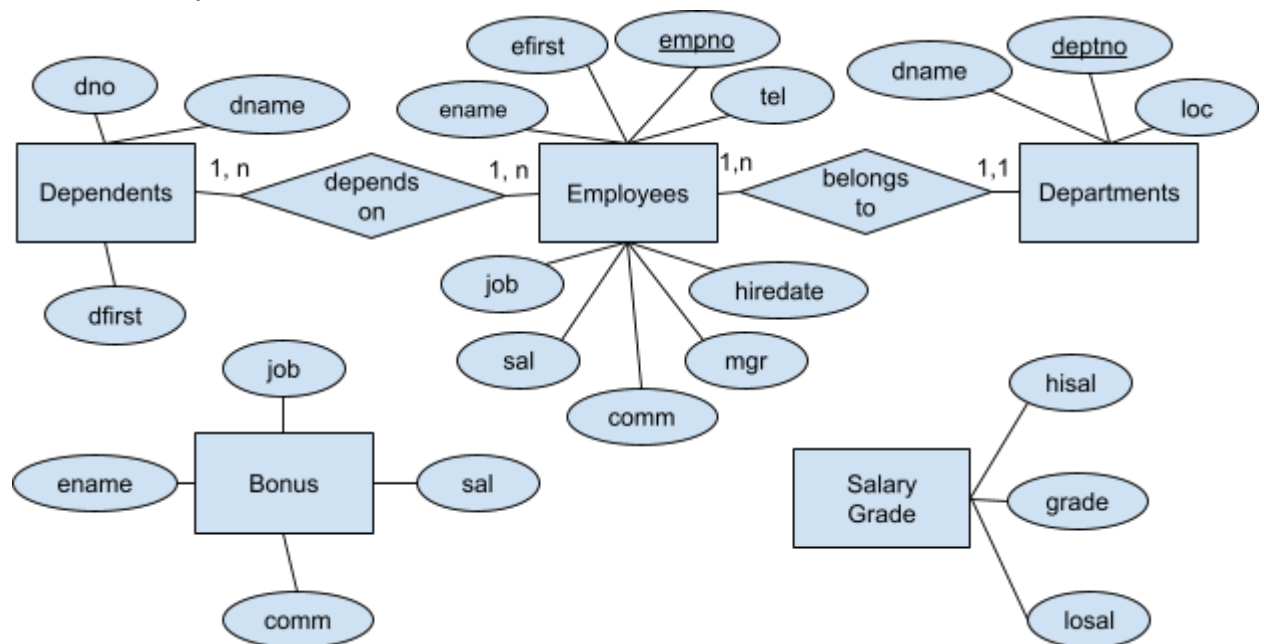


ADV. DB & BIG DATA TP1 REPORT

Exercise 1

1. After creating the database and tables, everything works correctly. Now let's do the exercises
2. Integrity constraints are every primary key (PK) which make every individual unique, also the salary of the employees must be a positive number. Foreign keys (FK) are also a constraint stating that one entity is related to another.

And here's the E/R diagram. We can see all entities and their attributes, along with their relationships:



Relationships:

- 1 Dependant depends on n Employees and 1 Employee has n Dependants. (Many to many). We can note this since the PK of Dependants is a tuple that permits any combination like (1, 2) and (1,3) but also (1,2) and (2,2) where the first one is dno and the second one is empno.
- 1 Employee belongs to 1 department and 1 Department has n Employees. This is because of the foreign key on employees, which allows an employee to belong to an apartment but you can have the same deptno for two employees.

3.

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

By doing this we modify the table to add the constraint without needing to drop and create it again. Same applied for next constraints.

4.

```
ALTER TABLE EMP  
ADD COLUMN MOBTEL char(10);
```

```
ALTER TABLE EMP  
ADD CONSTRAINT MOBTEL_check CHECK (MOBTEL LIKE '06%');
```

5.

```
ALTER TABLE EMP  
DROP CONSTRAINT fk_emp_dept;
```

```
ALTER TABLE EMP  
ADD CONSTRAINT fk_emp_dept FOREIGN KEY (DEPTNO)  
REFERENCES DEPT (DEPTNO) ON DELETE CASCADE;
```

6. Error fixed: Row with first name "KING" has mgr collumn null, which breaks the constraint NOT NULL. To fix it I will put a random value to make it meet the constraint.

```
INSERT INTO EMP VALUES  
(7839, 'KING', 'GUY', 'PRESIDENT', 7839,  
TO_DATE('17-11-1981', 'DD-MM-YYYY'), 5000, NULL, '0149545241',10);
```

7.

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

8.

```
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'JOHNSON',  
'MARY', 7369);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'BROWN',  
'JAMES', 7876);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'WILLIAMS',  
'PATRICIA', 7900);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'JONES',  
'MICHAEL', 7934);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'JOHNSON',  
'ALICE', 7369);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'WILSON',  
'ROBERT', 7900);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'DAVIS',  
'EMILY', 7698);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'BROWN',  
'MICHAEL', 7876);  
INSERT INTO DEPENDENTS VALUES (nextval('DNO_seq'), 'MILLER',  
'SARAH', 7788);
```

9.

Following the documentation given at the question, serial is a type which is added instead of “integer” and you can do auto-increment with it, but it has some weird behaviours that makes some things a bit cumbersome, so maybe is not the best option and it's already out of the standard.

Identity is simpler and is in the standard. It's added at creating table too but instead of a type it is an option added as PRIMARY KEY or NOT NULL, etc like: "GENERATED AS IDENTITY".

Sequence is the one used in exercise 7. It's a separated object which increments its value and it can be used when generating data, in the column needed with "nextval(seq_name)".

In my opinion, between identity and sequence, it is better to use identity since it's the standard but also it's simpler, when generating data identity generates the value but with sequence you need to put nextval. Maybe if you need more control with what value you want to be there, sequence could be better since you can use setval to change the sequence value between more options.

