

II.2314 / II.2414

Advanced Databases and Big Data

Lab 1 & 2 & 3

GUO Xiaofan

<https://github.com/heyPetiteF/ISEP/tree/main/2402-2406/2-BIG%20DATA/LAB/TP3/CODE>

CONTENT

Lab 1	4
Exercice 1: DDL queries	4
1. Copy	4
2. E/R diagram:	5
3. First integrity Constraint:	5
4. Second integrity Constraint.....	6
5. Third integrity Constraint on Delete	6
6. Explain Errors for Integrity Constraint	8
7. Define a sequence	10
8. Use Sequence:.....	11
Exercise 2: DML queries Answer the following queries using SQL.....	11
1. List the content of all tables to see the attributes names.	11
2. Select employees whose commission is higher than their salary.	12
3. Select employees earning between 1200 and 2400 (earning is sal + commision).....	12
4. Select employees who are CLERK or ANALYST	13
5. Select employees whose name begins by M.....	13
6. Select employees whose name includes a L in second position.....	14
7. Select employees who are MANAGER/CLERK in the department10 &salary is greater than 1500.....	14
8. Select employees whose commission is NULL.....	15
9. Select employees by ascending order (by hiredate).....	15
10. Select employees ordered by job, and for each job, by decreasing salary.....	16
11. Select departments without employees.....	16
12. List employees indicating for each the name of his/her manager.....	17
13. List employees earning more than JONES	18
14. List employees displaying in the same column salary and commission.....	18
15. List department numbers which are both in table EMP and in table DEPT	19
16. List employees working in CHICAGO and having the same job than JONES	19
17. List employees who don't work in the same department than their manager.....	20
18. List employees working in a department having at least one CLERK	20
19. List employees of department 10 having the same job than someone from the depart SALES	21
20. List employees having the same job than JONES or a salary greater than FORD's salary.....	21
21. List employees having a salary greater than all employees of department 20	22
Exercice 3: Join Table	23
1. Create a Table for employee's Projects:.....	23
2. Create the Join Table:	23
3. List all employees (by empno) who work on all projects?	25
4. Options on View creation:.....	25
5. View Creation:.....	26
6. Division Query:.....	27
7. Analyze:	28
Lab 2-Advanced SQL.....	29
Exercice 1: Analytics Queries . Window Queries	29
1. Gets the 2 persons per department, who have arrived the latest in the company.....	29
2. Show your analytical Skill and Invents an interesting query using Windows Functions.	30
Exercice 2: Index & Explain Plan	31
1. Execute this script.....	31
2. The goal of all exercice will be to tune the following query:.....	31

3. 1 Use EXPLAIN plan to analyze the query.....	32
3. 2 Activate stats.....	32
Exercice 3: Data Dictionary	38
Exercice 4: Use postgres via CLI.....	39
Exercice 5: Transaction Part 1 – Beginner	39
Lab 2-Part 2 PL/SQL.....	41
Exercice 1: Functions	41
Exercice 2: Procedure & Display & Cursors.....	42
Exercice 3. Procedure & Update	43
Exercice 4. Procedure	44
Exercice 5. SELECT for UPDATE	45
Exercice 6. Condition on EXIT loop	48
Exercice 7. Triggers	49
1. Create the triggers & 2. Function to analyze results.....	49
Lab 3	50
Part I: A Graphic of Table: communication with database.....	50
1. Load Driver - 2. Connect to bdd - 3. Make a request.....	50
Exercise 1: Modify the query above to add the location of the department.....	55
Exercice 2: Move Department.....	55
Exercice 3: Generic display of Tables.....	57
Exercice 4: Security:.....	57
Exercice 5:Modify moveDepartement() to use PrepareStatemen	58
Exercice 6: Try with displayTable.....	59
Exercice 7: With J2EE	60
Part II : DAO.....	60
Exercice 8: What are cons of JDBC as used above?	60
Exercice 9 : Write the Bean for Department.....	61
Exercice 10 : Write Implementation for method find for departement DAO.....	62
Exercice 11: Write Implementation for method find for employee DAO.....	65
Exercice 12 : Create DAOFactory and use it in previous example	67
Part III : Spring Boot & JPA.....	68
Exercice III.2 : Activate JPA and connect to your Database	69
Exercise III.3 : Read Data in your Database	70
Exercise III.4 : Create an Object via JPA and expose it on a REST endpoint	71
Exercise III.5 : GET by ID / Update by id / Delete by ID / Get All, via JPA / REST	72

Lab 1

Exercice 1: DDL queries

1. Copy

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying a tree structure of database objects for the 'Lab1&2' schema. The 'Tables' node under 'Tables (5)' is currently selected. The right pane is a query editor window titled 'Lab1&2/postgres@Lab1&2'. It contains several CREATE TABLE statements for tables like EMP, DEPENDENTS, BONUS, and SALGRADE. Below the code, the 'Messages' tab is active, showing the message 'CREATE TABLE' and 'Query returned successfully in 79 msec.'

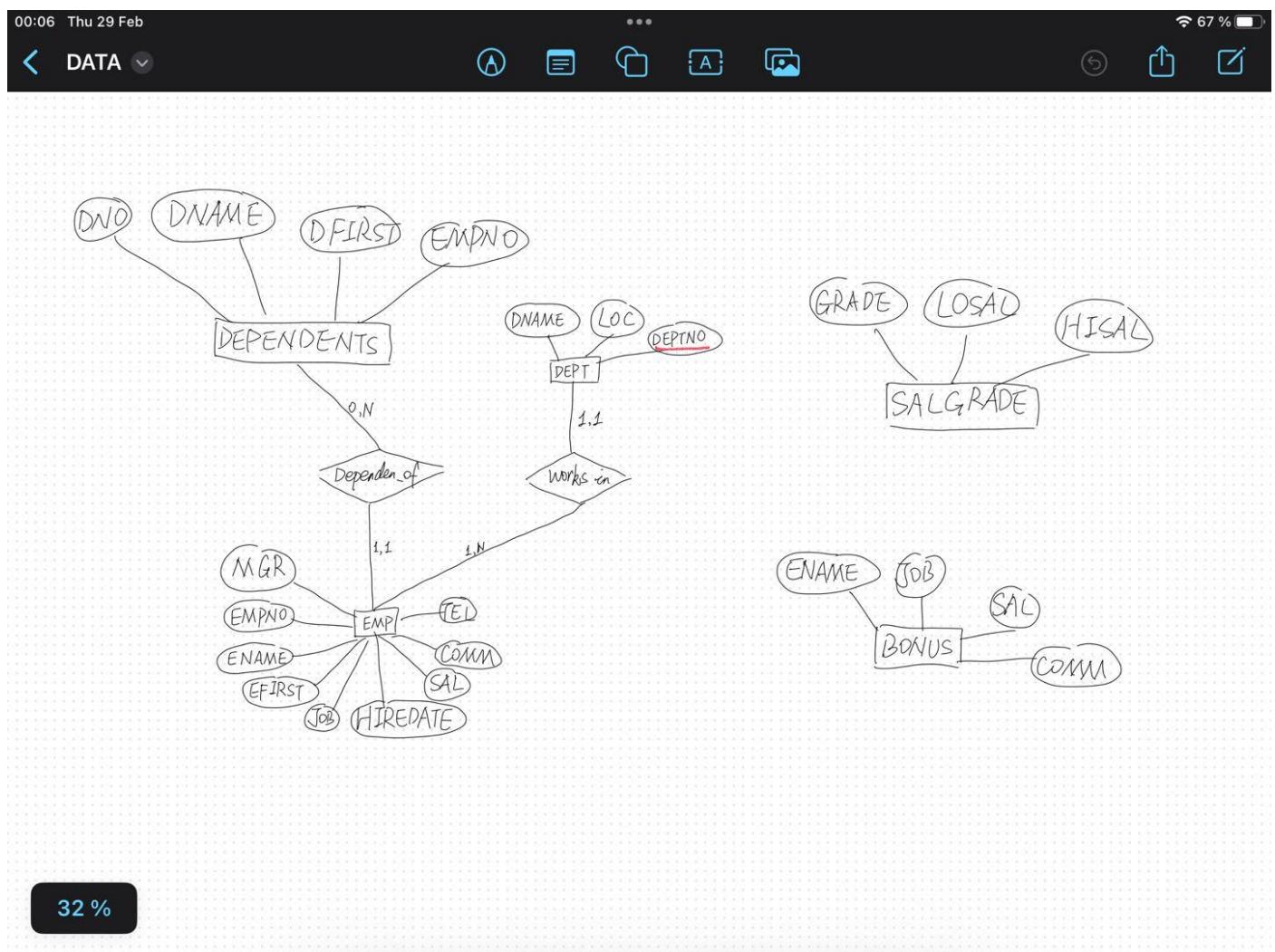
```
6
7 CREATE TABLE EMP
8     (EMPNO integer constraint pk_emp primary key,
9      ENAME VARCHAR(10),
10     EFIRST VARCHAR(10),
11     JOB VARCHAR(9),
12     MGR integer not null,
13     HIREDATE DATE,
14     SAL integer constraint ck_sal check (SAL>=0),
15     COMM integer,
16     TEL char(10),
17     DEPTNO integer,
18     constraint fk_emp_dept foreign key(DEPTNO) references DEPT (DEPTNO);
19
20 CREATE TABLE DEPENDENTS
21     (DNO integer,
22      DNAME VARCHAR(10),
23      DFIRST VARCHAR(10),
24      EMPNO integer,
25      constraint pk_dependent primary key (DNO, EMPNO),
26      constraint fk_dependent_emp foreign key(EMPNO) references EMP (EMPNO));
27
28
29 CREATE TABLE BONUS
30     (ENAME VARCHAR(10),
31      JOB VARCHAR(9),
32      SAL integer,
33      COMM integer);
34
35 CREATE TABLE SALGRADE
36     (GRADE integer,
37      LOSAL integer,
38      HISAL integer);
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 79 msec.

2. E/R diagram:



3. First integrity Constraint:

Lab1&2/postgres@Lab1

```
ALTER TABLE EMP
ADD CONSTRAINT emp_unique UNIQUE(ENAME,EFIRST,TEL);
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 63 msec.

4. Second integrity Constraint

Lab1&2/postgres@Lab1&2* X

```
1 ALTER TABLE EMP
2 ADD MOBILE_TEL CHAR(10);
3
4 ALTER TABLE EMP
5 ADD CONSTRAINT chk_mobile_tel CHECK (MOBILE_TEL LIKE '01%');
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 58 msec.

5. Third integrity Constraint on Delete

1) remove the referential constraint fk_dependent_emp

Lab1&2/postgres@Lab1&2* X

```
1 ALTER TABLE DEPENDENTS
2 DROP CONSTRAINT fk_dependent_emp;
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 37 msec.

2) delete an employee and remove all his dependents on the same time.

```
1 ALTER TABLE DEPENDENTS
2 ADD CONSTRAINT fk_dependent_emp
3 foreign key(EMPNO)
4 references EMP ON DELETE CASCADE;
```

Data Output Messages Notifications

ALTER TABLE

Query returned successfully in 36 msec.

ON DELETE CASCADE is an option used when setting foreign key constraints in the database. It defines that when a record in the referenced table (parent table) is deleted, for the table (child table) containing the referenced key (foreign key) The behavior of how associated records should be processed.

6. Explain Errors for Integrity Constraint

The screenshot shows a pgAdmin 4 interface with a query editor containing several SQL statements. The statements include various INSERT INTO commands for the EMP and SALGRADE tables, along with some commented-out code. Below the editor, there are tabs for Data Output, Messages, and Notifications. The Messages tab displays two error messages related to the NOT NULL constraint on the MGR column of the EMP table.

```
35      (7844, 'TURNER', 'PETER', 'SALESMAN', 7698,
36      TO_DATE('18-09-1981', 'DD-MM-YYYY'), 1500, 0, '0149548243', 30);
37 INSERT INTO EMP VALUES
38      (7876, 'ADAMS', 'JOSEPH', 'CLERK', 7788,
39      TO_DATE('12-01-1983', 'DD-MM-YYYY'), 1100, NULL, '0149565243', 20);
40 INSERT INTO EMP VALUES
41      (7900, 'JAMES', 'ALAN', 'CLERK', 7698,
42      TO_DATE('3-12-1981', 'DD-MM-YYYY'), 950, NULL, '0149545564', 30);
43 INSERT INTO EMP VALUES
44      (7902, 'FORD', 'MARIA', 'ANALYST', 7566,
45      TO_DATE('3-12-1981', 'DD-MM-YYYY'), 3000, NULL, '0149785243', 20);
46 INSERT INTO EMP VALUES
47      (7934, 'MILLER', 'ALICE', 'CLERK', 7782,
48      TO_DATE('23-01-1982', 'DD-MM-YYYY'), 1300, NULL, '0199545243', 10);
49
50 INSERT INTO SALGRADE VALUES (1, 700, 1200);
51 INSERT INTO SALGRADE VALUES (2, 1201, 1400);
52 INSERT INTO SALGRADE VALUES (3, 1401, 2000);
53 INSERT INTO SALGRADE VALUES (4, 2001, 3000);
54 INSERT INTO SALGRADE VALUES (5, 3001, 9999);
```

Data Output Messages Notifications

ERROR: Failing row contains (7839, KING, GUY, PRESIDENT, null, 1981-11-17, 5000, null, 0149545241, 10, null).null value in column "mgr" of relation "emp" violates not-null constraint

ERROR: null value in column "mgr" of relation "emp" violates not-null constraint
SQL state: 23502
Detail: Failing row contains (7839, KING, GUY, PRESIDENT, null, 1981-11-17, 5000, null, 0149545241, 10, null).

If run the code directly, we will find that due to the existence of null information, then need to do the following:

The screenshot shows a pgAdmin 4 interface with a query editor containing an ALTER TABLE statement. The statement uses the ALTER COLUMN clause to drop the NOT NULL constraint from the MGR column of the EMP table. Below the editor, there are tabs for Data Output, Messages, and Notifications. The Data Output tab shows the executed ALTER TABLE command.

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

Data Output Messages Notifications

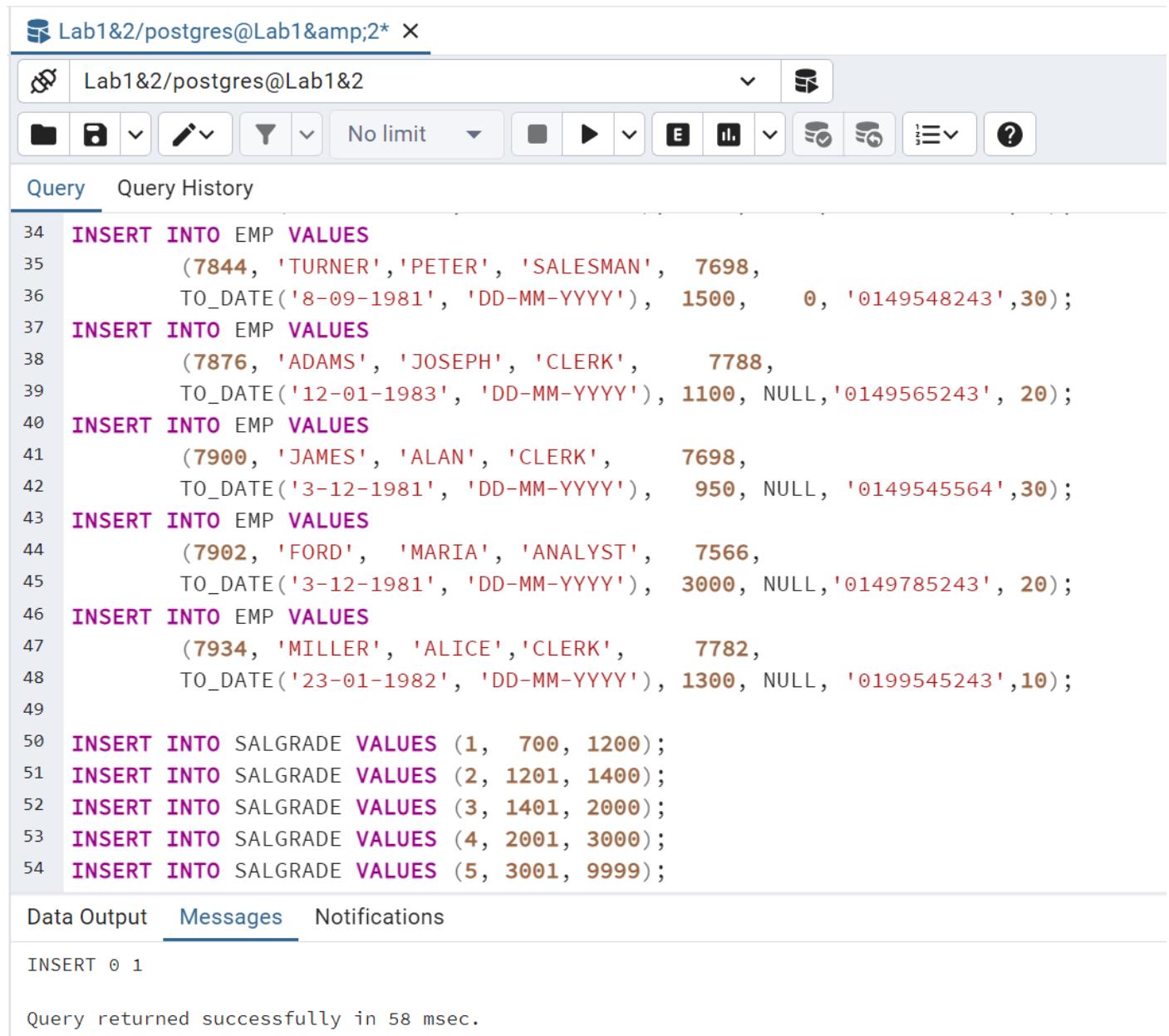
ALTER TABLE

Query returned successfully in 84 msec.

