

# 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.

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres@Local server\*

postgres/postgres@Local server

No limit ▾

Query History

Query Scratch Pad

```
1 ✓ CREATE TABLE DEPT
2     (DEPTNO integer constraint pk_dept primary key,
3      DNAME VARCHAR(14),
4      LOC VARCHAR(13) );
5
6
7 ✓ CREATE TABLE EMP
8     (EMPNO integer constraint pk_emp primary key,
9      ENAME VARCHAR(10),
10     EFIRST VARCHAR(10),
11     JOB VARCHAR(9),
12     MGR integer not null,
13     HIREDATE DATE,
14     SAL integer constraint ck_sal check (SAL>=0),
15     COMM integer,
16     TEL char(10),
17     DEPTNO integer,
18     constraint fk_emp_dept foreign key(DEPTNO) references DEPT (DEPTNO));
19
20 ✓ CREATE TABLE DEPENDENTS
```

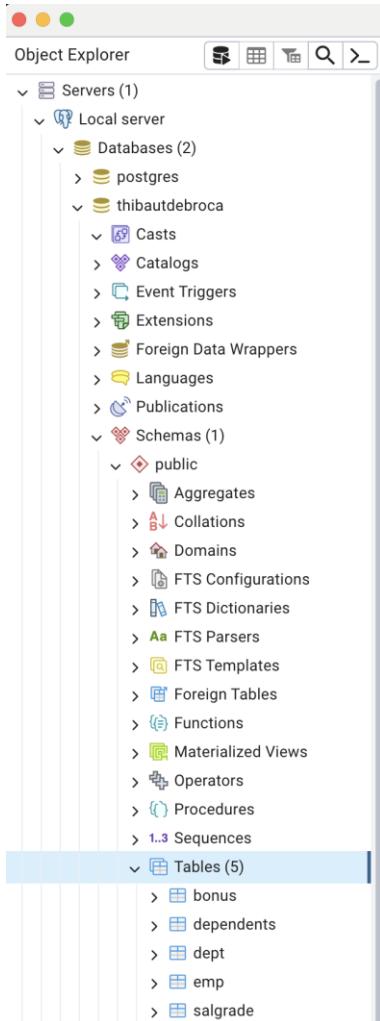
Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 62 msec.

Total rows: Query complete 00:00:00.062 CRLF Ln 38, Col 25

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,

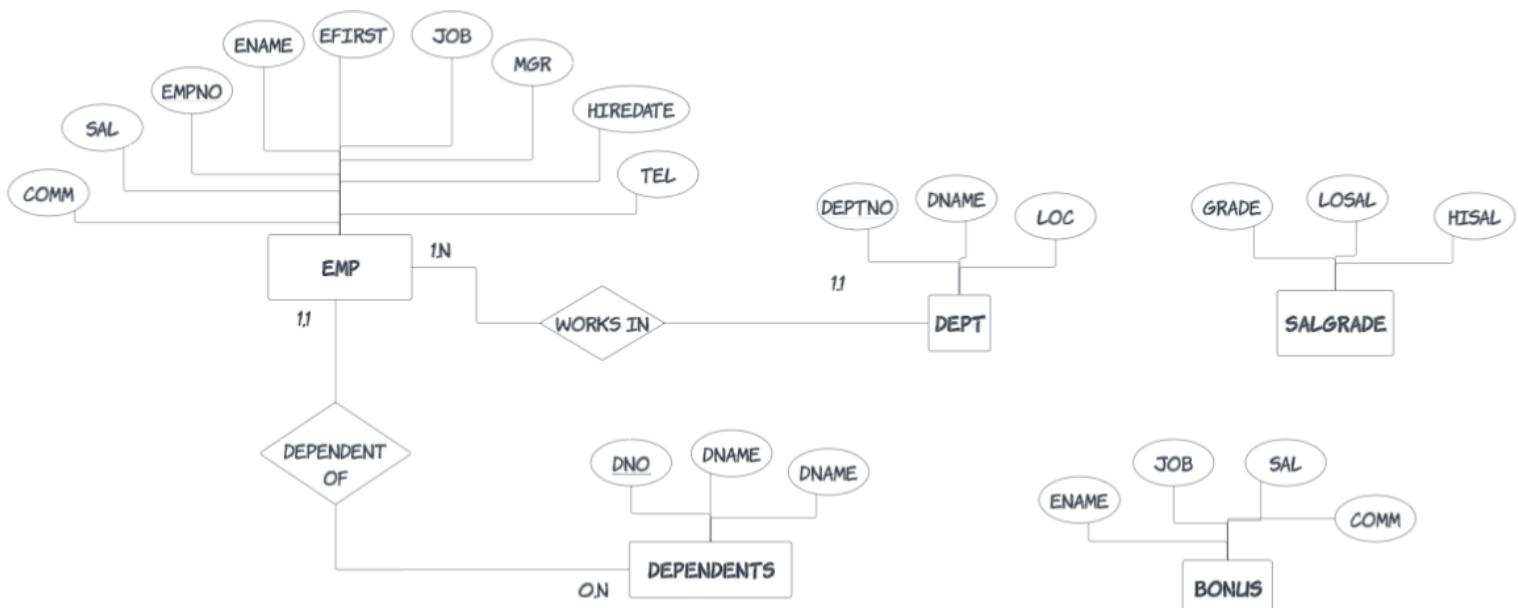
constraint fk\_emp\_dept foreign key(DEPTNO) references DEPT (DEPTNO)

### DEPENDENTS:

constraint pk\_dependent primary key (DNO, EMPNO)

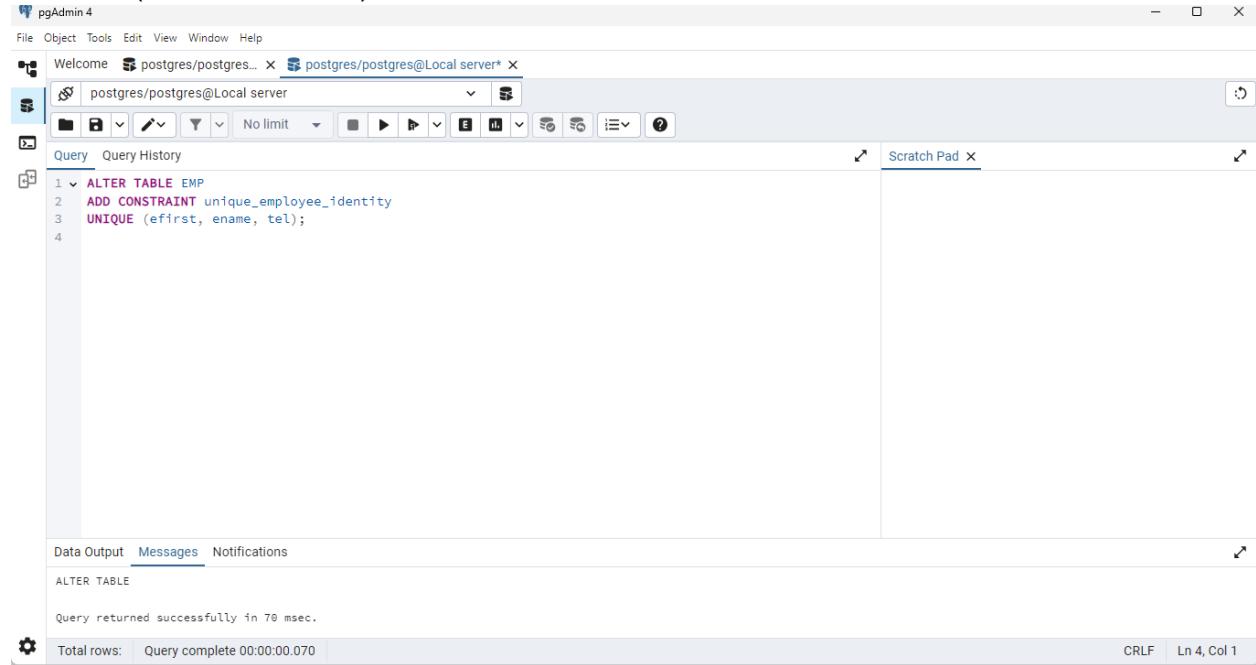
constraint fk\_dependent\_emp foreign key(EMPNO) references EMP (EMPNO)

## 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%');
```

```

pgAdmin 4
File Object Tools Edit View Window Help
Welcome postgres/postgres... × postgres/postgres@Local server* ×
postgres/postgres@Local server No limit ▾
Query History Scratch Pad ×
Query
1 -- Add a new column for mobile number
2 ALTER TABLE emp
3 ADD mobile_phone VARCHAR(15);
4
5 -- Add a CHECK constraint to ensure the mobile number starts with '06'
6 ALTER TABLE emp
7 ADD CONSTRAINT chk_mobile_phone_format
8 CHECK (mobile_phone LIKE '06%');
9
Data Output Messages Notifications
ALTER TABLE
Query returned successfully in 65 msec.
Total rows: Query complete 0:00:00.065
CRLF Ln 6, Col 16

```

**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\_dependent\_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;

The screenshot shows the pgAdmin 4 interface. In the top-left corner, it says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". There are two tabs open: "Welcome" and "postgres/postgres@Local server". The "postgres/postgres@Local server" tab is active, showing a query editor with the following SQL code:

```
1 -- Drop the existing foreign key constraint
2 \v ALTER TABLE DEPENDENTS
3 DROP CONSTRAINT fk_dependent_emp;
4
5 -- Add a new foreign key with cascading delete
6 \v ALTER TABLE DEPENDENTS
7 ADD CONSTRAINT fk_dependent_emp
8 FOREIGN KEY (EMPNO) REFERENCES EMP (EMPNO)
9 ON DELETE CASCADE;
10
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Messages" tab is selected, displaying the message "ALTER TABLE". Underneath, it says "Query returned successfully in 54 msec." and "Total rows: Query complete 00:00:00.054". To the right, a green box indicates "Query returned successfully in 54 msec." with a checkmark.

**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;
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server\* ×

postgres/postgres@Local server

No limit ▾

Query History

Query Scratch Pad

```
1 ✓ ALTER TABLE EMP
2   ALTER COLUMN MGR DROP NOT NULL;
3
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 57 msec.

Total rows: Query complete 00:00:00.057 CRLF Ln 3, 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 History

Query Scratch Pad

```
1 INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
2 INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
3 INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
4 INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
5
6
7 ✓ INSERT INTO EMP VALUES
8   (7369, 'SMITH', 'JOHN', 'CLERK',    7902,
9     TO_DATE('17-12-1980', 'DD-MM-YYYY'),  800, NULL, '0149545243', 20);
10 ✓ INSERT INTO EMP VALUES
11   (7499, 'ALLEN', 'BOB', 'SALESMAN', 7698,
12     TO_DATE('20-02-1981', 'DD-MM-YYYY'), 1600, 300, '0149547243', 30);
13 ✓ INSERT INTO EMP VALUES
14   (7521, 'WARD', 'PETER', 'SALESMAN', 7698,
15     TO_DATE('22-02-1981', 'DD-MM-YYYY'), 1250, 500, '0149545247', 30);
16 ✓ INSERT INTO EMP VALUES
17   (7566, 'JONES', 'JOHN', 'MANAGER', 7839,
18     TO_DATE('12-04-1981', 'DD-MM-YYYY'), 2975, NULL, '0149545456', 20);
19 ✓ INSERT INTO EMP VALUES
20   (7654, 'MARTIN', 'JOE', 'SALESMAN', 7698,
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 65 msec.

Total rows: Query complete 00:00:00.065 CRLF Ln 56, Col 2

**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|NO CYCLE];
```

```

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

```

The screenshot shows the pgAdmin 4 interface with a single query window open. The query is:

```

1 CREATE SEQUENCE dependent_seq
2 START WITH 8000
3 INCREMENT BY 1
4 MINVALUE 8000
5 NO CYCLE;
6

```

The 'Messages' tab is selected, displaying the message:

CREATE SEQUENCE

Query returned successfully in 58 msec.

In the bottom status bar, it says "Total rows: 0 Query complete 00:00:00.058". A green notification bar at the bottom right indicates "Query returned successfully in 58 msec." with a timestamp of "CRLF Ln 5, Col 4".

**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);

```

```

1 ✓ INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
2   VALUES (nextval('dependent_seq'), 'JIN', 'JANE', 7369);
3
4 ✓ INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
5   VALUES (nextval('dependent_seq'), 'AFFIONG', 'JOHN', 7499);
6
7 ✓ INSERT INTO DEPENDENTS (DNO, DNAME, DFIRST, EMPNO)
8   VALUES (nextval('dependent_seq'), 'AKPAN', 'MATT', 7521);
9

```

Query returned successfully in 46 msec.

Total rows: Query complete 00:00:00.046 CRLF Ln 8, Col 56

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:

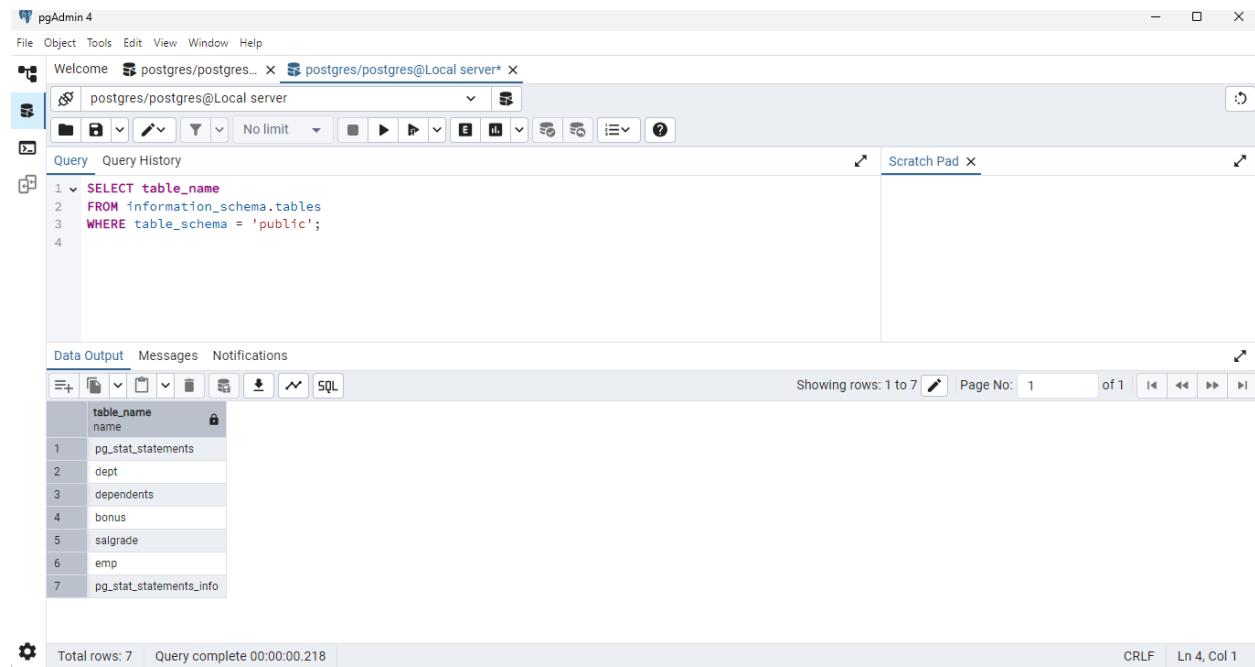
1. **For PostgreSQL 10 and later**, the recommended approach is to use the `IDENTITY` column. It is the SQL-standard method for defining auto-incrementing columns, providing a clean and declarative solution. The `IDENTITY` column is easier to manage, less error-prone, and aligns with modern database 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';
```



The screenshot shows the pgAdmin 4 interface. In the top-left corner, it says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". There are two tabs open in the top center: "Welcome" and "postgres/postgres@Local server". The "postgres/postgres@Local server" tab is active. Below the tabs is a toolbar with various icons for database management. The main area is divided into two panes: "Query" on the left and "Scratch Pad" on the right. The "Query" pane contains the following SQL code:

```
1 SELECT table_name  
2 FROM information_schema.tables  
3 WHERE table_schema = 'public';  
4
```

Below the query pane is a "Data Output" tab, which displays the results of the query:

table_name
pg_stat_statements
dept
dependents
bonus
salgrade
emp
pg_stat_statements_info

The results show 7 rows. At the bottom of the pgAdmin window, there is a status bar with "Total rows: 7" and "Query complete 00:00:00.218".

2. Select employees whose commission is higher than their salary

```
SELECT *  
FROM EMP  
WHERE COMM > SAL;
```

pgAdmin 4

Welcome postgres/postgres@Local server\* postgres/postgres@Local server\*

Query History

```
1 ✓ SELECT *
2   FROM EMP
3   WHERE COMM > SAL;
4
```

Data Output Messages Notifications

SQL Showing rows: 1 to 1 Page No: 1 of 1

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobile_phone character varying (15)
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784	30	[null]

Total rows: 1 Query complete 00:00:00.131 CRLF Ln 1, Col 9

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

```
SELECT *
FROM EMP
WHERE (SAL + COALESCE(COMM, 0)) BETWEEN 1200 AND 2400;
```

pgAdmin 4

Welcome postgres/postgres@Local server\* postgres/postgres@Local server\*

Query History

```
1 ✓ SELECT *
2   FROM EMP
3   WHERE (SAL + COALESCE(COMM, 0)) BETWEEN 1200 AND 2400;
4
```

Data Output Messages Notifications

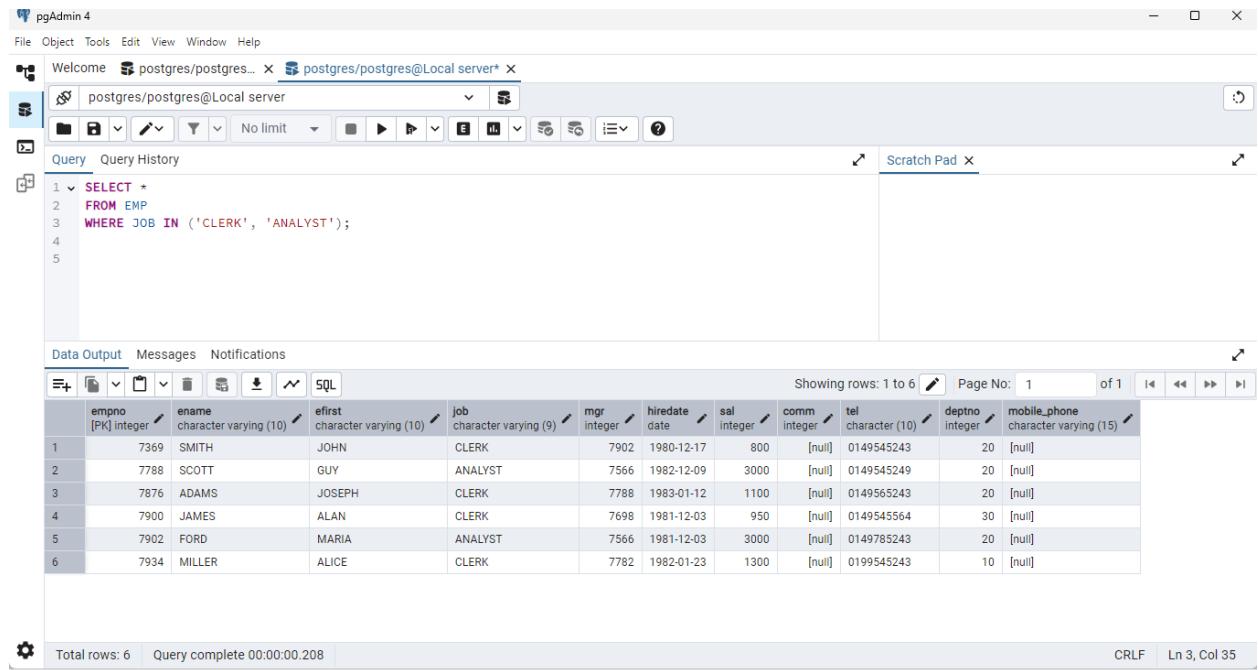
SQL Showing rows: 1 to 4 Page No: 1 of 1

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobile_phone character varying (15)
1	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
2	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247	30	[null]
3	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243	30	[null]
4	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]

Total rows: 4 Query complete 00:00:00.096 CRLF Ln 1, Col 2

4. Select employees who are CLERK or ANALYST

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



The screenshot shows the pgAdmin 4 interface with a query editor window. The query is:

```
1  SELECT *
2  FROM EMP
3  WHERE JOB IN ('CLERK', 'ANALYST');
4
5
```

The results table has the following data:

	empno	ename	efirst	job	mgr	hiredate	sal	comm	tel	deptno	mobile_phone
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
2	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]
4	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
5	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
6	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]

Total rows: 6 Query complete 00:00:00.208 CRLF Ln 3, Col 35

5. Select employees whose name begins by M

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

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server\*

postgres/postgres@Local server

No limit ▾

Query History

Scratch Pad

```
1 ✓ SELECT *
2   FROM EMP
3   WHERE ENAME LIKE '%M%';
4
5
```

Data Output Messages Notifications

SQL

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobile_phone character varying (15)
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784	30	[null]
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]

Total rows: 2 Query complete 00:00:00.153 CRLF Ln 1, Col 9

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

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

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server\*

postgres/postgres@Local server

No limit ▾

Query History

Scratch Pad

```
1 ✓ SELECT *
2   FROM EMP
3   WHERE ENAME LIKE '_L%';
4
```

Data Output Messages Notifications

SQL

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobile_phone character varying (15)
1	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
3	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]

✓ Successfully run. Total query runtime: 93 msec. 3 rows affected.

Total rows: 3 Query complete 00:00:00.093 CRLF Ln 1, Col 9

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;
```

The screenshot shows the pgAdmin 4 interface. In the top navigation bar, there are tabs for 'File', 'Object', 'Tools', 'Edit', 'View', 'Window', and 'Help'. Below the navigation bar, there are two tabs: 'Welcome' and 'postgres/postgres@Local server'. The 'postgres/postgres@Local server' tab is active. On the left side of the main window, there is a tree view under the heading 'Query History'. A single query entry is listed, which is the one shown in the code block above. To the right of the tree view is a 'Scratch Pad' tab. The main area of the window is divided into two panes. The left pane contains the SQL query, and the right pane displays the results of the query. The results are presented in a table with the following columns: empno [PK] integer, ename character varying (10), efirst character varying (10), job character varying (9), mgr integer, hiredate date, sal integer, comm integer, tel character (10), deptno integer, and mobile\_phone character varying (15). There is one row of data: empno 7782, ename CLARK, efirst JOHN, job MANAGER, mgr 7839, hiredate 1981-06-09, sal 2450, comm [null], tel 0149545245, deptno 10, and mobile\_phone [null]. At the bottom of the results pane, there is a message: 'Successfully run. Total query runtime: 85 msec. 1 rows affected.' Below the results pane, there is a status bar with the text 'Total rows: 1 Query complete 00:00:00.085' and a CRLF/Ln 1, Col 9 indicator.

empno	ename	efirst	job	mgr	hiredate	sal	comm	tel	deptno	mobile_phone
7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]

#### 8. Select employees whose commission is NULL

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





pgAdmin 4

Welcome postgres/postgres@Local server postgres/postgres@Local server\*

Query History

```
1 ✓ SELECT D.DEPTNO, D.DNAME, D.LOC
2   FROM DEPT D
3   LEFT JOIN EMP E ON D.DEPTNO = E.DEPTNO
4   WHERE E.EMPNO IS NULL;
5
6
```

Data Output Messages Notifications

deptno	dname	loc
40	OPERATIONS	BOSTON

Total rows: 1 Query complete 00:00:00.099

Successfully run. Total query runtime: 99 msec. 1 rows affected.

