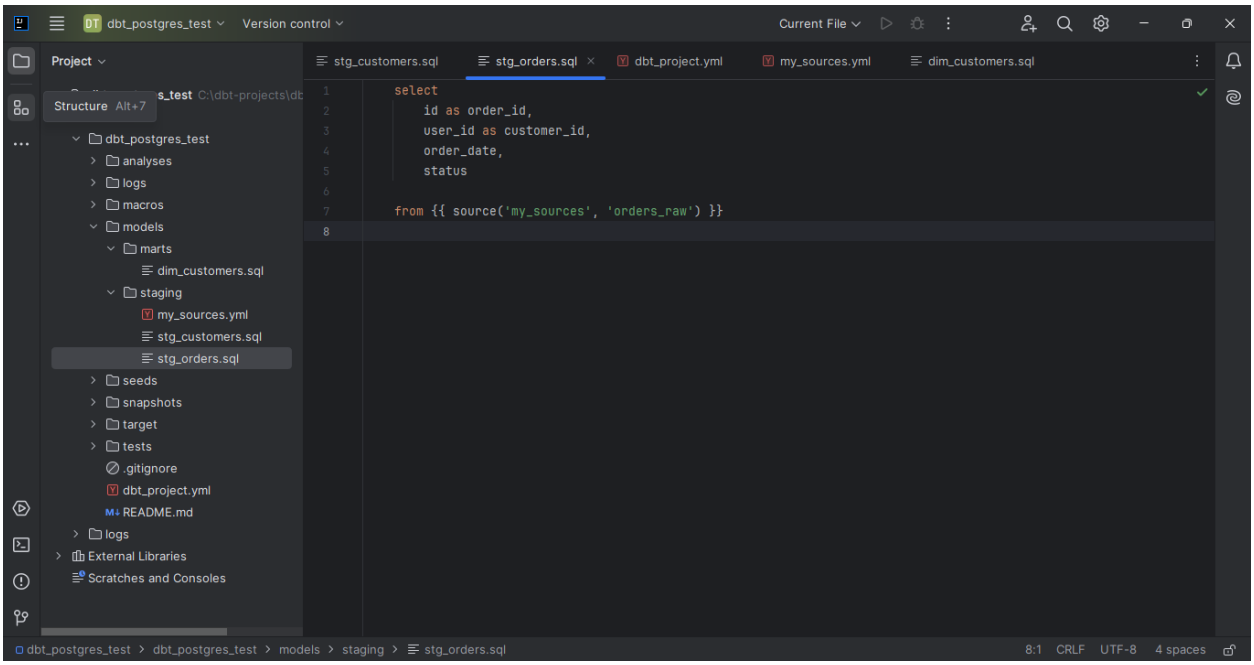
Check lineage:

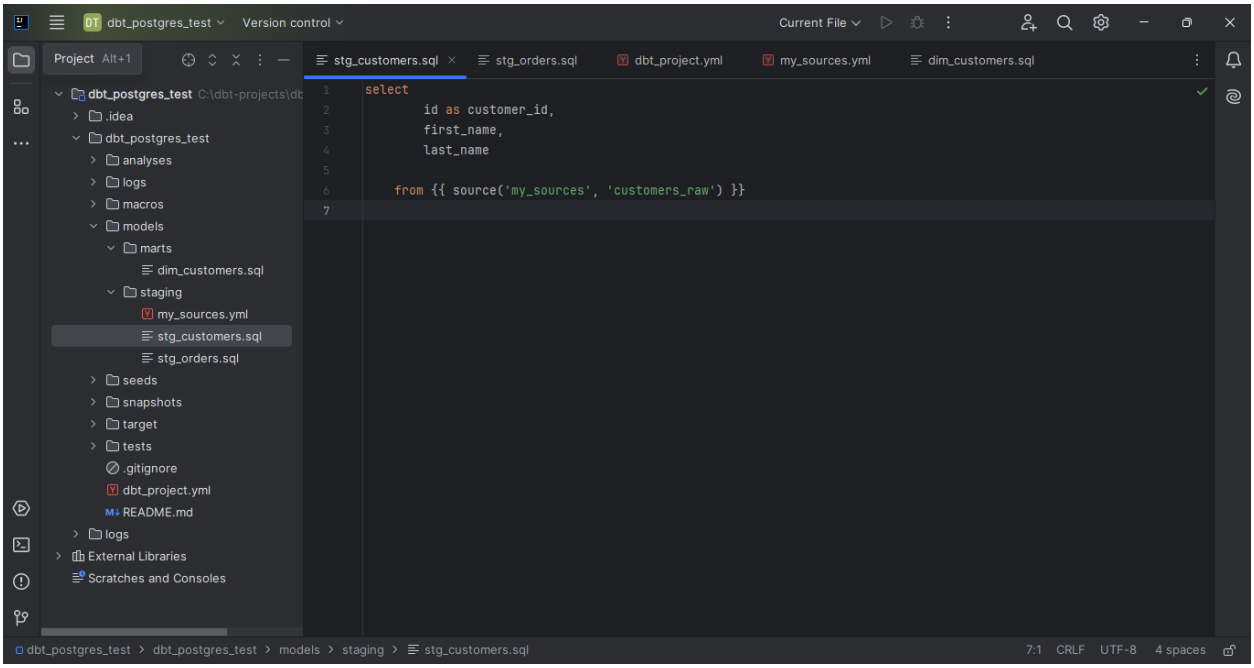


Episode 6: Sources



Then you can reference them in the staging files like this:

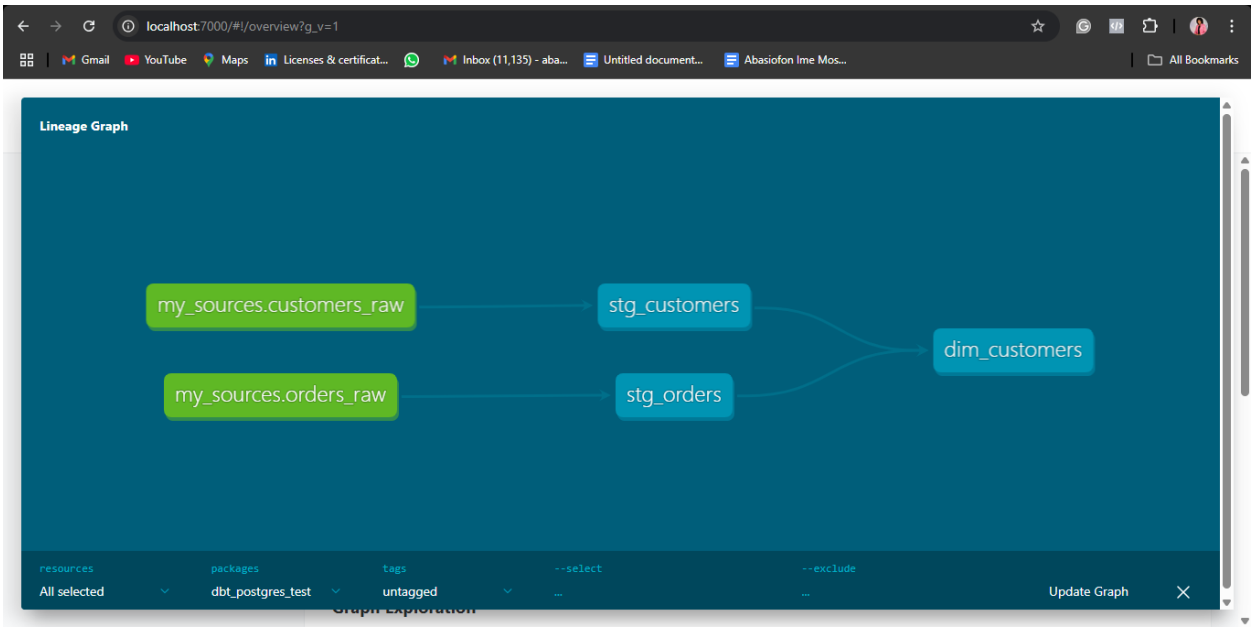

dbt_postgres_test
Version control
Current File
Project
Structure Alt+7
stg_customers.sql
stg_orders.sql
dbt_project.yml
my_sources.yml
dim_customers.sql
select
id as order_id,
user_id as customer_id,
order_date,
status
from {{ source('my_sources', 'orders_raw') }}
dbt_postgres_test
analyses
logs
macros
models
marts
dim_customers.sql
staging
my_sources.yml
stg_customers.sql
stg_orders.sql
seeds
snapshots
target
tests
.gitignore
dbt_project.yml
README.md
logs
External Libraries
Scratches and Consoles
dbt_postgres_test > dbt_postgres_test > models > staging > stg_orders.sql
8:1 CRLF UTF-8 4 spaces