Exercise 2

1.

```
SELECT
    t.table_name,
    array_agg(c.column_name::text) AS COLUMNS
FROM
    information_schema.tables t
INNER JOIN information_schema.columns c ON
    t.table_name = c.table_name
WHERE
    t.table_schema = 'public'
    AND t.table_type= 'BASE TABLE'
    AND c.table_schema = 'public'
GROUP BY t.table_name;
```

	table_name name	columns text[]
1	bonus	{sal,job,comm,ename}
2	dependents	{dfirst,dno,dname,empno}
3	dept	{deptno,loc,dname}
4	emp	{ename,efirst,job,mgr,hiredate,sal,comm,tel,deptno,mobtel,empno}
5	project	{pname,budget,projno,startdate}
6	project_emp	{projno,empno}
7	salgrade	{hisal,losal,grade}

2.

```
SELECT * FROM EMP
WHERE comm > sal;
```

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

3.

```
SELECT * FROM EMP
WHERE (comm + sal) BETWEEN 1200 AND 2400;
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
2	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247	30	[null]
3	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243	30	[null]

4.

```
SELECT * FROM EMP
WHERE job = 'CLERK' OR job = 'ANALYST';
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
2	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]
4	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
5	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
6	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]

5.

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

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

6.

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

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

7.

```
SELECT * FROM EMP
WHERE (job = 'MANAGER' OR job = 'CLERK')
AND deptno = 10
AND sal > 1500;
```

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

8.

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

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
2	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
3	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
4	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]
5	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
6	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]
7	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
8	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
9	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]
10	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]

GITHUB REPO FOR ALL 3 TPs: https://github.com/AbrahanGautiel/AdvDBandBD_Lab1_2_3.git
IT HAS ALL SQL AND JAVA FILES

9.

SELECT * FROM EMP
ORDER BY hiredate ASC;

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247	30	[null]
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
5	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
6	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]
7	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243	30	[null]
8	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784	30	[null]
9	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]
10	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
11	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
12	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]
13	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
14	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]

10.

SELECT * FROM EMP
ORDER BY job, sal DESC;

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
2	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
3	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]
4	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]
5	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
6	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
7	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
8	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
9	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]
10	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]
11	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
12	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243	30	[null]
13	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784	30	[null]
14	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247	30	[null]

11.

```
SELECT * FROM DEPT
WHERE deptno NOT IN (
    SELECT deptno FROM EMP
);
```

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

12.

```
SELECT e1.empno, e1.ename, e1.efirst, e1.job, e1.mgr, e2.ename
AS mgr_name, e1.hiredate, e1.sal, e1.comm, e1.tel, e1.deptno, e1.mobtel
FROM EMP e1
LEFT JOIN EMP e2 ON e1.mgr = e2.empno;
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	mgr_name character varying (10)	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7369	SMITH	JOHN	CLERK	7902	FORD	1980-12-17	800	[null]	0149545243	20	[null]
2	7499	ALLEN	BOB	SALESMAN	7698	BLAKE	1981-02-20	1600	300	0149547243	30	[null]
3	7521	WARD	PETER	SALESMAN	7698	BLAKE	1981-02-22	1250	500	0149545247	30	[null]
4	7566	JONES	JOHN	MANAGER	7839	KING	1981-04-02	2975	[null]	0149545456	20	[null]
5	7654	MARTIN	JOE	SALESMAN	7698	BLAKE	1981-09-28	1250	1400	0149545784	30	[null]
6	7698	BLAKE	BOB	MANAGER	7839	KING	1981-05-01	2850	[null]	0149545254	30	[null]
7	7782	CLARK	JOHN	MANAGER	7839	KING	1981-06-09	2450	[null]	0149545245	10	[null]
8	7788	SCOTT	GUY	ANALYST	7566	JONES	1982-12-09	3000	[null]	0149545249	20	[null]
9	7844	TURNER	PETER	SALESMAN	7698	BLAKE	1981-09-08	1500	0	0149548243	30	[null]
10	7876	ADAMS	JOSEPH	CLERK	7788	SCOTT	1983-01-12	1100	[null]	0149565243	20	[null]
11	7900	JAMES	ALAN	CLERK	7698	BLAKE	1981-12-03	950	[null]	0149545564	30	[null]
12	7902	FORD	MARIA	ANALYST	7566	JONES	1981-12-03	3000	[null]	0149785243	20	[null]
13	7934	MILLER	ALICE	CLERK	7782	CLARK	1982-01-23	1300	[null]	0199545243	10	[null]
14	7839	KING	GUY	PRESIDENT	7839	KING	1981-11-17	5000	[null]	0149545241	10	[null]

13. Since JONES comm is null we can ignore it.

```
SELECT * FROM EMP
WHERE (sal +
    CASE WHEN comm IS NULL THEN 0 ELSE comm END) >
(SELECT sal FROM EMP WHERE ename = 'JONES');
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
2	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
3	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]

14.

```
SELECT empno, ename, efirst, job, mgr, hiredate,
(sal, CASE WHEN comm IS NULL THEN 0 ELSE comm END) AS earning,
tel, deptno, mobtel FROM EMP
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	earning record	tel character (10)	deptno integer
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800,0	0149545243	20
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600,300	0149547243	30
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250,500	0149545247	30
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975,0	0149545456	20
5	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250,1400	0149545784	30
6	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850,0	0149545254	30
7	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450,0	0149545245	10
8	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000,0	0149545249	20
9	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500,0	0149548243	30
10	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100,0	0149565243	20
11	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950,0	0149545564	30
12	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000,0	0149785243	20
13	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300,0	0199545243	10
14	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000,0	0149545241	10

15.

```
SELECT deptno FROM DEPT
WHERE deptno IN (SELECT deptno FROM EMP);
```

	deptno [PK] integer
1	10
2	20
3	30

16.

```
SELECT e.empno, e.ename, e.efirst, e.job, e.mgr, e.hiredate, e.sal,
       e.comm, e.tel, e.deptno, e.mobtel
FROM EMP e JOIN DEPT d on e.deptno = d.deptno
WHERE d.loc = 'CHICAGO'
AND e.job = (
    SELECT job FROM EMP
    WHERE ename = 'JONES'
);
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]

17.

```
SELECT e1.empno, e1.ename, e1.efirst, e1.job, e1.mgr, e1.hiredate,
       e1.sal, e1.comm, e1.tel, e1.deptno, e1.mobtel
FROM EMP e1 JOIN EMP e2 ON e1.mgr = e2.empno
WHERE e1.deptno != e2.deptno;
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]

18.