CRLF Ln 5, Col 1

## 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;
```

pgAdmin 4

Welcome postgres/postgres@Local server postgres/postgres@Local server\*

Query History

```
1 ✓ SELECT E.EMPNO, E.ENAME AS Employee, M.ENAME AS Manager
2   FROM EMP E
3   LEFT JOIN EMP M ON E.MGR = M.EMPNO;
4
```

Data Output Messages Notifications

empno	employee	manager
7369	SMITH	FORD
7499	ALLEN	BLAKE
7521	WARD	BLAKE
7566	JONES	KING
7654	MARTIN	BLAKE
7698	BLAKE	KING
7782	CLARK	KING
7788	SCOTT	JONES
7839	KING	[null]
7844	TURNER	BLAKE

Total rows: 14 Query complete 00:00:00.079

### 13. List employees earning more than JONES

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

The screenshot shows the pgAdmin 4 interface. At the top, there are tabs for 'Welcome', 'postgres/postgres...' (selected), and 'postgres/postgres@Local server\*'. Below the tabs is a toolbar with various icons. The main area has two panes: 'Query' on the left containing the SQL code, and 'Scratch Pad' on the right which is currently empty. The 'Query' pane contains the following SQL:

```
1 SELECT EMPNO, ENAME, JOB, SAL, COMM  
2 FROM EMP  
3 WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME = 'JONES');  
4
```

Below the query pane is a 'Data Output' tab where the results are displayed as a table:

	empno	ename	job	sal	comm
	[PK] integer	character varying (10)	character varying (9)	integer	integer
1	7788	SCOTT	ANALYST	3000	[null]
2	7839	KING	PRESIDENT	5000	[null]
3	7902	FORD	ANALYST	3000	[null]

At the bottom of the interface, there is a status bar with the message 'Successfully run. Total query runtime: 128 msec. 3 rows affected.' and other system information like 'Total rows: 3' and 'Query complete 00:00:00.128'.

### 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;
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server\* ×

postgres/postgres@Local server

No limit ▾

Query History

Scratch Pad

```
1 SELECT empno, ename, CONCAT('Salary: ', SAL, ', Commission: ', COALESCE(COMM, 0)) AS Salary_And_Commission
2 FROM EMP;
3
```

Data Output Messages Notifications

Showing rows: 1 to 14 | Page No: 1 of 1 | 14 << >> | 1

	empno	ename	salary_and_commission
1	7369	SMITH	Salary: 800, Commission: 0
2	7499	ALLEN	Salary: 1600, Commission: 300
3	7521	WARD	Salary: 1250, Commission: 500
4	7566	JONES	Salary: 2975, Commission: 0
5	7654	MARTIN	Salary: 1250, Commission: 1400
6	7698	BLAKE	Salary: 2850, Commission: 0
7	7782	CLARK	Salary: 2450, Commission: 0
8	7788	SCOTT	Salary: 3000, Commission: 0
9	7839	KING	Salary: 5000, Commission: 0
10	7844	TURNER	Salary: 1500, Commission: 0
11	7876	ADAMS	Salary: 1100, Commission: 0

Total rows: 14 Query complete 00:00:00.187 CRLF Ln 3, Col 1

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;
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server\* ×

postgres/postgres@Local server

No limit ▾

Query History

Scratch Pad

```
1 SELECT DISTINCT E.DEPTNO
2 FROM EMP E
3 JOIN DEPT D ON E.DEPTNO = D.DEPTNO;
4
```

Data Output Messages Notifications

Showing rows: 1 to 3 | Page No: 1 of 1 | 14 << >> | 1

	deptno
1	30
2	10
3	20

Total rows: 3 Query complete 00:00:00.140 ✓ Successfully run. Total query runtime: 140 msec. 3 rows affected. CRLF Ln 4, Col 1

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');
```

The screenshot shows the pgAdmin 4 interface with a single query window open. The query is the one provided above, selecting employees from the EMP and DEPT tables where they work in CHICAGO and have the same job as 'JONES'. The result set is displayed in a table with columns: empno, ename, job, sal, and deptno. There is one row returned, corresponding to employee 7698 (BLAKE) who works in department 30 and has the job 'MANAGER'. A message at the bottom indicates the query was successfully run.

empno	ename	job	sal	deptno
7698	BLAKE	MANAGER	2850	30

Total rows: 1    Query complete 00:00:00.130    CRLF    Ln 6, Col 1

Successfully run. Total query runtime: 130 msec. 1 rows affected.

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;
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres@Local server \* postgres/postgres@Local server \*

postgres/postgres@Local server No limit Query History Scratch Pad

```
1 v SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO AS EMP_DEPT, M.ENAME AS MANAGER_NAME, M.DEPTNO AS MANAGER_DEPT
2 FROM EMP E
3 JOIN EMP M ON E.MGR = M.EMPNO
4 WHERE E.DEPTNO != M.DEPTNO;
5
6
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	empno	ename	job	emp_dept	manager_name	manager_dept
1	7566	JONES	MANAGER	20	KING	10
2	7698	BLAKE	MANAGER	30	KING	10

Total rows: 2 Query complete 00:00:00.076 CRLF Ln 5, Col 1

### 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');
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... \* postgres/postgres@Local server \*

postgres/postgres@Local server No limit Query History Scratch Pad

```
1 v SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO
2 FROM EMP E
3 WHERE E.DEPTNO IN (SELECT DISTINCT DEPTNO FROM EMP WHERE JOB = 'CLERK');
4
5
```

Data Output Messages Notifications

Showing rows: 1 to 14 Page No: 1 of 1

	empno	ename	job	deptno
1	7369	SMITH	CLERK	20
2	7499	ALLEN	SALESMAN	30
3	7521	WARD	SALESMAN	30
4	7566	JONES	MANAGER	20
5	7654	MARTIN	SALESMAN	30
6	7698	BLAKE	MANAGER	30
7	7782	CLARK	MANAGER	10
8	7788	SCOTT	ANALYST	20
9	7839	KING	PRESIDENT	10
10	7844	TURNER	SALESMAN	30
11	7876	ADAMS	CLERK	20

Successfully run. Total query runtime: 101 msec. 14 rows affected. CRLF Ln 4, Col 1

Total rows: 14 Query complete 00:00:00.101

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);
```

The screenshot shows the pgAdmin 4 interface with a single query window open. The query is:

```
1 v SELECT E.EMPNO, E.ENAME, E.JOB, E.DEPTNO  
2   FROM EMP E  
3   WHERE E.DEPTNO = 10  
4     AND E.JOB IN (SELECT DISTINCT JOB FROM EMP WHERE DEPTNO = 30);  
5  
6
```

The results table shows two rows:

	empno	ename	job	deptno
1	7782	CLARK	MANAGER	10
2	7934	MILLER	CLERK	10

At the bottom, a message bar indicates: "Successfully run. Total query runtime: 192 msec. 2 rows affected." and "Total rows: 2 Query complete 00:00:00.192".

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');
```

The screenshot shows the pgAdmin 4 interface. In the top-left corner, it says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". There are two tabs open: "postgres/postgres@Local server" and "postgres/postgres@Local server\*". The main window contains a query editor with the following SQL code:

```
1 v SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL
2   FROM EMP E
3 WHERE E.JOB = (SELECT JOB FROM EMP WHERE ENAME = 'JONES')
4     OR E.SAL > (SELECT SAL FROM EMP WHERE ENAME = 'FORD');
5
6
```

Below the query editor is a "Data Output" tab showing the results of the query:

	empno	ename	job	sal
1	7566	JONES	MANAGER	2975
2	7698	BLAKE	MANAGER	2850
3	7782	CLARK	MANAGER	2450
4	7839	KING	PRESIDENT	5000

At the bottom of the pgAdmin window, there is a message box indicating success: "Successfully run. Total query runtime: 119 msec. 4 rows affected." and "CRLF Ln 3, Col 13".

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);
```

```

SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO
FROM EMP E
WHERE E.SAL > ALL (SELECT SAL FROM EMP WHERE DEPTNO = 20);

```

	empno	ename	sal	deptno
1	7839	KING	5000	10

Total rows: 1    Query complete 00:00:00.165    CRLF    Ln 4, Col 1

Successfully run. Total query runtime: 165 msec. 1 rows affected.

## 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)
);

```

```

CREATE TABLE project (
    projno INTEGER PRIMARY KEY,
    pname VARCHAR(100),
    startdate DATE,
    budget DECIMAL(10, 2)
);

```

Query returned successfully in 84 msec.

Total rows: Query complete 00:00:00.084 CRLF Ln 7, Col 1

**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)

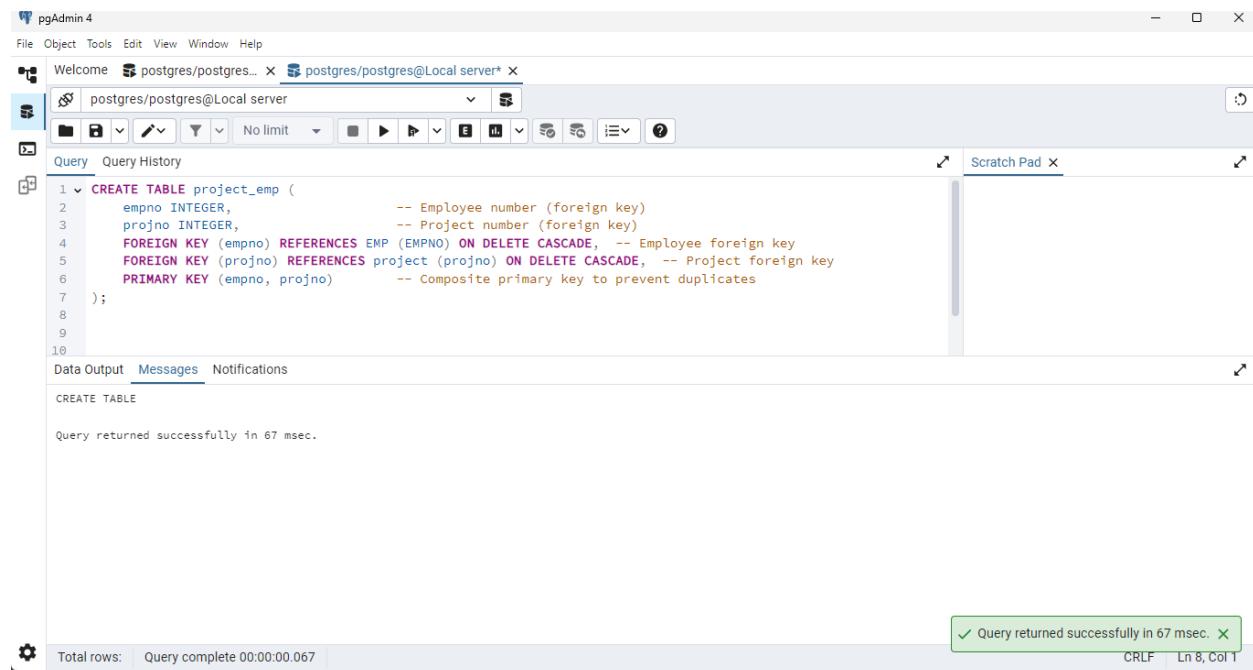
    FOREIGN KEY (empno) REFERENCES EMP (EMPNO) ON DELETE CASCADE, -- Employee
    foreign key

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

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

```

');



The screenshot shows the pgAdmin 4 interface. In the top navigation bar, the 'File' menu is open. Below the menu, there are two tabs: 'Welcome' and 'postgres/postgres@Local server'. The 'postgres/postgres@Local server' tab is active. In the main pane, there is a 'Query' tab containing the following SQL code:

```
1 v CREATE TABLE project_emp (
2     empno INTEGER,                      -- Employee number (foreign key)
3     projno INTEGER,                     -- Project number (foreign key)
4     FOREIGN KEY (empno) REFERENCES EMP (EMPNO) ON DELETE CASCADE, -- Employee foreign key
5     FOREIGN KEY (projno) REFERENCES project (projno) ON DELETE CASCADE, -- Project foreign key
6     PRIMARY KEY (empno, projno)        -- Composite primary key to prevent duplicates
7 );
8
9
10
```

Below the code, the 'Data Output' tab is selected, showing the output of the query:

```
CREATE TABLE
```

Query returned successfully in 67 msec.

In the bottom status bar, it says 'Total rows: 0' and 'Query complete 00:00:00.067'. To the right, a green message box indicates 'Query returned successfully in 67 msec.' with a checkmark icon.

**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);
```

The screenshot shows the pgAdmin 4 interface with two tabs open: 'postgres/postgres@Local server' and 'postgres/postgres@Local server\*'. The main query window contains the following SQL code:

```
1 v INSERT INTO project (projno, pname, startdate, budget) VALUES
2   (101, 'Project A', TO_DATE('2023-01-01', 'YYYY-MM-DD'), 100000.00),
3   (102, 'Project B', TO_DATE('2023-03-01', 'YYYY-MM-DD'), 150000.00),
4   (103, 'Project C', TO_DATE('2023-06-01', 'YYYY-MM-DD'), 200000.00),
5   (104, 'Project D', TO_DATE('2023-09-01', 'YYYY-MM-DD'), 120000.00);
6
7
8
9
10
```

The 'Data Output' tab shows the result of the query: 'INSERT 0 4'. Below it, a message states 'Query returned successfully in 93 msec.'.

**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),

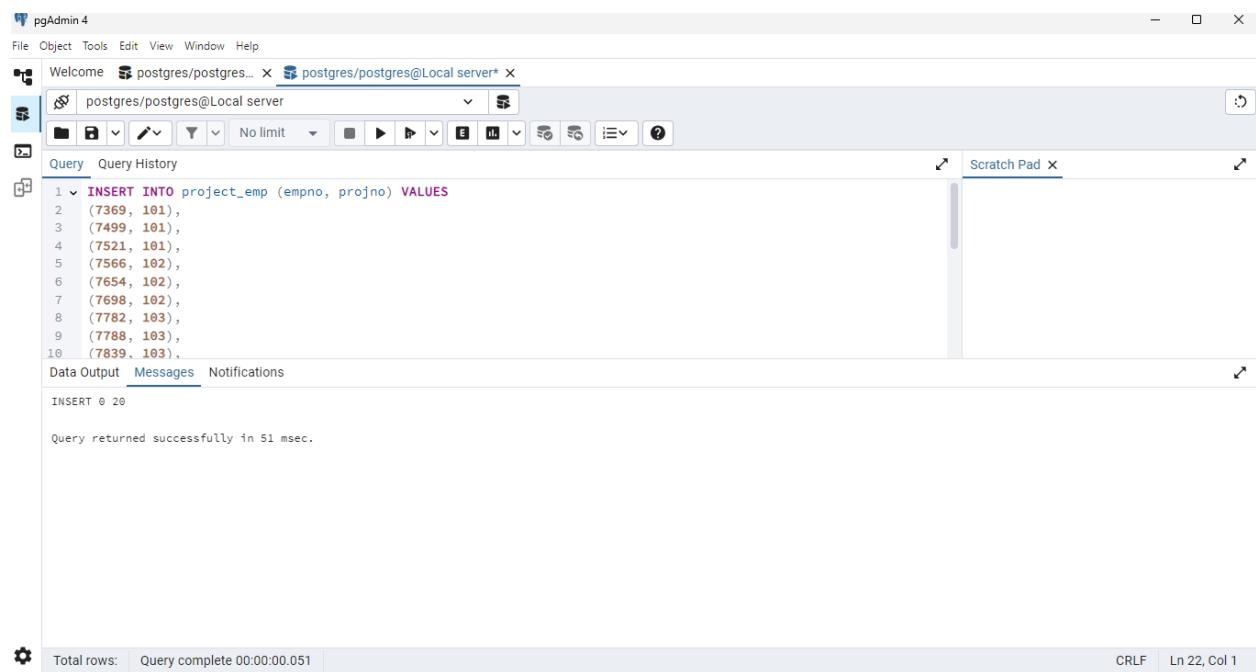
(7566, 103),

(7654, 101),

(7698, 104),

(7782, 102),

(7902, 103);



The screenshot shows the pgAdmin 4 interface. In the top navigation bar, 'File', 'Object', 'Tools', 'Edit', 'View', 'Window', and 'Help' are visible. There are two tabs open in the main window: 'Welcome' and 'postgres/postgres@Local server'. The 'postgres/postgres@Local server' tab is active. Below the tabs is a toolbar with various icons. The main area contains a 'Query' tab and a 'Scratch Pad' tab. The 'Query' tab has the following content:

```
1 ✓ INSERT INTO project_emp (empno, projno) VALUES
2   (7369, 101),
3   (7499, 101),
4   (7521, 101),
5   (7566, 102),
6   (7654, 102),
7   (7698, 102),
8   (7782, 103),
9   (7788, 103),
10  (7839, 103).
```

Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the message: 'INSERT 0 20'. At the bottom of the pgAdmin window, status information includes 'Total rows: 0', 'Query complete 00:00:00.051', 'CRLF', and 'Ln 22, Col 1'.

### 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);

```

The screenshot shows the pgAdmin 4 interface. In the top navigation bar, 'File', 'Object', 'Tools', 'Edit', 'View', 'Window', and 'Help' are visible. Below the bar, there are tabs for 'Welcome', 'postgres/postgres...', and 'postgres/postgres@Local server\*'. The main window has a toolbar with various icons. A 'Query' tab is active, containing the following SQL code:

```

1 v SELECT empno
2   FROM project_emp
3   GROUP BY empno
4 HAVING COUNT(DISTINCT projno) = (SELECT COUNT(*) FROM project);
5

```

Below the query, the results pane shows a single column named 'empno' with the data type 'Integer'. At the bottom of the interface, a status bar displays 'Total rows: 0' and 'Query complete 00:00:00.074'. A green success message in the status bar says 'Successfully run. Total query runtime: 74 msec. 0 rows affected.'.

#### 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 History

Scratch Pad

CREATE VIEW sales\_staff AS  
SELECT empno, ename, deptno  
FROM emp  
WHERE deptno = 10  
WITH CHECK OPTION;

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 History

Scratch Pad

INSERT INTO sales\_staff VALUES (7584, 'OSTER', 10);

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 screenshot shows the pgAdmin 4 interface. In the top-left corner, it says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". There are two tabs open: "Welcome" and "postgres/postgres@Local server\*". The "postgres/postgres@Local server\*" tab is active. The main area has a toolbar with various icons. Below the toolbar, there are two tabs: "Query" (selected) and "Query History". The "Query" tab contains the following SQL code:

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

Below the query, the "Messages" tab is selected, showing the error message:

```
ERROR: new row violates check option for view "sales_staff"  
Failing row contains (7591, WILLIAMS, null, null, null, null, null, null, null, 30, null).  
  
SQL state: 44000  
Detail: Failing row contains (7591, WILLIAMS, null, null, null, null, null, null, null, 30, null).
```

At the bottom of the pgAdmin window, there is a status bar with "Total rows: 0" and "Query complete 00:00:00.139". On the right side of the status bar, it says "CRLF" and "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;
```

```

SELECT empno, COUNT(DISTINCT projno) AS num_projects
FROM project_emp
GROUP BY empno
HAVING COUNT(DISTINCT projno) >= 2;

```

	empno	num_projects
1	7369	2
2	7499	2
3	7521	2
4	7566	2
5	7654	2
6	7698	2
7	7782	2

Successfully run. Total query runtime: 159 msec. 7 rows affected.

## 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;

The screenshot shows the pgAdmin 4 interface. In the top-left corner, there's a 'File' menu with options like Object, Tools, Edit, View, Window, Help. Below the menu bar, there are two tabs: 'Welcome' and 'postgres/postgres@Local server'. The 'postgres/postgres@Local server' tab is active, showing a toolbar with icons for file operations, search, and navigation. A 'Query History' section displays a previous query. To the right of the history is a 'Scratch Pad' tab. The main area contains a SQL query editor with the following code:

```
1 ✓ SELECT empno
2   FROM project_emp
3   WHERE projno = 1
4
5 INTERSECT
6
7 SELECT empno
8   FROM project_emp
9   WHERE projno = 2;
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. Under 'Data Output', a table is shown with one row:

empno
integer

In the bottom status bar, it says 'Total rows: 0' and 'Query complete 00:00:00.232'. On the right side of the status bar, there's a green message box that says 'Successfully run. Total query runtime: 232 msec. 0 rows affected.' with a close button. Next to it are buttons for 'CRLF' and 'Ln 8, Col 1'.

**SELECT empno**

**FROM project\_emp**

**WHERE projno = 3**

**EXCEPT**

**SELECT empno**

**FROM project\_emp**

**WHERE projno = 4;**

The screenshot shows the pgAdmin 4 interface. A query window titled 'postgres/postgres@Local server' contains the following SQL code:

```
1 ✓ SELECT empno
2   FROM project_emp
3  WHERE projno = 3
4  EXCEPT
5  SELECT empno
6   FROM project_emp
7  WHERE projno = 4;
```

The results pane shows a single column named 'empno' with the value 'integer'. A status bar at the bottom indicates 'Successfully run. Total query runtime: 167 msec. 0 rows affected.'