dbt_postgres_test
Version control
Current File
Project Alt+1
stg_customers.sql
stg_orders.sql
dbt_project.yml
my_sources.yml
dim_customers.sql
select
id as customer_id,
first_name,
last_name
from {{ source('my_sources', 'customers_raw') }}
dbt_postgres_test
.idea
dbt_postgres_test
analyses
logs
macros
models
marts
dim_customers.sql
staging
my_sources.yml
stg_customers.sql
stg_orders.sql
seeds
snapshots
target
tests
.gitignore
dbt_project.yml
README.md
logs
External Libraries
Scratches and Consoles
dbt_postgres_test > dbt_postgres_test > models > staging > stg_customers.sql
7:1 CRLF UTF-8 4 spaces

Re run:

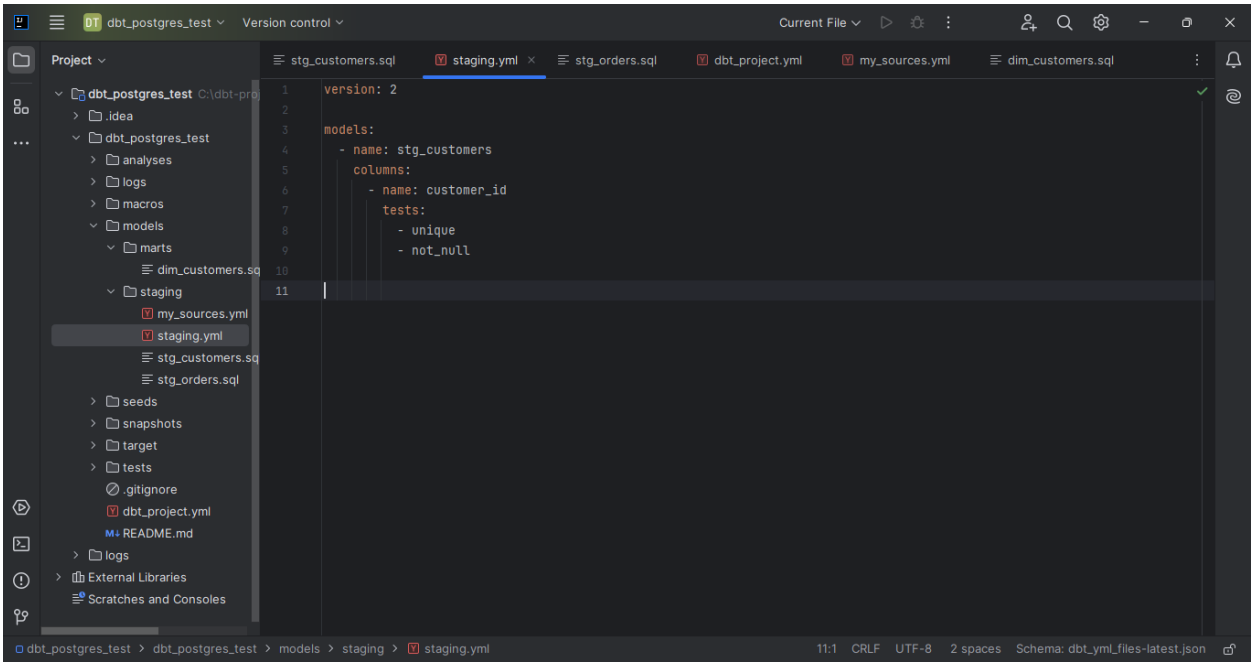
```
    httpd.serve_forever()
    ~~~~~~~~~~~~~~~~~~~~^^
  File "C:\Users\Abasiofon Ime Moses\AppData\Local\Programs\Python\Python313\Lib\socketserver.py", line 235, in serve_forever
    ready = selector.select(poll_interval)
  File "C:\Users\Abasiofon Ime Moses\AppData\Local\Programs\Python\Python313\Lib\selectors.py", line 314, in select
    r, w, _ = self._select(self._readers, self._writers, [], timeout)
              ~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Abasiofon Ime Moses\AppData\Local\Programs\Python\Python313\Lib\selectors.py", line 305, in _select
    r, w, x = select.select(r, w, w, timeout)
              ~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
KeyboardInterrupt


(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt docs generate
18:37:03  Running with dbt=1.9.4
18:37:03  Registered adapter: postgres=1.9.0
18:37:04  Found 3 models, 2 sources, 433 macros
18:37:04
18:37:04  Concurrency: 1 threads (target='dev')
18:37:04
18:37:05  Building catalog
18:37:05  Catalog written to C:\dbt-projects\dbt_postgres_test\dbt_postgres_test\target\catalog.json

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt docs serve --port 7000
18:37:26  Running with dbt=1.9.4
Serving docs at 7000
To access from your browser, navigate to: http://localhost:7000



Press Ctrl+C to exit.
127.0.0.1 - - [05/Apr/2025 20:37:27] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Apr/2025 20:37:27] "GET /manifest.json?cb=1743878247464 HTTP/1.1" 200 -
127.0.0.1 - - [05/Apr/2025 20:37:27] "GET /catalog.json?cb=1743878247464 HTTP/1.1" 200 -
```