```
SELECT * FROM EMP
WHERE deptno IN (
    SELECT deptno FROM EMP
    WHERE job = 'CLERK'
);
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0149545243	20	[null]
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0149547243	30	[null]
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0149545247	30	[null]
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
5	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0149545784	30	[null]
6	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
7	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]
8	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0149545249	20	[null]
9	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0149548243	30	[null]
10	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0149565243	20	[null]
11	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0149545564	30	[null]
12	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0149785243	20	[null]
13	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]
14	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]

19.

```
SELECT * FROM EMP
WHERE deptno = 10 AND job IN (
    SELECT job FROM EMP
    WHERE deptno IN (
        SELECT deptno FROM DEPT
        WHERE dname = 'SALES'
    )
);
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0199545243	10	[null]

20.

```
SELECT * FROM EMP
WHERE job IN (
    SELECT job FROM EMP
    WHERE ename = 'JONES'
)
UNION
SELECT * FROM EMP
WHERE sal > (
    SELECT sal FROM EMP
    WHERE ename = 'FORD'
);
```

	empno integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0149545456	20	[null]
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0149545254	30	[null]
3	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]
4	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0149545245	10	[null]

21.

```
SELECT * FROM EMP
WHERE sal > ALL (
    SELECT sal FROM EMP
    WHERE deptno = 20
)
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	job character varying (9)	mgr integer	hiredate date	sal integer	comm integer	tel character (10)	deptno integer	mobtel character (10)
1	7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000	[null]	0149545241	10	[null]

Exercise 3

1.

```
CREATE TABLE IF NOT EXISTS PROJECT (  
    PROJNO integer constraint pk_project primary key,  
    PNAME varchar(20),  
    STARTDATE DATE,  
    BUDGET integer  
);
```

2.

```
CREATE TABLE IF NOT EXISTS PROJECT_EMP (  
    EMPNO integer,  
    PROJNO integer,  
    constraint pk_project_emp primary key (EMPNO, PROJNO),  
    constraint fk_emp foreign key (EMPNO) references EMP (EMPNO),  
    constraint fk_project foreign key (PROJNO) references PROJECT  
(PROJNO)  
);
```

```
INSERT INTO PROJECT VALUES (1, 'INNOVATION',  
    TO_DATE('15-01-2023', 'DD-MM-YYYY'), 50000);  
INSERT INTO PROJECT VALUES (2, 'RESEARCH', TO_DATE('12-07-2023',  
    'DD-MM-YYYY'), 80000);  
INSERT INTO PROJECT VALUES (3, 'MARKETINGCAMPAIGNS',  
    TO_DATE('04-01-2024', 'DD-MM-YYYY'), 75000);  
INSERT INTO PROJECT VALUES (4, 'FINANTIALAUDITS',  
    TO_DATE('15-03-2024', 'DD-MM-YYYY'), 25000);
```

```
INSERT INTO PROJECT_EMP VALUES (7566, 1);  
INSERT INTO PROJECT_EMP VALUES (7566, 2);  
INSERT INTO PROJECT_EMP VALUES (7566, 3);  
INSERT INTO PROJECT_EMP VALUES (7566, 4);  
INSERT INTO PROJECT_EMP VALUES (7902, 4);  
INSERT INTO PROJECT_EMP VALUES (7499, 1);  
INSERT INTO PROJECT_EMP VALUES (7369, 2);  
INSERT INTO PROJECT_EMP VALUES (7521, 3);  
INSERT INTO PROJECT_EMP VALUES (7934, 4);  
INSERT INTO PROJECT_EMP VALUES (7839, 2);  
INSERT INTO PROJECT_EMP VALUES (7839, 1);  
INSERT INTO PROJECT_EMP VALUES (7788, 1);  
INSERT INTO PROJECT_EMP VALUES (7902, 1);  
INSERT INTO PROJECT_EMP VALUES (7782, 2);
```

```
INSERT INTO PROJECT_EMP VALUES (7844, 2);
INSERT INTO PROJECT_EMP VALUES (7654, 3);
INSERT INTO PROJECT_EMP VALUES (7900, 3);
INSERT INTO PROJECT_EMP VALUES (7698, 4);
INSERT INTO PROJECT_EMP VALUES (7782, 4);
INSERT INTO PROJECT_EMP VALUES (7876, 4);
```

3.

```
SELECT empno FROM PROJECT_EMP
GROUP BY empno
HAVING COUNT(projno) = (SELECT COUNT(*) FROM PROJECT);
```

	empno integer
1	7566

4. The view represents information (empno, ename and deptno) from employees who work at sales department (deptno = 10). WITH CHECK OPTION ensures that any addition to this view should meet the condition of the view, in this case, deptno should be 10.

```
CREATE VIEW sales_staff AS
SELECT empno, ename, deptno
FROM emp
WHERE deptno = 10 WITH CHECK OPTION;
```

5. It is expected that since Williams is at department 30 throws an error which says something about the condition, but it seems it's trying to add those two employees to the EMP table too. However, since there is just 3 data from those 2 employees and we remember at least mgr should'n be null with our configuration, that's why i get a different error, which says mgr is null. (Sorry that the error is in Spanish but it's basically what I'm explaining).

NOTICE: la relación «project» ya existe, omitiendo

NOTICE: la relación «project_emp» ya existe, omitiendo

ERROR: La fila que falla contiene (7584, OSTER, null, null, null, null, null, null, null, 10, null).el valor nulo en la columna «mgr» de la relación «emp» viola la restricción de no nulo

ERROR: el valor nulo en la columna «mgr» de la relación «emp» viola la restricción de no nulo

SQL state: 23502

Detail: La fila que falla contiene (7584, OSTER, null, null, null, null, null, null, null, 10, null).