ALTER COLUMN MGR DROP NOT NULL is used to modify column attributes in database tables. This statement is used to remove the NOT NULL constraint on the

MGR column in the EMP table. In SQL, the NOT NULL constraint is used to ensure that a column cannot contain null values (NULL), that is, the column must have a value in each row.

Then run the before code again:



The screenshot shows a PostgreSQL database client interface. The title bar says "Lab1&2/postgres@Lab1&2* X". The toolbar includes icons for connection, refresh, search, and various database operations. Below the toolbar is a menu bar with "Query" and "Query History" tabs, where "Query" is selected. The main area contains a multi-line text input field displaying a series of SQL INSERT statements for the EMP and SALGRADE tables. The statements are numbered from 34 to 54. The "Messages" tab is selected at the bottom, showing the output "INSERT 0 1" and the message "Query returned successfully in 58 msec."

```
34 INSERT INTO EMP VALUES
35     (7844, 'TURNER', 'PETER', 'SALESMAN', 7698,
36     TO_DATE('8-09-1981', 'DD-MM-YYYY'), 1500, 0, '0149548243', 30);
37 INSERT INTO EMP VALUES
38     (7876, 'ADAMS', 'JOSEPH', 'CLERK', 7788,
39     TO_DATE('12-01-1983', 'DD-MM-YYYY'), 1100, NULL, '0149565243', 20);
40 INSERT INTO EMP VALUES
41     (7900, 'JAMES', 'ALAN', 'CLERK', 7698,
42     TO_DATE('3-12-1981', 'DD-MM-YYYY'), 950, NULL, '0149545564', 30);
43 INSERT INTO EMP VALUES
44     (7902, 'FORD', 'MARIA', 'ANALYST', 7566,
45     TO_DATE('3-12-1981', 'DD-MM-YYYY'), 3000, NULL, '0149785243', 20);
46 INSERT INTO EMP VALUES
47     (7934, 'MILLER', 'ALICE', 'CLERK', 7782,
48     TO_DATE('23-01-1982', 'DD-MM-YYYY'), 1300, NULL, '0199545243', 10);
49
50 INSERT INTO SALGRADE VALUES (1, 700, 1200);
51 INSERT INTO SALGRADE VALUES (2, 1201, 1400);
52 INSERT INTO SALGRADE VALUES (3, 1401, 2000);
53 INSERT INTO SALGRADE VALUES (4, 2001, 3000);
54 INSERT INTO SALGRADE VALUES (5, 3001, 9999);
```

Data Output Messages Notifications

INSERT 0 1

Query returned successfully in 58 msec.

salgrade->view data check results:

The screenshot shows the pgAdmin interface with two tabs open: 'Lab1&2/postgres...' and 'public.salgrade/Lab1&2/postgres@Lab1&2'. The second tab is active, displaying the results of a query. The query is:

```
1 SELECT * FROM public.salgrade
```

The results are displayed in a table with three columns: grade, losal, and hisal. The data is as follows:

	grade integer	losal integer	hisal integer
1	1	700	1200
2	2	1201	1400
3	3	1401	2000
4	4	2001	3000
5	5	3001	9999

7. Define a sequence

The screenshot shows the pgAdmin interface with two tabs open: 'Lab1&2/postgres@Lab1&2*' and 'Lab1&2/postgres@Lab1&2'. The second tab is active, displaying the results of a query. The query is:

```
1 CREATE SEQUENCE SequenceDepl
2 START with 8000
3 INCREMENT BY 1
```

The results are displayed in a table with three columns: Data Output, Messages, and Notifications. The 'Messages' tab is active, showing the message:

CREATE SEQUENCE

Query returned successfully in 36 msec.

8. Use Sequence:

The screenshot shows the pgAdmin 4 interface. At the top, there's a connection bar with the connection name "Lab1&2/postgres@Lab1&2". Below it is a toolbar with various icons for database management. The main area has tabs for "Query" and "Query History", with "Query" currently selected. A code editor window contains the following SQL queries:

```
1 SELECT nextval('SequenceDepl');
2 INSERT INTO DEPENDENTS VALUES (NEXTVAL('SequenceDepl'), 'xxx', 'yyy', 7654);
```

Below the code editor is a "Data Output" tab, which is also selected. It displays a table with one row of data:

	nextval	bigint
	7654	

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

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

The screenshot shows the pgAdmin 4 interface. The connection bar and toolbar are identical to the previous screenshot. The "Query" tab is selected in the main area, and a code editor window contains the following SQL query:

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

Below the code editor is a "Data Output" tab, which is selected. It displays a table with the names of all tables in the public schema:

	table_name
1	dept
2	dependents
3	bonus
4	salgrade
5	emp

2. Select employees whose commission is higher than their salary.

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

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

The results grid displays one row of data:

	empno	ename	efirst	job	mgr	hiredate	sal	comm	tel
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784

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

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

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

The results grid displays four rows of data:

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

4. Select employees who are CLERK or ANALYST

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer.

Query:

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

Data Output:

	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
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243
2	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243
4	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564
5	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243
6	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243

5. Select employees whose name begins by M.

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer.

Query:

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

Data Output:

	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
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243

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

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer. The query is:

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

The data output shows three rows of employee data:

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

7. Select employees who are MANAGER or CLERK in the department 10 and whose salary is greater than 1500.

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer. The query is:

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

The data output shows one row of employee data:

	empno	ename	efirst	job	mgr	hiredate	sal	comm	tel
	[PK] integer	character varying (10)	character varying (10)	character varying (9)	integer	date	integer	integer	character
1	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245

8. Select employees whose commission is NULL

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer. The query is:

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

The data output shows 10 rows of employee data, all of which have a null value in the 'comm' column.

	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
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243
2	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456
3	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254
4	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245
5	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
6	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241
7	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243
8	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564
9	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243
10	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243

9. Select employees by ascending order (by hiredate)

The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer. The query is:

```
1 SELECT * FROM EMP
2 ORDER BY HIREDATE ASC;
```

The data output shows 14 rows of employee data, sorted by hiredate in ascending order.

	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
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456
5	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254
6	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245
7	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243
8	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784
9	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241
10	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243
11	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564
12	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243
13	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
14	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243

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

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

```
1 SELECT * FROM EMP
2 ORDER BY JOB asc,
3 SAL DESC;
```

The results grid displays 14 rows of employee data 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
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
2	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243
3	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243
4	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243
5	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564
6	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243
7	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456
8	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254
9	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245
10	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241
11	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243
12	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243
13	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247
14	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784

11. Select departments without employees

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

```
1 SELECT * FROM DEPT
2 WHERE DEPTNO
3 NOT IN (SELECT DEPTNO FROM EMP);
```

The results grid displays 1 row of department data with the following columns:

	deptno [PK] integer	dname character varying (14)	loc character varying (13)
1	40	OPERATIONS	BOSTON

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

Lab1&2/postgres@Lab1&2* X

Lab1&2/postgres@Lab1&2 No limit E II ✓

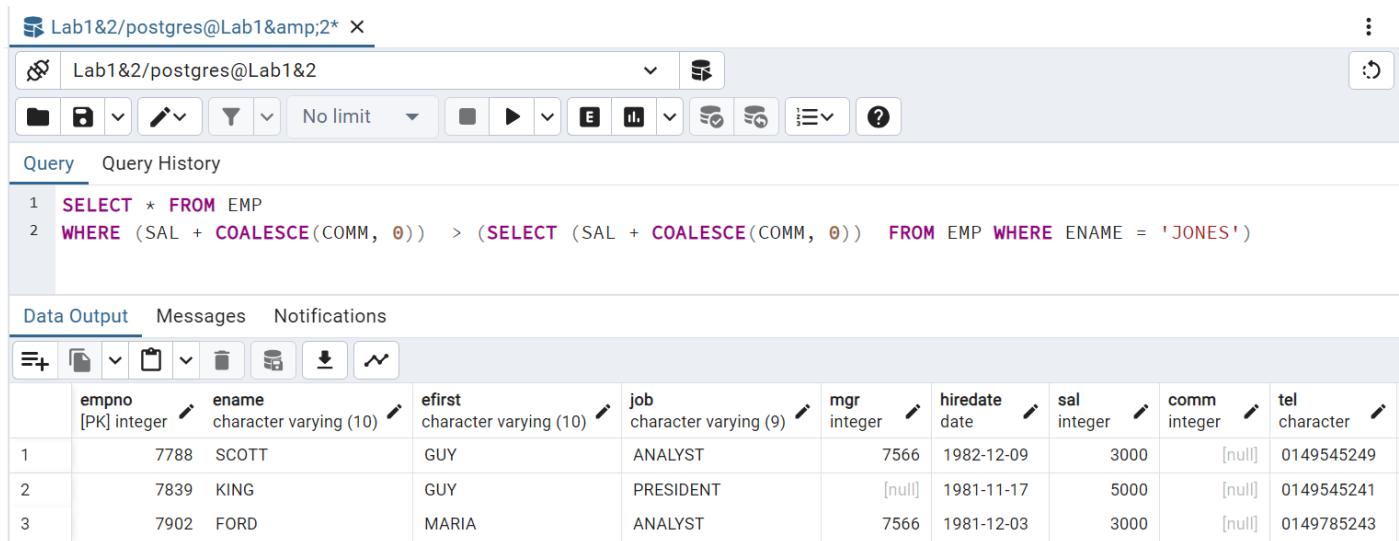
Query History

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

Data Output Messages Notifications

	empno [PK] integer	ename character varying (10)	manager_name character varying (10) 🔒
1	7369	SMITH	FORD
2	7499	ALLEN	BLAKE
3	7521	WARD	BLAKE
4	7566	JONES	KING
5	7654	MARTIN	BLAKE
6	7698	BLAKE	KING
7	7782	CLARK	KING
8	7788	SCOTT	JONES
9	7839	KING	[null]
10	7844	TURNER	BLAKE
11	7876	ADAMS	SCOTT
12	7900	JAMES	BLAKE
13	7902	FORD	JONES
14	7934	MILLER	CLARK

13. List employees earning more than JONES



The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer.

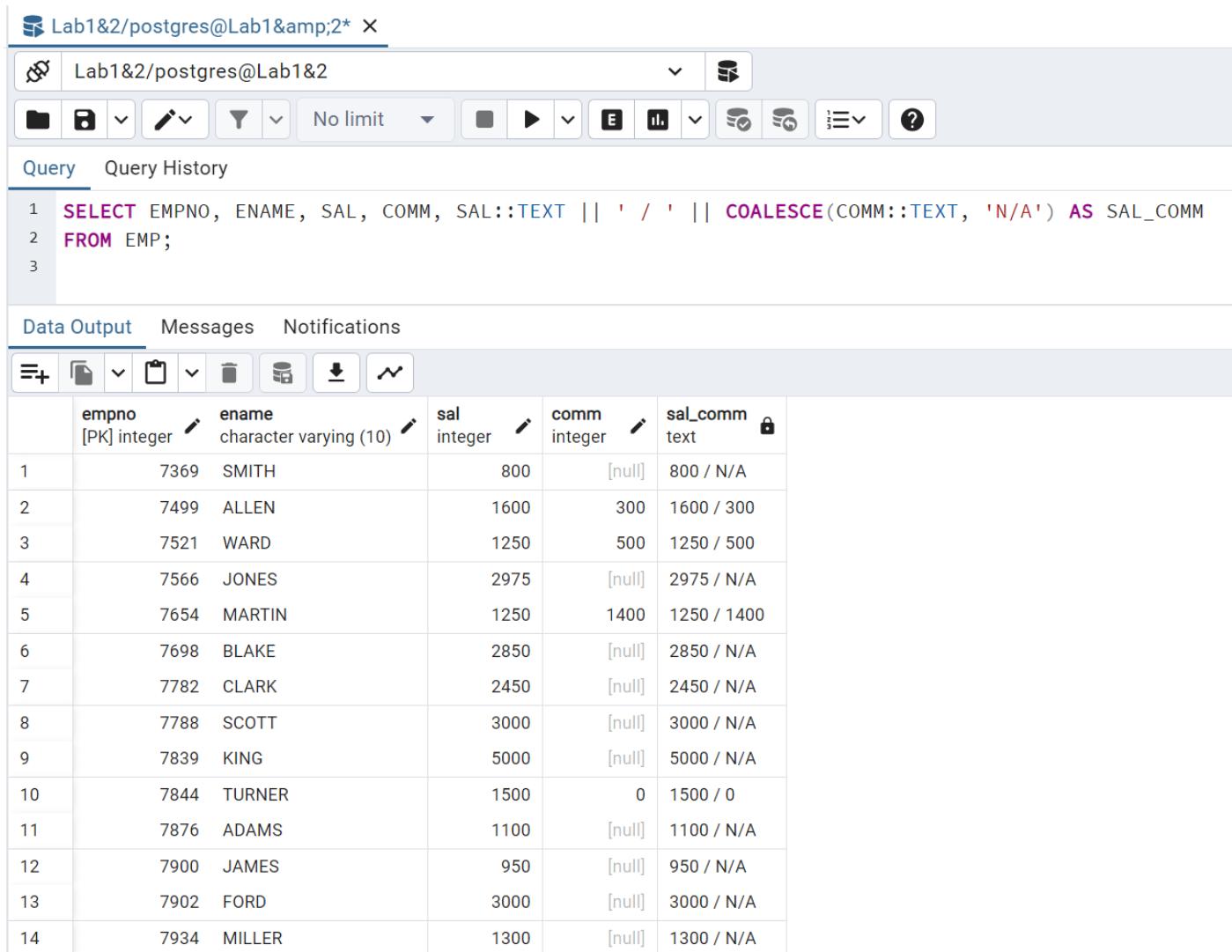
Query:

```
1 SELECT * FROM EMP
2 WHERE (SAL + COALESCE(COMM, 0)) > (SELECT (SAL + COALESCE(COMM, 0)) FROM EMP WHERE ENAME = 'JONES')
```

Data Output:

	empno	ename	efirst	job	mgr	hiredate	sal	comm	tel
	[PK] integer	character varying (10)	character varying (10)	character varying (9)	integer	date	integer	integer	character
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
2	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241
3	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243

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



The screenshot shows the pgAdmin 4 interface with a query editor and a data output viewer.