Lineage has changed by even showing you the sources in the lineage:



Episode 7: Tests

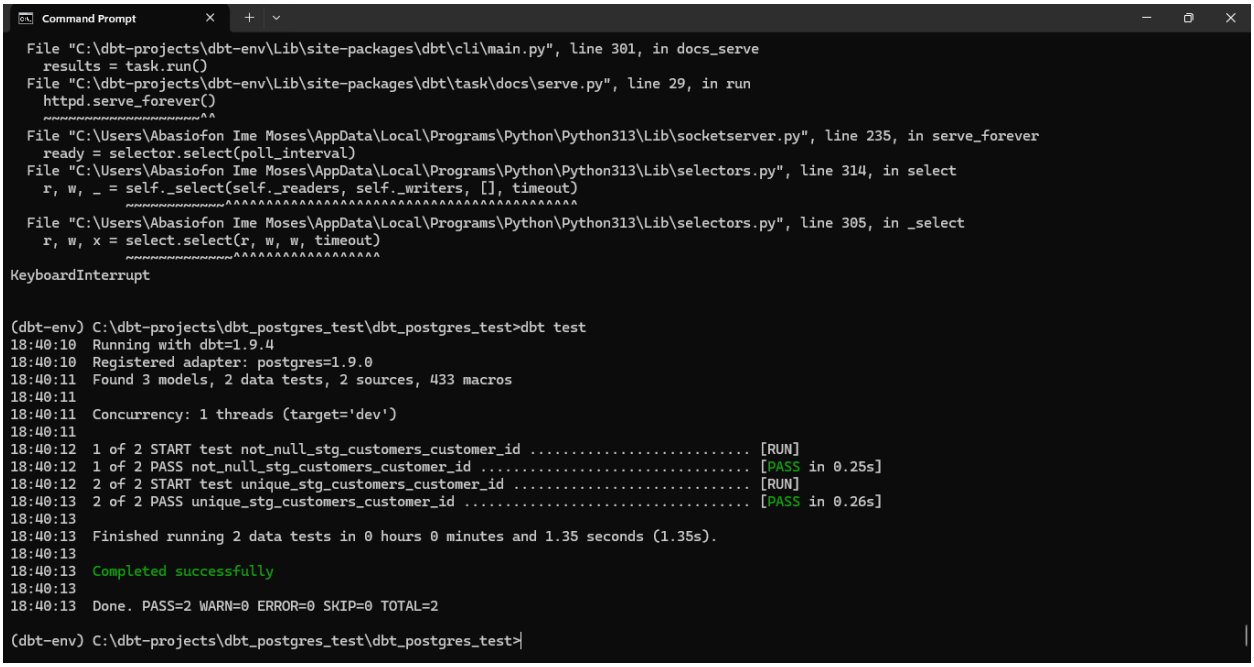
Generic Test



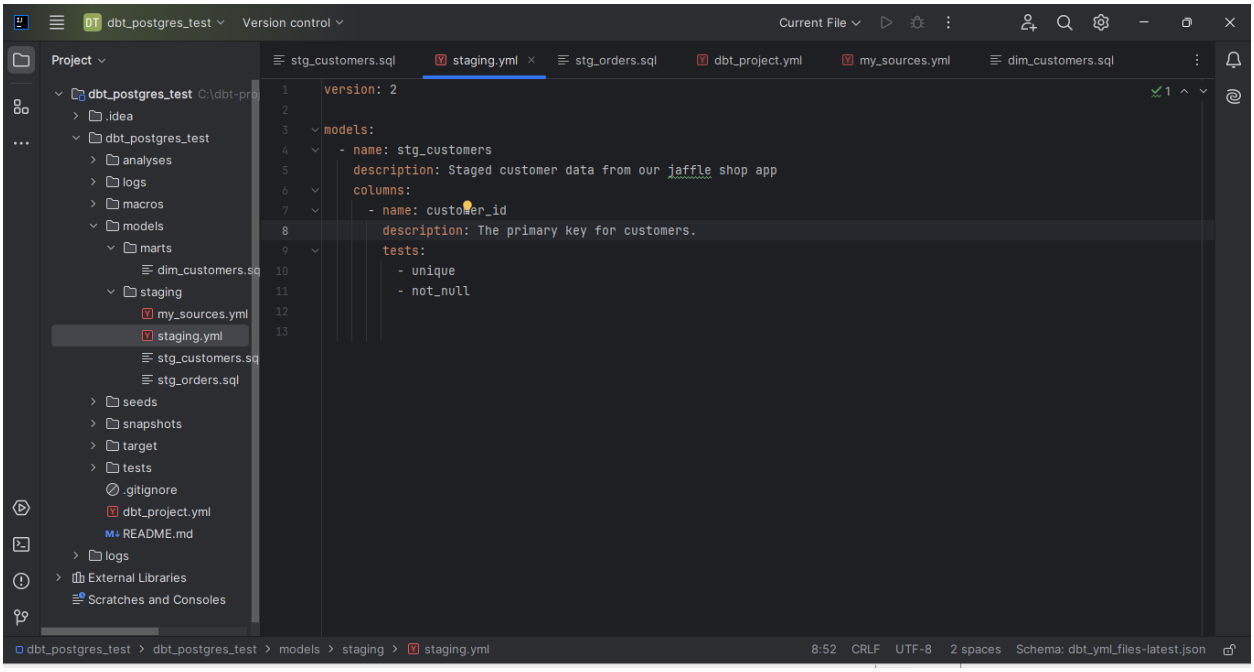