6.

```
SELECT empno, COUNT(projno) FROM PROJECT_EMP  
GROUP BY empno  
HAVING COUNT(projno) >= 2;
```

	empno integer	count bigint
1	7839	2
2	7902	2
3	7782	2
4	7566	4

7.

```
SELECT empno FROM PROJECT_EMP  
WHERE projno = 1  
INTERSECT  
SELECT empno FROM PROJECT_EMP  
WHERE projno = 2;
```

```
SELECT empno FROM PROJECT_EMP  
WHERE projno = 3  
EXCEPT  
SELECT empno FROM PROJECT_EMP  
WHERE projno = 4;
```

	empno integer
1	7839
2	7566

	empno integer
1	7521
2	7900
3	7654

8.

```
SELECT empno, projno, sal
FROM (
  SELECT empno, projno, sal
  FROM PROJECT_EMP pe1 NATURAL JOIN EMP e1
  WHERE (
    SELECT COUNT(*)
    FROM PROJECT_EMP pe2 NATURAL JOIN EMP e2
    WHERE pe2.projno = pe1.projno AND e2.sal > e1.sal
  ) < 3
) AS TopEmployees
ORDER BY projno, sal DESC;
```

	empno integer	projno integer	sal integer
1	7839	1	5000
2	7902	1	3000
3	7788	1	3000
4	7839	2	5000
5	7566	2	2975
6	7782	2	2450
7	7566	3	2975
8	7521	3	1250
9	7654	3	1250
10	7902	4	3000
11	7566	4	2975
12	7698	4	2850

9.

.1

```
SELECT empno,  
       ROUND(COUNT(projno) * 100/(SELECT COUNT(*)  
       FROM PROJECT), 2) AS percentage FROM PROJECT_EMP  
GROUP BY empno
```

	empno integer	percentage numeric
1	7839	50.00
2	7902	50.00
3	7698	25.00
4	7369	25.00
5	7499	25.00
6	7900	25.00
7	7788	25.00
8	7876	25.00
9	7782	50.00
10	7844	25.00
11	7934	25.00
12	7566	100.00
13	7521	25.00
14	7654	25.00

.2

```
SELECT empno, percentage,
CASE
  WHEN percentage = 0 THEN 'Empty'
  WHEN percentage BETWEEN 10 AND 49 THEN 'Small'
  WHEN percentage BETWEEN 50 AND 79 THEN 'Medium'
  WHEN percentage BETWEEN 80 AND 99 THEN 'Large'
  WHEN percentage = 100 THEN 'Total'
END AS scope_size
FROM (
  SELECT empno,
    ROUND(COUNT(projno) * 100.0 / (SELECT COUNT(*)
      FROM PROJECT), 2) AS percentage
  FROM PROJECT_EMP
  GROUP BY empno
```

	empno integer	percentage numeric	scope_size text
1	7839	50.00	Medium
2	7902	50.00	Medium
3	7698	25.00	Small
4	7369	25.00	Small
5	7499	25.00	Small
6	7900	25.00	Small
7	7788	25.00	Small
8	7876	25.00	Small
9	7782	50.00	Medium
10	7844	25.00	Small
11	7934	25.00	Small
12	7566	100.00	Total
13	7521	25.00	Small
14	7654	25.00	Small

ADV. DB & BIG DATA TP2 REPORT

Exercise 1

1. We rank the employees ordering by hiredate and we partition then by deptno, then we take the 2 first of each.

```
SELECT *  
FROM (  
    SELECT empno, ename, efirst, deptno, hiredate,  
           RANK() OVER (PARTITION BY deptno ORDER BY hiredate DESC)  
    AS rank  
    FROM EMP  
) where rank < 3;
```

- 2.

```
SELECT empno, ename, sal,  
       SUM(sal) OVER (  
           ORDER BY empno  
           ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING  
       ) AS cumulative_sal  
FROM EMP;
```

Exercise 2


1. Table created, now let's begin with the questions.
2. After running the query, we have this runtime:

Total query runtime: 156 msec.

- 3.

3.1

As said at the lab, the msec is not really a good metric to analyze queries. We can use the EXPLAIN option before SELECT so we get a more detailed description of the query.

	QUERY PLAN	
	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=49 widt...	
4	Filter: (manager_id = 7)	

3.2

Now, we can run `SHOW data_directory;` to see where the postgres files are.

	data_directory text
1	C:/Program Files/PostgreSQL/17/data

That directory in my case is where the postgresql.conf is located. I opened it and edited the variable requested:

```
#session_preload_libraries = ''  
shared_preload_libraries = 'pg_stat_statements' # (change requires restart)  
#jit_provider = 'llvmjit' # JIT library to use
```

Now, after running create extension pg_stat_statements and then re-running the query 10 times we get a mean of 0.3

4. (5 and 6) After adding the index we get a mean of 0.05 it seems now the mean is way smaller than without the index, as expected.

7.

	QUERY PLAN text
1	GroupAggregate (cost=0.28..5.40 rows=2 width=10)
2	Group Key: gender
3	-> Index Only Scan using manager_id_gender_index on emp_medium_table (cost=0.28..5.14 rows=49 width=...
4	Index Cond: (manager_id = 7)

It seems the cost is less now, before was 144, now was 0.28

Exercise 3

First create the table:

```
CREATE TABLE IF NOT EXISTS MY_OBJECTS(  
    Object VARCHAR(255);  
    Type VARCHAR(50);  
);
```



Then, insert the views that contain the objects, taking the columns that contain the name and adding the type.

```
INSERT INTO MY_OBJECTS (Object, Type)  
SELECT table_name AS Object, 'Table' AS Type  
FROM information_schema.tables  
WHERE table_schema = 'public'  
  
UNION  
  
SELECT column_name AS Object, 'Column' AS Type  
FROM information_schema.columns  
WHERE table_schema = 'public'  
  
UNION  
  
SELECT conname AS Object, 'Constraint' AS Type  
FROM pg_constraint  
  
WHERE connamespace = (SELECT oid FROM pg_namespace WHERE nsname  
= 'public');
```

And last, we view the table:

```
SELECT * FROM MY_OBJECTS ORDER BY Type
```

And here I show some rows of the table:

	object character varying (25) 	type character varying (15) 
79	mgr	Column
80	min_exec_time	Column
81	temp_blk_write_time	Column
82	pk_project_emp	Constraint
83	pk_emp	Constraint
84	pk_dependent	Constraint
85	ck_sal	Constraint
86	fk_dependent_emp	Constraint
87	mobtel_check	Constraint
88	pk_dept	Constraint
89	fk_project	Constraint
90	pk_project	Constraint
91	fk_emp	Constraint
92	fk_emp_dept	Constraint
93	efirst_ename_tel_unique	Constraint
94	emp_medium_table	Table
95	project	Table
96	sales_staff	Table
97	dept	Table
98	dependents	Table
99	emp	Table