Query:

```
1 SELECT EMPNO, ENAME, SAL, COMM, SAL::TEXT || ' / ' || COALESCE(COMM::TEXT, 'N/A') AS SAL_COMM
2 FROM EMP;
```

Data Output:

	empno	ename	sal	comm	sal_comm
	[PK] integer	character varying (10)	integer	integer	text
1	7369	SMITH	800	[null]	800 / N/A
2	7499	ALLEN	1600	300	1600 / 300
3	7521	WARD	1250	500	1250 / 500
4	7566	JONES	2975	[null]	2975 / N/A
5	7654	MARTIN	1250	1400	1250 / 1400
6	7698	BLAKE	2850	[null]	2850 / N/A
7	7782	CLARK	2450	[null]	2450 / N/A
8	7788	SCOTT	3000	[null]	3000 / N/A
9	7839	KING	5000	[null]	5000 / N/A
10	7844	TURNER	1500	0	1500 / 0
11	7876	ADAMS	1100	[null]	1100 / N/A
12	7900	JAMES	950	[null]	950 / N/A
13	7902	FORD	3000	[null]	3000 / N/A
14	7934	MILLER	1300	[null]	1300 / N/A

Note: The COALESCE function requires all parameters to be of the same type

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

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

```
1 SELECT DISTINCT DEPTNO
2 FROM EMP
3 WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT);
```

The results grid displays the following data:

deptno
30
10
20

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

The screenshot shows the pgAdmin 4 interface with a query editor and a results grid. The query is:

```
1 SELECT E.* FROM EMP E
2 JOIN DEPT D ON E.DEPTNO = D.DEPTNO
3 WHERE D.LOC = 'CHICAGO' AND E.JOB = (SELECT JOB FROM EMP WHERE ENAME = 'JONES');
```

The results grid displays the following data:

empno	ename	efirst	job	mgr	hiredate	sal	comm	tel
7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254

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

Lab1&2/postgres@Lab1&2* ×

Lab1&2/postgres@Lab1&2

No limit

	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
1	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254

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

Lab1&2/postgres@Lab1&2* ×

Lab1&2/postgres@Lab1&2

No limit

	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
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456
5	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784
6	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254
7	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245
8	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249
9	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241
10	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243
11	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243
12	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564
13	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243
14	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243

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

```

Lab1&2/postgres@Lab1&2* 
Lab1&2/postgres@Lab1&2
No limit
SELECT *
FROM EMP
WHERE JOB IN (SELECT JOB FROM EMP WHERE DEPTNO = (SELECT DEPTNO FROM DEPT WHERE DNAME = 'SALES')) AND DEPTNO = 10;

```

Data Output Messages Notifications

	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	deptno integer
1	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	1
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	1

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

```

Lab1&2/postgres@Lab1&2* 
Lab1&2/postgres@Lab1&2
No limit
SELECT *
FROM EMP
WHERE JOB = (SELECT JOB FROM EMP WHERE ENAME = 'JONES')
OR SAL > (SELECT SAL FROM EMP WHERE ENAME = 'FORD');

```

Data Output Messages Notifications

	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	deptno integer
1	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	2
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	2
3	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	1
4	7839	KING	GUY	PRESIDENT	[null]	1981-11-17	5000	[null]	0149545241	1

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

Lab1&2/postgres@Lab1&2* X

Lab1&2/postgres@Lab1&2 No limit

Query History

```
1 SELECT E1.ENAME, E1.ENAME, E1.TEL, E2.ENAME AS MANAGER
2 FROM EMP E1 INNER JOIN EMP E2 ON E1.MGR = E2.EMPNO;
```

Data Output Messages Notifications

	ename character varying (10) 	ename character varying (10) 	tel character 	manager character varying (10) 
1	SMITH	SMITH	0149545243	FORD
2	ALLEN	ALLEN	0149547243	BLAKE
3	WARD	WARD	0149545247	BLAKE
4	JONES	JONES	0149545456	KING
5	MARTIN	MARTIN	0149545784	BLAKE
6	BLAKE	BLAKE	0149545254	KING
7	CLARK	CLARK	0149545245	KING
8	SCOTT	SCOTT	0149545249	JONES
9	TURNER	TURNER	0149548243	BLAKE
10	ADAMS	ADAMS	0149565243	SCOTT
11	JAMES	JAMES	0149545564	BLAKE
12	FORD	FORD	0149785243	JONES
13	MILLER	MILLER	0199545243	CLARK

Exercice 3: Join Table

1. Create a Table for employee's Projects:

The screenshot shows the pgAdmin 4 interface with the connection set to 'Lab1&2/postgres@Lab1&2'. The toolbar includes icons for connection, refresh, search, and various database operations. The main area has tabs for 'Query' (selected) and 'Query History'. The query editor contains the following SQL code:

```
1 CREATE TABLE project (
2     projno INTEGER PRIMARY KEY,
3     pname VARCHAR(255),
4     startdate DATE,
5     budget DECIMAL
6 );
7
```

Below the query editor, there are tabs for 'Data Output', 'Messages' (selected), and 'Notifications'. The message pane shows the output of the query:

CREATE TABLE

Query returned successfully in 69 msec.

2. Create the Join Table:

Create table project_emp:

The screenshot shows the pgAdmin 4 interface with the connection set to 'Lab1&2/postgres@Lab1&2'. The toolbar and tabs are identical to the previous screenshot. The query editor contains the following SQL code:

```
1 CREATE TABLE project_emp (
2     empno INTEGER,
3     projno INTEGER,
4     PRIMARY KEY (empno, projno),
5     FOREIGN KEY (empno) REFERENCES EMP (EMPNO),
6     FOREIGN KEY (projno) REFERENCES project (projno)
7 );
8
```

Below the query editor, there are tabs for 'Data Output', 'Messages' (selected), and 'Notifications'. The message pane shows the output of the query:

CREATE TABLE

Query returned successfully in 75 msec.

Insert project data into the project table:

The screenshot shows the pgAdmin 4 interface. The title bar says "Lab1&2/postgres@Lab1&2*". The toolbar has various icons for database management. Below the toolbar, there are tabs for "Query" and "Query History", with "Query" selected. The main area contains a numbered SQL query:

```
1 INSERT INTO project (projno, pname, startdate, budget) VALUES
2 (1, 'Project A', '2024-01-01', 100000),
3 (2, 'Project B', '2024-02-01', 200000),
4 (3, 'Project C', '2024-03-01', 300000),
5 (4, 'Project D', '2023-04-01', 400000);
6
```

Below the query, there are three tabs: "Data Output", "Messages", and "Notifications", with "Messages" selected. The message pane shows the output of the query:

INSERT 0 4

Query returned successfully in 73 msec.

Insert project data into the project_emp table:

The screenshot shows the pgAdmin 4 interface. The title bar says "Lab1&2/postgres@Lab1&2*". The toolbar has various icons for database management. Below the toolbar, there are tabs for "Query" and "Query History", with "Query" selected. The main area contains a numbered SQL query:

```
1 INSERT INTO project_emp (empno, projno) VALUES
2 (7369, 1),
3 (7369, 2),
4 (7369, 3),
5 (7369, 4);
6
7 TNSERT TINTO project_emp (empno, projno) VALUES
```

Below the query, there are three tabs: "Data Output", "Messages", and "Notifications", with "Messages" selected. The message pane shows the output of the query:

INSERT 0 26

Query returned successfully in 45 msec.

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

The screenshot shows the pgAdmin 4 interface. At the top, there are two tabs: 'Lab1&2/postgres@Lab1&2*' and 'public.emp/Lab1...'. Below the tabs is a toolbar with various icons for file operations, search, and navigation. The main area has tabs for 'Query' (which is selected) and 'Query History'. Under the 'Query' tab, the following SQL code is displayed:

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

Below the code, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is selected, showing a results grid. The grid has one column labeled 'empno' with a lock icon. There is one row with the value '7369'.

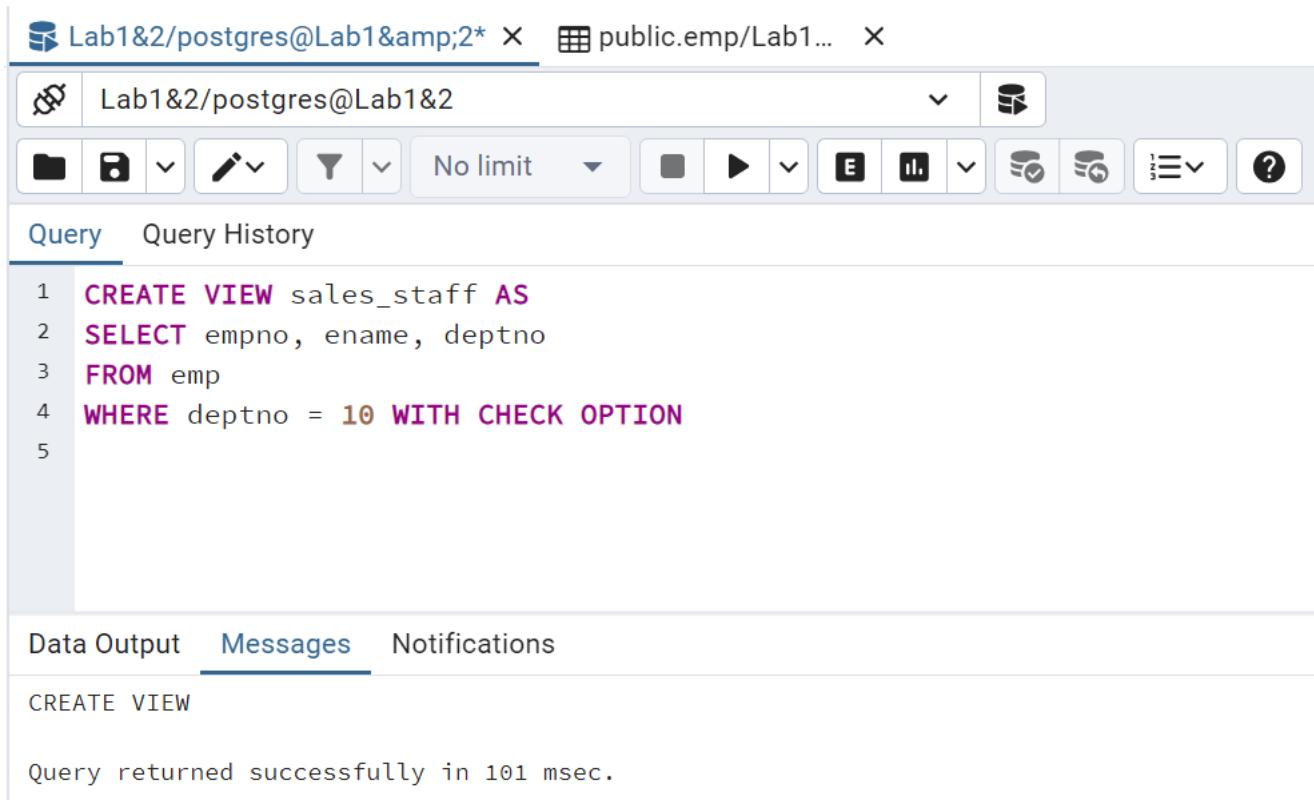
empno
7369

4. Options on View creation:

“CREATE VIEW” statement creates a view named sales_staff, Select the data from the three fields empno, ename and deptno from the emp table, which only contains information about employees with department number 10.

“WITH CHECK OPTION” ensures that all data modifications through the view (including insert and update operations) will comply with the “WHERE “condition in the view definition: all insert or update operations through the sales_staff view must ensure that deptno remains 10.

5. View Creation:

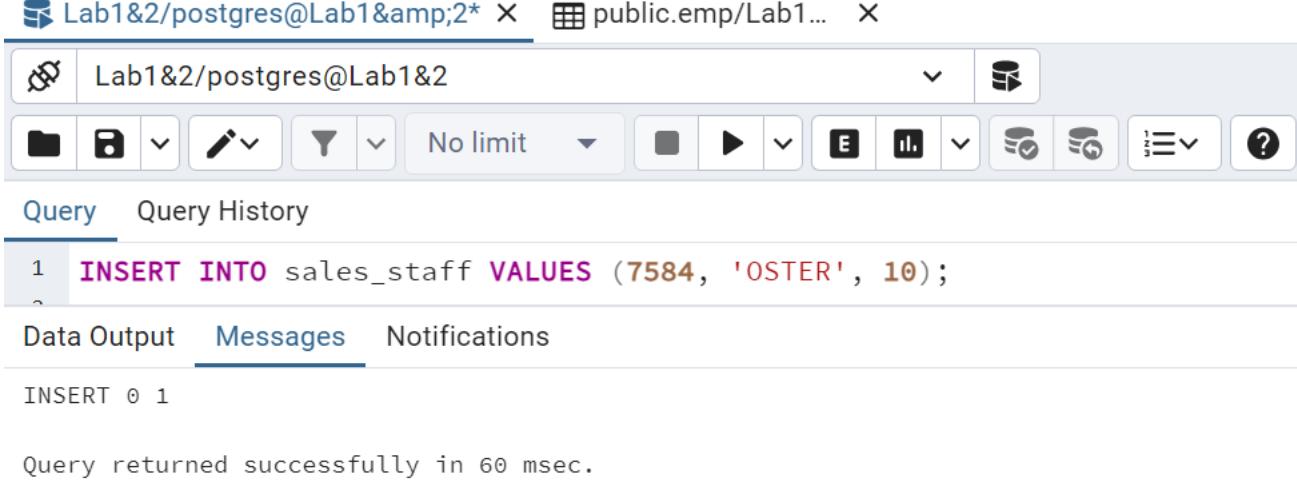


The screenshot shows the pgAdmin 4 interface with a connection to Lab1&2/postgres@Lab1&2. A new query window is open, titled 'public.emp/Lab1...'. The query editor contains the following SQL code:

```
1 CREATE VIEW sales_staff AS
2 SELECT empno, ename, deptno
3 FROM emp
4 WHERE deptno = 10 WITH CHECK OPTION
5
```

The 'Messages' tab is selected, showing the execution results:

```
CREATE VIEW
Query returned successfully in 101 msec.
```



The screenshot shows the pgAdmin 4 interface with the same connection. A new query window is open, titled 'public.emp/Lab1...'. The query editor contains the following SQL code:

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

The 'Messages' tab is selected, showing the execution results:

```
INSERT 0 1
Query returned successfully in 60 msec.
```

Insert a new record with deptno 10 into the sales_staff view. Insert operations are allowed.

Lab1&2/postgres@Lab1&2* X public.emp/Lab1... X

Lab1&2/postgres@Lab1&2

No limit

Query History

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

Data Output Messages Notifications

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

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

Insert a record with deptno 30 into the sales_staff view. This violates “deptno=10” as well as “WITH CHECK OPTION”. Therefore, this insert operation is rejected.

6. Division Query:

Lab1&2/postgres@Lab1&2* X public.emp/Lab1... X

Lab1&2/postgres@Lab1&2

No limit

Query History

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

Data Output Messages Notifications

	empno	integer
1	7369	

7. Analyze:

Lab1&2/postgres@Lab1&2* X public.emp/Lab1... X

Lab1&2/postgres@Lab1&2

No limit

Query History

```
1 SELECT empno, COUNT(projno) AS num_projects
2 FROM project_emp
3 GROUP BY empno
4 HAVING COUNT(projno) >= 2;
```

Data Output Messages Notifications

	empno integer	num_projects bigint
1	7839	2
2	7902	2
3	7698	2
4	7369	4
5	7499	2
6	7900	2
7	7788	2
8	7876	2
9	7782	2
10	7844	2
11	7934	2
12	7566	2
13	7521	2
14	7654	2

Lab 2-Advanced SQL

Exercice 1: Analytics Queries . Window Queries

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

```
1 SELECT * FROM (
2   SELECT EMPNO, EFIRST, ENAME, HIREDATE, DEPTNO,
3   RANK() OVER (partition by DEPTNO order by HIREDATE desc) AS RANK
4   FROM EMP
5 )
6 WHERE RANK <= 2;
```

Data Output Messages Notifications

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

2. Show your analytical Skill and Invents an interesting query using Windows Functions.

Query Query History

```

1 SELECT EMPNO, EFIRST, ENAME, HIREDATE, DEPTNO, SAL,
2   MIN (SAL) OVER (ORDER BY HIREDATE
3 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS min
4 FROM EMP

```

Data Output Messages Notifications

	empno [PK] integer	efirst character varying (10)	ename character varying (10)	hiredate date	deptno integer	sal integer	min integer
1	7369	JOHN	SMITH	1980-12-17	20	800	800
2	7499	BOB	ALLEN	1981-02-20	30	1600	800
3	7521	PETER	WARD	1981-02-22	30	1250	1250
4	7566	JOHN	JONES	1981-04-02	20	2975	1250
5	7698	BOB	BLAKE	1981-05-01	30	2850	2450
6	7782	JOHN	CLARK	1981-06-09	10	2450	1500
7	7844	PETER	TURNER	1981-09-08	30	1500	1250
8	7654	JOE	MARTIN	1981-09-28	30	1250	1250
9	7839	GUY	KING	1981-11-17	10	5000	950
10	7900	ALAN	JAMES	1981-12-03	30	950	950
11	7902	MARIA	FORD	1981-12-03	20	3000	950
12	7934	ALICE	MILLER	1982-01-23	10	1300	1300
13	7788	GUY	SCOTT	1982-12-09	20	3000	1100
14	7876	JOSEPH	ADAMS	1983-01-12	20	1100	1100
15	7584	[null]	OSTER	[null]	10	[null]	1100

Exercice 2: Index & Explain Plan

1. Execute this script.

Query Query History

```
27 FROM generate_series(1,5000) as S;
28 -----
29 ---- Create Join table: Project-Employee -
30 -----
31 CREATE TABLE PROJECT_EMP_MEDIUM_TABLE
32 (PROJECTNO integer,
33 EMPNO integer);
34 INSERT INTO PROJECT_EMP_MEDIUM_TABLE
35 SELECT ROUND(random()*100) AS PROJECTNO,
36 ROUND(random()*100) AS EMPNO
37 FROM generate_series(1,5000) as S;
38
```

Data Output Messages Notifications

INSERT 0 5000

Query returned successfully in 262 msec.