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;

```

The screenshot shows the pgAdmin 4 interface with a query editor and a results table.

**Query Editor:**

```

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

```

**Results Table:**

	projno	empno	ename	sal	rank
1	101	7499	ALLEN	1600	1
2	101	7654	MARTIN	1250	2
3	101	7521	WARD	1250	3
4	102	7566	JONES	2975	1
5	102	7698	BLAKE	2850	2
6	102	7782	CLARK	2450	3
7	103	7839	KING	5000	1
8	103	7902	FORD	3000	2
9	103	7780	SCOTT	3000	3
10	104	7876	SIMPSON	3000	1
11	104	7787	KING	5000	2
12	104	7886	BLAKE	2850	3

**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;

```

The screenshot shows the pgAdmin 4 interface. The top window is titled 'Welcome' and has tabs for 'postgres/postgres...' and 'postgres/postgres@Local server'. Below the tabs is a toolbar with various icons. The main area is a 'Query' window containing the provided SQL code. To the right of the Query window is a 'Scratch Pad' window. Below the Query window is a 'Data Output' window displaying the results of the query. The results are presented in a table with three columns: 'empno', 'ename', and 'project\_percentage'. The data shows eight rows of employees and their project percentages.

	empno	ename	project_percentage
1	7698	BLAKE	50.00%
2	7499	ALLEN	50.00%
3	7782	CLARK	50.00%
4	7521	WARD	50.00%
5	7369	SMITH	50.00%
6	7566	JONES	50.00%
7	7654	MARTIN	50.00%
8	7839	KING	25.00%

Total rows: 15    Query complete 00:00:00.345    CRLF    Ln 9, Col 1

## 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 History Scratch Pad

```
1 ✓ SELECT
2     e.empno,
3     e.ename,
4     ROUND(COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project), 2) || '%' AS project_percentage,
5     CASE
6         WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 0 THEN 'Empty'
7         WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 10 AND 50 THEN 'Small'
8         WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 50 AND 80 THEN 'Medium'
9         WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) BETWEEN 80 AND 100 THEN 'Large'
10        WHEN COUNT(pe.projno) * 100.0 / (SELECT COUNT(*) FROM project) = 100 THEN 'Total'
11    END AS scope_size
12  FROM emp e
13  LEFT JOIN project_emp pe ON e.empno = pe.empno
14  GROUP BY e.empno, e.ename
15  ORDER BY project_percentage DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 15 | Page No: 1 of 1 | < << > >> | CRLF | Ln 16, Col 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

# 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();
}
...
```

The screenshot shows the IntelliJ IDEA interface with a Java file named 'Main.java' open. The code contains a main method that loads the PostgreSQL JDBC driver and prints a stack trace if a class not found exception occurs. The run tool window at the bottom shows the application was run successfully with exit code 0. The status bar indicates the file was saved with LF line endings and 4 spaces.

```
4 //TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
5 // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
6
7 public class Main {
8     public static void main(String[] args) {
9
10         /* Load JDBC Driver. */
11         try {
12             Class.forName("org.postgresql.Driver");
13         } catch ( ClassNotFoundException e ) {
14             e.printStackTrace();
15         }
16     }
17 }
18 }
```

## 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:

```
Project lab3 Version control Current File Run Main.java AI Assistant
Main.java
public class Main {
    public static void main(String[] args) {
        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();
                }
        }
    }
}

Run Main
Bdd Connected
Process finished with exit code 0
lab3 > src > Main > main
28:37 LF UTF-8 4 spaces
```

```
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 success
  - 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();
    }
}
```

The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The code connects to a PostgreSQL database and prints department details to the console. The run output shows the application successfully connecting to the database and printing four department records.

```
public class Main {
    public static void main(String[] args) {
        }
        resultat.close();
    }
}
```

```
Bdd Connected
Department 10 is for ACCOUNTING and located in ?
Department 20 is for RESEARCH and located in ?
Department 30 is for SALES and located in ?
Department 40 is for OPERATIONS and located in ?
Process finished with exit code 0
```

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

The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The code has been modified to include a column for location in the query result. The run output shows the application successfully connecting to the database and printing four department records, each including its location.

```
public class Main {
    public static void main(String[] args) {
        }
        resultat.close();
    }
}
```

```
Bdd Connected
Department 10 is for ACCOUNTING and located in NEW YORK
Department 20 is for RESEARCH and located in DALLAS
Department 30 is for SALES and located in CHICAGO
Department 40 is for OPERATIONS and located in BOSTON
Process finished with exit code 0
```

## Exercice 2: Move Department

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

The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows a project named "lab3" containing files like ".idea", "out", "src", ".gitignore", and "lab3.iml".
- Main.java Content:** The code implements a static method `moveDepartment` that updates the `deptno` field in the `emp` table for a given `empno` to a new `newDeptno`. It prints a message to `System.out` indicating whether the move was successful or not.
- Output Window:** Displays the results of running the program, showing four department entries: "Department 10 is for ACCOUNTING and located in NEW YORK", "Department 20 is for RESEARCH and located in DALLAS", "Department 30 is for SALES and located in CHICAGO", and "Department 40 is for OPERATIONS and located in BOSTON".
- Status Bar:** Shows the file path "lab3 > src > Main", encoding "UTF-8", and a character count of "67:1 LF".

*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](#)



The screenshot shows the IntelliJ IDEA interface with the Main.java file open. The code connects to a database and prints employee data to the console. The output window shows the results of the query:

```

Bdd Connected
empno ename  efirst job mgr hiredate    sal comm      tel deptno  mobile_phone
-----
7499 ALLEN BOB SALESMAN 7698 1981-02-20 1600 1200 0149547243 30 null
7521 WARD PETER SALESMAN 7698 1981-02-22 1250 1200 0149545247 30 null
7566 JONES JOHN MANAGER 7839 1981-04-02 2975 1500 0149545456 20 null
7698 BLAKE BOB MANAGER 7839 1981-05-01 2850 1500 0149545254 30 null
7782 CLARK JOHN MANAGER 7839 1981-06-09 2450 1500 0149545245 10 null
7788 SCOTT GUY ANALYST 7566 1982-12-09 3000 1500 0149545249 20 null
7839 KING GUY PRESIDENT null 1981-11-17 5000 1500 0149545241 10 null
7844 TURNER PETER SALESMAN 7698 1981-09-08 1500 1200 0149548243 30 null
7876 ADAMS JOSEPH CLERK 7788 1983-01-12 1100 1200 0149565243 20 null
7900 JAMES ALAN CLERK 7698 1981-12-03 950 800 0149545564 30 null

```

## 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();
```

```

public class Main {
    public static void main(String[] args) {
        System.out.println("Enter first name:");
        String firstName = sc.nextLine(); // Capture user input for first name
        System.out.println("Enter last name:");
        String lastName = sc.nextLine(); // Capture user input for last name

        // Create the PreparedStatement to securely query the database
        String query = "SELECT * FROM emp WHERE efirst = ? AND ename = ?";
        PreparedStatement preparedStatement = connexion.prepareStatement(query);

        // Set the values in the PreparedStatement
        preparedStatement.setString(parameterIndex: 1, firstName); // Bind the first name to the first parameter
        preparedStatement.setString(parameterIndex: 2, lastName); // Bind the last name to the second parameter

        // Execute the query
    }
}

```

Run Main

```

BOB
Enter last name:
ALLEN
Employee ID: 7499
Employee First Name: BOB
Employee Last Name: ALLEN
Process finished with exit code 0

```

## Exercice 5: Modify moveDepartement() to use PreparedStatement

```

public class Main {
    public static void moveDepartment(Connection connexion, int empno, int deptno) throws SQLException {
        // SQL query with placeholders (?)
        String command = "UPDATE EMP SET DEPTNO = ? WHERE EMPNO = ?";

        // Use PreparedStatement to execute the query with parameters
        try (PreparedStatement updateEmp = connexion.prepareStatement(command)) {
            // Set the values for the placeholders in the query
            updateEmp.setInt(parameterIndex: 1, deptno); // Set the department number (new department)
            updateEmp.setInt(parameterIndex: 2, empno); // Set the employee number (whose department needs to be updated)

            // Execute the update query
            int rowsUpdated = updateEmp.executeUpdate();
        }
    }
}

Run Main
Enter department ID to move to:
20
Employee NO: 7499 has been moved to Dept NO: 20
Process finished with exit code 0

```

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

## Exercice 6: Try with displayTable

Try to modify your method `displayTable` by using `PreparedStatement` with `tableName` as a parameter of `PreparedStatement`.

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.



## 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.

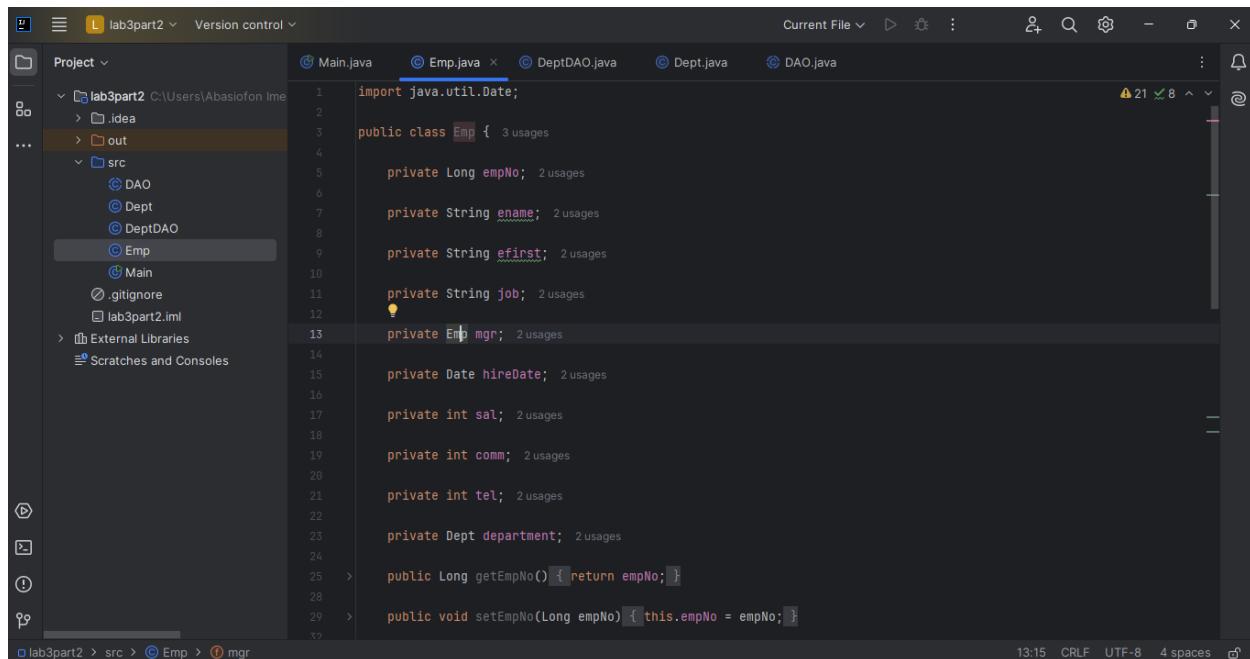
7. **No Separation of Concerns:** The business logic, database logic, and presentation logic are tightly coupled in the servlet. By using a DAO pattern, the database logic is separated from business logic, making the code easier to maintain, understand, and extend. This separation of concerns also makes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



```
import java.util.Date;

public class Emp {
    private Long empNo;
    private String ename;
    private String efirst;
    private String job;
    private Emp mgr;
    private Date hireDate;
    private int sal;
    private int comm;
    private int tel;
    private Dept department;
    public Long getEmpNo() { return empNo; }
    public void setEmpNo(Long empNo) { this.empNo = empNo; }
}
```

### Exercice 9 : Write the Bean for Department

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

The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows the project structure under "lab3part2". The "src" folder contains "DAO", "Dept", "DeptDAO", "Emp", and "Main" packages.
- Editor Tab:** The active tab is "Dept.java".
- Code Content:** The code defines a class named "Dept" with the following methods:
  - Constructor: "Dept() { }"
  - Private fields: "private int Deptno; private String Dname; private String loc;"
  - Public methods:
    - "public String getDname() { return Dname; }"
    - "public void setDname(String dname) { Dname = dname; }"
    - "public int getDeptno() { return Deptno; }"
    - "public void setDeptno(int deptno) { Deptno = deptno; }"
    - "public String getLoc() { return loc; }"
    - "public void setLoc(String loc) { this.loc = loc; }"
- Bottom Status Bar:** Shows the file path "lab3part2 > src > Dept > Dept.java", and the status "3 2 8".
- Bottom Right:** Displays the time "30:92" and encoding "CRLF UTF-8 4 spaces".

## 2. DAO Implementation

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

The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows the project structure under "lab3part2". The "src" folder contains "DAO", "Dept", "DeptDAO", "Emp", and "Main" packages.
- Editor Tab:** The active tab is "DAO.java".
- Code Content:** The code defines an abstract class named "DAO<T>" with the following methods:
  - Protected field: "protected Connection connect = null;"
  - Constructor: "public DAO(Connection connect) { this.connect = connect; }"
  - Abstract methods:
    - "public abstract T find(int id);"
    - "public abstract boolean create(T object);"
    - "public abstract boolean update(T object);"
    - "public abstract boolean delete(T object);"
- Bottom Status Bar:** Shows the file path "lab3part2 > src > DAO > DAO.java", and the status "4 4 4".
- Bottom Right:** Displays the time "14:46" and encoding "CRLF UTF-8 4 spaces".

## Exercice 10 : Write Implementation for method find for department DAO.

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

Then test it:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure with modules like lab3part2, .idea, out, and src containing DAO, Dept, DeptDAO, Emp, and Main classes, along with .gitignore and lab3part2.iml files.
- Main.java Content:** The code implements a main method that connects to a PostgreSQL database using JDBC, retrieves a department by ID (DeptNo 20), and prints the result. The code includes try-catch blocks and a finally block to handle database resources.
- Run Tab:** Shows the run configuration for Main.
- Output Window:** Displays the command used to run the application ("C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.5\lib\idea\_rt.jar=60664" -Dfile.encoding=UTF-8 Main) and the resulting output: "DeptNo : 20 Department Name: RESEARCH Department Location: DALLAS".
- Status Bar:** Shows the file path (lab3part2 > src > Main > main), encoding (UTF-8), and code style settings (32:39, LF, 4 spaces).

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

This one will be trickier 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

The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure for "lab3part2" located at "C:\Users\Abasiofon.lme". It includes a ".idea" folder, "out", and a "src" folder containing classes: DAO, Dept, DeptDAO, Emp, EmpDAO, and Main.
- Main.java Content:** The code implements the Main class with a main method. It connects to a database, retrieves a department (ID 20), and prints its details. Then, it retrieves an employee (ID 7369) and prints their details.
- Run Tab:** Displays the output of the program:

```
Fetching Employee with ID 7369...
Employee Details: SMITH
Job: CLERK
Salary: 7000
Manager: FORD
Department: RESEARCH
```
- Status Bar:** Shows the time as 16:65, file encoding as LF, and code style as 4 spaces.

### 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 screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "lab3part2".
- Main.java:** The active file contains Java code for testing DAO methods. It includes imports for DAO, DAOFactory, Dept, DeptDAO, Emp, EmpDAO, and Main.
- Run Tab:** Shows the run configuration for "Main".
- Output Tab:** Displays the console output of the program execution, showing the retrieval of employee details for employee ID 7369.
- Status Bar:** Shows the time as 32:34, encoding as LF, and file save status as 4 spaces.

```
public class Main {
    public static void main(String[] args) {
        // Step 4: Test Department Retrieval
        DeptDAO departmentDAO = DAOFactory.getDeptDAO(connexion);
        System.out.println("Fetching Department with ID 20...");
        Dept dept20 = departmentDAO.find( Deptno: 20 );
        System.out.println(dept20); // Should print department details

        // Step 5: Test Employee Retrieval
        EmpDAO employeeDAO = DAOFactory.getEmpDAO(connexion);
        System.out.println("\nFetching Employee with ID 7369...");
        Emp emp7369 = employeeDAO.find( empno: 7369 );
        System.out.println("Employee Details: " + emp7369.getEname());
        System.out.println("Job: " + emp7369.getJob());
        System.out.println("Salary: " + emp7369.getSal());

        // Step 6: Test Manager Retrieval
    }
}
```

```
Employee Details: SMITH
Job: CLERK
Salary: 7000
Manager: FORD
Department: RESEARCH
```

The codes can be found here: <https://github.com/DatagirIX/JDBCPart2>

## Part III : Spring Boot & JPA.

### Exercice III.1 : Init your Spring project on IntelliJ

The screenshot shows the IntelliJ IDEA interface with the project 'testjpa' selected. The central editor window displays the `pom.xml` file for the 'testjpa' project. The code is as follows:

```
<?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>
```

The screenshot shows the IntelliJ IDEA interface with the project 'TTP3\_testjpa' selected. The central editor window displays the `TestjpaApplication.java` file. The code is as follows:

```
package com.isep.testjpa;

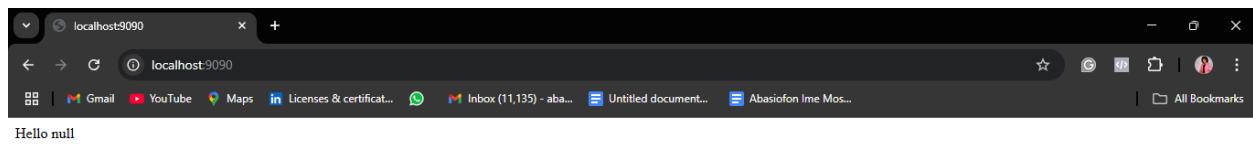
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TestjpaApplication {

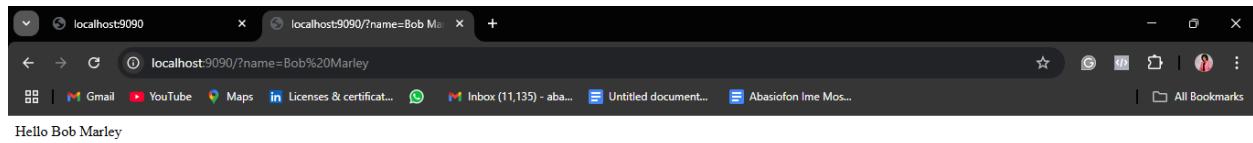
    public static void main(String[] args) {
        SpringApplication.run(TestjpaApplication.class, args);
    }
}
```

The bottom panel shows the 'Run' tool window with the process 'TestjpaApplication' running. The output log shows the application starting up:

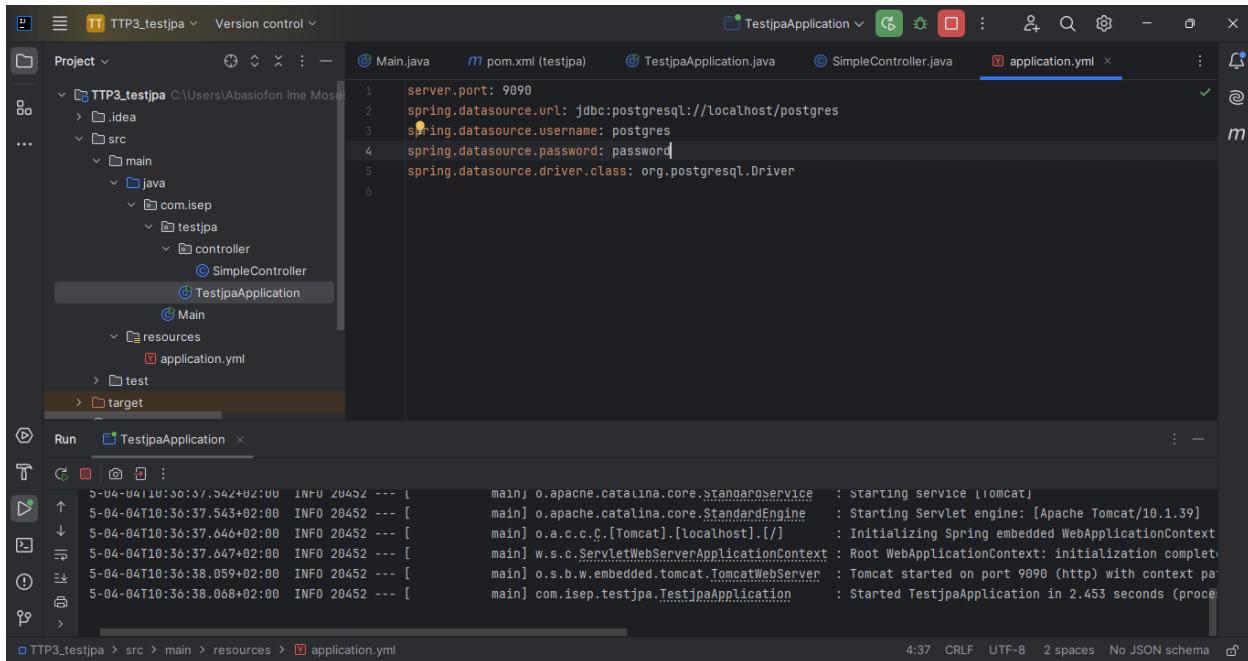
```
2025-04-04T10:37:48.981+02:00 INFO 21124 --- [           main] o.apache.catalina.core.StandardService   : Starting service Tomcat
2025-04-04T10:37:48.981+02:00 INFO 21124 --- [           main] o.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-04-04T10:37:49.035+02:00 INFO 21124 --- [           main] o.a.c.c.c.Tomcat@localhost.[]          : Initializing Spring embedded WebApplicationContext
2025-04-04T10:37:49.036+02:00 INFO 21124 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1 ms
2025-04-04T10:37:49.474+02:00 INFO 21124 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9890 (http) with context path ''
2025-04-04T10:37:49.481+02:00 INFO 21124 --- [           main] com.isep.testjpa.TestjpaApplication   : Started TestjpaApplication in 2.236 seconds (pr)
```



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



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

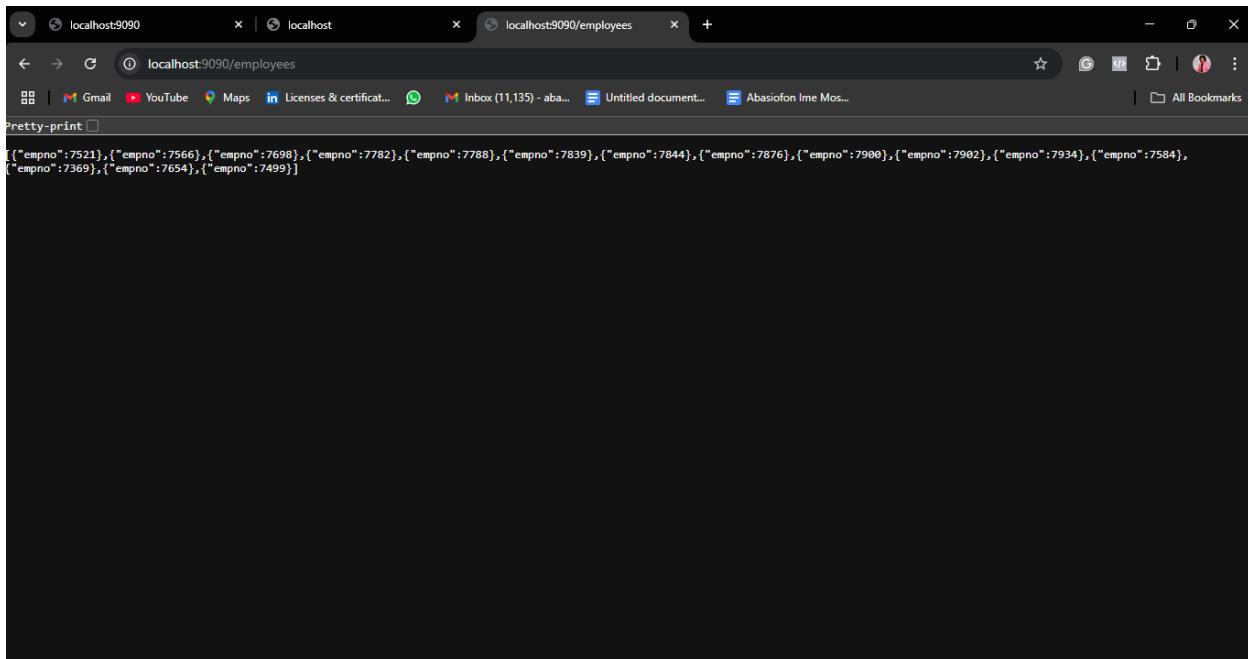


The screenshot shows the IntelliJ IDEA interface with the project 'TTP3\_testjpa' open. The 'application.yml' file is selected in the editor. The code defines a Spring datasource configuration:

```
server.port: 9090
spring.datasource.url: jdbc:postgresql://localhost/postgres
spring.datasource.username: postgres
spring.datasource.password: password
spring.datasource.driver.class: org.postgresql.Driver
```

The project structure on the left shows the directory tree: src/main/java/com/isep/testjpa/controller/TestjpaApplication.java, resources/application.yml, and target.

## Exercise III.3 : Read Data in your Database



The screenshot shows a web browser window with the URL 'localhost:9090/employees'. The page displays a JSON array of employee records:

```
[{"empno":7521}, {"empno":7566}, {"empno":7698}, {"empno":7782}, {"empno":7788}, {"empno":7839}, {"empno":7844}, {"empno":7876}, {"empno":7900}, {"empno":7902}, {"empno":7934}, {"empno":7584}, {"empno":7369}, {"empno":7654}, {"empno":7499}]
```

```

[{"empno":7521,"ename":"WARD","efirst":"PETER","job":"SALESMAN","mgr":7698,"sal":1250}, {"empno":7698,"ename":"BLAKE","efirst":"BOB","job":"MANAGER","mgr":7839,"sal":2850}, {"empno":7782,"ename":"CLARK","efirst":"JOHN","job":"MANAGER","mgr":7839,"sal":2450}, {"empno":7788,"ename":"SCOTT","efirst":"GUY","job":"ANALYST","mgr":7566,"sal":3000}, {"empno":7839,"ename":"KING","efirst":"GUY","job":"PRESIDENT","mgr":null,"sal":5000}, {"empno":7844,"ename":"TURNER","efirst":"PETER","job":"SALESMAN","mgr":7698,"sal":1500}, {"empno":7876,"ename":"ADAMS","efirst":"JOSEPH","job":"CLERK","mgr":7788,"sal":1100}, {"empno":7900,"ename":"JAMES","efirst":"ALAN","job":"CLERK","mgr":7698,"sal":950}, {"empno":7962,"ename":"FORD","efirst":"MARIA","job":"ANALYST","mgr":7566,"sal":3000}, {"empno":7934,"ename":"MILLER","efirst":"ALICE","job":"CLERK","mgr":7782,"sal":1300}, {"empno":7584,"ename":"OSIER","efirst":null,"job":null,"mgr":null,"sal":null}, {"empno":7369,"ename":"SMITH","efirst":"DONN","job":"CLERK","mgr":7982,"sal":7000}, {"empno":7654,"ename":"MARTIN","efirst":"JOE","job":"SALESMAN","mgr":7698,"sal":1400}, {"empno":7499,"ename":"ALLEN","efirst":"BOB","job":"SALESMAN","mgr":7698,"sal":1600}]

```

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

```

package com.isep.testjpa.controller;

import ...;

@RestController no usages
public class SimpleController {

    @Autowired 2 usages
    private EmpRepository empRepository;

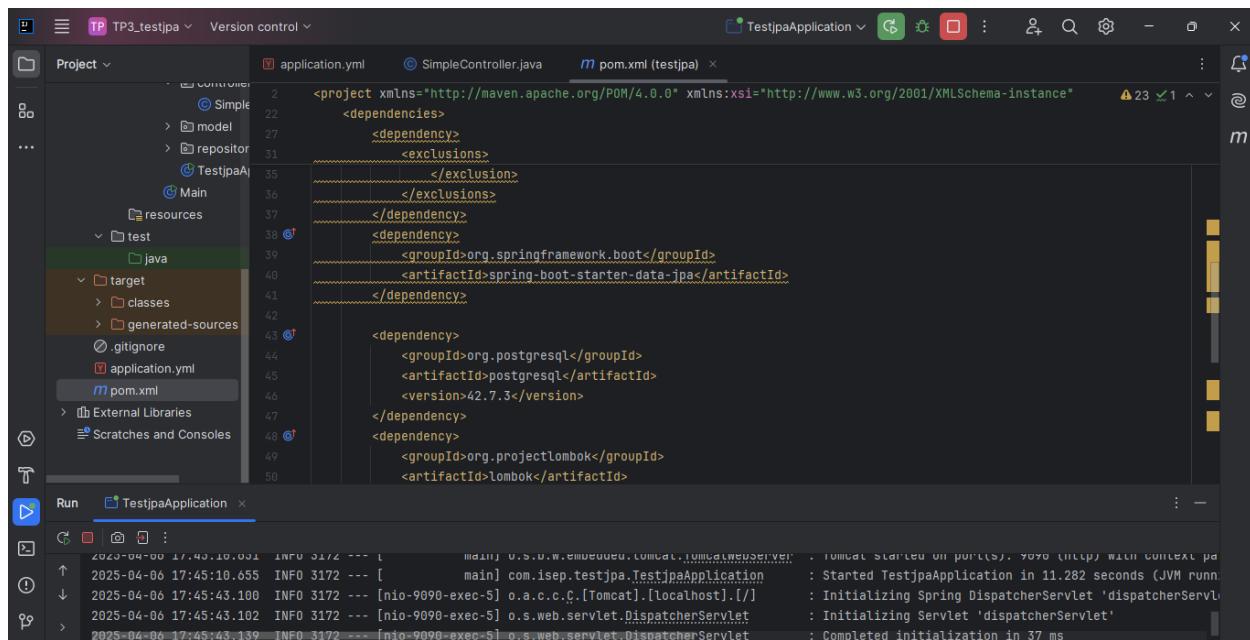
    @RequestMapping(value = "/", method = RequestMethod.GET) no usages
    public String hello(@Param("name") String name) { return "Hello " + name; }

    @RequestMapping(value = "/employees", method = RequestMethod.GET) no usages
    public List<Emp> getEmployees() { return empRepository.findAll(); }

    @PostMapping(value = "/employees") no usages
}

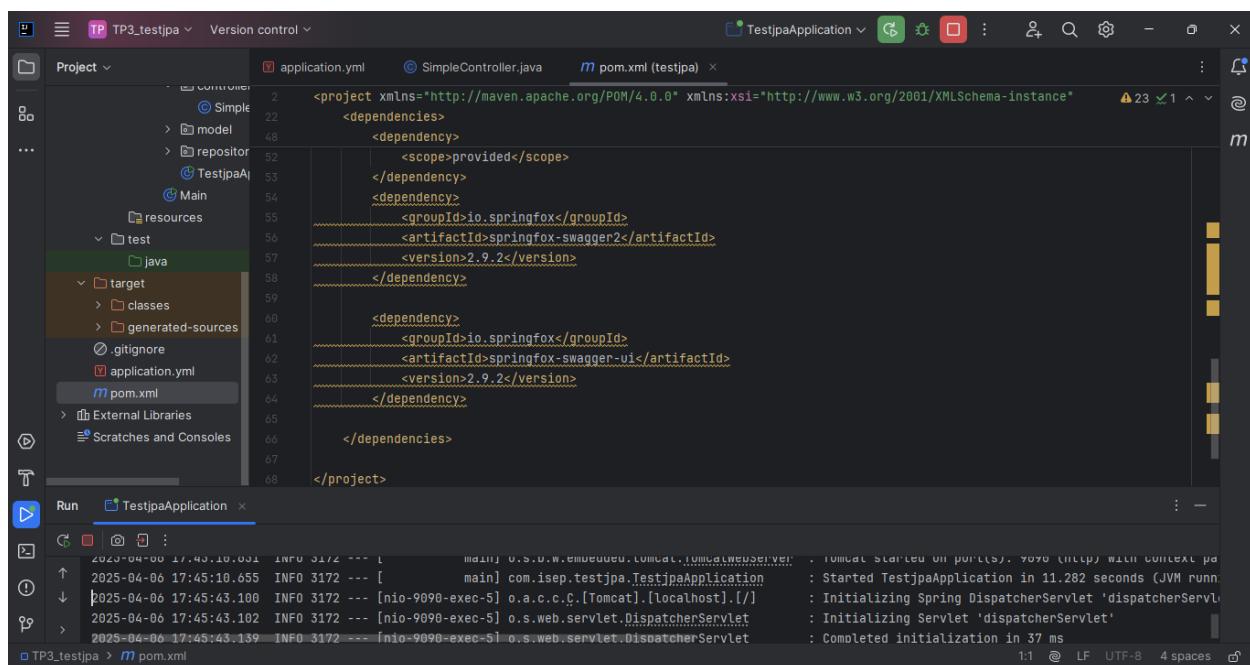
```

The screenshot shows the IntelliJ IDEA interface with the code editor open to SimpleController.java. The code defines a controller with two methods: a hello endpoint that returns a greeting with a parameter and an employees endpoint that returns a list of all employees from the database. The empRepository is autowired. The bottom part of the screenshot shows the terminal output of the application running on port 9090, displaying logs related to the application's startup and configuration.



```
<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 http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.isep.testjpa</groupId>
    <artifactId>TestjpaApplication</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <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>
            <version>1.18.20</version>
        </dependency>
    </dependencies>

```



```
<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 http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.isep.testjpa</groupId>
    <artifactId>TestjpaApplication</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <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>

```

The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure under "main".
  - src/main/java/com/isep/testjpa/configuration:** Contains the `SpringFoxConfig` class.
  - src/main/resources:** Contains the `pom.xml` file.
  - src/test/java:** Contains the `TestjpaApplication` class.
- Code Editor:** Displays the `SpringFoxConfig.java` file content:

```
package com.isep.testjpa.configuration;
import ...;
@Configuration no usages
@EnableSwagger2
public class SpringFoxConfig {
    @Bean no usages
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build();
    }
}
```
- Run Tab:** Shows the application is running as `TestjpaApplication`.
- Console Tab:** Shows log output from Tomcat starting up and initializing the application.
- Status Bar:** Shows the current time as 13:14, file encoding as CRLF, and character set as UTF-8.

The screenshot shows a web browser displaying the Swagger UI API documentation at `localhost:9090/swagger-ui.html`. The page includes:

- Header:** Shows the URL as `localhost:9090/swagger-ui.html#` and various browser tabs.
- Toolbar:** Includes a "Select a spec" dropdown set to "default".
- Section Headers:** "Api Documentation 1.0", "[ Base URL: localhost:9090/ ]", and links to "Terms of service" and "Apache 2.0".
- API Endpoints:** A list of endpoints:
  - `basic-error-controller` Basic Error Controller
  - `simple-controller` Simple Controller
- Models:** A section labeled "Models" with a right-pointing arrow.

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

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** TP3\_testjpa
- Code Editor:** SimpleController.java (selected), showing Java code for a REST controller.
- Run Log:** TestjpaApplication
 

```

2025-04-06 18:41:18.370 INFO 12444 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Context refreshed
2025-04-06 18:41:18.412 INFO 12444 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Found 1 custom documentation plugin(s)
2025-04-06 18:41:18.475 INFO 12444 --- [           main] s.d.s.w.s.ApiListingReferenceScanner : Scanning for api listing references
2025-04-06 18:41:18.844 INFO 12444 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 (http) with context path ''
2025-04-06 18:41:18.847 INFO 12444 --- [           main] com.isep.testjpa.TestjpaApplication : Started TestjpaApplication in 6.425 seconds (JVM running for 7.011)
```
- Bottom Status Bar:** 70:1, CRLF, UTF-8, 4 spaces

## Part IV : Connections Pool (Bonus)

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

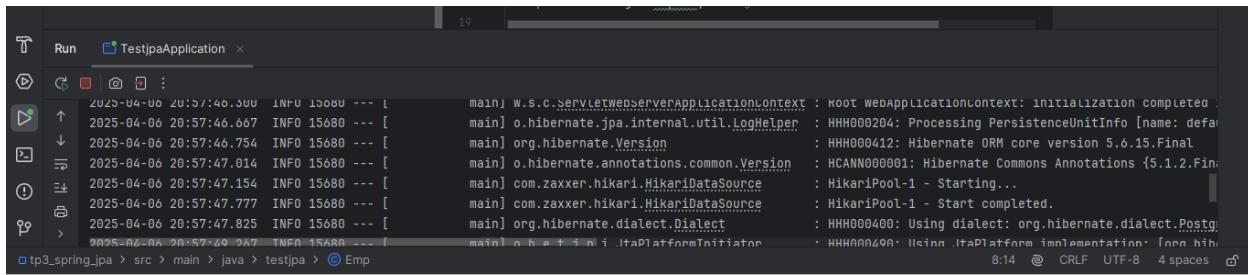
### 1. A Connection Originally Kept

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.

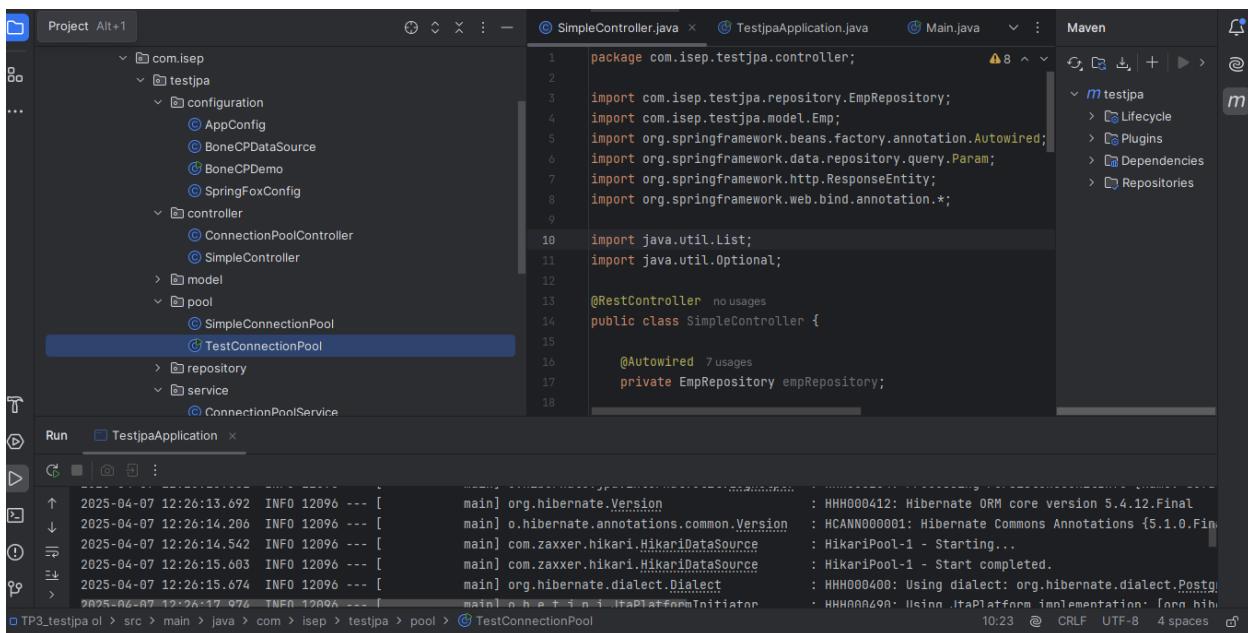
At the start of your server, you'll open a certain number of Connections and store them in a pool (a list as a stack for example). A client c1 will take a Connection do its request and once he has finished with it (at the moment of the close() statement) , he will replace it in the pool for the use of another client.

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



```
2025-04-06 20:57:40.300 INFO 15680 --- [main] W.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed : HHH000204: Processing PersistenceUnitInfo [name: default, ...]
2025-04-06 20:57:46.754 INFO 15680 --- [main] o.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.15.Final
2025-04-06 20:57:47.014 INFO 15680 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2025-04-06 20:57:47.154 INFO 15680 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-04-06 20:57:47.777 INFO 15680 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-04-06 20:57:47.825 INFO 15680 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2025-04-06 20:57:49.347 INFO 15680 --- [main] o.h.e.t.t.i.HibernatePlatformInitiator : HHH000490: Using .HibPlatform implementation: [org.hibernate.dialect.PostgreSQLDialect]
```

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



```
package com.isep.testjpa.controller;
import com.isep.testjpa.repository.EmpRepository;
import com.isep.testjpa.model.Emp;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.repository.query.Param;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;

@RestController
public class SimpleController {

    @Autowired
    private EmpRepository empRepository;
```

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

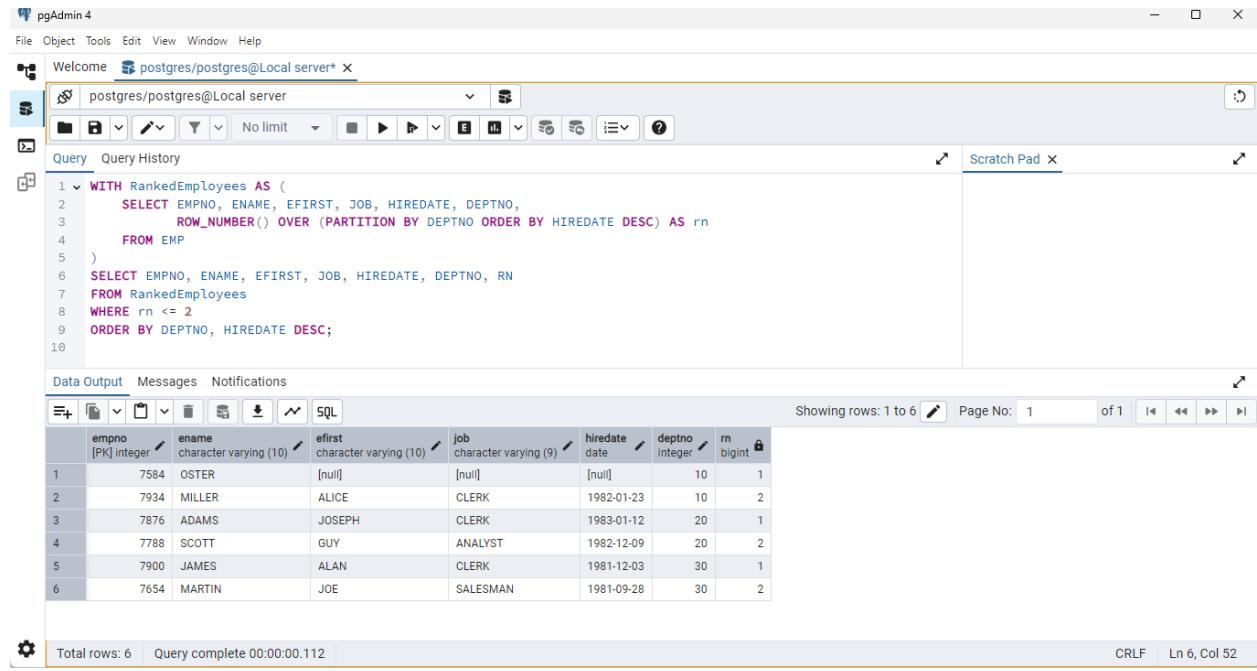
# Advanced Database - TP

## Advanced SQL

### Exercice 1. Analytics Queries . Window Queries

If you don't remember about Analytical / Windows Queries (very important!), please [read this chapter](#).

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



The screenshot shows the pgAdmin 4 interface. In the top-left, it says "pgAdmin 4" and "Welcome postgres/postgres@Local server". The main window has tabs for "Query" and "Scratch Pad". The "Query" tab contains a SQL script:

```
1 v WITH RankedEmployees AS (
2     SELECT EMPNO, ENAME, EFIRST, JOB, HIREDATE, DEPTNO,
3            ROW_NUMBER() OVER (PARTITION BY DEPTNO ORDER BY HIREDATE DESC) AS rn
4     FROM EMP
5 )
6     SELECT EMPNO, ENAME, EFIRST, JOB, HIREDATE, DEPTNO, RN
7     FROM RankedEmployees
8     WHERE rn <= 2
9     ORDER BY DEPTNO, HIREDATE DESC;
10
```

The "Data Output" tab shows the results of the query:

empno	[PK] integer	ename	character varying (10)	efirst	character varying (10)	job	character varying (9)	hiredate	date	deptno	integer	rn	bigint
1	7584	OSTER	[null]	[null]	[null]	CLERK	ANALYST	1982-01-23	1982-01-12	10	20	1	1
2	7934	MILLER	ALICE	JOSEPH	GUY	CLERK	ANALYST	1981-12-03	1981-09-28	20	30	2	2
3	7876	ADAMS	SCOTT	ALAN	JOE	CLERK	SALESMAN	1982-12-09	1983-01-12	10	20	1	1
4	7788	JAMES											
5	7900	MARTIN											
6	7654												

Total rows: 6 Query complete 00:00:00.112 CRLF Ln 6, Col 52

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".