Exercise 4

To access from windows, we need the command psql from cmd like this, and then, type the password:

```
psql -U postgres
```

Then, write:

```
\c TP_1
```

And we are inside the database. We simply run a query to make sure all works:

```
SELECT * FROM emp;
```

empno	ename	efirst	job	mgr	hiredate	sal	comm	tel	deptno	mobtel
7369	SMITH	JOHN	CLERK	7902	1980-12-17	800		0149545243	20	
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	
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	
7839	KING	GUY	PRESIDENT	7839	1981-11-17	5000		0149545241	10	

(14 filas)

Exercise 5

On the first terminal:

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

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

```
TP_1=# UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369;
UPDATE 1
TP_1=*# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
UPDATE 1
TP_1=*# |
```

If then we try to see the update, we see it's updated:

```
SELECT sal FROM EMP WHERE EMPNO = 7369;
```

```
sal
-----
7000
(1 fila)
```

On the second terminal, we won't see it until terminal 1 does the command COMMIT; then, the second one will be able to see it.

We will try to do an update on the second terminal:

```
TP_1=# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
```

As we see, terminal 2 is locked (It doesn't let you type after it tries to update). This happens to avoid inconsistency, since the autocommit is off, terminal 2 isn't able to see the changes of terminal 1, so it could lead to various inconsistencies if we try to do transactions from both, so this database is configured so if a client is doing a transaction, other clients can't do one until the active one is committed or aborted.

The name of this mechanism is lock.

So, we do as said before, in first terminal, we do COMMIT:

```
TP_1=*# COMMIT;  
COMMIT  
TP_1=# |
```

And then terminal 2 unlocks and finishes the update that locked it, and of course also will be able to see all changes terminal 1 did.

```
TP_1=# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;  
UPDATE 1  
TP_1=*# |
```

ADV. DB & BIG DATA TP3 REPORT

Note: The main function will be at the end, the rest of the code is fragmented while explaining it (only things not included are the imports). The full code will be at the github.

Part I: A Graphic Of Table: communication with database

Note: I put connection as a static object to make it accessible from any other function without the need to ask it as an arg.

Exercise 1:

To retrieve the location of each department, we need to change the query to:

```
"SELECT deptno, dname, loc FROM DEPT"
```

and then add the line:

```
String loc = result.getString( "loc" );
```

and of course include `loc` into the print function. The function looks like this:

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

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

        System.out.println("Department " + deptno + " is
for "
            + dname + " and located in " + loc);
    }
    result.close();
}
```

And the result will look like this:

```
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
```

Exercise 2:

For this action we do an update with the empno and newDeptno:

```
"UPDATE EMP SET deptno = " + newDeptno + " WHERE empno = " + empno
```

Since it will be an update, now I'll use the method executeUpdate() and now the result will be an int. This is how the function looks like:

```
public static void moveDepartment(int empno, int newDeptno) throws SQLException {
    Statement statement = connexion.createStatement();
    int result = statement.executeUpdate( "UPDATE EMP SET deptno = " + newDeptno + " WHERE empno = " + empno );
    if (result > 0){
        System.out.println("Employee " + empno + " moved to department " + newDeptno);
    } else {
        System.out.println("Something occurred when trying to update");
    }
}
```

It will display this if the update was success:

```
Select an employee to change their dept (e.g 7369)
7369
Select which department you want to move them (10, 20, 30, 40)
20
Employee 7369 moved to department 20
```


Exercise 3:

We will need a loop to display all column names and another to display the rows:

```
public static void displayTable(String tableName) throws
SQLException {

    Statement statement = connexion.createStatement();
    ResultSet result = statement.
        executeQuery( "SELECT * FROM " + tableName );

    ResultSetMetaData rsmd = result.getMetaData();
    int columnsNumber = rsmd.getColumnCount();
    //Print columns header
    for (int i = 1 ; i <= columnsNumber ; i++){
        String columnName = rsmd.getColumnName(i);
        System.out.print( columnName.toUpperCase() + " |
    ");
    }

    //Print rows
    while (result.next()){
        System.out.print("\n");
        for (int i = 1 ; i <= columnsNumber ; i++){
            System.out.print(result.getString(i) + " | ");
        }
    }

    result.close();
}
```

Returns:

```
Select the table to display
emp
EMPNO | ENAME | EFIRST | JOB | MGR | HIREDATE | SAL | COMM | TEL | DEPTNO | MOBTEL |
7499 | ALLEN | BOB | SALESMAN | 7698 | 1981-02-20 | 1600 | 300 | 0149547243 | 30 | null |
7521 | WARD | PETER | SALESMAN | 7698 | 1981-02-22 | 1250 | 500 | 0149545247 | 30 | null |
7566 | JONES | JOHN | MANAGER | 7839 | 1981-04-02 | 2975 | null | 0149545456 | 20 | null |
7654 | MARTIN | JOE | SALESMAN | 7698 | 1981-09-28 | 1250 | 1400 | 0149545784 | 30 | null |
7698 | BLAKE | BOB | MANAGER | 7839 | 1981-05-01 | 2850 | null | 0149545254 | 30 | null |
7782 | CLARK | JOHN | MANAGER | 7839 | 1981-06-09 | 2450 | null | 0149545245 | 10 | null |
7788 | SCOTT | GUY | ANALYST | 7566 | 1982-12-09 | 3000 | null | 0149545249 | 20 | null |
7844 | TURNER | PETER | SALESMAN | 7698 | 1981-09-08 | 1500 | 0 | 0149548243 | 30 | null |
7876 | ADAMS | JOSEPH | CLERK | 7788 | 1983-01-12 | 1100 | null | 0149565243 | 20 | null |
7900 | JAMES | ALAN | CLERK | 7698 | 1981-12-03 | 950 | null | 0149545564 | 30 | null |
7902 | FORD | MARIA | ANALYST | 7566 | 1981-12-03 | 3000 | null | 0149785243 | 20 | null |
7934 | MILLER | ALICE | CLERK | 7782 | 1982-01-23 | 1300 | null | 0199545243 | 10 | null |
7839 | KING | GUY | PRESIDENT | 7839 | 1981-11-17 | 5000 | null | 0149545241 | 10 | null |
7369 | SMITH | JOHN | CLERK | 7902 | 1980-12-17 | 7000 | null | 0149545243 | 20 | null |
```

Exercise 4:

The security flow here is the user isn't restricted to just to put a table name at the exercise before, they can write more code after. This is called SQL injection and can make a very big mess if we don't fix this vulnerability. With basic native methods we just have raw and fixed code, which may be repetitive and less efficient.

Exercise 5:

So I Commented the unsecured statement and added the prepared statement, and used "?" to set the values later with "set" method:

```
public static void moveDepartment(int empno, int
newDeptno) throws SQLException {
    /*Statement statement = connexion.createStatement();
    int result = statement.
        executeUpdate( "UPDATE EMP SET deptno = " +
newDeptno +
                                " WHERE empno = " + empno
    );
    */
    PreparedStatement preparedStatement = connexion.
        prepareStatement("UPDATE emp SET deptno = ?
WHERE empno = ?");
    preparedStatement.setInt(1, newDeptno);
    preparedStatement.setInt(2, empno);
    int result = preparedStatement.executeUpdate();
    if (result > 0){
        System.out.println("Employee " + empno + " moved
to department " + newDeptno);
    } else {
        System.out.println("Something occurred when trying
to update");
    }
}
```

Exercise 6:

With this query is not possible because table names are not parameterizable with "?" so we will need another method to filter these queries.

PART II: DAO

Exercise 8:

The cons I see are:

1. The repetitive code that I had to write every time I wanted a new function: create a statement object, do the query, extract what I want from it and close the connection, which also makes it too manual, having to open and close every connection.
2. No abstraction.
3. Just simple relations and transaction management.
4. Very limited scalability

Exercise 9:

To create the POJO class for Department, we need to create as attributes as columns the table have (deptno, dname, loc) and their getters and setters:

```
public class Dept {  
    private int deptNo;  
  
    private String dname;  
  
    private String loc;  
  
    public int getDeptNo() {  
        return deptNo;  
    }  
  
    public void setDeptNo(int deptNo) {  
        this.deptNo = deptNo;  
    }  
  
    public String getDname() {  
        return dname;  
    }  
  
    public void setDname(String dname) {  
        this.dname = dname;  
    }  
  
    public String getLoc() {  
        return loc;  
    }  
  
    public void setLoc(String loc) {  
        this.loc = loc;  
    }  
  
    @Override  
    public String toString() {  
        return "Dept{" +  
            "deptNo=" + deptNo +  
            ", dname='" + dname + '\'' +  
            ", loc='" + loc + '\'' +  
            '}';  
    }  
}
```

Exercise 10:

Implemented find method for DeptDAO (and toString added to the Dept class:

```
public class DeptDAO extends DAO<Dept> {

    public DeptDAO(Connection connect) {
        super(connect);
    }

    @Override
    public Dept find(int id) {
        Dept dept = null;
        try {
            PreparedStatement preparedStatement =
connect.prepareStatement(
                "SELECT * FROM DEPT WHERE deptno = ?"
            );
            preparedStatement.setInt(1, id);
            ResultSet result =
preparedStatement.executeQuery();
            if(result.next()){
                dept = new Dept();
                dept.setDeptNo(result.getInt(1));
                dept.setDname(result.getString(2));
                dept.setLoc(result.getString(3));
            }
            result.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return dept;
    }
}
```

The result:

```
Dept{deptNo=20, dname='RESEARCH', loc='DALLAS'}
```

Exercise 11:

As said, I had to find the managers recursively until finding the president, who in this case has no managers on top of him. In this case, as I put himself as his own manager, when the mgr column has the same number as the employee, it means we got to the president, so it will return null and will exit the recursion.

For the department I just created a DeptDAO and used find to get the department.

```
public class EmpDAO extends DAO<Emp> {

    public EmpDAO(Connection connect) {
        super(connect);
    }

    @Override
    public Emp find(int id) {
        Emp emp = null;
        try {
            PreparedStatement preparedStatement =
connect.prepareStatement(
                "SELECT * FROM EMP WHERE empno = ?"
            );
            preparedStatement.setInt(1, id);
            ResultSet result =
preparedStatement.executeQuery();
            if(result.next()){
                emp = new Emp();
                emp.setEmpNo(result.getLong(1));
                emp.setName(result.getString(2));
                emp.setEfirst(result.getString(3));
                emp.setJob(result.getString(4));
                int mgrNo = result.getInt(5);
                if (emp.getEmpNo() != mgrNo){
                    Emp emp2 = find(mgrNo);
                    emp.setMgr(emp2);
                } else {
                    emp.setMgr(null);
                }
                emp.setHireDate(result.getDate(6));
                emp.setSal(result.getInt(7));
                emp.setComm(result.getInt(8));
                emp.setTel(result.getInt(9));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return emp;
    }
}
```

GITHUB REPO FOR ALL 3 TPs: https://github.com/AbrahamGautiel/AdvDBandBD_Lab1_2_3.git
IT HAS ALL SQL AND JAVA FILES

```
        int deptNo = result.getInt(10);
        DAO<Dept> DeptDao = new DeptDAO(connect);
        Dept dept = DeptDao.find(deptNo);
        emp.setDepartment(dept);
    }
    result.close();
} catch (SQLException e) {
    throw new RuntimeException(e);
}
return emp;
}
```

The result has a manager inside the other inside the other, so I can't display all the information, but it will look like this:

```
Emp{empNo=7369, ename='SMITH', efirst='JOHN', job='CLERK', mgr=Emp{empNo=7902, ename='FORD'
```

and it will continue with the manager data, until it displays the president data, and then it will display the rest of the employee data after managers.

Exercise 12:

Using the tutorial given at the TP, we create our Factory. First we need an interface, which is already implemented like a superclass in the project.

Then, we create the concrete classes which are EmpDAO and DeptDAO from previous exercises.

For the Factory, the tutorial uses one method, but in the schema the DAOFactory uses three different ones, it's the same. Here is how I did it:

```
public class DAOFactory {  
  
    private final Connection connection;  
  
    public DAOFactory(Connection connection) {  
        this.connection = connection;  
    }  
  