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

Lab1&2/postgres@Lab1&2

No limit

Query Query History

```
1 SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;
2
```

Data Output Messages Notifications

	gender	count
1	M	21
2	F	29

✓ Successfully run. Total query runtime: 59 msec. 2 rows affected. ✘

Total rows: 2 of 2 Query complete 00:00:00.059 Ln 2, Col 1

3. 1 Use EXPLAIN plan to analyze the query

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for 'Query' (which is selected) and 'Query History'. Below the tabs is a code editor containing the following SQL query:

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

Below the code editor are three tabs: 'Data Output' (selected), 'Messages', and 'Notifications'. A toolbar with various icons is located above the results area. The results area displays the 'QUERY PLAN' for the query, with the text:

```
1 HashAggregate (cost=114.75..114.77 rows=2 width=10)
2   Group Key: gender
3     -> Seq Scan on emp_medium_table (cost=0.00..114.50 rows=50 width=...
4       Filter: (manager_id = 7)
```

3. 2 Activate stats.

```
#local preload libraries =
#session preload libraries =
shared preload libraries = 'pg_stat_statements' # (change requires restart)
#jit provider = 'llvmjit'           # JIT library to use
```

Run:

The screenshot shows a PostgreSQL query editor interface. At the top, it says 'Lab1&2/postgres@Lab1&2'. Below the connection info is a toolbar with icons for file operations and a dropdown menu set to 'No limit'. There are tabs for 'Query' (selected) and 'Query History'. The code editor contains the following SQL command:

```
1 create extension pg_stat_statements;
```

Below the code editor are three tabs: 'Data Output' (selected), 'Messages', and 'Notifications'. The 'Messages' tab shows the output of the command:

```
CREATE EXTENSION
```

And the 'Notifications' tab shows:

```
Query returned successfully in 154 msec.
```

Services

File Action View Help

Services (Local)

PostgreSQL Scheduling Agent - pgagent-pg16****

[Stop](#) the service
[Pause](#) the service
[Restart](#) the service

Description:
Provides the ability to schedule tasks within a Postgres database server.

Name	Description	Status	Startup Type	Loc
Payments and NFC/SE Mana...	Manages pa...	Manual (Trigg...	Manual (Trigg...	Loc
Peer Name Resolution Proto...	Enables serv...	Manual	Manual	Loc
Peer Networking Grouping	Enables mul...	Manual	Manual	Loc
Peer Networking Identity M...	Provides ide...	Manual	Manual	Loc
PenService_43d40	Pen Service	Manual (Trigg...	Manual (Trigg...	Loc
Performance Counter DLL H...	Enables rem...	Manual	Manual	Loc
Performance Logs & Alerts	Performance...	Manual	Manual	Loc
Phone Service	Manages th...	Manual (Trigg...	Manual (Trigg...	Loc
PimIndexMaintenanceSvc_4...	Indexes cont...	Running	Manual	Loc
Plug and Play	Enables a co...	Running	Manual	Loc
PNRP Machine Name Public...	This service ...	Manual	Manual	Loc
Portable Device Enumerator ...	Enforces gro...	Manual (Trigg...	Manual (Trigg...	Loc
PostgreSQL Scheduling Age...	Provides the...	Running	Automatic	\pc
postgresql-x64-16	Provides role...	Running	Automatic	N/A

Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>net stop postgresql-x64-16
The postgresql-x64-16 - PostgreSQL Server 16 service is stopping.
The postgresql-x64-16 - PostgreSQL Server 16 service was stopped successfully.

C:\Windows\System32>net start postgresql-x64-16
The postgresql-x64-16 - PostgreSQL Server 16 service is starting.
The postgresql-x64-16 - PostgreSQL Server 16 service was started successfully.

C:\Windows\System32>
```

Lab1&2/postgres@Lab1&2* X

Lab1&2/postgres@Lab1&2

Query History

```
1 SELECT * FROM pg_stat_statements
2 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
3
4
```

Data Output Messages Notifications

plan_time	calls	total_exec_time	min_exec_time	max_exec_time	mean_exec_time	stddev_exec_time	rows
precision	bigint	double precision	bigint				

Re-run 10 times:

The screenshot shows the pgAdmin interface with a query editor window. The title bar says "Lab1&2/postgres@Lab1&2*". The toolbar has various icons for database management. Below the toolbar, there are tabs for "Query" and "Query History", with "Query" selected. A single-line query is entered: "SELECT gender, count(*) from EMP_MEDIUM_TABLE where MANAGER_ID = 7 group by gender;".

Data Output Messages Notifications

A table titled "Data Output" showing the results of the query. It has two columns: "gender" and "count". There are two rows: one for gender M with count 21, and one for gender F with count 29.

	gender character varying (2)	count bigint
1	M	21
2	F	29

Lab1&2/postgres@Lab1&2* X

The screenshot shows the pgAdmin interface with a second query editor window. The title bar says "Lab1&2/postgres@Lab1&2". The toolbar has various icons for database management. Below the toolbar, there are tabs for "Query" and "Query History", with "Query" selected. Two lines of a query are visible: "SELECT * FROM pg_stat_statements" and "WHERE query like '%group by gender' and query not like 'EXPLAIN%'".

Data Output Messages Notifications

A table titled "Data Output" showing the results of the query. It has eight columns: n_time_cision, calls, total_exec_time, min_exec_time, max_exec_time, mean_exec_time, and stddev_exec_time. There is one row of data.

n_time_cision	calls	total_exec_time	min_exec_time	max_exec_time	mean_exec_time	stddev_exec_time
0	15	9.669500000000003	0.4254	1.1879	0.6446333333333333	0.2167003020046093

4. Add a covering index on both columns fetched:

The screenshot shows the pgAdmin 4 interface. The title bar says "Lab1&2/postgres@Lab1&2". The toolbar has various icons for file operations, search, and database management. Below the toolbar, there are tabs for "Query" and "Query History", with "Query" selected. A single line of SQL code is present in the query editor: "1 create index MANAGER_ID_GENDER_INDEX ON EMP_MEDIUM_TABLE(MANAGER_ID, GENDER);". Below the editor, there are tabs for "Data Output", "Messages", and "Notifications", with "Messages" selected. The message area displays the output of the query: "CREATE INDEX" and "Query returned successfully in 193 msec."

5. Reset the stats:

The screenshot shows the pgAdmin 4 interface. The title bar says "Lab1&2/postgres@Lab1&2". The toolbar has various icons for file operations, search, and database management. Below the toolbar, there are tabs for "Query History" and "Messages", with "Query History" selected. A single line of SQL code is present in the query editor: "1 Alt O nnect pg_stat_statements_reset()". Below the editor, there are tabs for "Data Output", "Messages", and "Notifications", with "Data Output" selected. The data output area displays the result of the function call: "pg_stat_statements_reset void" and the number "1".

Lab1&2/postgres@Lab1&2* X

Lab1&2/postgres@Lab1&2

No limit

Query History

```
1 SELECT * FROM pg_stat_statements
2 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
```

Data Output Messages Notifications

userid	dbid	toplevel	queryid	query	plans	total_plan_time	min_plan_time
oid	oid	boolean	bigint	text	bigint	double precision	double precision

6. Re-run query the same query 15 times:

Lab1&2/postgres@Lab1&2* X

Lab1&2/postgres@Lab1&2

No limit

Query History

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

Data Output Messages Notifications

gender	count
character varying (2)	bigint
M	21
F	29

7. Use the EXPLAIN plan again and compare the plan before the INDEX

The screenshot shows the pgAdmin 4 interface. In the top bar, the connection is set to 'Lab1&2/postgres@Lab1&2'. The 'Query' tab is active, displaying the following SQL code:

```

1 SELECT * FROM pg_stat_statements
2 WHERE query like '%group by gender' and query not like 'EXPLAIN%'
3

```

The 'Data Output' tab is also active, showing the results of the query:

_time	calls	total_exec_time	min_exec_time	max_exec_time	mean_exec_time	stddev_exec_time	row
0	15	1.4356999999999998	0.0316	0.7213999999999999	0.0957133333333333	0.16768726394359496	big

For the query performance recorded in the “pg_stat_statements” view before and after the creation of the index “MANAGER_ID_GENDER_INDEX”:

- 1) the “Calls” remained at 15 times.
- 2) the “Mean Exec Time” dropped significantly.
- 3) the value shows that after the creation of the index, the average execution of the query time is reduced, which is direct evidence that indexes optimize query performance.
- 4) By creating indexes on the MANAGER_ID and GENDER columns, PostgreSQL can locate these specific rows with MANAGER_ID 7 more quickly and group and count more efficiently because the indexes provide This reduces the number of data pages that must be read and the amount of work that queries must perform.

Exercice 3: Data Dictionary

Lab1&2/postgres@Lab1&2

Query History

```

1 DROP TABLE IF EXISTS MY_OBJECTS;
2
3 CREATE TABLE MY_OBJECTS (
4     Object_name VARCHAR(100),
5     Type VARCHAR(50)
6 );
7
8 INSERT INTO MY_OBJECTS (Object_name, Type)
9 SELECT table_name, 'Table'
10 FROM information_schema.tables
11 WHERE table_schema = 'public';
12
13 INSERT INTO MY_OBJECTS (Object_name, Type)
14 SELECT column_name, 'Column'
15 FROM information_schema.columns
16 WHERE table_schema = 'public';
17
18 INSERT INTO MY_OBJECTS (Object_name, Type)
19 SELECT conname, 'Constraint'
20 FROM pg_constraint
21 WHERE connamespace = 'public'::regnamespace;
22
23 SELECT * FROM MY_OBJECTS;

```

Data Output Messages Notifications

	object_name	type
	character varying (100)	character varying (50)
1	dept	Table
2	dependents	Table
3	bonus	Table
4	salgrade	Table
5	emp	Table
6	project_emp	Table
7	project	Table
8	sales_staff	Table
9	emp_medium_table	Table
10	project_medium_table	Table
11	project_emp_medium_table	Table
12	pg_stat_statements_info	Table
13	pg_stat_statements	Table
14	my_objects	Table
15	grade	Column

93	efirst	Column
94	jit_functions	Column
95	local_blks_hit	Column
96	ename	Column
97	mean_exec_time	Column
98	name	Column
99	losal	Column
100	tel	Column
101	dname	Column
102	userid	Column
103	sal	Column
104	shared_blks_written	Column
105	pk_dept	Constraint
106	ck_sal	Constraint
107	pk_emp	Constraint
108	fk_emp_dept	Constraint
109	pk_dependent	Constraint
110	emp_unique	Constraint
111	chk_mobile_tel	Constraint
112	fk_dependent_emp	Constraint
113	project_pkey	Constraint
114	project_emp_pkey	Constraint
115	project_emp_empno_fkey	Constraint
116	project_emp_projno_fkey	Constraint

Exercice 4: Use postgres via CLI

```
SQL Shell (psql) × + ▾
Server [localhost]: Database [postgres]: company Port [5432]: Username [postgres]: 用户 postgres 的口令：
pgsql (16.2)
输入 "help" 来获取帮助信息。

company=# SELECT * FROM EMP;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | efirst | job | mgr | hiredate | sal | comm | tel | deptno |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 7499 | ALLEN | BOB | SALESMAN | 7698 | 1981-02-20 | 1600 | 300 | 0149547243 | 30
| 7521 | WARD | PETER | SALESMAN | 7698 | 1981-02-22 | 1250 | 500 | 0149545247 | 30
| 7566 | JONES | JOHN | MANAGER | 7839 | 1981-04-02 | 2975 |  | 0149545456 | 20
| 7654 | MARTIN | JOE | SALESMAN | 7698 | 1981-09-28 | 1250 | 1400 | 0149545784 | 30
| 7698 | BLAKE | BOB | MANAGER | 7839 | 1981-05-01 | 2850 |  | 0149545254 | 30
| 7782 | CLARK | JOHN | MANAGER | 7839 | 1981-06-09 | 2450 |  | 0149545245 | 10
| 7788 | SCOTT | GUY | ANALYST | 7566 | 1982-12-09 | 3000 |  | 0149545249 | 20
| 7839 | KING | GUY | PRESIDENT |  | 1981-11-17 | 5000 |  | 0149545241 | 10
| 7844 | TURNER | PETER | SALESMAN | 7698 | 1981-09-08 | 1500 | 0 | 0149548243 | 30
| 7876 | ADAMS | JOSEPH | CLERK | 7788 | 1983-01-12 | 1100 |  | 0149565243 | 20
| 7900 | JAMES | ALAN | CLERK | 7698 | 1981-12-03 | 950 |  | 0149545564 | 30
| 7902 | FORD | MARIA | ANALYST | 7566 | 1981-12-03 | 3000 |  | 0149785243 | 20
| 7934 | MILLER | ALICE | CLERK | 7782 | 1982-01-23 | 1300 |  | 0199545243 | 10
| 7369 | SMITH | JOHN | CLERK | 7902 | 1980-12-17 | 800 |  | 0149545243 | 30
(14 行记录)
```

Exercice 5: Transaction Part 1 – Beginner

1.& 2. & 3.

```
SQL Shell (psql) × + ▾
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | efirst | job | mgr | hiredate | sal | comm | tel | deptno |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 7499 | ALLEN | BOB | SALESMAN | 7698 | 1981-02-20 | 1600 | 300 | 0149547243 | 30
| 7521 | WARD | PETER | SALESMAN | 7698 | 1981-02-22 | 1250 | 500 | 0149545247 | 30
| 7566 | JONES | JOHN | MANAGER | 7839 | 1981-04-02 | 2975 |  | 0149545456 | 20
| 7654 | MARTIN | JOE | SALESMAN | 7698 | 1981-09-28 | 1250 | 1400 | 0149545784 | 30
| 7698 | BLAKE | BOB | MANAGER | 7839 | 1981-05-01 | 2850 |  | 0149545254 | 30
| 7782 | CLARK | JOHN | MANAGER | 7839 | 1981-06-09 | 2450 |  | 0149545245 | 10
| 7788 | SCOTT | GUY | ANALYST | 7566 | 1982-12-09 | 3000 |  | 0149545249 | 20
| 7839 | KING | GUY | PRESIDENT |  | 1981-11-17 | 5000 |  | 0149545241 | 10
| 7844 | TURNER | PETER | SALESMAN | 7698 | 1981-09-08 | 1500 | 0 | 0149548243 | 30
| 7876 | ADAMS | JOSEPH | CLERK | 7788 | 1983-01-12 | 1100 |  | 0149565243 | 20
| 7900 | JAMES | ALAN | CLERK | 7698 | 1981-12-03 | 950 |  | 0149545564 | 30
| 7902 | FORD | MARIA | ANALYST | 7566 | 1981-12-03 | 3000 |  | 0149785243 | 20
| 7934 | MILLER | ALICE | CLERK | 7782 | 1982-01-23 | 1300 |  | 0199545243 | 10
| 7369 | SMITH | JOHN | CLERK | 7902 | 1980-12-17 | 800 |  | 0149545243 | 30
(14 行记录)

company=# \set AUTOCOMMIT off
company=#
company=# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369;
UPDATE 1
company=*# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
UPDATE 1
company=*#
```

This update will not take effect immediately to the database because the transaction has not yet been committed.

4.-5.

The screenshot shows two separate SQL shells. The left shell displays the contents of the 'emp' table from the 'company' database. The right shell shows the PostgreSQL command-line interface (psql) with the following session:

```
Server [localhost]:  
Database [postgres]: company  
Port [5432]:  
Username [postgres]:  
用户 postgres 的口令：  
psql (16.2)  
输入 "help" 来获取帮助信息。  
company=# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;  
|
```

Since the first client's transaction has not yet committed, this command may hang waiting because the first client's transaction has a lock on the record. To make the update available for the second client, the first client should COMMIT.

6.-7.

The screenshot shows two separate SQL shells. The left shell displays the contents of the 'emp' table from the 'company' database. The right shell shows the PostgreSQL command-line interface (psql) with the following session:

```
Server [localhost]:  
Database [postgres]: company  
Port [5432]:  
Username [postgres]:  
用户 postgres 的口令：  
psql (16.2)  
输入 "help" 来获取帮助信息。  
company=# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;  
UPDATE 1  
company=# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;  
UPDATE 1  
company=# COMMIT;  
COMMIT  
company=# |
```

The first client's transaction has been committed, and the second client's update operation has been successfully executed.

Lab 2-Part 2 PL/SQL

Exercice 1: Functions

The screenshot shows the pgAdmin 4 interface with two connections: 'company/postgres@Lab1&2*' and 'company/postgre...'. The left connection is active. The query editor displays the creation of a PL/SQL procedure named EXERSISE1. The code uses the plpgsql language and defines a local variable R_EMP of type EMP%ROWTYPE. It performs a SELECT INTO operation to fetch employee details where EMPNO matches the input parameter in_EMP_ID. A RAISE NOTICE statement is used to output the employee's name and salary. The procedure ends with an END; statement and a \$procedure\$ delimiter.