```

SELECT EMNO, ENAME, SAL, HIREDATE,
       ROUND(AVG(SAL) OVER (ORDER BY HIREDATE
                             ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING), 2) AS Rolling_Avg_Salary
  FROM EMP;

```

	emno [PK] integer	ename character varying (10)	sal integer	hiredate date	rolling_avg_salary numeric
1	7369	SMITH	800	1980-12-17	1200.00
2	7499	ALLEN	1600	1981-02-20	1216.67
3	7521	WARD	1250	1981-02-22	1941.67
4	7566	JONES	2975	1981-04-02	2358.33
5	7698	BLAKE	2850	1981-05-01	2758.33
6	7782	CLARK	2450	1981-06-09	2266.67
7	7844	TURNER	1500	1981-09-08	1733.33
8	7654	MARTIN	1250	1981-09-28	2583.33
9	7839	KING	5000	1981-11-17	2400.00
10	7899	JAMES	650	1981-12-03	2000.00

Total rows: 15    Query complete 00:00:00.133    CRLF    Ln 5, Col 1

## Exercice 2. Index & Explain Plan

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

```

CREATE TABLE EMP_MEDIUM_TABLE
(
    EMPO INTEGER,
    MANAGER_ID INTEGER,
    DEPT_ID VARCHAR(10),
    GENDER VARCHAR(2) NOT NULL,
    NAME VARCHAR(1000));

```

```

INSERT INTO EMP_MEDIUM_TABLE
SELECT
    S AS empo,
    ROUND(random()*100) AS manager_id,
    ROUND(random()*10) AS dept_id,
    (ARRAY['M','F'])[round(random())+1] AS gender,
    gen_random_uuid() AS name_hashed
FROM generate_series(1,5000) AS S;

```

INSERT 0 5000

Query returned successfully in 206 msec.

2. The goal of all exercise 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.

The screenshot shows a PostgreSQL query editor interface. The top pane displays the SQL code:

```
65  /x
66  CREATE TABLE PROJECT_EMP_MEDIUM_TABLE
67      (PROJECTNO integer,
68       EMPNO integer);
69
70  INSERT INTO PROJECT_EMP_MEDIUM_TABLE
71  SELECT      ROUND(random()*100) AS PROJECTNO,
72             ROUND(random()*100) AS EMPNO
73  FROM generate_series(1,5000) as S;
74  */
75
76  SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
77
```

The bottom pane shows the results of the last query:

	gender	count
1	F	16
2	M	23

Total rows: 2 | Query complete 00:00:00.087

3.1. Use EXPLAIN plan to analyze the query:

Keep the Execution Plan for this query

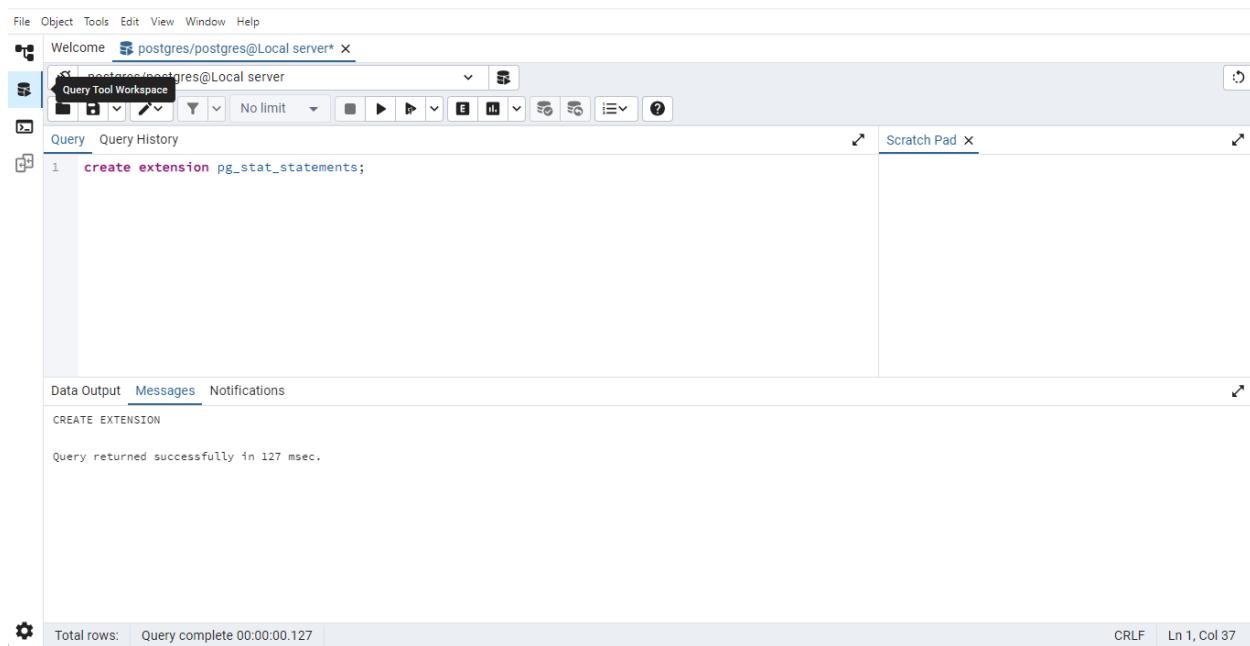
```

69
70     INSERT INTO PROJECT_EMP_MEDIUM_TABLE
71     SELECT      ROUND(random()*100) AS PROJECTNO,
72                 ROUND(random()*100) AS EMPNO
73     FROM generate_series(1,5000) as S;
74   */
75
76   /* 
77     SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
78   */
79
80   EXPLAIN SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
81
Data Output  Messages  Notifications
Showing rows: 1 to 6  Page No: 1
QUERY PLAN
text
1 GroupAggregate (cost=115.53..115.84 rows=2 width=10)
2  Group Key: gender
3    -> Sort (cost=115.53..115.63 rows=39 width=2)
4      Sort Key: gender
5        -> Seq Scan on emp_medium_table (cost=0.00..114.50 rows=39 ...
6          Filter: (manager_id = 7)

Total rows: 6 | Query complete 00:00:00.066

```

### 3.2. Activate stats



The screenshot shows the pgAdmin 4 interface with the following details:

- File Bar:** File, Object, Tools, Edit, View, Window, Help.
- Toolbar:** Welcome, postgres/postgres@Local server\*, Query Tool Workspace, Query History, Scratch Pad.
- Query Editor:** Contains the SQL command: `create extension pg_stat_statements;`.
- Data Output Tab:** Shows the execution results:
 

```

CREATE EXTENSION
Query returned successfully in 127 msec.

```
- Status Bar:** Total rows: 0, Query complete 00:00:00.127, CRLF, Ln 1, Col 37.

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”.

The screenshot shows the pgAdmin 4 interface. In the top-left, there's a navigation bar with File, Object, Tools, Edit, View, Window, Help. Below it is a toolbar with various icons. The main area has two tabs: 'Query' (selected) and 'Scratch Pad'. The 'Query' tab contains the following SQL code:

```
1 v SELECT * FROM pg_stat_statements
2 WHERE query like '%group by gender' and query not like 'EXPLAIN'
3
4 /*
5  SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
6 */
```

The 'Data Output' tab below shows the results of the query:

plan_time_precision	calls	total_exec_time	min_exec_time	max_exec_time	mean_exec_time	stddev_exec_time	rows	shared_blkshits	shared_blkssread	shared_blkssdirty	shared_blkssused
1	0	10	5.9207	0.2737	2.2405	0.59207	0.5538516174030731	20	468	52	0

At the bottom of the interface, it says "Total rows: 1" and "Query complete 00:00:00.092".

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.

File Object Tools Edit View Window Help

Welcome postgres/postgres@Local server\*

postgres/postgres@Local server No limit

Query History Scratch Pad

```

1 /* 
2  SELECT * FROM pg_stat_statements
3  WHERE query like '%group by gender' and query not like 'EXPLAIN%'
4  */
5 /*
6  SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
7 */
8
9 create index MANAGER_ID_GENDER_INDEX ON EMP_MEDIUM_TABLE(MANAGER_ID, GENDER);

```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 81 msec.

Total rows: Query complete 00:00:00.081 CRLF Ln 1, Col 3

✓ Query returned successfully in 81 msec. X

## 5. Reset the stats:

```
select pg_stat_statements_reset();
```

File Object Tools Edit View Window Help

Welcome postgres/postgres@Local server\*

postgres/postgres@Local server No limit

Query History Scratch Pad

```

3 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
4 */
5 /*
6  SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
7 */
8
9
10 /*
11  create index MANAGER_ID_GENDER_INDEX ON EMP_MEDIUM_TABLE(MANAGER_ID, GENDER);
12 */
13
14 select pg_stat_statements_reset();

```

Data Output Messages Notifications

pg_stat_statements_reset	timestamp with time zone
1	2025-03-11 15:29:38.388782+01

Showing rows: 1 to 1 Page No: 1 of 1

Total rows: 1 Query complete 00:00:00.077 CRLF Ln 5, Col 1

✓ Successfully run. Total query runtime: 77 msec. 1 rows affected. X

If you run:

```
SELECT * FROM pg_stat_statements
```

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

You should see nothing

The screenshot shows the pgAdmin 4 interface. The top menu bar includes File, Object, Tools, Edit, View, Window, Help. A toolbar with various icons is at the top. The main window has tabs for Query and Query History, with the Query tab selected. Below the tabs is a code editor containing the following SQL:

```
9
10 /*
11  create index MANAGER_ID_GENDER_INDEX ON EMP_MEDIUM_TABLE(MANAGER_ID, GENDER);
12 */
13
14 /*
15  select pg_stat_statements_reset();
16 */
17
18 SELECT * FROM pg_stat_statements
19 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
20
```

Below the code editor is a Data Output pane showing the structure of the pg\_stat\_statements table:

userid	dbid	toplevel	queryid	query	plans	total_plan_time	min_plan_time	max_plan_time	mean_plan_time	stddev_plan_time	calls	total_exec_time	min_exec_time
old	lock	old	lock	bigint	lock	double precision	lock	double precision	double precision				

At the bottom of the interface, there is a status bar with the message "Successfully run. Total query runtime: 85 msec. 0 rows affected." and other details like "Total rows: 0" and "Query complete 00:00:00.085".

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.

```

13
14 /*
15 select pg_stat_statements_reset();
16 */
17
18 /*
19 SELECT * FROM pg_stat_statements
20 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
21
22
23 SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
24

```

gender	count
F	16
M	23

Showing rows: 1 to 2 Page No: 1 of 1

✓ Successfully run. Total query runtime: 94 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 107 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 95 msec. 2 rows affected.

Total rows: 2 Query complete 00:00:00.094 CRLF Ln 18, Col 3

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

```

13
14 /*
15 select pg_stat_statements_reset();
16 */
17
18
19 /*
20 SELECT * FROM pg_stat_statements
21 WHERE query like '%group by gender%' and query not like 'EXPLAIN%'
22
23 */
24 SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;

```

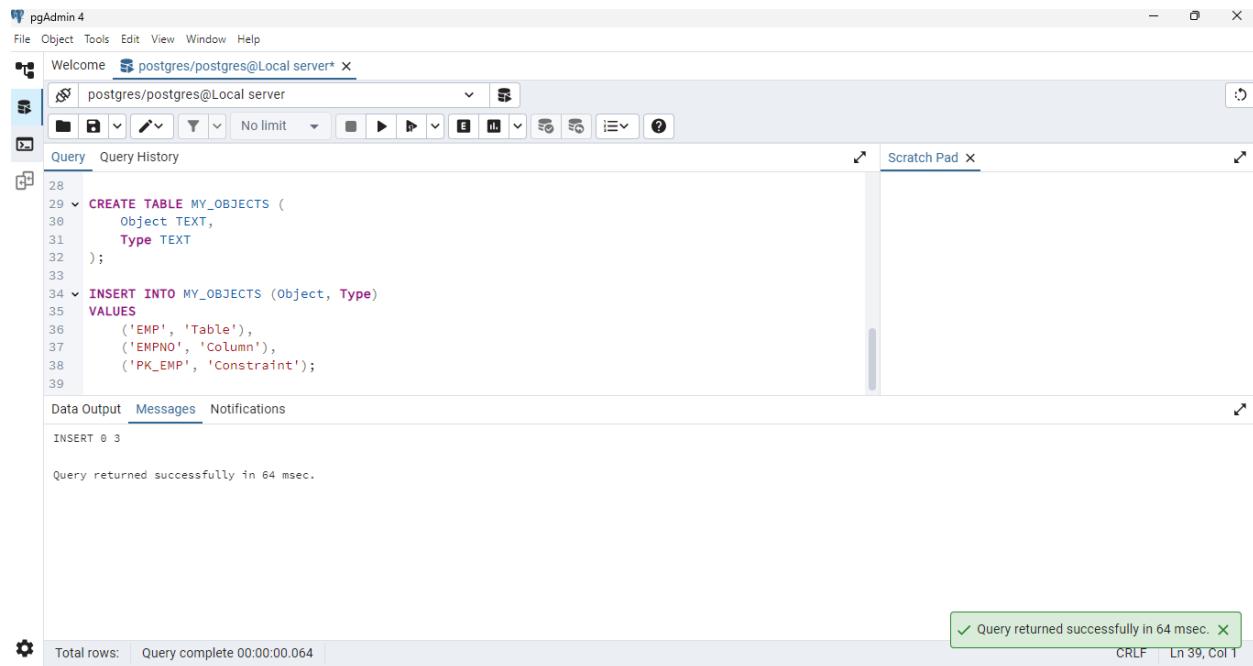
in_time	calls	total_exec_time	min_exec_time	max_exec_time	mean_exec_time	stddev_exec_time	rows	shared_blk_hit	shared_blk_read	shared_blk_dirtied	shared_blk_written
0	10	0.7745	0.0313	0.2757	0.07745	0.06782463048185372	20	28	2	0	

Showing rows: 1 to 1 Page No: 1 of 1

Total rows: 1 Query complete 00:00:00.094 CRLF Ln 21, Col 1

- **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



The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". The main window has a toolbar at the top with various icons. Below the toolbar, there are two tabs: "Query" (which is selected) and "Scratch Pad". The "Query" tab contains the following SQL code:

```
28 v CREATE TABLE MY_OBJECTS (
29     Object TEXT,
30     Type TEXT
31 );
32
33
34 v INSERT INTO MY_OBJECTS (Object, Type)
35 VALUES
36     ('EMP', 'Table'),
37     ('EMPNO', 'Column'),
38     ('PK_EMP', 'Constraint');
39
```

Below the code, there are three tabs: "Data Output" (selected), "Messages", and "Notifications". The "Data Output" tab shows the message "INSERT 0 3". Underneath it, it says "Query returned successfully in 64 msec.". At the bottom of the window, there are status bars for "Total rows:" and "Query complete 00:00:00.064". A green success message box is visible in the bottom right corner.

## 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.

The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4". The menu bar includes "File", "Object", "Tools", "Edit", "View", "Window", and "Help". A toolbar with icons for "File", "Edit", "Run", "Stop", and "Close" is visible. The main window has a tab titled "Welcome postgres/postgres@Local server". The status bar at the bottom shows "Microsoft Windows [Version 10.0.26100.3323] (c) Microsoft Corporation. All rights reserved." and "C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime>C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime\psql.exe" "host=localhost port=5432 dbname=postgres user=postgres sslmode=prefer connect\_timeout=10" 2>>61 psql (17.4)". A message box is open with the text: "WARNING: Console code page (437) differs from Windows code page (1252) 8-bit characters might not work correctly. See psql reference page "Notes for Windows users" for details. Type "help" for help." The command prompt shows "postgres=#".

## 2. On the CLI: Launch a

```
SELECT * FROM EMP;
```

The screenshot shows the pgAdmin 4 interface with the same connection details as the previous screenshot. The main window now displays the results of the "SELECT \* FROM EMP;" query. The results are presented in a table with 14 rows and 11 columns. The columns are: empno, ename, efirst, job, mgr, hiredate, sal, comm, tel, and deptno. The data includes entries for employees like SMITH, ALLEN, WARD, JONES, MARTIN, BLAKE, CLARK, SCOTT, KING, TURNER, ADAMS, JAMES, FORD, and MILLER, each with their respective details.

empno	ename	efirst	job	mgr	hiredate	sal	comm	tel	deptno
7369	SMITH	JOHN	CLERK	7902	1980-12-17	800		0619545243	20
7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0619547243	30
7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0619545247	30
7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975		0619545456	20
7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0619545784	30
7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850		0619545254	30
7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450		0619545245	10
7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000		0619545249	20
7839	KING	GUY	PRESIDENT	0	1981-11-17	5000		0619545241	10
7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0619548243	30
7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100		0619565243	20
7900	JAMES	ALAN	CLERK	7698	1981-12-03	950		0619545564	30
7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000		0619785243	20
7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300		0619545243	10
(14 rows)									

postgres=#

## Exercice 5. Transaction Part 1 - Beginner

### 0. Open 2 sql clients on the same Database.

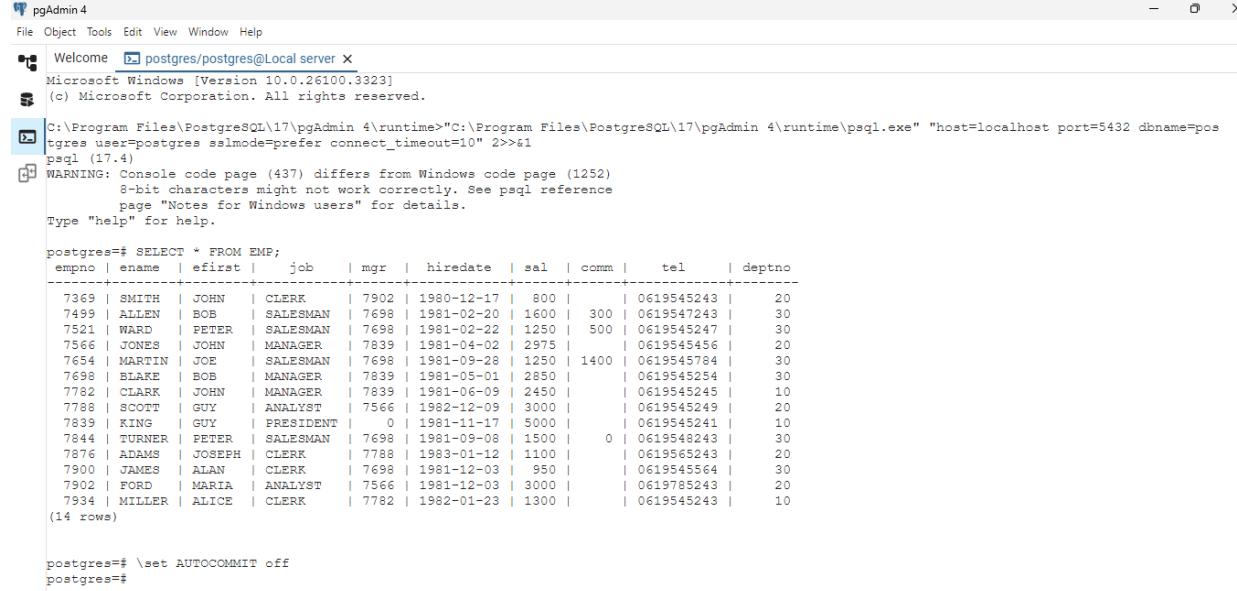
For this Exercise, you should have 2 clients either:

- 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
```



```
pgAdmin 4
File Object Tools Edit View Window Help
Welcome postgres/postgres@Local server X
Microsoft Windows [Version 10.0.26100.3323]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime>"C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime\psql.exe" "host=localhost port=5432 dbname=postgres user=postgres sslmode=prefer connect_timeout=10" 2>>61
psql (17.4)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.

postgres=# SELECT * FROM EMP;
   empno | ename  |  job   |  mgr  | hiredate | sal | comm |  tel   | deptno
-----+-----+-----+-----+-----+-----+-----+-----+
    7369 | SMITH  | CLERK | 7902 | 1980-12-17 |  800 |       | 0619545243 |    20
    7499 | ALLEN  | BOB    | 7698 | 1981-02-20 | 1600 | 300  | 0619547243 |    30
    7521 | WARD   | PETER  | 7698 | 1981-02-22 | 1250 | 500  | 0619545247 |    30
    7566 | JONES  | JOHN   | 7839 | 1981-04-02 | 2975 |       | 0619545456 |    20
    7654 | MARTIN | JOE    | 7698 | 1981-09-28 | 1250 | 1400 | 0619545784 |    30
    7698 | BLAKE  | BOB    | 7839 | 1981-05-01 | 2850 |       | 0619545254 |    30
    7782 | CLARK  | JOHN   | 7839 | 1981-06-09 | 2450 |       | 0619545245 |    10
    7788 | SCOTT  | GUY    | ANALYST | 7566 | 1982-12-09 | 3000 |       | 0619545249 |    20
    7839 | KING   | GUY    | PRESIDENT | 0  | 1981-11-17 | 5000 |       | 0619545241 |    10
    7844 | TURNER | PETER  | SALESMAN | 7698 | 1981-09-08 | 1500 | 0  | 0619548243 |    30
    7876 | ADAMS  | JOSEPH | CLERK  | 7788 | 1983-01-12 | 1100 |       | 0619565243 |    20
    7900 | JAMES  | ALAN   | CLERK  | 7698 | 1981-12-03 |  950 |       | 0619545564 |    30
    7902 | FORD   | MARIA  | ANALYST | 7566 | 1981-12-03 | 3000 |       | 0619785243 |    20
    7934 | MILLER | ALICE  | CLERK  | 7782 | 1982-01-23 | 1300 |       | 0619545243 |    10
(14 rows)

postgres=# \set AUTOCOMMIT off
postgres=#
```

**1.2 On the UI, untick that option:**



The screenshot shows the pgAdmin 4 interface. In the top navigation bar, the title is "pgAdmin 4" and the connection is "postgres/postgres@Local server". The main window has tabs for "Query" and "Query History". The "Query" tab contains the following SQL script:

```

32 );
33
34 INSERT INTO MY_OBJECTS (Object, Type)
35 VALUES
36 ('EMP', 'Table'),
37 ('EMPNO', 'Column'),
38 ('PK_EMP', 'Constraint');
39 */
40
41
42
43

```

Below the script, the "Data Output" tab is selected, showing the output of the "INSERT" command:

```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 64 msec.

Total rows: Query complete 00:00:00.064 CRLF Ln 42, Col 1

```

## 2. On the first client: Launch:

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

EMPNO	FNAME	MIDDLEINIT	LNAME	SAL	HIREDATE	COMM	DEPTNO		
7837	KING		CORY	5000	1981-12-01	1000	10		
7900	JAMES	A	ALAN	CLERK	7698	1981-12-03	950	0619545564	30
7902	FORD		MARIA	ANALYST	7566	1981-12-03	3000	0619785243	20
7934	MILLER	A	ALICE	CLERK	7782	1982-01-23	1300	0619545243	10

(14 rows)

```
postgres=# \set AUTOCOMMIT off
postgres=# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369
postgres=#

```

## 3. On the first client, launch:

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

```
^
postgres=!\# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369
postgres=!\#
```

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.