    DAO<Emp> getEmpDAO() {  
        return new EmpDAO(connection);  
    }  
  
    DAO<Dept> getDeptDAO() {  
        return new DeptDAO(connection);  
    }  
}
```

In my case I created a constructor for the factory to reuse the connection Main class uses to avoid creating a new one.

Now when testing it with the previous exercises:

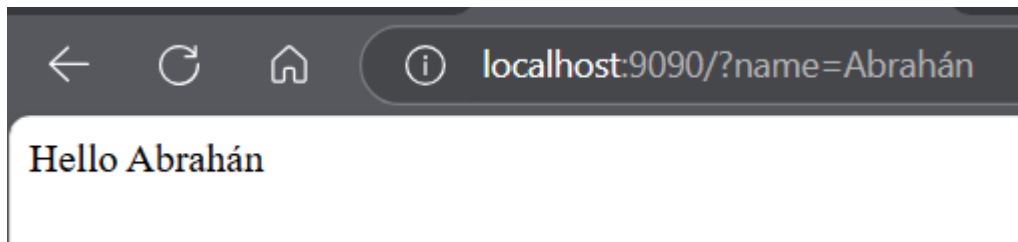
```
DAOFactory factory = new DAOFactory(connexion);  
  
DAO<Dept> DeptDao = factory.getDeptDAO();  
Dept dept20 = DeptDao.find(20);  
System.out.println(dept20);  
  
DAO<Emp> EmpDao = factory.getEmpDAO();  
Emp emp7369 = EmpDao.find(7369);  
System.out.println(emp7369);
```

We can create the DAO objects from a same factory instead of using a different class (EmpDAO and DeptDAO) to create the DAO objects.

Part III: Spring Boot & JPA.

Exercise III.1:

Following the steps provided at the TP, I could do the Spring project successfully. Here's my Spring project's Hello World:



I don't see necessary putting the code since it's the same as the provided at the TP, but it all will be

Exercise III.2

Done. Only thing I want to point out. There's a typo at the TP Document. In the postgres dependency it's written "org.postre" instead of "org.postgre" the "g" is missing.

Exercise III.3

I have configured lombok so it adds the getters and setters on the run. Only things I had to do was add the lombok dependency, enable Annotation processing and just in case, because it was throwing some errors, installed the lombok plugin. This is how the page looks:

```
[
  {
    "empno": 7499,
    "ename": "ALLEN",
    "efirst": "BOB",
    "job": "SALESMAN",
    "mgr": 7698,
    "sal": 1600
  },
  {
    "empno": 7521,
    "ename": "WARD",
    "efirst": "PETER",
    "job": "SALESMAN",
```

```
"mgr": 7698,
"sal": 1250
},
{
  "empno": 7566,
  "ename": "JONES",
  "efirst": "JOHN",
  "job": "MANAGER",
  "mgr": 7839,
  "sal": 2975
},
{
  "empno": 7654,
  "ename": "MARTIN",
  "efirst": "JOE",
  "job": "SALESMAN",
  "mgr": 7698,
  "sal": 1250
},
{
  "empno": 7698,
  "ename": "BLAKE",
  "efirst": "BOB",
  "job": "MANAGER",
  "mgr": 7839,
  "sal": 2850
},
{
  "empno": 7782,
  "ename": "CLARK",
  "efirst": "JOHN",
  "job": "MANAGER",
  "mgr": 7839,
  "sal": 2450
},
{
  "empno": 7788,
  "ename": "SCOTT",
  "efirst": "GUY",
  "job": "ANALYST",
  "mgr": 7566,
  "sal": 3000
},
{
  "empno": 7844,
  "ename": "TURNER",
  "efirst": "PETER",
  "job": "SALESMAN",
  "mgr": 7698,
  "sal": 1500
},
{
```

GITHUB REPO FOR ALL 3 TPs: https://github.com/AbrahanGautiel/AdvDBandBD_Lab1_2_3.git
IT HAS ALL SQL AND JAVA FILES

```
    "empno": 7876,  
    "ename": "ADAMS",  
    "efirst": "JOSEPH",  
    "job": "CLERK",  
    "mgr": 7788,  
    "sal": 1100  
  },  
  {  
    "empno": 7900,  
    "ename": "JAMES",  
    "efirst": "ALAN",  
    "job": "CLERK",  
    "mgr": 7698,  
    "sal": 950  
  },  
  {  
    "empno": 7902,  
    "ename": "FORD",  
    "efirst": "MARIA",  
    "job": "ANALYST",  
    "mgr": 7566,  
    "sal": 3000  
  },  
  {  
    "empno": 7934,  
    "ename": "MILLER",  
    "efirst": "ALICE",  
    "job": "CLERK",  
    "mgr": 7782,  
    "sal": 1300  
  },  
  {  
    "empno": 7839,  
    "ename": "KING",  
    "efirst": "GUY",  
    "job": "PRESIDENT",  
    "mgr": 7839,  
    "sal": 5000  
  },  
  {  
    "empno": 7369,  
    "ename": "SMITH",  
    "efirst": "JOHN",  
    "job": "CLERK",  
    "mgr": 7902,  
    "sal": 7000  
  }  
]
```

Exercise III.4

After following all steps I ended up with the UI:

The screenshot shows the 'Api Documentation' page for version 1.0. It includes the base URL 'localhost:9090/' and a link to 'http://localhost:9090/v2/api-docs'. Below this, there are links for 'Terms of service' and 'Apache 2.0'. The main content area lists two controllers: 'basic-error-controller' (Basic Error Controller) and 'simple-controller' (Simple Controller). The 'simple-controller' is expanded, showing three endpoints: a GET endpoint for '/hello', a GET endpoint for '/employees' (getEmployees), and a POST endpoint for '/employees' (addEmployee). At the bottom, there is a 'Models' section with a right arrow.

Api Documentation ^{1.0}

[Base URL: localhost:9090/]
<http://localhost:9090/v2/api-docs>

Api Documentation
[Terms of service](#)
[Apache 2.0](#)

basic-error-controller Basic Error Controller >

simple-controller Simple Controller ✓

GET / hello

GET /employees getEmployees

POST /employees addEmployee

Models >

One thing I had to add is since the code is done so the empno is auto-generated, hibernate looks for a sequence called “hibernate_sequence” which didn’t find and it wouldn’t let me use the endpoint, so I created it manually, starting from the higher empno (7934) to avoid it creating employees with already existing empno:

```
CREATE SEQUENCE hibernate_sequence START 7935;
```

GITHUB REPO FOR ALL 3 TPs: https://github.com/AbrahamGautiel/AdvDBandBD_Lab1_2_3.git
IT HAS ALL SQL AND JAVA FILES

How did I find the issue:

I had to create application.properties and add this code to know why it was throwing the error, because at the UI it just said “could not extract ResultSet in hibernate” and after creating that file with these lines:

```
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format_sql=true  
logging.level.org.hibernate.SQL=DEBUG  
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Now, after trying the POST endpoint now, at the server console it showed that the sequence didn't exist.