```
1 CREATE OR REPLACE PROCEDURE EXERSISE1(in_EMP_ID NUMERIC)
2 LANGUAGE plpgsql
3 AS $procedure$
4 DECLARE
5     R_EMP EMP%ROWTYPE;
6 BEGIN
7     SELECT EMP.* INTO R_EMP FROM EMP WHERE EMPNO = in_EMP_ID;
8     --Print the employee name
9     RAISE NOTICE 'Employee Name : %, Employee Salary : %',
10     R_EMP.EFIRST, R_EMP.SAL;
11
12 END;
13 $procedure$
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 161 msec.

The screenshot shows the pgAdmin 4 interface with the same two connections. The right connection is active. The query editor runs the CALL command on the public.exercise1 procedure, passing the value 7499 for the IN parameter in_emp_id. The output shows a NOTICE message indicating the employee's name and salary.

```
1 CALL public.exercise1(
2     --<IN in_emp_id numeric>
3     7499
4 )
```

Data Output Messages Notifications

NOTICE: Employee Name : BOB, Employee Salary : 1600

CALL

Query returned successfully in 68 msec.

Exercice 2: Procedure & Display & Cursors

The screenshot shows a PostgreSQL query editor interface. The top bar displays the connection information "company/postgres@Lab1&2" and various toolbar icons. Below the toolbar, the tabs "Query" and "Query History" are visible, with "Query" being the active tab.

```
1 CREATE OR REPLACE PROCEDURE EXERCISE2(in_EMP_ID NUMERIC)
2 LANGUAGE plpgsql
3 AS $$ 
4 DECLARE
5     R_EMP RECORD;
6     AVG_SAL NUMERIC;
7
8 BEGIN
9
10    SELECT * FROM EMP WHERE EMPNO = in_EMP_ID;
11    SELECT AVG(SAL) into AVG_SAL FROM EMP
12        WHERE JOB IN(SELECT JOB FROM EMP WHERE EMPNO = in_EMP_ID);
13
14    RAISE NOTICE 'Employee Name: %, Employee Salary: %', R_EMP.ENAME, R_EMP.SAL;
15    RAISE NOTICE 'Average Salary for Job: %', AVG_SAL;
16 EXCEPTION
17 WHEN OTHERS THEN
18     RAISE NOTICE 'SQL ERROR OG QUERY : % ', SQLERRM;
19
20 END;
21 $$;
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 305 msec.

The screenshot shows a PostgreSQL query editor interface with two tabs open: "company/postgres@Lab1&2" and "company/postgres@Lab1&2*". The "Messages" tab is selected. The "Query" tab shows the execution of the procedure.

```
1 CALL public.exercise2(
2     --<IN in_emp_id numeric>
3     7499
4 )
```

Data Output Messages Notifications

NOTICE: SQL ERROR OG QUERY : query has no destination for result data
CALL

Query returned successfully in 115 msec.

Exercice 3. Procedure & Update

The screenshot shows the pgAdmin 4 interface with two tabs at the top: 'company/postgres@Lab1&2*' and 'company/postgre... X'. The left tab is active. Below the tabs is a toolbar with various icons. The main area is titled 'Query' and contains the following SQL code:

```
1 CREATE OR REPLACE PROCEDURE exercise3(in_EMP_ID NUMERIC)
2 LANGUAGE plpgsql
3 AS $procedure$
4 DECLARE
5     R_EMP EMP%ROWTYPE;
6     AVG_SAL NUMERIC;
7 BEGIN
8     SELECT * INTO R_EMP FROM emp WHERE EMPNO = in_EMP_ID;
9
10    SELECT AVG(sal) INTO AVG_SAL FROM emp;
11
12    RAISE NOTICE 'Initial salary: %', R_EMP.sal;
13
14    IF R_EMP.sal >= AVG_SAL THEN
15        UPDATE emp SET sal = sal * 1.10 WHERE EMPNO = in_EMP_ID;
16    ELSE
17        UPDATE emp SET sal = AVG_SAL WHERE EMPNO = in_EMP_ID;
18    END IF;
19
20    SELECT sal INTO R_EMP.sal FROM emp WHERE EMPNO = in_EMP_ID;
21    RAISE NOTICE 'Resulting salary: %', R_EMP.sal;
22 END;
$procedure$
```

Below the code, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the output of the CREATE PROCEDURE command:

CREATE PROCEDURE

Query returned successfully in 123 msec.

The screenshot shows the pgAdmin 4 interface with two tabs at the top: 'company/postgre... X' and 'company/postgres@Lab1&2*'. The right tab is active. Below the tabs is a toolbar with various icons. The main area is titled 'Query' and contains the following SQL code:

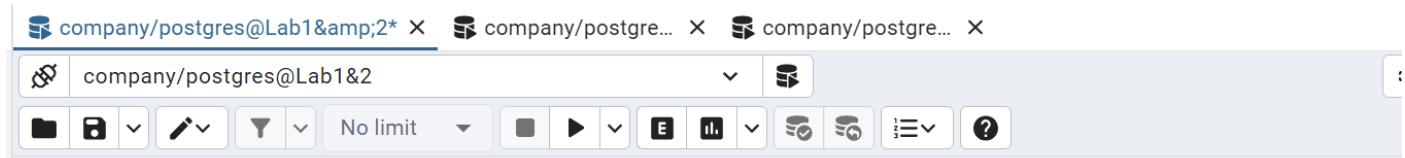
```
1 CALL public.exercise3(
2     --<IN in_emp_id numeric>
3     7499
4 )
```

Below the code, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the output of the procedure execution:

NOTICE: Initial salary: 1600
NOTICE: Resulting salary: 2516
CALL

Query returned successfully in 108 msec.

Exercice 4. Procedure



The screenshot shows the pgAdmin 4 interface with three tabs open in the background. The foreground tab is titled "company/postgres@Lab1&2" and contains a PostgreSQL query for creating a procedure named "exercise4()". The query includes logic to alter the "bonus" table if it doesn't exist, add a numeric column if it's missing, and calculate a bonus based on the employee's job type using a CASE statement. It also includes an RAISE NOTICE statement and a FOR loop to iterate over the "emp" table.

```
1 CREATE OR REPLACE PROCEDURE exercise4()
2 LANGUAGE plpgsql
3 AS $procedure$
4 DECLARE
5     R_EMP RECORD;
6     bonus NUMERIC;
7 BEGIN
8
9 IF NOT EXISTS (SELECT 1 FROM information_schema.columns
10                 WHERE table_name = 'bonus' AND column_name = 'job' AND data_type = 'text')
11 THEN
12     ALTER TABLE bonus ALTER COLUMN job TYPE TEXT;
13 END IF;
14
15
16 IF NOT EXISTS (SELECT 1 FROM information_schema.columns
17                 WHERE table_name = 'bonus' AND column_name = 'bonus' AND data_type = 'numeric')
18 THEN
19     ALTER TABLE bonus ADD COLUMN bonus NUMERIC;
20 END IF;
21
22 FOR R_EMP IN SELECT ename, job, comm, sal FROM emp LOOP
23     CASE R_EMP.job
24         WHEN 'SALESMAN' THEN bonus := R_EMP.comm * 2;
25         WHEN 'CLERK' THEN bonus := R_EMP.sal * 0.15;
26         WHEN 'MANAGER' THEN bonus := R_EMP.sal * 0.18;
27         ELSE bonus := 0;
28     END CASE;
29
30     INSERT INTO bonus (ename, job, bonus) VALUES (R_EMP.ename, R_EMP.job, bonus);
31
32
33     RAISE NOTICE 'Employee Name: %, Job: %, Bonus: %', R_EMP.ename, R_EMP.job, bonus;
34 END LOOP;
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 78 msec.

The screenshot shows a pgAdmin 4 interface with three tabs open in the top bar: 'company/postgre...', 'company/postgres@Lab1&2*', and 'company/postgre... X'. The active tab is 'company/postgres@Lab1&2'. The main area contains a query editor with the following content:

```
1 CALL public.exercise4()
```

Below the query editor is a message log:

```
NOTICE: Employee Name: WARD, Job: SALESMAN, Bonus: 2400
NOTICE: Employee Name: JONES, Job: MANAGER, Bonus: 535.50
NOTICE: Employee Name: MARTIN, Job: SALESMAN, Bonus: 2400
NOTICE: Employee Name: BLAKE, Job: MANAGER, Bonus: 513.00
NOTICE: Employee Name: CLARK, Job: MANAGER, Bonus: 441.00
NOTICE: Employee Name: SCOTT, Job: ANALYST, Bonus: 0
NOTICE: Employee Name: KING, Job: PRESIDENT, Bonus: 0
NOTICE: Employee Name: TURNER, Job: SALESMAN, Bonus: 2400
NOTICE: Employee Name: ADAMS, Job: CLERK, Bonus: 165.00
NOTICE: Employee Name: JAMES, Job: CLERK, Bonus: 142.50
NOTICE: Employee Name: FORD, Job: ANALYST, Bonus: 0
NOTICE: Employee Name: MILLER, Job: CLERK, Bonus: 195.00
NOTICE: Employee Name: SMITH, Job: CLERK, Bonus: 1050.00
NOTICE: Employee Name: ALLEN, Job: SALESMAN, Bonus: 2400
CALL
```

At the bottom of the message log, it says 'Query returned successfully in 93 msec.'

Exercice 5. SELECT for UPDATE

The screenshot shows a pgAdmin 4 interface with three tabs open in the top bar: 'company/postgres@Lab1&2*', 'company/postgre...', and 'company/postgre... X'. The active tab is 'company/postgres@Lab1&2'. The main area contains a query editor with the following content:

```
1 UPDATE EMP SET COMM = 0;
```

Below the query editor is a message log:

```
UPDATE 14
```

At the bottom of the message log, it says 'Query returned successfully in 60 msec.'

company/postgres@Lab1* X company/postgre... X company/postgre... X

company/postgres@Lab1&2

No limit ▾ E II ▾

Query History

```
1 CREATE OR REPLACE PROCEDURE EXERCISE5()
2 language plpgsql
3 AS $procedure$  
4  
5 DECLARE
6     ONE_EMP EMP%ROWTYPE;
7     NEW_COMM numeric;
8     E_SAL numeric;
9     SAL_EMP CURSOR FOR SELECT * FROM EMP FOR UPDATE;  
10  
11 BEGIN
12     --Open the cursor
13     OPEN SAL_EMP;
14     --Looping in the cursor
15     LOOP
16         FETCH SAL_EMP into ONE_EMP;
17         EXIT WHEN NOT FOUND;
18         --Cursor usage
19         E_SAL := ONE_EMP.SAL;
20         IF E_SAL <= 1000 THEN
21             NEW_COMM := 800;
22         ELSE IF E_SAL <= 2000 THEN
23             NEW_COMM := 1200;
24         ELSE
25             NEW_COMM := 1500;
26         END IF;
27         END IF;  
28  
29         UPDATE EMP SET COMM = NEW_COMM WHERE EMPNO = ONE_EMP.EMPNO;
30         raise notice 'Employee Number: %, Employee Salary : %, New Commission: %,' ,
31         ONE_EMP.EMPNO, ONE_EMP.SAL, NEW_COMM;  
32  
33     END LOOP;
34     --Closing the cursor
35     CLOSE SAL_EMP;  
36  
37 END
38 $procedure$
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 74 msec.

company/postgre... X company/postgres@Lab1&2*



company/postgres@Lab1&2



No limit ▾



Query Query History

1 `CALL public.exercise5()`

Data Output Messages Notifications

NOTICE: Employee Number; 7521, Employee Salary : 1250, New Commission: 1200,
NOTICE: Employee Number; 7566, Employee Salary : 2975, New Commission: 1500,
NOTICE: Employee Number; 7654, Employee Salary : 1250, New Commission: 1200,
NOTICE: Employee Number; 7698, Employee Salary : 2850, New Commission: 1500,
NOTICE: Employee Number; 7782, Employee Salary : 2450, New Commission: 1500,
NOTICE: Employee Number; 7788, Employee Salary : 3000, New Commission: 1500,
NOTICE: Employee Number; 7839, Employee Salary : 5000, New Commission: 1500,
NOTICE: Employee Number; 7844, Employee Salary : 1500, New Commission: 1200,
NOTICE: Employee Number; 7876, Employee Salary : 1100, New Commission: 1200,
NOTICE: Employee Number; 7900, Employee Salary : 950, New Commission: 800,
NOTICE: Employee Number; 7902, Employee Salary : 3000, New Commission: 1500,
NOTICE: Employee Number; 7934, Employee Salary : 1300, New Commission: 1200,
NOTICE: Employee Number; 7369, Employee Salary : 7000, New Commission: 1500,
NOTICE: Employee Number; 7499, Employee Salary : 2516, New Commission: 1500,
CALL

Query returned successfully in 145 msec.

Exercice 6. Condition on EXIT loop

The screenshot shows the pgAdmin interface with two tabs open: 'company/postgres@Lab1&2*' and 'company/postgres@Lab1&2'. The left tab contains the SQL code for creating the procedure:

```
1 CREATE OR REPLACE PROCEDURE exercise6()
2 LANGUAGE plpgsql
3 AS $procedure$
4 DECLARE
5     R_EMP EMP%ROWTYPE;
6     counter INT := 0;
7 BEGIN
8
9     RAISE NOTICE 'Highest salary employees!';
10    FOR R_EMP IN SELECT * FROM EMP ORDER BY sal DESC LIMIT 5 LOOP
11        RAISE NOTICE 'Employee Number: %, Employee Salary: %', R_EMP.empno, R_EMP.sal;
12    END LOOP;
13
14    RAISE NOTICE 'Lowest salary employees!';
15    FOR R_EMP IN SELECT * FROM EMP ORDER BY sal ASC LIMIT 5 LOOP
16        RAISE NOTICE 'Employee Number: %, Employee Salary: %', R_EMP.empno, R_EMP.sal;
17    END LOOP;
18 END;
19 $procedure$;
```

The right tab shows the results of the query:

CREATE PROCEDURE

Query returned successfully in 61 msec.

The screenshot shows the pgAdmin interface with two tabs open: 'company/postgres... X' and 'company/postgres@Lab1&2* X'. The left tab contains the SQL code for calling the procedure:

```
1 CALL public.exercise6()
```

The right tab shows the results of the query:

NOTICE: Highest salary employees:

NOTICE: Employee Number: 7369, Employee Salary: 7000

NOTICE: Employee Number: 7839, Employee Salary: 5000

NOTICE: Employee Number: 7788, Employee Salary: 3000

NOTICE: Employee Number: 7902, Employee Salary: 3000

NOTICE: Employee Number: 7566, Employee Salary: 2975

NOTICE: Lowest salary employees:

NOTICE: Employee Number: 7900, Employee Salary: 950

NOTICE: Employee Number: 7876, Employee Salary: 1100

NOTICE: Employee Number: 7654, Employee Salary: 1250

NOTICE: Employee Number: 7521, Employee Salary: 1250

NOTICE: Employee Number: 7934, Employee Salary: 1300

CALL

Query returned successfully in 62 msec.

Exercice 7. Triggers

1. Create the triggers & 2. Function to analyze results

Create - Table

General Columns Advanced Constraints Partitions Parameters Security SQL

Inherited from table(s) Select to inherit from...

Columns

		Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
⋮	-pencil	date_user	date	1 ⏺		<input type="checkbox"/>	<input type="checkbox"/>	
⋮	-pencil	user1	name	1 ⏺		<input type="checkbox"/>	<input type="checkbox"/>	
⋮	-pencil	ID	serial	1 ⏺		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Query Query History