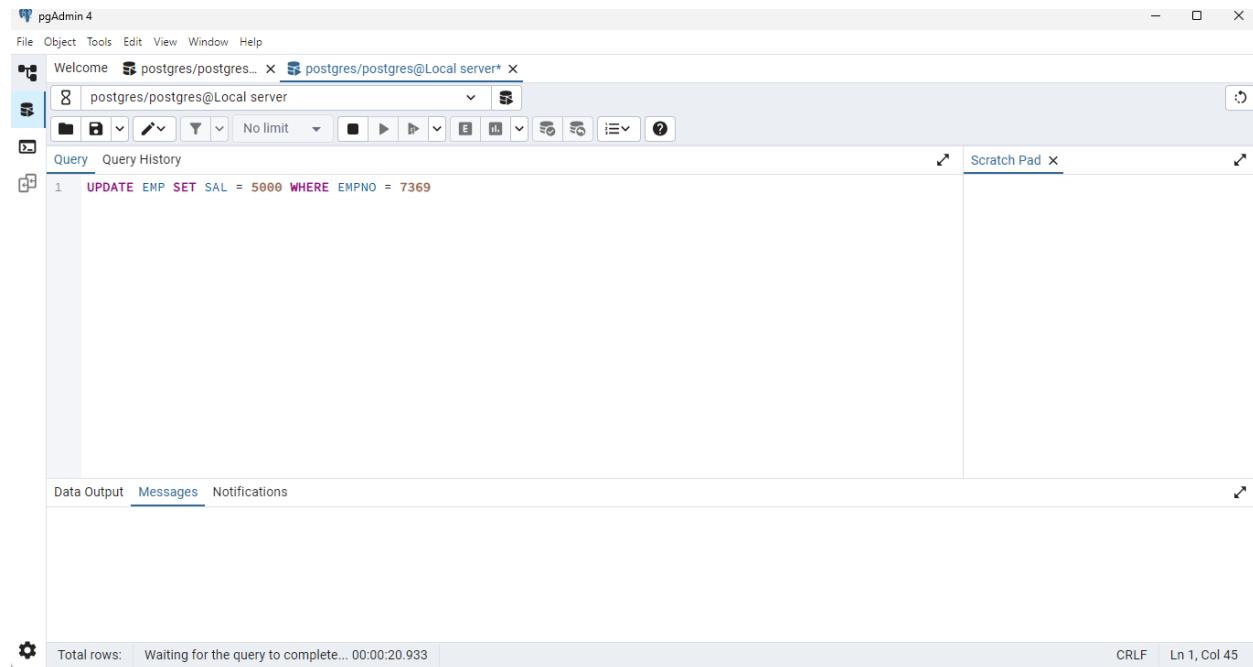
4. On the second client, launch:

```
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 ?

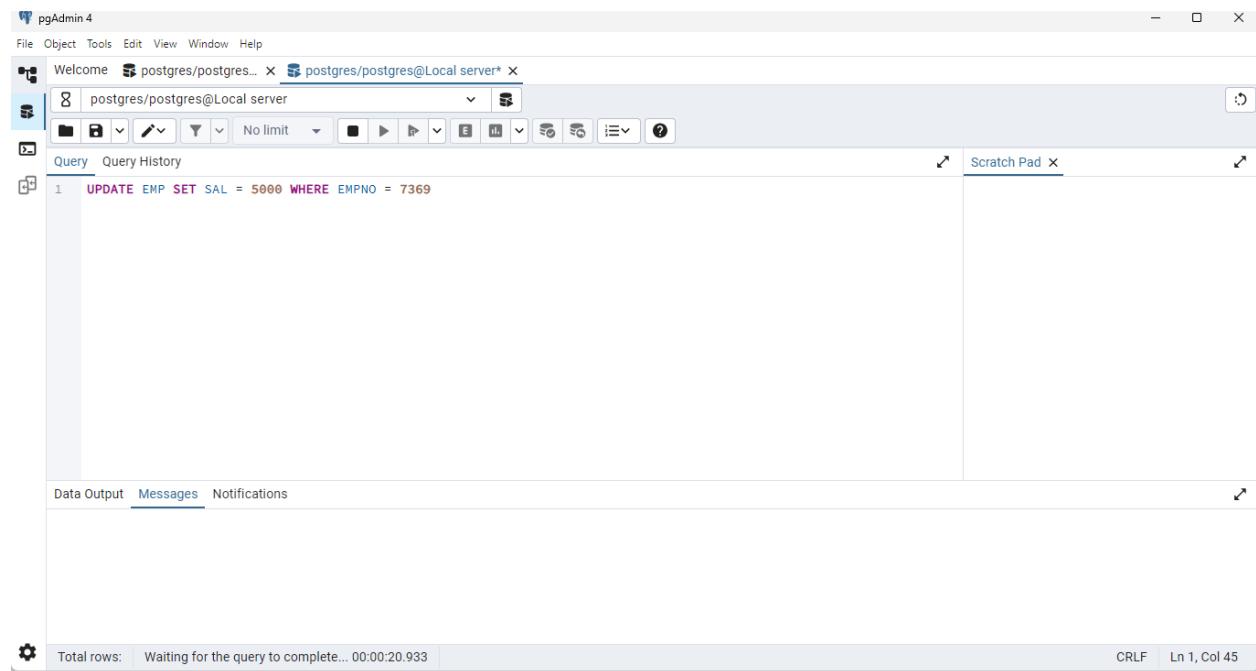
What is the name of this mechanism ? (hint: l\*\*k)

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
```

```
C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime>"C:\Program Files\PostgreSQL\17\pgAdmin 4\runtime\psql.exe" "host=localhost port=5432 dbname=postgres user=postgres salmode=prefer connect_timeout=10" 2>>61
psql (17.4)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \set AUTOCOMMIT off
postgres# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369
postgres# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369
postgres# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369
postgres# COMMIT
postgres#
```

## 7. On the second client, launch:

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

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

Data Output Messages Notifications

```
UPDATE 1
Query returned successfully in 83 msec.
```

Total rows: Query complete 00:00:00.083 CRLF Ln 1, Col 46

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;
```

The screenshot shows the pgAdmin 4 interface. In the top navigation bar, 'File', 'Object', 'Tools', 'Edit', 'View', 'Window', and 'Help' are visible. The main window has a title bar 'Welcome postgres/postgres@Local server\*' and a toolbar with various icons. A query editor window titled 'Query' contains the following SQL code:

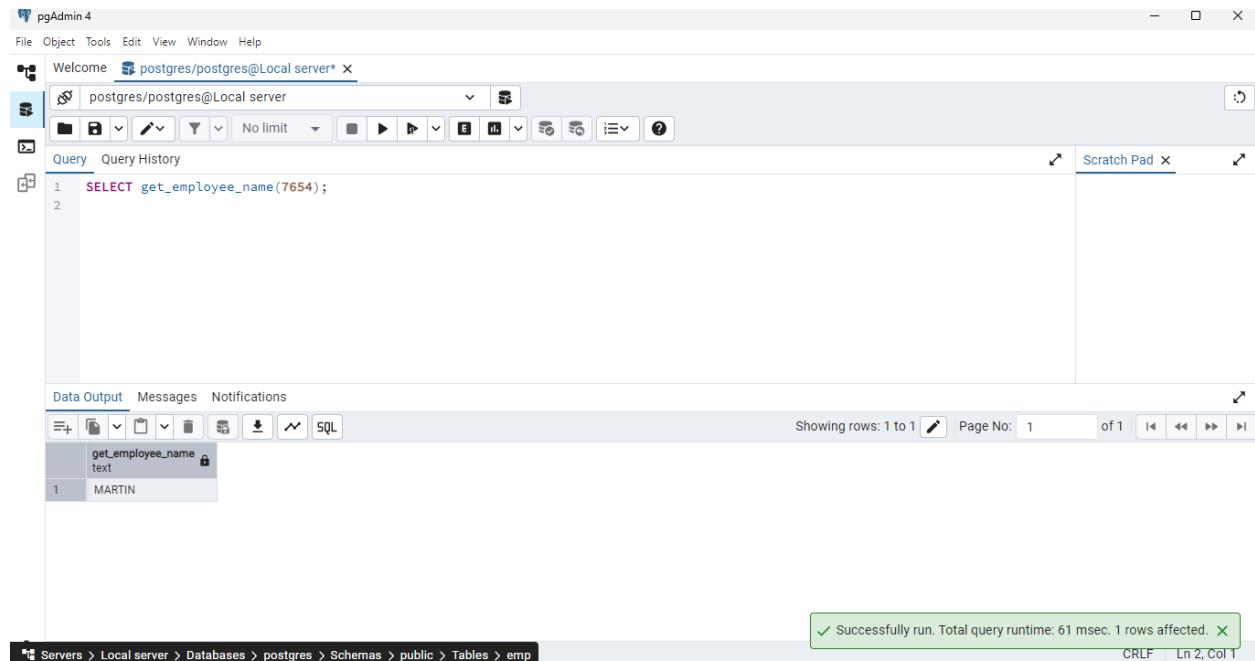
```
1 CREATE OR REPLACE FUNCTION get_employee_name(p_empno INT)
2 RETURNS TEXT AS $$ 
3 DECLARE
4     v_ename TEXT;
5 BEGIN
6     SELECT ename
7     INTO v_ename
8     FROM emp
9     WHERE empno = p_empno;
10
11    -- Handle case where employee is not found
12    IF NOT FOUND THEN
13        RETURN 'Employee Not Found';
14    END IF;
15
16    RETURN v_ename;
17 END;
18 $$ LANGUAGE plpgsql;
19
```

Below the code, tabs for 'Data Output', 'Messages', and 'Notifications' are shown. Under 'Messages', it says 'CREATE FUNCTION'. At the bottom, a message box indicates 'Query returned successfully in 87 msec.' with a green checkmark icon.

## 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;

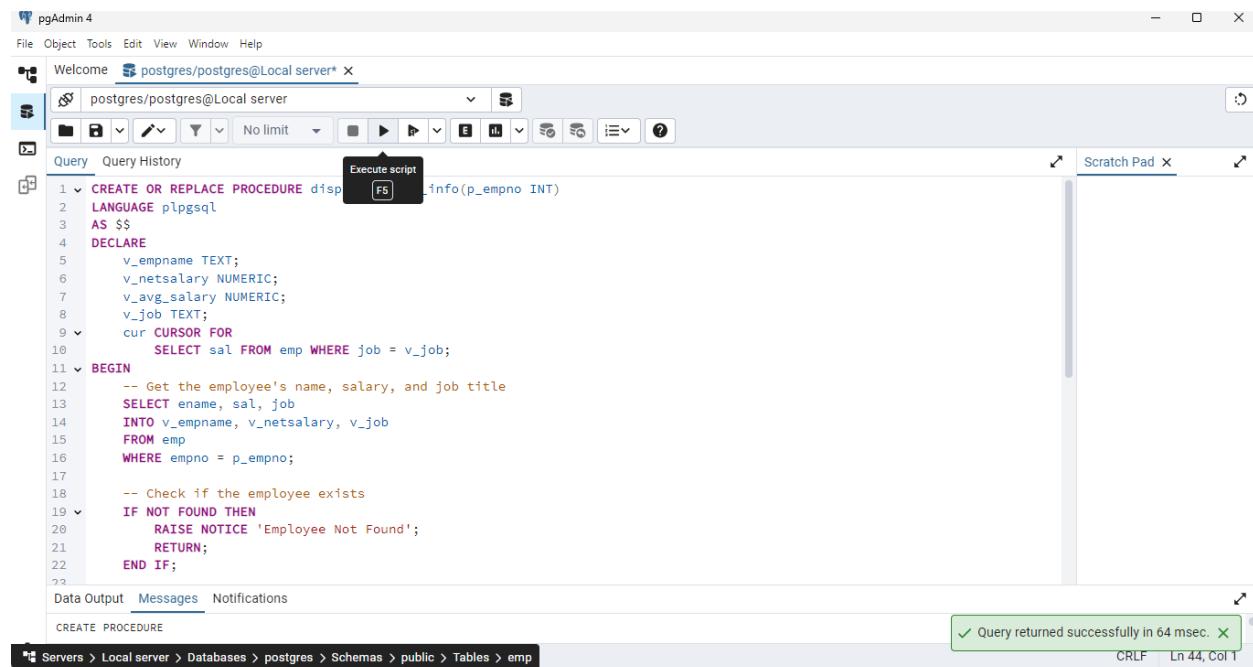
    -- Cursor to fetch all salaries for the same job

    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;
$$;

```



The screenshot shows the pgAdmin 4 interface with a query editor window. The query is a CREATE OR REPLACE PROCEDURE named 'disp'. The procedure takes a parameter 'p\_empno' of type INT. It declares variables for employee name (v\_empname), net salary (v\_netsalary), average salary (v\_avg\_salary), and job title (v\_job). It then creates a cursor 'cur' that selects salary from the 'emp' table where the job matches the input job. It begins a BEGIN block to handle the cursor. Inside the BEGIN block, it selects the employee's name, salary, and job title into the declared variables. It then checks if the employee exists using an IF NOT FOUND THEN block. If found, it raises a notice 'Employee Not Found' and returns. Otherwise, it ends the IF block. The status bar at the bottom right indicates the query was executed successfully in 64 msec.

```

CREATE OR REPLACE PROCEDURE disp(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;
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;

$$;
```

pgAdmin 4

Welcome **postgres/postgres@Local server\***

File Object Tools Edit View Window Help

postgres/postgres@Local server No limit

Query History

```

22 -- calculate the average salary for employees with the same job
23   SELECT AVG(sal)
24     INTO v_avg_salary
25   FROM emp
26  WHERE job = v_job;
27
28  -- Determine the new salary
29  IF v_old_salary >= v_avg_salary THEN
30    v_new_salary := v_old_salary * 1.10; -- Increase by 10%
31  ELSE
32    v_new_salary := v_avg_salary; -- Set to the average salary
33  END IF;
34
35  -- Update the employee's salary
36  UPDATE emp
37    SET sal = v_new_salary
38   WHERE empno = p_empno;
39
40  -- Display the initial and resulting salary
41  RAISE NOTICE 'Employee %: Initial Salary = %, New Salary = %', p_empno, v_old_salary, v_new_salary;
42
43 END;
44 $$;
```

Data Output Messages Notifications

CREATE PROCEDURE

Servers > Local server > Databases > postgres > Schemas > public > Tables > emp

✓ Query returned successfully in 130 msec. CRLF Ln 44, Col 1

`CALL update_salary(7654);`

pgAdmin 4

Welcome **postgres/postgres@Local server\***

File Object Tools Edit View Window Help

postgres/postgres@Local server No limit

Query History

```

1 CALL update_salary(7654);
2
```

Data Output Messages Notifications

NOTICE: Employee 7654: Initial Salary = 1250, New Salary = 1400.0000000000000000  
CALL

Query returned successfully in 70 msec.

✓ Query returned successfully in 70 msec. CRLF Ln 2, Col 1

Servers > Local server > Databases > postgres > Schemas > public > Tables > emp

## 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;

$$;
```

pgAdmin 4

Welcome postgres/postgres... × postgres/postgres@Local server\*

Query History

```

1 ✓ CREATE OR REPLACE PROCEDURE calculate_bonus(p_empno INT)
2 LANGUAGE plpgsql
3 AS $$
4 DECLARE
5     v_ename TEXT;
6     v_job TEXT;
7     v_salary NUMERIC;
8     v_comm NUMERIC;
9     v_bonus NUMERIC;
10    BEGIN
11        -- Fetch employee details
12        SELECT ename, job, sal, comm
13        INTO v_ename, v_job, v_salary, v_comm
14        FROM emp
15        WHERE empno = p_empno;
16
17        -- Check if employee exists
18        IF NOT FOUND THEN
19            RAISE NOTICE 'Employee Not Found';
20            RETURN;
21        END IF;

```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 51 msec.

Total rows: Query complete 00:00:00.051

✓ Query returned successfully in 51 msec. CRLF Ln 45, Col 1

CALL calculate\_bonus(7654);

pgAdmin 4

Welcome postgres/postgres... × postgres/postgres@Local server\*

Query History

Execute script F5

```

1 CALL calculate_bonus(7654);
2

```

Data Output Messages Notifications

NOTICE: Bonus for Employee MARTIN (SALESMAN): 2800

CALL

Query returned successfully in 54 msec.

✓ Query returned successfully in 54 msec. CRLF Ln 2, Col 1

Total rows: Query complete 00:00:00.054

## 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;  
$$;
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar says "pgAdmin 4". The main area displays the following SQL code:

```

1 CREATE OR REPLACE PROCEDURE UpdateCommission()
2 LANGUAGE plpgsql
3 AS $$
4 DECLARE
5     emp_cursor CURSOR FOR
6         SELECT empno, sal, comm FROM emp FOR UPDATE; -- Lock the rows
7     v_empno INT;
8     v_sal NUMERIC;
9     v_new_comm NUMERIC;
10    BEGIN
11        -- Open cursor
12        OPEN emp_cursor;
13
14        -- Loop through employees
15        LOOP
16            FETCH emp_cursor INTO v_empno, v_sal, v_new_comm;
17            EXIT WHEN NOT FOUND;
18
19            -- Determine new commission based on salary
20            UPDATE emp SET comm = v_new_comm WHERE empno = v_empno;
21        END LOOP;
22    END;
23
24

```

The status bar at the bottom shows "Total rows: 0" and "Query complete 00:00:00.050". A green message box in the bottom right corner says "Query returned successfully in 50 msec. CRLF Ln 41, Col 1".

`CALL UpdateCommission();`

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar says "pgAdmin 4". The main area displays the following SQL code:

```

1 CALL UpdateCommission();
2

```

The status bar at the bottom shows "Total rows: 0" and "Query complete 00:00:00.050". The message area below the status bar shows several "NOTICE" messages indicating employee updates:

- NOTICE: Updated Employee 7499: New COMM = 1200
- NOTICE: Updated Employee 7521: New COMM = 1200
- NOTICE: Updated Employee 7566: New COMM = 1500
- NOTICE: Updated Employee 7698: New COMM = 1500
- NOTICE: Updated Employee 7782: New COMM = 1500
- NOTICE: Updated Employee 7788: New COMM = 1500
- NOTICE: Updated Employee 7839: New COMM = 1500
- NOTICE: Updated Employee 7844: New COMM = 1200
- NOTICE: Updated Employee 7876: New COMM = 1200
- NOTICE: Updated Employee 7900: New COMM = 800

A green message box in the bottom right corner says "Query returned successfully in 50 msec. CRLF Ln 2, Col 1".

## 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;
$$;
```

pgAdmin 4

Welcome postgres/postgres... × postgres/postgres@Local server\*

Query History

```

1 CREATE OR REPLACE PROCEDURE Top5HighLowSalaries()
2 LANGUAGE plpgsql
3 AS $$
4 DECLARE
5     v_empno INT;
6     v_ename TEXT;
7     v_job TEXT;
8     v_sal NUMERIC;
9
10    -- Cursors for top 5 highest and lowest salaries
11    high_salary_cursor CURSOR FOR
12        SELECT empno, ename, job, sal FROM emp ORDER BY sal DESC LIMIT 5;
13
14    low_salary_cursor CURSOR FOR
15        SELECT empno, ename, job, sal FROM emp ORDER BY sal ASC LIMIT 5;
16
17    BEGIN
18        -- Display Top 5 Highest Salaries
19        RAISE NOTICE 'Top 5 Employees with Highest Salaries:';
20        OPEN high_salary_cursor;
21
22    LOOP
23        FETCH high_salary_cursor INTO v_empno, v_ename, v_job, v_sal;

```

Data Output Messages Notifications

CREATE PROCEDURE

Total rows: Query complete 00:00:00.048

✓ Query returned successfully in 48 msec. CRLF Ln 42, Col 1

CALL Top5HighLowSalaries();

pgAdmin 4

Welcome postgres/postgres... × postgres/postgres@Local server\*

Query History

```

1 CALL Top5HighLowSalaries();
2

```

Data Output Messages Notifications

NOTICE: Top 5 Employees with Highest Salaries:  
NOTICE: EmpNo: 7584, Name: OSTER, Job: <NULL>, Salary: <NULL>  
NOTICE: EmpNo: 7369, Name: SMITH, Job: CLERK, Salary: 7600  
NOTICE: EmpNo: 7839, Name: KING, Job: PRESIDENT, Salary: 5000  
NOTICE: EmpNo: 7788, Name: SCOTT, Job: ANALYST, Salary: 3000  
NOTICE: EmpNo: 7902, Name: FORD, Job: ANALYST, Salary: 3000  
NOTICE: Top 5 Employees with Lowest Salaries:  
NOTICE: EmpNo: 7900, Name: JAMES, Job: CLERK, Salary: 950  
NOTICE: EmpNo: 7876, Name: ADAMS, Job: CLERK, Salary: 1100  
NOTICE: EmpNo: 7521, Name: WARD, Job: SALESMAN, Salary: 1250  
NOTICE: EmpNo: 7934, Name: MILLER, Job: CLERK, Salary: 1300

Total rows: Query completed 00:00:00.094

✓ Query returned successfully in 94 msec. CRLF Ln 2, Col 1

## 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;
```

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server ×

postgres/postgres@Local server

Query History Execute script F5

```

1 CREATE OR REPLACE FUNCTION log_emp_changes()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     IF TG_OP = 'INSERT' THEN
5         INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
6             VALUES (NEW.empno, NEW.ename, NEW.job, NEW.sal, 'INSERT', CURRENT_USER, CURRENT_TIMESTAMP);
7
8     ELSIF TG_OP = 'UPDATE' THEN
9         INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
10            VALUES (NEW.empno, NEW.ename, NEW.job, NEW.sal, 'UPDATE', CURRENT_USER, CURRENT_TIMESTAMP);
11
12    ELSIF TG_OP = 'DELETE' THEN
13        INSERT INTO audit_emp (empno, ename, job, sal, action_type, modified_by, modified_at)
14            VALUES (OLD.empno, OLD.ename, OLD.job, OLD.sal, 'DELETE', CURRENT_USER, CURRENT_TIMESTAMP);
15    END IF;
16
17    RETURN NULL; -- Triggers of type AFTER don't modify data
18 END;
19 $$ LANGUAGE plpgsql;
20

```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 52 msec.

Total rows: Query complete 00:00:00.052 CRLF Ln 12, Col 32

pgAdmin 4

File Object Tools Edit View Window Help

Welcome postgres/postgres... × postgres/postgres@Local server ×

postgres/postgres@Local server

Query History Execute script F5

```

1 CREATE TRIGGER emp_insert_trigger
2 AFTER INSERT ON emp
3 FOR EACH ROW EXECUTE FUNCTION log_emp_changes();
4
5 CREATE TRIGGER emp_update_trigger
6 AFTER UPDATE ON emp
7 FOR EACH ROW EXECUTE FUNCTION log_emp_changes();
8
9 CREATE TRIGGER emp_delete_trigger
10 AFTER DELETE ON emp
11 FOR EACH ROW EXECUTE FUNCTION log_emp_changes();
12

```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 53 msec.

Total rows: Query complete 00:00:00.053 CRLF Ln 12, Col 1

**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 History

Scratch Pad

```
1 ✓ CREATE OR REPLACE FUNCTION AnalyzeActivity(p_user TEXT DEFAULT NULL, p_date DATE DEFAULT NULL)
2 RETURNS TABLE (
3     user_name TEXT,
4     action_type TEXT,
5     operation_count INT
6 ) AS $$ 
7 BEGIN
8     RETURN QUERY
9     SELECT modified_by, action_type, COUNT(*)
10    FROM audit_emp
11   WHERE (p_user IS NULL OR modified_by = p_user) -- Filter by user if provided
12     AND (p_date IS NULL OR DATE(modified_at) = p_date) -- Filter by date if provided
13   GROUP BY modified_by, action_type
14   ORDER BY modified_by, action_type;
15 END;
16 $$ LANGUAGE plpgsql;
17
```

Data Output Messages Notifications

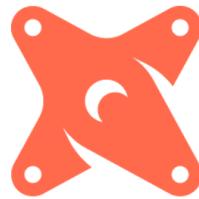
CREATE FUNCTION

Query returned successfully in 101 msec.

✓ Query returned successfully in 101 msec. ✎

CRLF Ln 17, Col 1

Servers > Local server > Databases > postgres > Schemas > public > Tables > emp



# dbt

---

By Thibaut de Broca  
**GROOPTOWN**

DBeaver 25.0.1

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator Projects Enter a part of object name here

postgres localhost:5432

Databases

- Schemas
  - dbt\_postgres\_test
  - pgagent
  - public
- Event Triggers
- Extensions
- Storage
- System Info
- Roles

Administrator System Info

New Database Connection Ctrl+Shift+N  
Last edited SQL script Ctrl+Enter  
Open File... Ctrl+O  
Show Key Assist Ctrl+Shift+L  
Find Actions Ctrl+3

Project - General DataSource

Name Bookmarks Dashboards Diagrams Scripts

CET en

## Install DBT