Run

```
dbt test
```

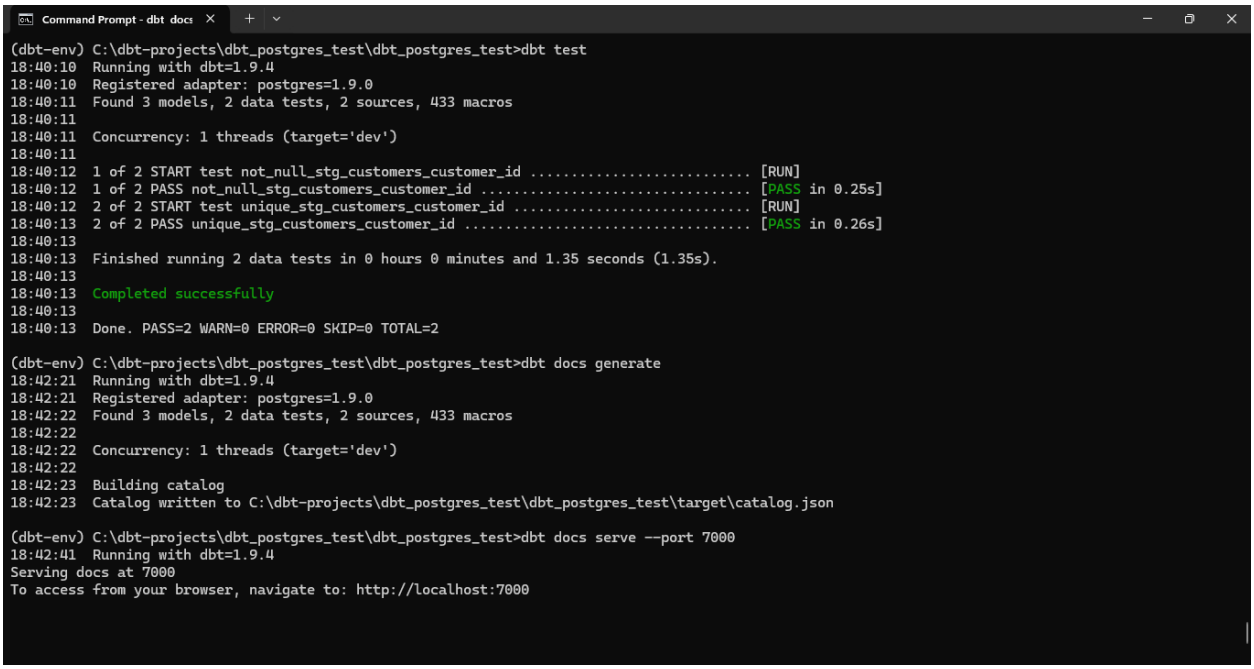
You can see your tests are successful



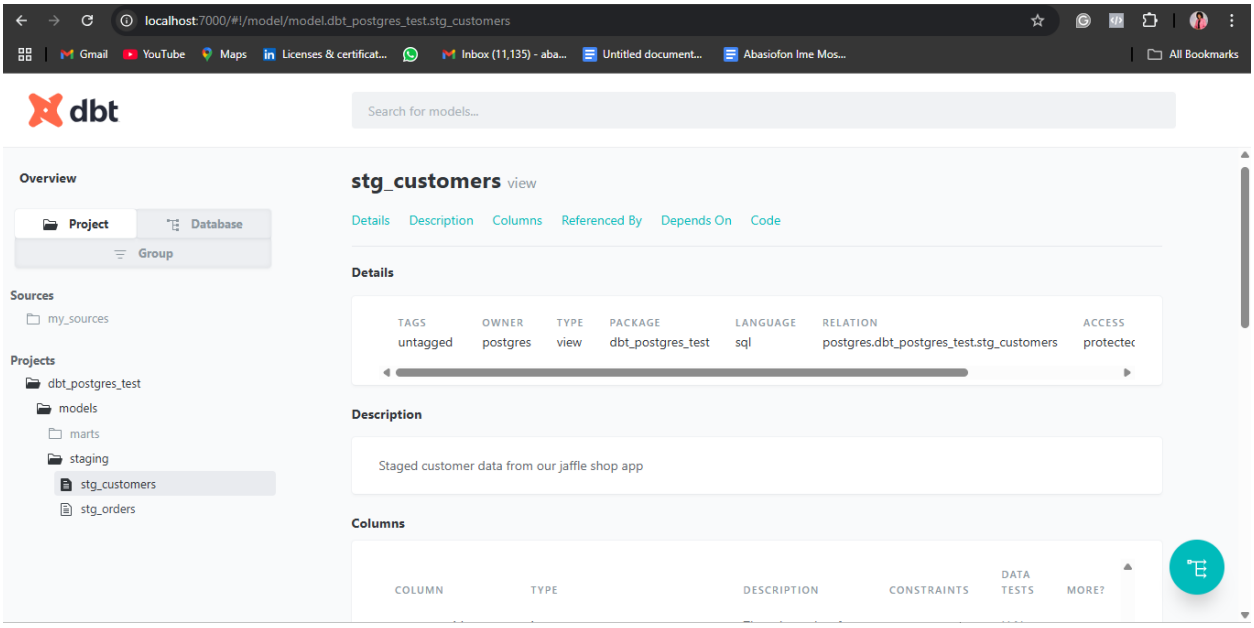
Episode 8: Lineage and Documentation



Run:



You can see the descriptions appearing in the UI:



The codes can be found here: https://github.com/DatagirlX/DBT

https://github.com/DatagirlX/DBTproject