```
1 CREATE OR REPLACE FUNCTION auto_insert() RETURNS TRIGGER AS $body$  
2  
3 BEGIN  
4     raise notice 'user doing the motification : %', current_user;  
5     raise notice 'date of motification : % ', current_date;  
6  
7     insert into monitor(user1, date_user) VALUES (current_user, current_date);  
8 RETURN NULL;  
9 END  
$body$  
11 LANGUAGE plpgsql;  
12  
13 CREATE OR REPLACE TRIGGER MYTRIGGER  
14 AFTER INSERT OR UPDATE OR DELETE ON EMP  
15 EXECUTE PROCEDURE auto_insert();  
16  
17 UPDATE EMP  
18 SET EFIRST = 'JEAN' WHERE EMPNO = 7369;  
19
```

Data Output Messages Notifications

NOTICE: user doing the motification : postgres
NOTICE: date of motification : 2024-04-01
UPDATE 1

Query returned successfully in 75 msec.

Lab 3

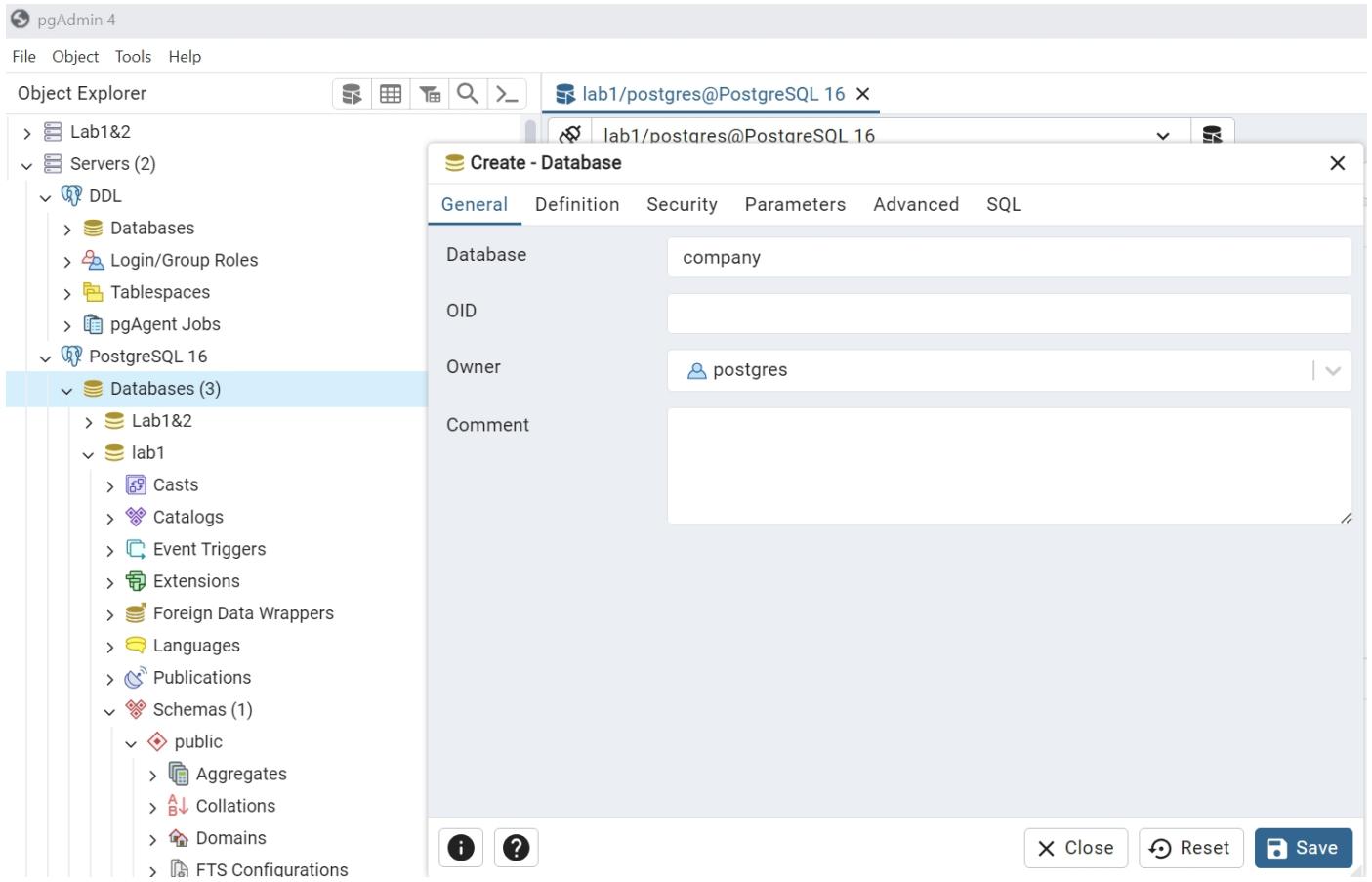
Part I: A Graphic of Table: communication with database

1. Load Driver - 2. Connect to bdd - 3. Make a request.

The screenshot shows a Java development environment with the following interface elements:

- Top Bar:** Includes tabs for "TP TP3" and "Version control", and icons for "Current File", "Run", "Settings", and "Exit".
- Left Sidebar:** Shows a file tree with "Main.java" selected.
- Code Editor:** Displays the content of Main.java. The code is a Java program that connects to a PostgreSQL database and prints department information. A syntax error is highlighted at the line: `Statement statement = connexion.createStatement();`. The error message "relation \"dept\" does not exist" is shown in the terminal below.
- Terminal:** Shows the command run in the terminal: `C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Fi`. The output shows the error: `Bdd Connected` followed by the stack trace of the `PSQLException`.
- Bottom Status Bar:** Shows the project path "TP3 > src > Main", the build status "20:27 LF UTF-8 4 spaces", and other system information.

- In order to solve: Create new Database -> "company".



● Close "NULL"

● Add table

The screenshot shows the pgAdmin interface with the Object Explorer on the left and a query editor on the right.

Object Explorer:

- > Aa FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > 1..3 Sequences
- Tables (5)
 - > bonus
 - > dependents
 - > dept
 - > emp
 - Columns (10)
 - empno
 - ename
 - efirst
 - job
 - mgr
 - hiredate
 - sal
 - comm
 - tel
 - deptno
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
 - > salgrade
- > Trigger Functions
- > Tuner

Query Editor:

```

CREATE TABLE DEPT
    (DEPTNO integer constraint pk_dept primary key,
     DNAME VARCHAR(14),
     LOC VARCHAR(13));

CREATE TABLE EMP
    (EMPNO integer constraint pk_emp primary key,
     ENAME VARCHAR(10),
     EFIRST VARCHAR(10),
     JOB VARCHAR(9),
     MGR integer not null,
     HIREDATE DATE,
     SAL integer constraint ck_sal check (SAL>=0),
     COMM integer,
     TEL char(10),
     DEPTNO integer,
     constraint fk_emp_dept foreign key(DEPTNO) references DEPT (DEPTNO));

CREATE TABLE DEPENDENTS
    (DNO integer,
     DNAME VARCHAR(10),
     DFIRST VARCHAR(10),
     EMPNO integer,
     constraint pk_dependent primary key (DNO, EMPNO),
     constraint fk_dependent_emp foreign key(EMPNO) references EMP (EMPNO));

CREATE TABLE BONUS
    (ENAME VARCHAR(10),
     JOB VARCHAR(9),
     MGR integer,
     HIREDATE DATE,
     SAL integer,
     COMM integer,
     DEPTNO integer,
     constraint fk_bonus_dept foreign key(DEPTNO) references DEPT (DEPTNO));
  
```

Total rows: 0 of 0 Query complete 00:00:00.071 Ln 38, Col 25

● Add Info

The screenshot shows the pgAdmin interface with the Object Explorer on the left and a query editor on the right.

Object Explorer:

- > Aa FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > 1..3 Sequences
- Tables (5)
 - > bonus
 - > dependents
 - > dept
 - > emp
 - Columns (10)
 - empno
 - ename
 - efirst
 - job
 - mgr
 - hiredate
 - sal
 - comm
 - tel
 - deptno
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
 - > salgrade
- > Trigger Functions
- > Tuner

Query Editor:

```

INSERT INTO EMP VALUES
    (7900, 'JAMES', 'ALAN', 'CLERK', 7698,
     TO_DATE('3-12-1981', 'DD-MM-YYYY'), 950, NULL, '0149545564', 30);

INSERT INTO EMP VALUES
    (7902, 'FORD', 'MARIA', 'ANALYST', 7566,
     TO_DATE('3-12-1981', 'DD-MM-YYYY'), 3000, NULL, '0149785243', 20);

INSERT INTO EMP VALUES
    (7934, 'MILLER', 'ALICE', 'CLERK', 7782,
     TO_DATE('23-01-1982', 'DD-MM-YYYY'), 1300, NULL, '0199545243', 10);

INSERT INTO SALGRADE VALUES (1, 700, 1200);
INSERT INTO SALGRADE VALUES (2, 1201, 1400);
INSERT INTO SALGRADE VALUES (3, 1401, 2000);
INSERT INTO SALGRADE VALUES (4, 2001, 3000);
INSERT INTO SALGRADE VALUES (5, 3001, 9999);
  
```

Data Output:

INSERT 0 1

Query returned successfully in 53 msec.

Message Bar:

✓ Query returned successfully in 53 msec. ✘

Total rows: 0 of 0 Query complete 00:00:00.053 Ln 54, Col 45

- Modifier the JAVA code to link to “company”

```
String url = "jdbc:postgresql://localhost/company";
String user = "postgres";
String pass = "123456";
```

- Re-run the code

The screenshot shows a Java development environment with the following details:

- File Explorer:** Shows the project structure with a file named `Main.java`.
- Code Editor:** Displays the `Main.java` file containing JDBC connection code and a `main` method.
- Terminal:** Shows the command used to run the application and the resulting output. The output includes four questions about department names and locations, followed by the message "Process finished with exit code 0".

```
java.sql.*;
class Main {
    public static void main(String[] args) throws SQLException {
        /* Load JDBC Driver. */
        try {
            Class.forName(className: "org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    String url = "jdbc:postgresql://localhost/company";
    String user = "postgres";
    String pass = "123456";
    Connection connexion = null;
    Statement statement = null;
}
trv:
```

```
C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Fi
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 an IDE interface with a code editor and a terminal window. The code editor contains a Java class with a static method 'displayDepartment'. The method uses a database connection to execute a SQL query ('SELECT deptno, dname, loc FROM dept') and prints the results to the console. The terminal window shows the output of the program, which lists four departments: ACCOUNTING (deptno 10), RESEARCH (deptno 20), SALES (deptno 30), and OPERATIONS (deptno 40), each with its name and location.

```
1 usage
45 @
46
47
48
49
50
51
52
53
54
55
56
57
58
```

```
public static void displayDepartment(Connection connexion) throws SQLException {
    Statement statement = connexion.createStatement();
    ResultSet resultat = statement.
        executeQuery( sql: "SELECT deptno, dname, loc FROM dept" );
    while ( resultat.next() ) {
        int deptno = resultat.getInt( columnLabel: "deptno" );
        String dname = resultat.getString( columnLabel: "dname" );
        String location = resultat.getString( columnLabel: "loc" );
        System.out.println("Department " + deptno + " is for "
            + dname + " and located in " + location);
    }
    resultat.close();
}
```

Run Main ×

```
C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.1\lib\idea_rt.jar=5334,C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.1\bin" -Dfile.encoding=UTF-8
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.

- Add the move-function

The screenshot shows an IDE interface with a code editor. The code editor contains a Java class with a static method 'moveDepartment'. This method takes a database connection, an employee number ('Emp'), and a department number ('Deptno') as parameters. It constructs an SQL UPDATE statement to change the department for the specified employee. The code uses a try-with-resources block to handle the preparation of the statement and its execution. A catch block is included to handle any exceptions.

```
1 usage
public static void moveDepartment(Connection connection,int Emp, int Deptno) throws SQLException{
    String command = "UPDATE EMP set DEPTNO = ? WHERE EMPNO = ?";
    try(PreparedStatement updateEmp = connection.prepareStatement(command)){
        updateEmp.setInt( parameterIndex: 1,Deptno);
        updateEmp.setInt( parameterIndex: 2,Emp);
        updateEmp.execute();
        System.out.println("\nEmployee No." + Emp + " is in the department " + Deptno);
    }
    catch (Exception exception){
        exception.printStackTrace();
    }
}
```

- Call the function in the Main and run it

The screenshot shows the IntelliJ IDEA interface with two tabs open: 'Main.java' and 'Main'. The 'Main.java' tab displays Java code for updating an employee's department. The 'Main' tab shows the terminal output of the program execution.

```
>Main.java
1 usage
60     public static void moveDepartment(Connection connection,int Emp, int Deptno) throws SQLException{
61         String command = "UPDATE EMP set DEPTNO = ? WHERE EMPNO = ?";
62         try(PreparedStatement updateEmp = connection.prepareStatement(command)){
63             updateEmp.setInt( parameterIndex: 1,Deptno);
64             updateEmp.setInt( parameterIndex: 2,Emp);
65             updateEmp.execute();
66             System.out.println("\nEmployee No." + Emp + " is in the department " + Deptno);
67         }
68         catch (Exception exception){
69             exception.printStackTrace();
70         }
71     }
72 }
73

>Main
C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar" -Dfile.encoding=UTF-8
Bdd Connected

Employee No.7369 is in the department 30

Process finished with exit code 0
```

Exercice 3: Generic display of Tables:

The screenshot shows the IntelliJ IDEA interface with the code editor and run terminal.

Code Editor (Main.java):