```

Command Prompt

Downloading annotated_types-0.7.0-py3-none-any.whl (13 kB)
Downloading typing_inspection-0.4.0-py3-none-any.whl (14 kB)
Downloading zipp-3.21.0-py3-none-any.whl (9.6 kB)
Building wheels for collected packages: daff
  Building wheel for daff (pyproject.toml) ... done
    Created wheel for daff: filename=daff-1.3.46-py3-none-any.whl size=146174 sha256=79f27be02675dbc01a34ff0edb3bfa2421175e1d557214aca2988d6b9c88f362
    Stored in directory: c:\users\abasiofon\appdata\local\pip\cache\wheels\ed\60\27\9b49e83466a7ad65476a93af9909a249e7a1862cd16eba0c20
Successfully built daff
Installing collected packages: text-unidecode, pytz, pytimeparse, parsedatetime, leather, daff, urllib3, tzdata, typing-extensions, sqlparse, six, rpds-py, pyyaml, python-slugify, psycopg2-binary, protobuf, pathspec, packaging, ordered-set, networkx, msgpack, more-itertools, MarkupSafe, idna, dbt-extractor, colorama, charset-normalizer, certifi, Babel, attrs, annotated-types, typing-inspection, requests, referencing, python-dateutil, pydantic-core, mashumaro, jinja2, isodate, importlib-metadata, deepdiff, click, snowplow-tracker, pydantic, jsonschema-specifications, agate, jsonschema, dbt-semantic-interfaces, dbt-common, dbt-adapters, dbt-core, dbt-postgres
Successfully installed Babel-2.17.0 MarkupSafe-3.0.2 agate-1.9.1 annotated-types-0.7.0 attrs-25.3.0 certifi-2025.1.31 charset-normalizer-3.4.1 click-8.1.8 colorama-0.4.6 daff-1.3.46 dbt-adapters-1.14.4 dbt-common-1.17.0 dbt-core-1.9.4 dbt-extractor-0.5.1 dbt-postgres-1.9.0 dbt-semantic-interface-0.7.4 deepdiff-7.0.1 idna-3.10 importlib-metadata-6.11.0 isodate-0.6.1 jinja2-3.1.6 jsonschema-4.23.0 jsonschema-specifications-2024.10.1 leather-0.4.0 mashumaro-3.14 more-itertools-10.6.0 msgpack-1.1.0 networkx-3.4.2 ordered-set-4.1.0 packaging-24.2 parsedatetime-2.6 pathspec-0.12.1 protobuf-5.29.4 psycopg2-binary-2.9.10 pydantic-2.11.2 pydantic-core-2.33.1 python-dateutil-2.9.0.post0 python-slugify-8.0.4 pytimeparse-1.1.8 pytz-2025.2 pyyaml-6.0.2 referencing-0.36.2 requests-2.32.3 rpds-py-0.24.0 six-1.17.0 snowplow-tracker-1.1.0 sqlparse-0.5.3 text-unidecode-1.3 typing-extensions-4.13.1 typing-inspection-0.4.0 tzdata-2025.2 urllib3-2.3.0 zipp-3.21.0

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip

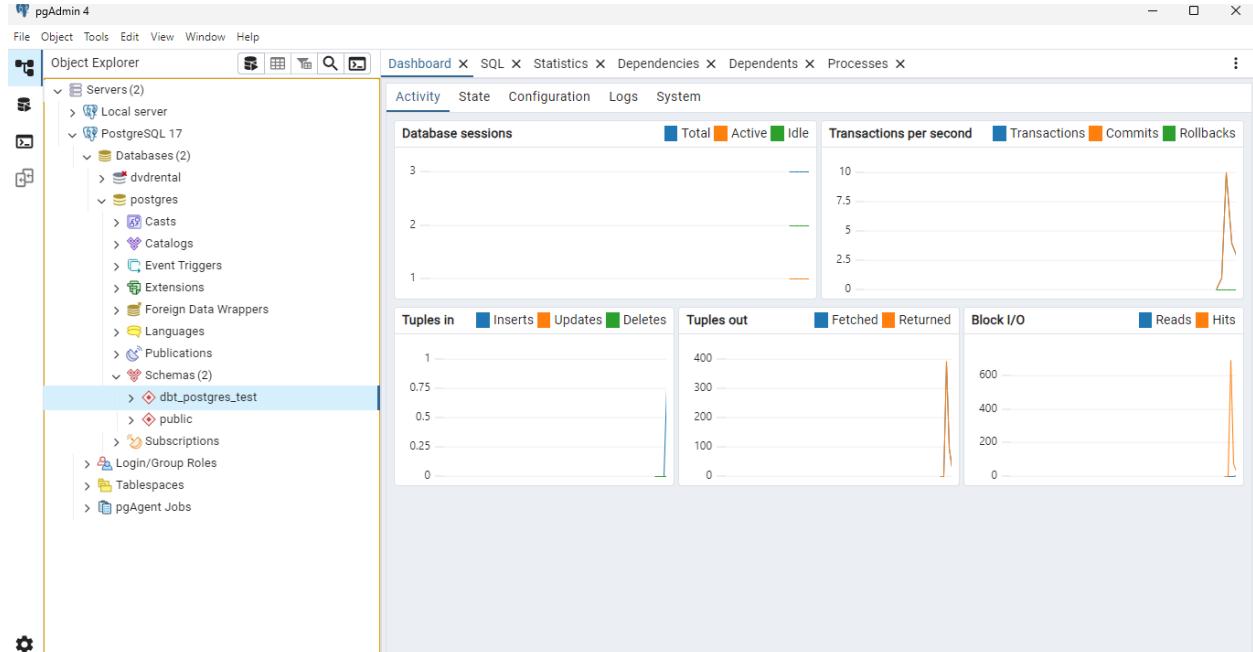
(dbt-env) C:\dbt-projects>dbt --version
Core:
- installed: 1.9.4
- latest: 1.9.4 - Up to date!

Plugins:
- postgres: 1.9.0 - Up to date!

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

```

## 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 X Projects

Enter a part of object name here

postres localhost:5432

Databases

postres

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 X

Name DataSource

Bookmarks Dashboards Diagrams Scripts

\*<postgres> Script X

```
-- 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 X

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 Sel: 1

DBeaver 25.0.1 - <postgres> Script

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator X Projects

Enter a part of object name here

postres localhost:5432

Databases

postres

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 X

Name DataSource

Bookmarks Dashboards Diagrams Scripts

\*<postgres> Script X

```
COPY orders_raw(ID, USER_ID, ORDER_DATE, STATUS, PRICE)
FROM 'C:/Users/Public/orders.csv'
DELIMITER ','
CSV HEADER;
```

Statistics 1 X

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 Sel: 1

DBeaver 25.0.1 - <postgres> Script

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator X Projects

Enter a part of object name here

postgres localhost:5432

Databases

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 X

Name DataSource

Bookmarks

Dashboards

Diagrams

Scripts

\*<postgres> Script X

CREATE TABLE customers\_raw (

ID SERIAL,

FIRST\_NAME VARCHAR(255),

LAST\_NAME VARCHAR(255),

PRIMARY KEY (ID)

);

Statistics 1 X

Name	Value
Updated Rows	0
Execute time	0.010s
Start time	Sat Apr 05 15:41:49 CEST 2025
Finish time	Sat Apr 05 15:41:49 CEST 2025
Query	CREATE TABLE customers_raw (
	ID SERIAL,
	FIRST_NAME VARCHAR(255),
	LAST_NAME VARCHAR(255),
	PRIMARY KEY (ID)
	)

CET en Writable Smart Insert 7:1:131 Set: t

DBeaver 25.0.1 - <postgres> Script

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator X Projects

Enter a part of object name here

postgres localhost:5432

Databases

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 X

Name DataSource

Bookmarks

Dashboards

Diagrams

Scripts

\*<postgres> Script X

COPY customers\_raw(ID, FIRST\_NAME, LAST\_NAME)

FROM 'C:/Users/Public/customers.csv'

DELIMITER ','

CSV HEADER

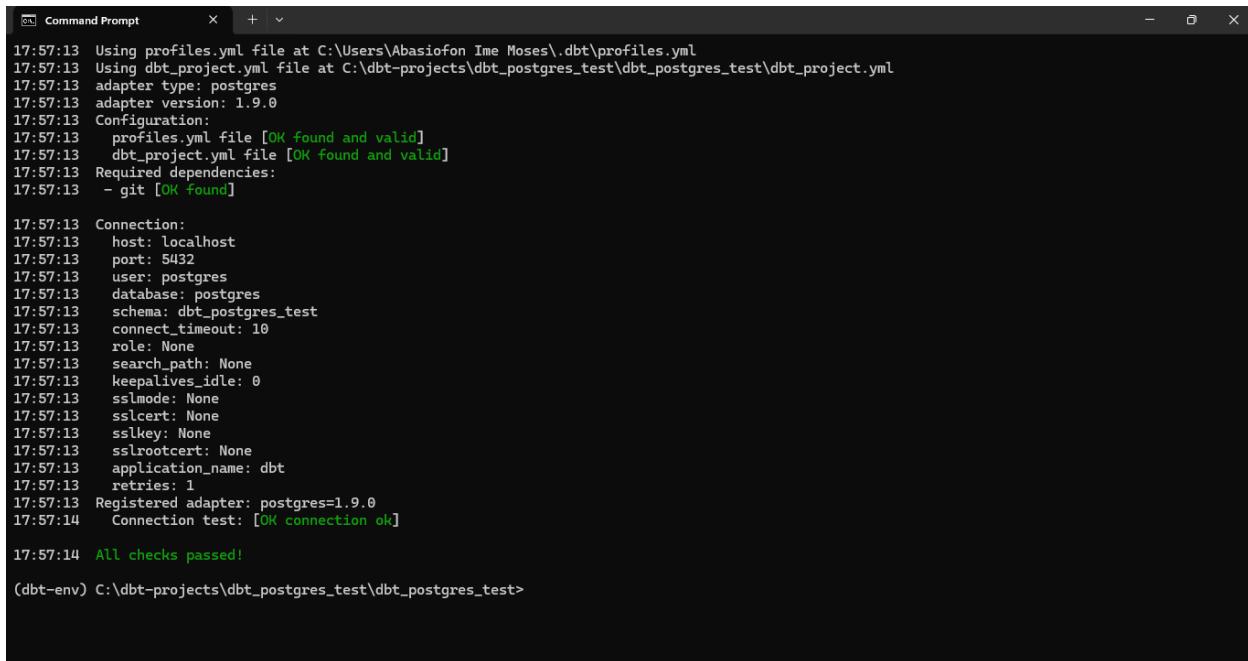
Statistics 1 X

Name	Value
Updated Rows	100000
Execute time	0.892s
Start time	Sat Apr 05 15:42:19 CEST 2025
Finish time	Sat Apr 05 15:42:19 CEST 2025
Query	COPY customers_raw(ID, FIRST_NAME, LAST_NAME)
	FROM 'C:/Users/Public/customers.csv'
	DELIMITER ','
	CSV HEADER

CET en Writable Smart Insert 4:12:111 Set: t

## Episode 4: Dbt Init

## Init your project



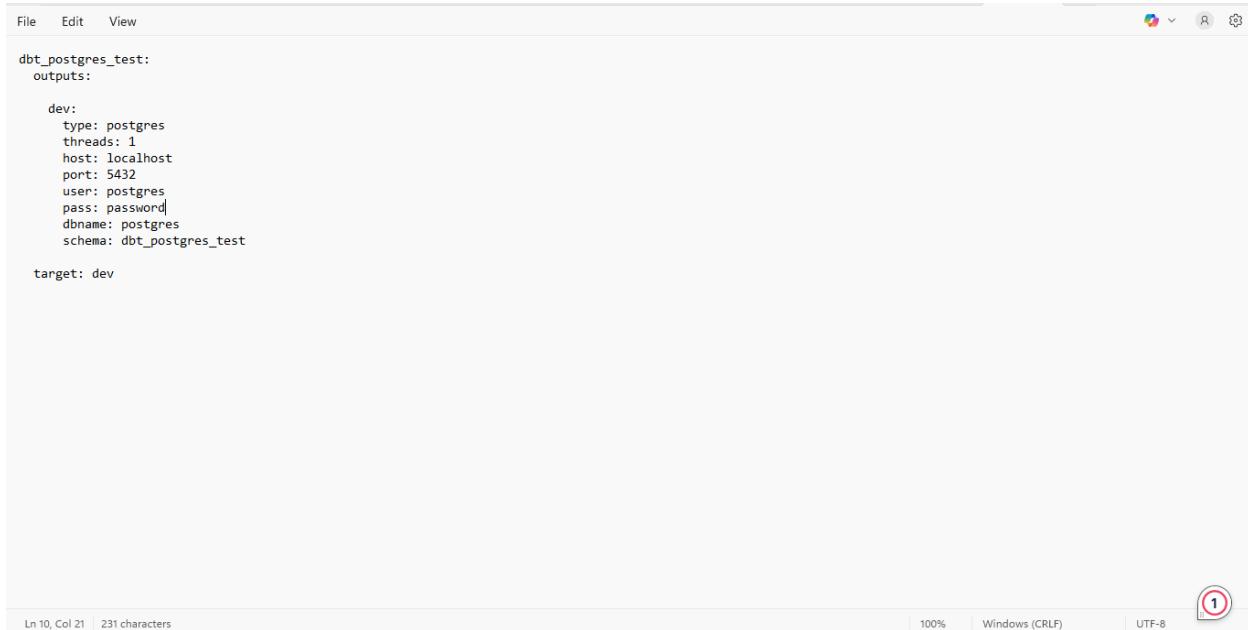
```
Command Prompt + X - X
17:57:13 Using profiles.yml file at C:\Users\Abasiofon Ime Moses\.dbt\profiles.yml
17:57:13 Using dbt_project.yml file at C:\dbt-projects\dbt_postgres_test\dbt_postgres_test\dbt_project.yml
17:57:13 adapter type: postgres
17:57:13 adapter version: 1.9.0
17:57:13 Configuration:
17:57:13   profiles.yml file [OK found and valid]
17:57:13   dbt_project.yml file [OK found and valid]
17:57:13 Required dependencies:
17:57:13   - git [OK found]

17:57:13 Connection:
17:57:13   host: localhost
17:57:13   port: 5432
17:57:13   user: postgres
17:57:13   database: postgres
17:57:13   schema: dbt_postgres_test
17:57:13   connect_timeout: 10
17:57:13   role: None
17:57:13   search_path: None
17:57:13   keepalives_idle: 0
17:57:13   sslmode: None
17:57:13   sslcert: None
17:57:13   sslkey: None
17:57:13   sslrootcert: None
17:57:13   application_name: dbt
17:57:13   retries: 1
17:57:13 Registered adapter: postgres=1.9.0
17:57:14 Connection test: [OK connection ok]

17:57:14 All checks passed!

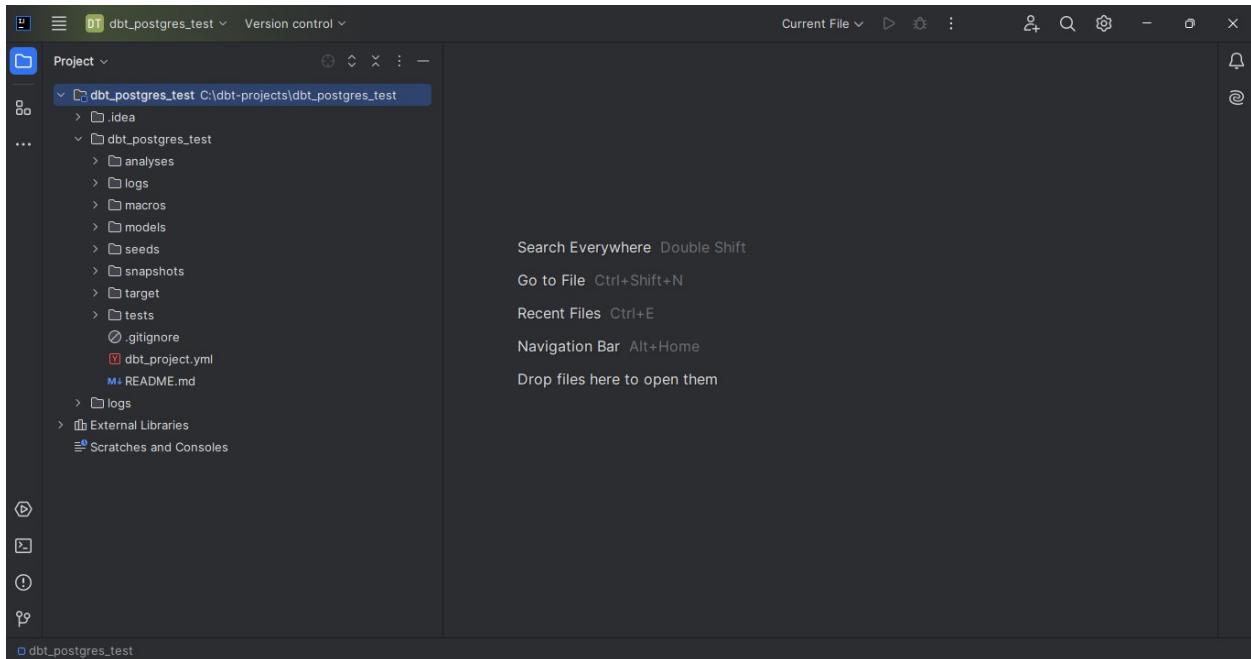
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>
```

## Configure your DBT profile



```
File Edit View
dbt_postgres_test:
  outputs:
    dev:
      type: postgres
      threads: 1
      host: localhost
      port: 5432
      user: postgres
      pass: password
      dbname: postgres
      schema: dbt_postgres_test
  target: dev
```

## Open in a IDE



## 1st Launch

```
Command Prompt
18:03:21 All checks passed!
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>cd C:\dbt-projects\dbt_postgres_test\dbt_postgres_test
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt run
18:06:47 Running with dbt=1.9.4
18:06:47 Registered adapter: postgres=1.9.0
18:06:49 Found 2 models, 4 data tests, 433 macros
18:06:49 Concurrency: 1 threads (target='dev')
18:06:49
18:06:50 1 of 2 START sql table model dbt_postgres_test.my_first_dbt_model ..... [RUN]
18:06:51 1 of 2 OK created sql table model dbt_postgres_test.my_first_dbt_model ..... [SELECT 2 in 0.56s]
18:06:51 2 of 2 START sql view model dbt_postgres_test.my_second_dbt_model ..... [RUN]
18:06:51 2 of 2 OK created sql view model dbt_postgres_test.my_second_dbt_model ..... [CREATE VIEW in 0.33s]
18:06:51
18:06:51 Finished running 1 table model, 1 view model in 0 hours 0 minutes and 2.47 seconds (2.47s).
18:06:51
18:06:51 Completed successfully
18:06:51
18:06:51 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>
```

# Episode 5: Create Models

The screenshot shows the IntelliJ IDEA interface with the project 'dbt\_postgres\_test' open. The left sidebar displays the project structure, including 'models' and 'dim\_customers.sql'. The main editor window shows the SQL code for the 'dim\_customers' model:

```
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 (
    select
```

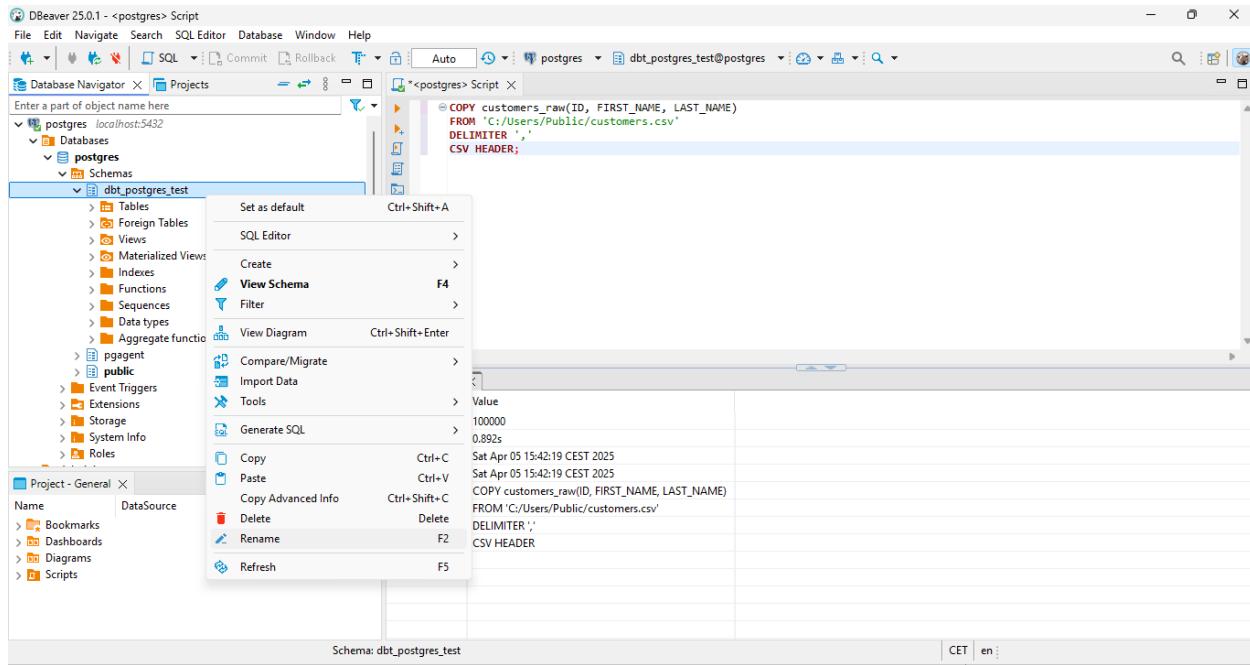
The screenshot shows a Command Prompt window with the following log output:

```
18:06:51 1 of 2 OK created sql table model dbt_postgres_test.my_first_dbt_model ..... [SELECT 2 in 0.56s]
18:06:51 2 of 2 START sql view model dbt_postgres_test.my_second_dbt_model ..... [RUN]
18:06:51 2 of 2 OK created sql view model dbt_postgres_test.my_second_dbt_model ..... [CREATE VIEW in 0.33s]
18:06:51
18:06:51 Finished running 1 table model, 1 view model in 0 hours 0 minutes and 2.47 seconds (2.47s).
18:06:51
18:06:51 Completed successfully
18:06:51
18:06:51 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>C:\dbt-projects\dbt_postgres_test\dbt_postgres_test\models
'C:\dbt-projects\dbt_postgres_test\dbt_postgres_test\models' is not recognized as an internal or external command,
operable program or batch file.

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt run
18:10:51 Running with dbt=1.9.4
18:10:51 Registered adapter: postgres=1.9.0
18:10:52 Found 3 models, 4 data tests, 433 macros
18:10:52
18:10:52 Concurrency: 1 threads (target='dev')
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>
```



DBeaver 25.0.1 - dim\_customers

customer_id	first_name	last_name	first_order_date	most_r
0	Shelly	Freeman	[NULL]	
1	Sarah	Schmidt	[NULL]	
2	Andrew	Walton	[NULL]	
3	Tommy	Gay	[NULL]	
4	Amanda	Macdonald	[NULL]	
5	Tracey	Johnson	[NULL]	
6	Nathan	Barnes	[NULL]	
7	Lauren	Jones	[NULL]	
8	April	Robinson	[NULL]	
9	Noah	Dawson	[NULL]	
10	Megan	Morton	[NULL]	
11	Sharon	Galloway	[NULL]	
12	Robert	Newman	[NULL]	
13	Kenneth	Johnson	[NULL]	
14	Anthony	Young	[NULL]	
15	Jonathan	James	[NULL]	
16	Lindsay	Hendricks	[NULL]	
17	Karen	Singleton	[NULL]	
18	Denise	Berry	[NULL]	
19	John	Anderson	[NULL]	
20	Kevin	Monica	[NULL]	
21	Kenneth	Rush	[NULL]	
22	Chad	Hunt	[NULL]	
23	Melanie	Walker	[NULL]	

The status bar at the bottom right indicates the time is 'Sat Apr 05 15:42:19 CEST 2025'.