Exercise III.5

-Get by ID: Search the employee specified at the URL

```
@GetMapping(value="/employees/{id}")
public Emp getEmployeeById(@PathVariable Long id) {
    return empRepository.findById(id).map(emp ->
        ResponseEntity.ok().body(emp)
    ).orElse(ResponseEntity.notFound().build()).getBody();
}
```

When trying with the UI:

The screenshot displays a web application interface for testing HTTP requests. It is divided into several sections: 'Curl' at the top, followed by 'Request URL', 'Server response', and a table for 'Response body'. The 'Curl' section contains the command: `curl -X GET "http://localhost:9090/employees/7934" -H "accept: */*"`. The 'Request URL' section shows `http://localhost:9090/employees/7934`. The 'Server response' section shows a status code of 200. The 'Response body' section displays a JSON object: `{ "empno": 7934, "ename": "MILLER", "efirst": "ALICE", "job": "CLERK", "mgr": 7782, "sal": 1300 }`. A 'Download' button is located at the bottom right of the response body area.

Code	Details
200	<pre>{ "empno": 7934, "ename": "MILLER", "efirst": "ALICE", "job": "CLERK", "mgr": 7782, "sal": 1300 }</pre>

-Update by ID: search the employee at the URL and edits it

```
@PutMapping("/employees/{id}")
public ResponseEntity<Emp> updateEmployee(@PathVariable Long
id, @RequestBody Emp empDetails) {
    return empRepository.findById(id)
        .map(emp -> {
            emp.setName(empDetails.getName());
            emp.setEfirst(empDetails.getEfirst());
            emp.setJob(empDetails.getJob());
            emp.setMgr(empDetails.getMgr());
            emp.setSal(empDetails.getSal());
            return
ResponseEntity.ok(empRepository.save(emp));
        })
        .orElse(ResponseEntity.notFound().build());
}
```

When trying it: Updating the previous employee so now it has sales as job
and it has more salary

Curl

```
curl -X PUT "http://localhost:9090/employees/7934" -H "accept: /*/*" -H "Content-Type: application/json" -d
"{ \"empno\": 7934, \"ename\": \"MILLER\", \"efirst\": \"ALICE\", \"job\": \"SALES\", \"mgr\": 7782,
\"sal\": 1600}"
```

Request URL

http://localhost:9090/employees/7934

Server response

Code

200

Details

Response body

```
{
  "empno": 7934,
  "ename": "MILLER",
  "efirst": "ALICE",
  "job": "SALES",
  "mgr": 7782,
  "sal": 1600
}
```

Download

GITHUB REPO FOR ALL 3 TPs: https://github.com/AbrahamGautiel/AdvDBandBD_Lab1_2_3.git
IT HAS ALL SQL AND JAVA FILES

-Delete by ID:

```
@DeleteMapping("employees/{id}")
public ResponseEntity<Object> deleteEmployee(@PathVariable
Long id) {
    return empRepository.findById(id)
        .map(employee -> {
            empRepository.delete(employee); // Elimina el
            empleado de la base de datos
            return ResponseEntity.noContent().build(); //
            Retorna un 204 No Content
        })
        .orElse(ResponseEntity.notFound().build()); // Si
        no existe, retorna un 404 Not Found
}
```

When trying it: If it returns a 204 it means it deleted it successfully.

Curl

```
curl -X DELETE "http://localhost:9090/employees/7936" -H "accept: */*"
```

Request URL

```
http://localhost:9090/employees/7936
```

Server response

Code	Details
204	<p>Response headers</p> <pre>connection: keep-alive date: Sat, 29 Mar 2025 19:10:35 GMT keep-alive: timeout=60</pre>

GetAll is already implemented with the endpoint getEmployees.

Main function of Parts I and II:

```
public static void main(String[] args) {
    /* Load JDBC Driver. */
    try {
        Class.forName( "org.postgresql.Driver" );
    } catch ( ClassNotFoundException e ) {
        e.printStackTrace();
    }

    String url = "jdbc:postgresql://localhost/TP_1";
    String user = "postgres";
    String pass = "root";
    try {
        connexion = DriverManager.getConnection( url, user,
pass );

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

        /*Exercises are commented just to avoid exdcuting
them all, to reproduce an exercise,
        just move the * and / next to "Exercise i"
        */

        /*Exercise 1
displayDepartment();
        */

        Scanner sc = new Scanner(System.in);
        /*Exercise 2 and 5
        System.out.println("Select an employee to change
their dept (e.g 7369)");
        int empno = sc.nextInt();

        System.out.println("Select which department you
want to move them (10, 20, 30, 40)");
        int newDeptno = sc.nextInt();
        moveDepartment(empno, newDeptno);
        */

        /*Exercise 3 and 6
```

```
        System.out.println("Select the table to display");
        String tableName = sc.next();
        displayTable(tableName);
    */

    /*Exercise 10
    DAO<Dept> DeptDao = new DeptDAO(connexion);
    Dept dept20 = DeptDao.find(20);
    System.out.println(dept20);
    */

    /*Exercise 11
    DAO<Emp> EmpDao = new EmpDAO(connexion);
    Emp emp7369 = EmpDao.find(7369);
    System.out.println(emp7369);
    */

    /*Exescise 12*/
    DAOFactory factory = new DAOFactory(connexion);

    DAO<Dept> DeptDao = factory.getDeptDAO();
    Dept dept20 = DeptDao.find(20);
    System.out.println(dept20);

    DAO<Emp> EmpDao = factory.getEmpDAO();
    Emp emp7369 = EmpDao.find(7369);
    System.out.println(emp7369);
} catch ( SQLException e ) {
    e.printStackTrace();
} finally {
    if ( connexion != null )
        try {
            connexion.close();
        } catch ( SQLException ignore ) {
            ignore.printStackTrace();
        }
}
}
```