```
1 usage
public static void displayTable(String tableName) throws SQLException {
    String url = "jdbc:postgresql://localhost/company";
    String user = "postgres";
    String pass = "123456";
    try (Connection connection = DriverManager.getConnection(url, user, pass)) {
        try (Statement statement = connection.createStatement()) {
            try (ResultSet resultSet = statement.executeQuery(sql: "SELECT * FROM " + tableName)) {
                ResultSetMetaData metaData = resultSet.getMetaData();
                int columnCount = metaData.getColumnCount();
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(metaData.getColumnName(i) + "\t| ");
                }
                System.out.println();
                while (resultSet.next()) {
                    for (int i = 1; i <= columnCount; i++) {
                        System.out.print(resultSet.getString(i) + "\t| ");
                    }
                    System.out.println();
                }
            }
            resultSet.close();
            statement.close();
        }
    }
}
```

Run Terminal Output:

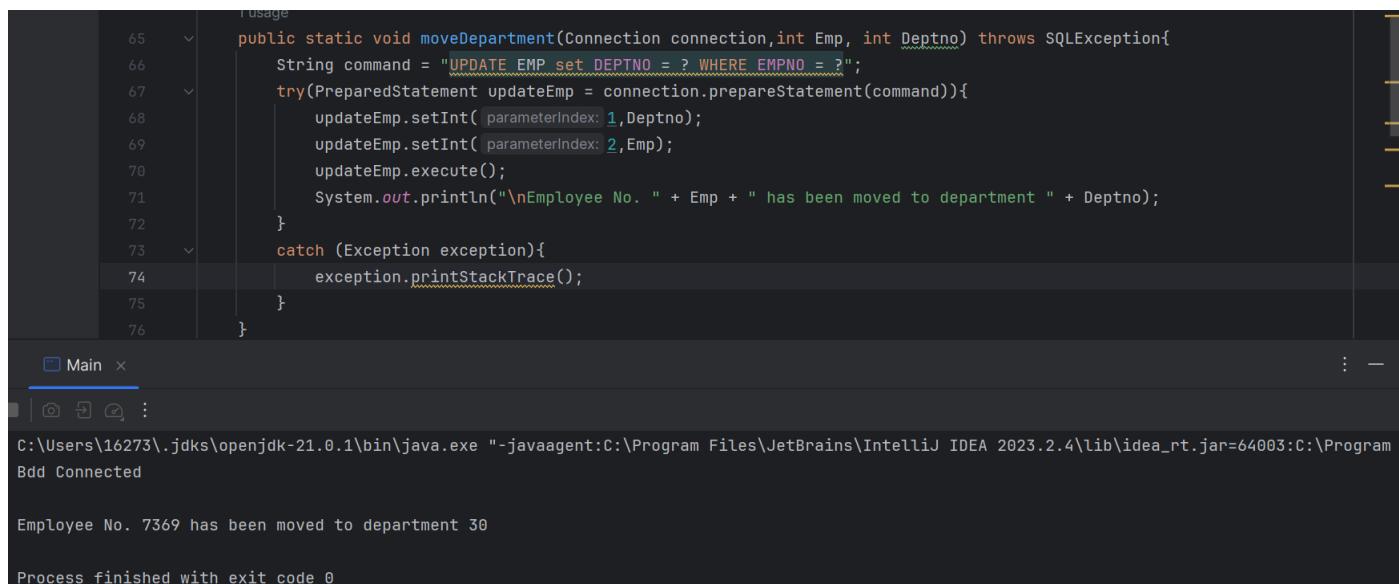
```
C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar" -Dfile.encoding=UTF-8 Main
Bdd Connected
+-----+
| empno | ename  | efirst   | job    | mgr     | hiredate | sal     | comm    | tel     | deptno |
+-----+
| 7521  | WARD   | PETER    | SALESMAN | 7698   | 1981-02-22 | 1250   | 1200   | 0149545247 | 30    |
| 7566  | JONES  | JOHN     | MANAGER  | 7839   | 1981-04-02 | 2975   | 1500   | 0149545456 | 20    |
| 7654  | MARTIN | JOE      | SALESMAN | 7698   | 1981-09-28 | 1250   | 1200   | 0149545784 | 30    |
| 7698  | BLAKE  | BOB      | MANAGER  | 7839   | 1981-05-01 | 2850   | 1500   | 0149545254 | 30    |
| 7782  | CLARK  | JOHN     | MANAGER  | 7839   | 1981-06-09 | 2450   | 1500   | 0149545245 | 10    |
| 7788  | SCOTT  | GUY      | ANALYST | 7566   | 1982-12-09 | 3000   | 1500   | 0149545249 | 20    |
| 7839  | KING   | GUY      | PRESIDENT | null   | 1981-11-17 | 5000   | 1500   | 0149545241 | 10    |
| 7844  | TURNER | PETER    | SALESMAN | 7698   | 1981-09-08 | 1500   | 1200   | 0149548243 | 30    |
| 7876  | ADAMS  | JOSEPH   | CLERK   | 7788   | 1983-01-12 | 1100   | 1200   | 0149565243 | 20    |
| 7900  | JAMES  | ALAN     | CLERK   | 7698   | 1981-12-03 | 950    | 800    | 0149545564 | 30    |
| 7902  | FORD   | MARIA    | ANALYST | 7566   | 1981-12-03 | 3000   | 1500   | 0149785243 | 20    |
| 7934  | MILLER | ALICE    | CLERK   | 7782   | 1982-01-23 | 1300   | 1200   | 0199545243 | 10    |
| 7499  | ALLEN  | BOB      | SALESMAN | 7698   | 1981-02-20 | 2516   | 1500   | 0149547243 | 30    |
| 7369  | SMITH  | JEAN     | CLERK   | 7902   | 1980-12-17 | 7000   | 1500   | 0149545243 | 30    |
+-----+
Process finished with exit code 0
```

Exercice 4: Security:

- What is the Security flow to the method above?

- 1) `Statement` objects may lead to SQL injection. If user input is spliced directly into an SQL statement, can modify the behaviour of the SQL statement by constructing special input, for example, to retrieve, modify, or delete data,
- What are others Cons from this basic native method?
 - 1) SQL statements are usually hard coded in the code: making modification difficult, and the readability and maintainability of the code are poor.
 - 2) Developers must ensure that all database-related resources are properly closed after use, otherwise problems such as memory leaks may occur.
 - 3) SQL injection and potential others wrong arguments passed in the query.

Exercice 5: Modify moveDepartement() to use PreparedStatement



```

65     public static void moveDepartment(Connection connection,int Emp, int Deptno) throws SQLException{
66         String command = "UPDATE EMP set DEPTNO = ? WHERE EMPNO = ?";
67         try(PreparedStatement updateEmp = connection.prepareStatement(command)){
68             updateEmp.setInt( parameterIndex: 1,Deptno);
69             updateEmp.setInt( parameterIndex: 2,Emp);
70             updateEmp.execute();
71             System.out.println("\nEmployee No. " + Emp + " has been moved to department " + Deptno);
72         }
73         catch (Exception exception){
74             exception.printStackTrace();
75         }
76     }
  
```

Main x

C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=64003:C:\Program
Bdd Connected

Employee No. 7369 has been moved to department 30

Process finished with exit code 0

Exercice 6: Try with displayTable

The screenshot shows the IntelliJ IDEA interface with the Main.java file open in the editor. The code implements a `displayTable` method that uses JDBC to print the contents of a database table to the console. The code includes logic for preparing statements and handling result sets. Below the editor is the Run tool window, which displays a stack trace for a `PSQLException` with the error message "ERROR: syntax error at or near \"\$1\"". The stack trace lists several method calls from the PostgreSQL JDBC driver, ending at the `Main.main` method.

```
public static void displayTable(String tableName) throws SQLException {
    String url = "jdbc:postgresql://localhost/company";
    String user = "postgres";
    String pass = "123456";

    String query = "SELECT * FROM ?"; //PreparedStatement

    try (Connection connection = DriverManager.getConnection(url, user, pass)) {
        /* try (Statement statement = connection.createStatement()) {
            try (ResultSet resultSet = statement.executeQuery("SELECT * FROM " + tableName)) {
                ResultSetMetaData metaData = resultSet.getMetaData();
                int columnCount = metaData.getColumnCount();*/
        /* -----PreparedStatement----- */
        try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {
            preparedStatement.setString(1, tableName);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                ResultSetMetaData metaData = resultSet.getMetaData();
                int columnCount = metaData.getColumnCount();
                /* -----PreparedStatement----- */
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(metaData.getColumnName(i) + "\t| ");
                }
                System.out.println();

                while (resultSet.next()) {
                    for (int i = 1; i <= columnCount; i++) {
                        System.out.print(resultSet.getString(i) + "\t| ");
                    }
                    System.out.println();
                }
                //resultSet.close();
                //statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Run Main

```
C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=Bdd Connected
org.postgresql.util.PSQLException: ERROR: syntax error at or near "$1"
  Position: 15
  at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2725)
  at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2412)
  at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:371)
  at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:502)
  at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:419)
  at org.postgresql.jdbc.PgPreparedStatement.executeWithFlags(PgPreparedStatement.java:194)
  at org.postgresql.jdbc.PgPreparedStatement.executeQuery(PgPreparedStatement.java:137)
  at Main.displayTable(Main.java:94)
  at Main.main(Main.java:29)

Process finished with exit code 0
```

JDBC does not allow table names or column names to be passed as parameters to PreparedStatement, preventing SQL injection and ensuring that only values can be dynamically inserted into SQL statements.

Exercice 7: With J2EE

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

- Project View:** Shows the project structure for "Lab3-1-J2EE". It includes a pom.xml file, Java source code (HelloApplication.java, HelloResource.java, DatabaseServlet.java), configuration files (web.xml), and resources (META-INF beans.xml).
- Code Editor:** Displays the Java code for the DatabaseServlet class. The code connects to a PostgreSQL database and prints the results to the response.
- Maven Tool Window:** Shows the Maven lifecycle with various goals like clean, validate, test, package, verify, install, site, and deploy.
- Run Tab:** Shows the run configuration for "Lab3-1-J2EE [install]".
- Output Tab:** Shows Maven validation issues detected during the build process.

```
public class DatabaseServlet extends HttpServlet {
    private static final String url = "jdbc:postgresql://localhost/company";
    private static final String user = "postgres";
    private static final String pass = "123456";

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        try {
            Class.forName("org.postgresql.Driver");
            Connection conn = DriverManager.getConnection(url, user, pass);
            Statement statement = conn.createStatement();
            ResultSet rs = statement.executeQuery("SELECT * FROM emp");
            while (rs.next()) {
                out.println("<p>" + rs.getString("ename") + "</p>");
            }
        } catch (Exception e) {
            out.println("<p>Error: " + e.getMessage() + "</p>");
            e.printStackTrace();
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

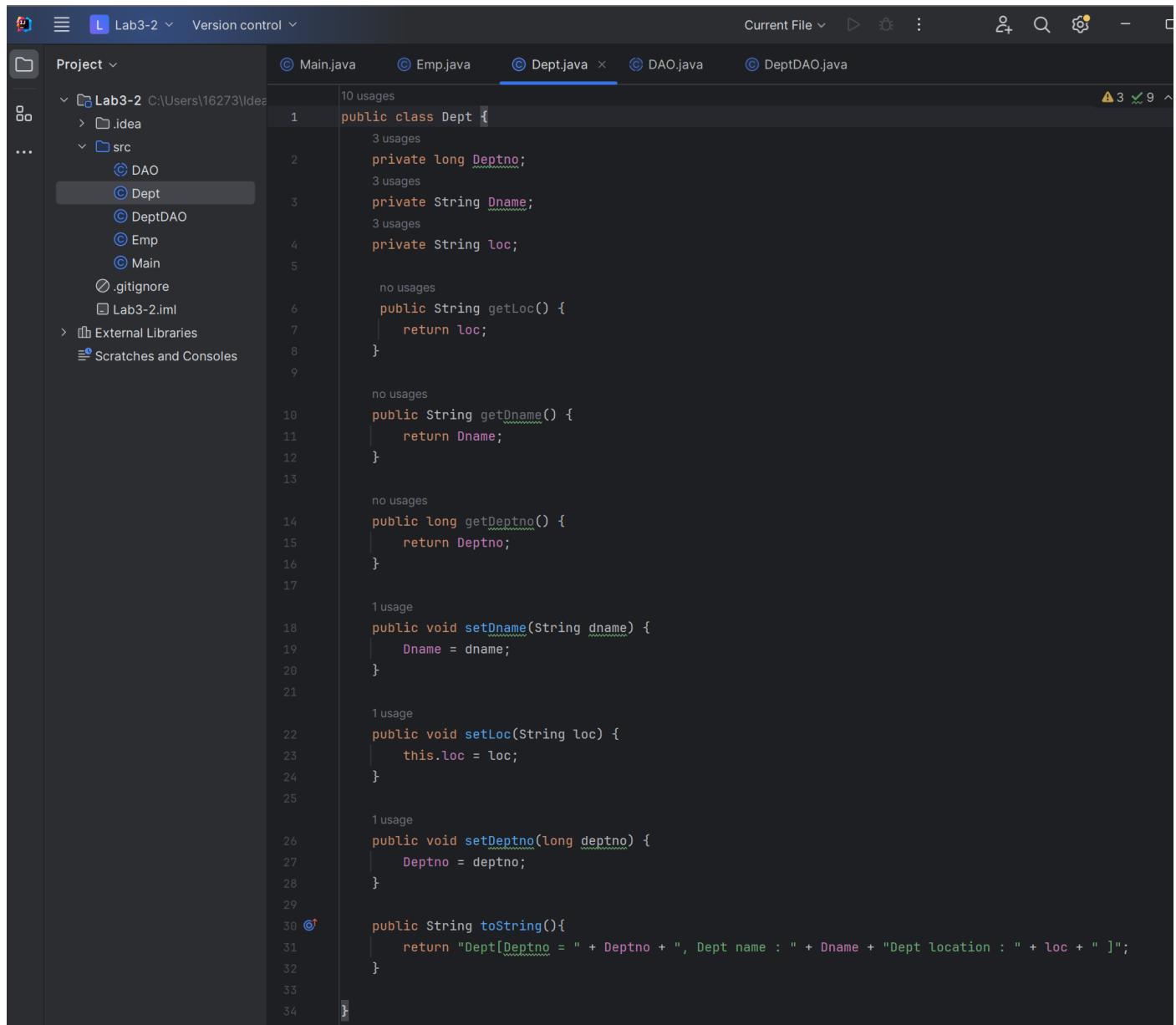
```
[WARNING] Plugin validation issues were detected in 3 plugin(s)
[WARNING]
[WARNING] * org.apache.maven.plugins:maven-compiler-plugin:3.10.1
[WARNING] * org.apache.maven.plugins:maven-resources-plugin:3.3.0
[WARNING] * org.apache.maven.plugins:maven-war-plugin:3.3.2
[WARNING]
[WARNING] For more or less details, use 'maven.plugin.validation' property with one of the values (case insensitive): [BRIEF, DEFAULT, VERBOSE]
[WARNING]
```

Part II : DAO.

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

- Boilerplate Code: JDBC requires verbose code for establishing connections, error handling, and resource cleanup.
- Error-Prone: Manually managing resources like connections, statements, and result sets increases the risk of resource leaks.
- Lack of Abstraction: JDBC operates at a low level, resulting in more effort for even simple queries and data manipulations.
- Maintenance Difficulty: Changes in the database schema often require updates in multiple parts of JDBC code, making maintenance challenging.

Exercice 9 : Write the Bean for Department



```
1 public class Dept {  
2     private long Deptno;  
3     private String Dname;  
4     private String loc;  
5  
6     public String getLoc() {  
7         return loc;  
8     }  
9  
10    public String getDname() {  
11        return Dname;  
12    }  
13  
14    public long getDeptno() {  
15        return Deptno;  
16    }  
17  
18    public void setDname(String dname) {  
19        Dname = dname;  
20    }  
21  
22    public void setLoc(String loc) {  
23        this.loc = loc;  
24    }  
25  
26    public void setDeptno(long deptno) {  
27        Deptno = deptno;  
28    }  
29  
30    public String toString(){  
31        return "Dept[Deptno = " + Deptno + ", Dept name : " + Dname + "Dept location : " + loc + "]";  
32    }  
33  
34}
```

Exercice 10 : Write Implementation for method find for department DAO

```
>Main.java    Emp.java    Dept.java    DAO.java    DeptDAO.java ×    :
1 import java.sql.Connection;
2 import java.sql.PreparedStatement;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5
6 1 usage
7 public class DeptDAO extends DAO<Dept> {
8 1 usage
9     public Dept find(int id) {
10         Dept dept = new Dept();
11         try {
12             PreparedStatement finddept = connect.prepareStatement(sql: "SELECT * FROM DEPT where DEPTNO = ?");
13             finddept.setInt(parameterIndex: 1, id);
14             ResultSet result = finddept.executeQuery();
15             while (result.next()){
16                 dept.setDeptno(result.getInt(columnLabel: "Deptno"));
17                 dept.setDname(result.getString(columnLabel: "Dname"));
18                 dept.setLoc(result.getString(columnLabel: "Loc"));
19             }
20         } catch (SQLException e) {
21             throw new RuntimeException(e);
22         }
23         return dept;
24     }
25
26 1 usage
27     @Override
28     public boolean create(Dept object) {
29         return false;
30     }
31
32     no usages
33     @Override
34     public boolean update(Dept object) {
35         return false;
36     }
37
38     no usages
39     @Override
40     public boolean delete(Dept object) {
41         return false;
42     }
43 }
```

Configure a version control for the project

Dept.java

10 usages

```
1 public class Dept {  
2     private long Deptno;  
3     private String Dname;  
4     private String loc;  
5  
6     no usages  
7     public String getLoc() {  
8         return loc;  
9     }  
10    no usages  
11    public String getDname() {  
12        return Dname;  
13    }  
14    no usages  
15    public long getDeptno() {  
16        return Deptno;  
17    }  
18    1 usage  
19    public void setDname(String dname) {  
20        Dname = dname;  
21    }  
22    1 usage  
23    public void setLoc(String loc) {  
24        this.loc = loc;  
25    }  
26    1 usage  
27    public void setDeptno(long deptno) {  
28        Deptno = deptno;  
29    }  
30    public String toString(){  
31        return "Dept[Deptno = " + Deptno + ", Dept name : " + Dname + "Dept location : " + loc + " ]";  
32    }  
33}  
34}
```

3 9

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

- Project View:** Shows the project structure under "Lab3-2". The "src" folder contains DAO, Dept, DeptDAO, Emp, and Main classes, along with .gitignore and Lab3-2.iml files.
- Main.java Content:** The code connects to a PostgreSQL database using JDBC. It prints "Bdd Connected" and then retrieves a department with ID 20, printing its details (Deptno = 20, Dept name : RESEARCH, Dept location : DALLAS).
- Run Tab:** Shows the output of the run command: "C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=50945:C:\Program". The output also includes the printed department information and "Process finished with exit code 0".