## Materialize final Table

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "dbt\_postgres\_test".
- Code Editor:** Displays the content of the `dim_customers.sql` file.
- Code Content:**

```

1 {{ config(materialized='table') }}
2
3 with customers as (
4
5     select
6         id as customer_id,
7         first_name,
8         last_name
9
10    from dbt_postgres_test.customers_raw
11
12 ),
13
14 orders as (
15
16     select
17         id as order_id,
18         user_id as customer_id,
19         order_date,
20         status
21
22    from dbt_postgres_test.orders_raw
23
24 ),
25
26 customer_orders as (

```
- Status Bar:** Shows the current time (10:41), encoding (CRLF), file encoding (UTF-8), and code style settings (4 spaces).

The screenshot shows a Command Prompt window with the following log output:

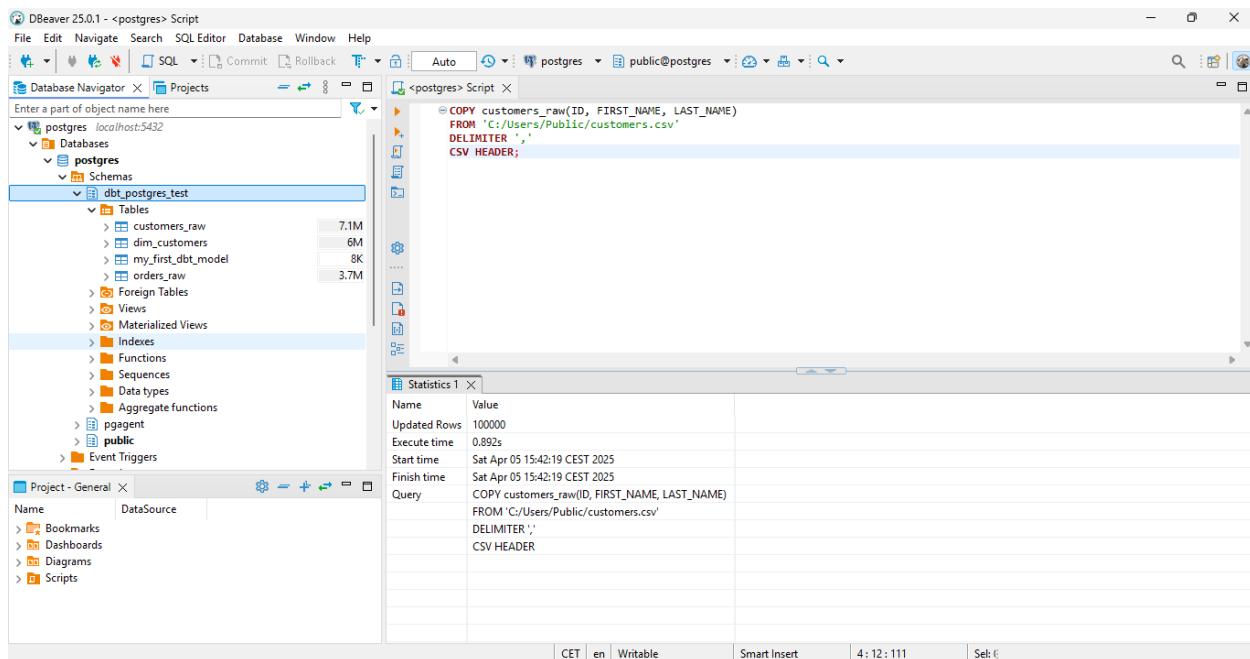
```

18:10:52
18:10:53 1 of 3 START sql view model dbt_postgres_test.dim_customers ..... [RUN]
18:10:53 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 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

```

with customers as (
    select * from {{ ref('stg_customers') }}
),
orders as (
    select * from {{ ref('stg_orders') }}
),
customer_orders as (
    select
        customer_id,
        min(order_date) as first_order_date,
        max(order_date) as most_recent_order_date,
        count(order_id) as number_of_orders
    from orders
    group by 1
),

```

The screenshot shows the IntelliJ IDEA interface with the project `dbt_postgres_test` open. The left sidebar displays the project structure, including `analyses`, `logs`, `macros`, `models` (which contains `marts` and `seeds`), `staging` (which contains `stg_customers.sql` and `stg_orders.sql`), and `tests`. The right panel shows the code editor with `stg_customers.sql` selected. The code in the editor is:

```
1 select
2     id as customer_id,
3     first_name,
4     last_name
5
6     from dbt_postgres_test.customers_raw
7
```

The screenshot shows the IntelliJ IDEA interface with the project `dbt_postgres_test` open. The left sidebar displays the project structure, identical to the previous screenshot. The right panel shows the code editor with `stg_orders.sql` selected. The code in the editor is:

```
1 select
2     id as order_id,
3     user_id as customer_id,
4     order_date,
5     status
6
7     from dbt_postgres_test.orders_raw
8
```

Re-run dbt:

```

Command Prompt x + v
18:22:04 2 of 3 START sql table model dbt_postgres_test.my_first_dbt_model ..... [RUN]
18:22:04 2 of 3 OK created sql table model dbt_postgres_test.my_first_dbt_model ..... [SELECT 2 in 0.24s]
18:22:04 3 of 3 START sql view model dbt_postgres_test.my_second_dbt_model ..... [RUN]
18:22:04 3 of 3 OK created sql view model dbt_postgres_test.my_second_dbt_model ..... [CREATE VIEW in 0.25s]
18:22:04
18:22:04 Finished running 2 table models, 1 view model in 0 hours 0 minutes and 2.61 seconds (2.61s).
18:22:05
18:22:05 Completed successfully
18:22:05
18:22:05 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt run
18:26:48 Running with dbt=1.9.4
18:26:48 Registered adapter: postgres=1.9.0
18:26:49 [WARNING]: Configuration paths exist in your dbt_project.yml file which do not apply to any resources.
There are 1 unused configuration paths:
- models.dbt_postgres_test.example
18:26:49 Found 3 models, 433 macros
18:26:49
18:26:49 Concurrency: 1 threads (target='dev')
18:26:49
18:26:50 1 of 3 START sql view model dbt_postgres_test.stg_customers ..... [RUN]
18:26:50 1 of 3 OK created sql view model dbt_postgres_test.stg_customers ..... [CREATE VIEW in 0.32s]
18:26:50 2 of 3 START sql view model dbt_postgres_test.stg_orders ..... [RUN]
18:26:51 2 of 3 OK created sql view model dbt_postgres_test.stg_orders ..... [CREATE VIEW in 0.21s]
18:26:51 3 of 3 START sql view model dbt_postgres_test.dim_customers ..... [RUN]
18:26:51 3 of 3 OK created sql view model dbt_postgres_test.dim_customers ..... [CREATE VIEW in 0.25s]
18:26:51
18:26:51 Finished running 3 view models in 0 hours 0 minutes and 1.78 seconds (1.78s).
18:26:51
18:26:51 Completed successfully
18:26:51
18:26:51 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3

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

```

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

DBeaver 25.0.1 - <postgres> Script

File Edit Navigate Search SQL Editor Database Window Help

Database Navigator X Projects

Enter a part of object name here

postgres localhost:5432

- Databases
  - postgres
- Schemas
  - dbt\_postgres\_test

Tables

- customers\_raw
- my\_first\_dbt\_model
- orders\_raw

Foreign Tables

Views

- dim\_customers
- my\_second\_dbt\_model
- stg\_customers
- stg\_orders

Materialized Views

Indexes

Functions

Sequences

Data types

Aggregate functions

Script X

```

COPY customers_raw(ID, FIRST_NAME, LAST_NAME)
FROM 'C:/Users/Public/customers.csv'
DELIMITER ','
CSV HEADER;
  
```

Statistics 1 X

Name	Value
Updated Rows	100000
Execute time	0.892s
Start time	Sat Apr 05 15:42:19 CEST 2025
Finish time	Sat Apr 05 15:42:19 CEST 2025

Project - General X

Name      DataSource

Bookmarks

Dashboards

Diagrams

Scripts

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

## Define global default materialization per folder

The screenshot shows a code editor interface with a sidebar containing a project tree. The tree includes a root folder 'dbt\_postgres\_test' which contains subfolders like '.idea', 'analyses', 'logs', 'macros', 'models' (which further contains 'marts' and 'staging' subfolders), 'seeds', 'snapshots', 'target', 'tests', '.gitignore', and 'dbt\_project.yml'. Below the tree are files 'README.md' and 'External Libraries'. The main editor area displays the 'dbt\_project.yml' file. The file content is as follows:

```
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

clean-targets:          # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/
# directory as views. These settings can be overridden in the individual model
# files using the `{{ config(...) }}` macro.

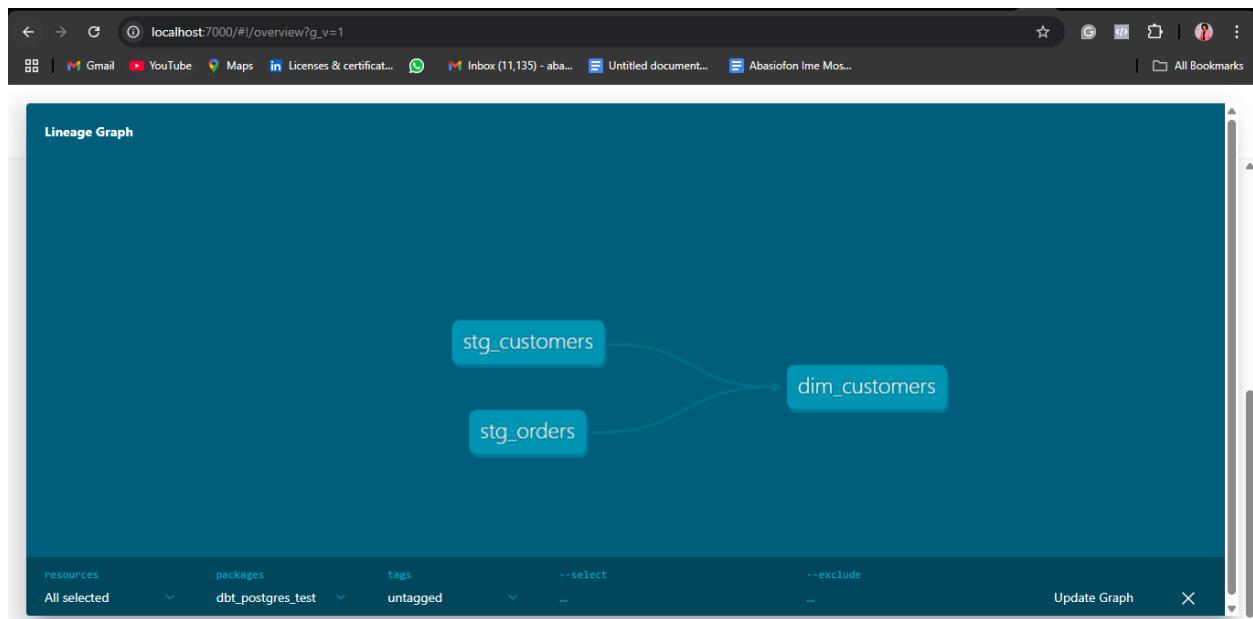
models:
  dbt_postgres_test:
    marts:
      +materialized: table
    staging:
      +materialized: view
```

At the bottom of the editor, status bars show '38:1 CRLF UTF-8 2 spaces Schema: dbt\_project-latest.json'.

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 .....
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 .....
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

The screenshot shows a code editor interface with a dark theme. On the left, there's a file tree for the project "dbt\_postgres\_test". The "models" directory contains "stg\_customers.sql", "stg\_orders.sql", "dbt\_project.yml", "my\_sources.yml" (which is currently selected), and "dim\_customers.sql". The "staging" directory contains "my\_sources.yml", "stg\_customers.sql", and "stg\_orders.sql". Other directories like "analyses", "logs", "macros", "seeds", "snapshots", "target", "tests", and "External Libraries" are also visible. The main editor area shows the content of "my\_sources.yml":

```

version: 2
sources:
  - name: my_sources
    database: postgres
    schema: dbt_postgres_test
    tables:
      - name: customers_raw
      - name: orders_raw
  
```

At the bottom of the editor, the status bar shows "10:1 CRLF UTF-8 2 spaces Schema: dbt\_yml\_files-latest.json".

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

The screenshot shows a code editor interface with a dark theme. The left sidebar displays the project structure:

- Project
- dbt\_postgres\_test
- ...
  - 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

The main editor area shows the content of the `stg_orders.sql` file:

```
1 select
2     id as order_id,
3     user_id as customer_id,
4     order_date,
5     status
6
7 from {{ source('my_sources', 'orders_raw') }}
```

File status indicators at the top right show a green checkmark and an '@' symbol.

The screenshot shows a code editor interface with a dark theme. The left sidebar displays the project structure:

- Project
- dbt\_postgres\_test
- ...
  - 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

The main editor area shows the content of the `stg_customers.sql` file:

```
1 select
2     id as customer_id,
3     first_name,
4     last_name
5
6 from {{ source('my_sources', 'customers_raw') }}
```

File status indicators at the top right show a green checkmark and an '@' symbol.

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

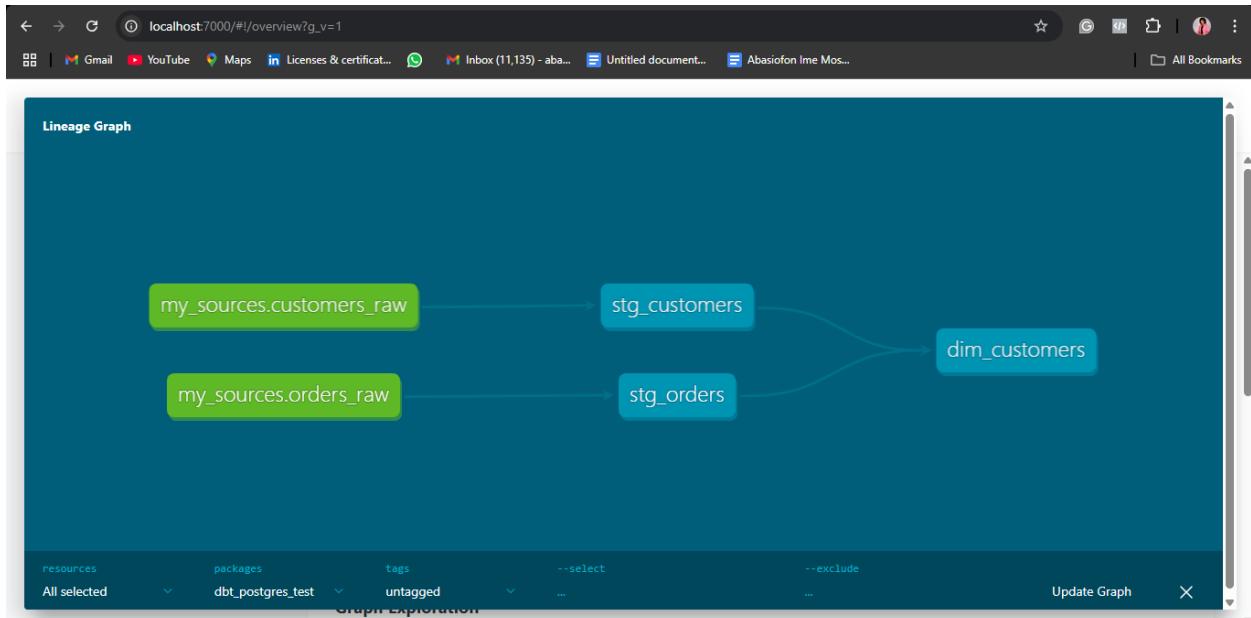
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt docs generate
18:37:03  Running with dbt=1.9.4
18:37:03  Registered adapter: postgres=1.9.0
18:37:04  Found 3 models, 2 sources, 433 macros
18:37:04  Concurrency: 1 threads (target='dev')
18:37:04
18:37:05  Building catalog
18:37:05  Catalog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

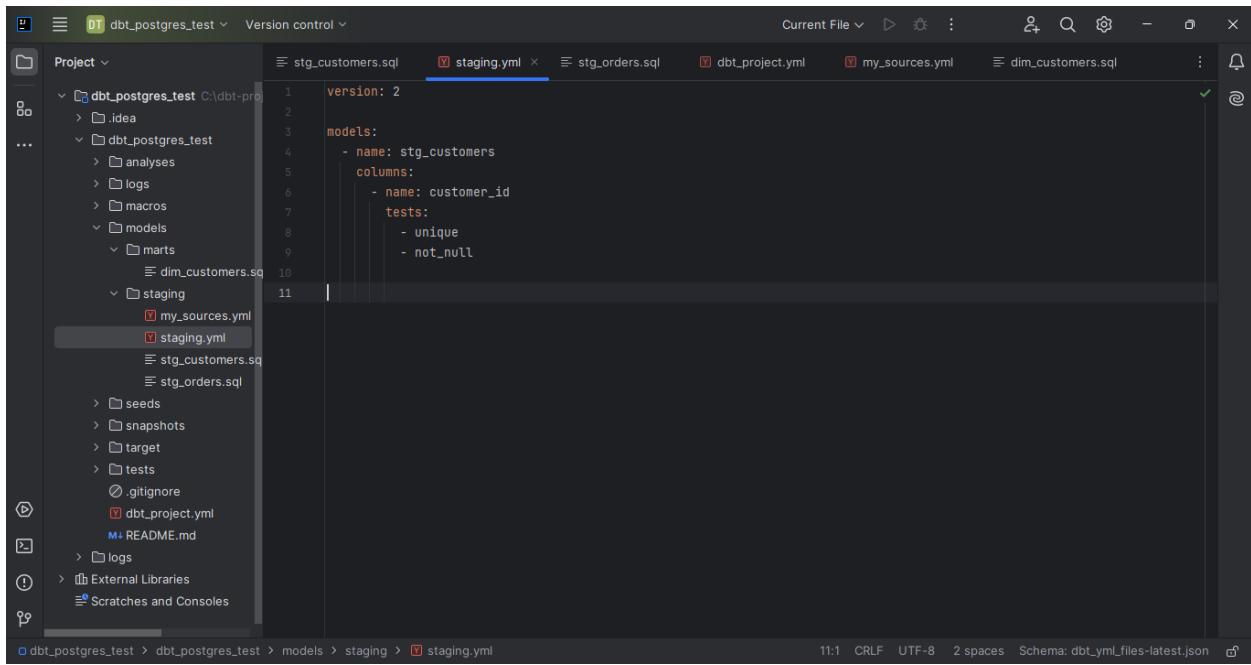
```

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



# Episode 7: Tests

## Generic Test



The screenshot shows a code editor interface with a dark theme. The left sidebar displays a project structure for a dbt project named "dbt\_postgres\_test". The "models" directory contains "stg\_customers.sql" and "stg\_orders.sql". The "staging" directory contains "my\_sources.yml", "staging.yml", "stg\_customers.sql", and "stg\_orders.sql". The "tests" directory contains ".gitignore" and "dbt\_project.yml". The "target" directory contains "README.md". The "seeds" and "snapshots" directories are also present. The bottom status bar shows the path "dbt\_postgres\_test > dbt\_postgres\_test > models > staging > staging.yml". The main editor area shows the content of the "staging.yml" file:

```
version: 2
models:
  - name: stg_customers
    columns:
      - name: customer_id
        tests:
          - unique
          - not_null
```

Run

```
dbt test
```

You can see your tests are successful

```

Command Prompt x + v
File "C:\dbt-projects\dbt-env\Lib\site-packages\dbt\cli\main.py", line 301, in docs_serve
    results = task.run()
File "C:\dbt-projects\dbt-env\Lib\site-packages\dbt\task\docs\serve.py", line 29, in 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 test
18:40:10 Running with dbt=1.9.4
18:40:10 Registered adapter: postgres=1.9.0
18:40:11 Found 3 models, 2 data tests, 2 sources, 433 macros
18:40:11
18:40:11 Concurrency: 1 threads (target='dev')
18:40:11
18:40:12 1 of 2 START test not_null_stg_customers_customer_id ..... [RUN]
18:40:12 1 of 2 PASS not_null_stg_customers_customer_id ..... [PASS in 0.25s]
18:40:12 2 of 2 START test unique_stg_customers_customer_id ..... [RUN]
18:40:13 2 of 2 PASS unique_stg_customers_customer_id ..... [PASS in 0.26s]
18:40:13
18:40:13 Finished running 2 data tests in 0 hours 0 minutes and 1.35 seconds (1.35s).
18:40:13
18:40:13 Completed successfully
18:40:13
18:40:13 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>

```

## Episode 8: Lineage and Documentation

```

version: 2
models:
  - name: stg_customers
    description: Staged customer data from our jaffle shop app
    columns:
      - name: customer_id
        description: The primary key for customers.
    tests:
      - unique
      - not_null

```

Run:

```

[dbt-env] Command Prompt - dbt docs + X
(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt test
18:40:10 Running with dbt=1.9.4
18:40:10 Registered adapter: postgres=1.9.0
18:40:11 Found 3 models, 2 data tests, 2 sources, 433 macros
18:40:11
18:40:11 Concurrency: 1 threads (target='dev')
18:40:11
18:40:12 1 of 2 START test not_null_stg_customers_customer_id ..... [RUN]
18:40:12 1 of 2 PASS not_null_stg_customers_customer_id ..... [PASS in 0.25s]
18:40:12 2 of 2 START test unique_stg_customers_customer_id ..... [RUN]
18:40:13 2 of 2 PASS unique_stg_customers_customer_id ..... [PASS in 0.26s]
18:40:13
18:40:13 Finished running 2 data tests in 0 hours 0 minutes and 1.35 seconds (1.35s).
18:40:13
18:40:13 Completed successfully
18:40:13
18:40:13 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2

(dbt-env) C:\dbt-projects\dbt_postgres_test\dbt_postgres_test>dbt docs generate
18:42:21 Running with dbt=1.9.4
18:42:21 Registered adapter: postgres=1.9.0
18:42:22 Found 3 models, 2 data tests, 2 sources, 433 macros
18:42:22
18:42:22 Concurrency: 1 threads (target='dev')
18:42:22
18:42:23 Building catalog
18:42:23 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
18:42:41 Running with dbt=1.9.4
Serving docs at 7000
To access from your browser, navigate to: http://localhost:7000

```

You can see the descriptions appearing in the UI:

The screenshot shows the dbt UI interface. At the top, there's a browser header with the URL `localhost:7000/#/model/model.dbt_postgres_test.stg_customers`. Below the header, the dbt logo is visible. The main area has a sidebar on the left containing sections for Overview, Sources, and Projects. The Overview section is currently active, showing a Project tab selected. The main content area displays the details for the `stg_customers` view. The title is `stg_customers view`. Below the title, there are tabs for Details, Description, Columns, Referenced By, Depends On, and Code. The Details tab is selected, showing the following table:

TAGS	OWNER	TYPE	PACKAGE	LANGUAGE	RELATION	ACCESS
untagged	postgres	view	dbt_postgres_test	sql	postgres.dbt_postgres_test.stg_customers	protected

Below the table, the Description section contains the text: "Staged customer data from our jaffle shop app". The Columns section is partially visible at the bottom.

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

<https://github.com/DatagirlX/DBTproject>