```
String url = "jdbc:postgresql://localhost/company";
String user = "postgres";
String pass = "123456";
Connection connexion = null;
Statement statement = null;

try {
    connexion = DriverManager.getConnection(url, user, pass);
    /* Requests to bdd will be here */
    System.out.println("Bdd Connected");

    statement = connexion.createStatement();
    DAO<Dept> departmentDao = new DeptDAO(connexion);
    Dept dept20 = departmentDao.find( id: 20 );
    System.out.println(dept20);

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

C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=50945:C:\Program
Bdd Connected
Dept[Deptno = 20, Dept name : RESEARCHDept location : DALLAS]
Process finished with exit code 0

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

```
© Main.java   © Emp.java   © EmpDAO.java   ×   © Dept.java   © DAO.java   © DeptDAO.java
1 import java.sql.Connection;
2 import java.sql.PreparedStatement;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5
6 no usages
7 public abstract class EmpDAO extends DAO<Emp> {
8     no usages
9     public EmpDAO(Connection connexion) {
10         super(connexion);
11
12     }
13
14     2 usages
15     @Override
16     public Emp find(int id) {
17         Emp emp = null;
18         try {
19             PreparedStatement findEmp = connect.prepareStatement(
20                 sql: "SELECT * FROM EMP WHERE EMPNO = ?");
21             findEmp.setInt( parameterIndex: 1, id);
22             ResultSet result = findEmp.executeQuery();
23             if (result.next()) {
24                 emp = new Emp();
25                 emp.setEmpNo((long) result.getInt( columnLabel: "empNo"));
26                 emp.setEname(result.getString( columnLabel: "ename"));
27
28                 int mgrId = result.getInt( columnLabel: "mgr");
29                 if (mgrId != 0) {
30                     emp.setMgr(find(mgrId)); // 递归调用, 设置经理
31                 }
32             }
33             return emp;
34         }
35     }
36 }
```

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

- Project View:** Shows the project structure with packages like DAO, Dept, DeptDAO, Emp, EmpDAO, and Main. The Main.java file is currently selected.
- Main.java Content:** The code implements a Main class with methods for creating, updating, and deleting employees using DAO interfaces. It also handles SQLExceptions and prints stack traces.
- Run Tab:** Displays the command used to run the application: "C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=51175:C:\Program".
- Output Log:** Shows the application's output:

```
Bdd Connected
null
Process finished with exit code 0
```

Exercice 12 : Create DAOFactory and use it in previous example

```
© Main.java      © DAOFactory.java ×   © Emp.java      © EmpDAO.java      © Dept.java      © DAO.java
1 import java.sql.Connection;
2
3 no usages
4 public class DAOFactory {
5     5 usages
6         private Connection connection;
7
8             no usages
9             public DAOFactory(Connection connection) {
10                 this.connection = connection;
11             }
12
13             no usages
14             public DeptDAO getDeptDAO() {
15                 return new DeptDAO(connection);
16             }
17             ①↑
18             no usages
19             @Override
20                 public boolean create(Emp object) {
21                     return false;
22             }
23
24             no usages
25             @Override
26                 public boolean update(Emp object) {
27                     return false;
28             }
29
30             no usages
31             @Override
32                 public boolean delete(Emp object) {
33                     return false;
34             }
35
36             no usages
37             public DeptDAO getdependentsDAO(){
38                 return new DeptDAO(connection);
39             }
40
41 }
```

```

72             */
73             /*-----Exercice 12-----*/
74             DAOFactory daoFactory = new DAOFactory(connexion);
75             DAO<Dept> departmentDao = daoFactory.getDeptDAO();
76             Dept dept20 = departmentDao.find( id: 20);
77             System.out.println(dept20);
78
79             DAO<Emp> employeeDao = daoFactory.getEmpDAO();
80             Emp emp = employeeDao.find( id: 20);
81             System.out.println(emp);
82             }catch (SQLException e) {
83                 e.printStackTrace();
84                 System.out.println("Connection failure.");
85             }
86         }
87     }
88

```

Run Main ×

C:\Users\16273\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.4\lib\idea_rt.jar=51433:C:\Program
Bdd Connected
Dept[Deptno = 20, Dept name : RESEARCHDept location : DALLAS]
null

Process finished with exit code 0

Part III : Spring Boot & JPA.

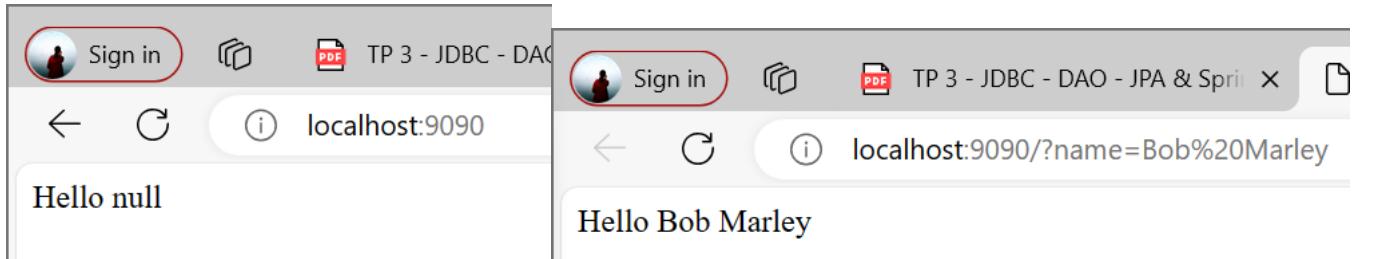
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "TP3_testjpa". It includes a "src" directory containing "main" (with "java", "com.isep", "testjpa", "controller", "SimpleController", "TestjpaApplication", and "Main" files), "resources", "test", "target", ".gitignore", "application.yml", and "pom.xml".
- Code Editor:** The active file is "TestjpaApplication.java". The code defines a main method that runs the "TestjpaApplication" class using SpringApplication.run().
- Run Tab:** The "TestjpaApplication" configuration is selected. The console output shows the application starting up, connecting to the database, and successfully finding a department record.
- Console Output:**

```

2024-04-01 03:47:01.390 INFO 15584 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-04-01 03:47:01.390 INFO 15584 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/9.0.31]
2024-04-01 03:47:01.524 INFO 15584 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-04-01 03:47:01.525 INFO 15584 --- [           main] o.s.web.context.ContextLoader           : Root WebApplicationContext: initialization completed in 1 ms
2024-04-01 03:47:01.825 INFO 15584 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2024-04-01 03:47:02.294 INFO 15584 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 (http) with context path ''
2024-04-01 03:47:02.298 INFO 15584 --- [           main] com.isep.testjpa.TestjpaApplication      : Started TestjpaApplication in 4.324 seconds (JVM running for 4.324)
2024-04-01 03:47:15.616 INFO 15584 --- [nio-9090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-04-01 03:47:15.618 INFO 15584 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-04-01 03:47:15.646 INFO 15584 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 28 ms

```
- Status Bar:** Shows file statistics: 9:2 CRLF, UTF-8, 4 spaces.



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

Idea Project Structure:

```

TP3_testjpa [testjpa] C:\Users\16273\IdeaProjects\TP3_testjpa
  > .idea
  > src
    > main
      > java
        > com.isep
          > testjpa
            > controller
              > SimpleController
              > TestjpaApplication
              > Main
            > resources
          > test
        > target
        > .gitignore
        > application.yml
        > pom.xml
      > External Libraries
    > Scratches and Consoles
  > pom.xml (testjpa) x
  > TestjpaApplication.java
  > SimpleController.java
  > Main.java
  > pom.xml
  > External Libraries
  > Scratches and Consoles

```

application.yml content:

```

server.port: 9090
spring.datasource.url: jdbc:postgresql://localhost/company
spring.datasource.username: postgres
spring.datasource.password: 123456
spring.datasource.driver.class: org.postgresql.Driver

```

pom.xml content:

```

<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>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
</dependencies>

```

Run TestjpaApplication output:

```

2024-04-01 03:59:06.700 INFO 13084 --- [           main] com.isep.testjpa.TestjpaApplication   : Starting TestjpaApplication on Petite-win w
2024-04-01 03:59:06.702 INFO 13084 --- [           main] com.isep.testjpa.TestjpaApplication   : No active profile set, falling back to defa
2024-04-01 03:59:07.253 INFO 13084 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories
2024-04-01 03:59:07.266 INFO 13084 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in
2024-04-01 03:59:07.657 INFO 13084 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9090 (http
2024-04-01 03:59:07.663 INFO 13084 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-01 03:59:07.663 INFO 13084 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0
2024-04-01 03:59:07.724 INFO 13084 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-01 03:59:07.724 INFO 13084 --- [           main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization...
2024-04-01 03:59:07.804 INFO 13084 --- [           main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-04-01 03:59:08.008 INFO 13084 --- [           main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2024-04-01 03:59:08.054 INFO 13084 --- [           main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [
2024-04-01 03:59:08.104 INFO 13084 --- [           main] o.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.1
2024-04-01 03:59:08.246 INFO 13084 --- [           main] o.hibernate.annotations.common.Version : HHH00000001: Hibernate Commons Annotations

```

Exercise III.3 : Read Data in your Database

The screenshot shows an IDE interface with several windows:

- Code Editor:** Displays a JSON array of employee records from the database. The array contains 44 elements, each representing an employee with fields like empno, name, job, and salary.
- Project Explorer:** Shows the project structure for "TP3_testjpa". It includes a "src" folder containing "main", "resources", and "test" folders. "main" contains "com.isep" and "testjpa" packages, with "SimpleController" and "EmpRepository" classes. "resources" contains "application.yml" and "pom.xml".
- Code Editor (TestjpaApplication.java):** Shows the Java code for the application. It includes imports for Spring framework beans, repository, and annotations. The `SimpleController` class has a `hello` method that returns "Hello " + name and a `getEmployees` method that returns a list of employees from the repository.
- Console:** Displays the application's log output. It shows the application starting up, connecting to a HikariDataSource, and initializing various Spring components like StandardEngine, WebApplicationContext, and EntityManagerFactory.
- Status Bar:** Shows the current time as 22:22, and file formats as CRLF, UTF-8, and 4 spaces.

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

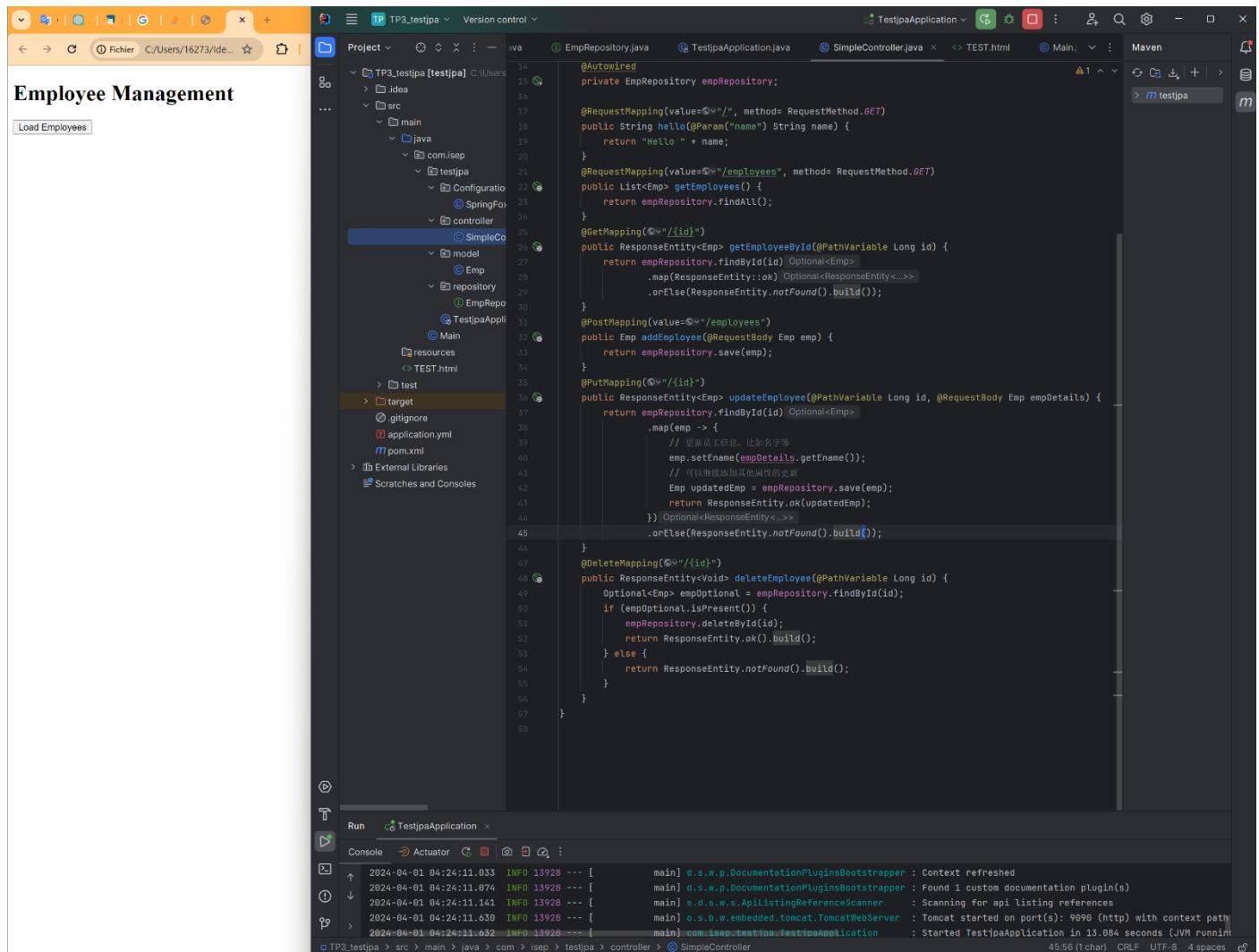
- Code Editor:** The left pane displays the `Emp.java` file, which contains Java code for a `TestipaApplication` class. The code includes methods for setting and getting employee details like empno, ename, efirst, job, mgr, and sal.
- Terminal:** The right pane shows the application's log output from the terminal window titled "Run TestipaApplication". It includes initialization logs and a warning about open-in-view being enabled by default.
- Project Structure:** The bottom-left pane shows the project structure for `TP3.testipa`, including `pom.xml`, `src` (containing `java`, `resources`, and `test`), and `target`.
- Console:** The bottom-right pane shows the Java console output, which is mostly empty except for the application's initialization logs.

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

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

- Code Editor:** The left pane displays the `SpringFoxConfig.java` file, which contains configuration for the `SpringFoxConfig` class. It imports various Springfox annotations and defines a `getDocumentation()` method.
- Terminal:** The right pane shows the application's log output from the terminal window titled "Run TestipaApplication". It includes initialization logs and a warning about documentation plugins.
- Project Structure:** The bottom-left pane shows the project structure for `TP3.testipa`, including `pom.xml`, `src` (containing `java`, `resources`, and `test`), and `target`.
- Console:** The bottom-right pane shows the Java console output, which is mostly empty except for the application's initialization logs.

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



The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "TP3_testjpa". It contains a "src" folder with "main" and "java" subfolders. The "java" folder contains packages like "com.isep.testjpa" and "com.isep.testjpa.controller".
- Code Editor:** The main editor window displays the "SimpleController.java" file. The code implements a REST API for managing employees using JPA annotations like @Autowired, @RequestMapping, and @GetMapping.
- Console:** The bottom panel shows the application's log output. It includes logs from "d.s.w.p.DocumentationPluginsBootstrapper", "s.d.s.w.s.ApiListingReferenceScanner", and "o.s.d.b.w.embedded.tomcat.TomcatWebServer". The log message indicates the application was started on port 9090.

```
2024-04-01 04:24:11.833 INFO 13928 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Context refreshed
2024-04-01 04:24:11.874 INFO 13928 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Found 1 custom documentation plugin(s)
2024-04-01 04:24:11.141 INFO 13928 --- [           main] s.d.s.w.s.ApiListingReferenceScanner : Scanning for api listing references
2024-04-01 04:24:11.630 INFO 13928 --- [           main] o.s.d.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 (http) with context path
2024-04-01 04:24:11.632 INFO 13928 --- [           main] com.isep.testjpa.TestjpaApplication   : Started TestjpaApplication in 13.084 seconds (JVM running
```