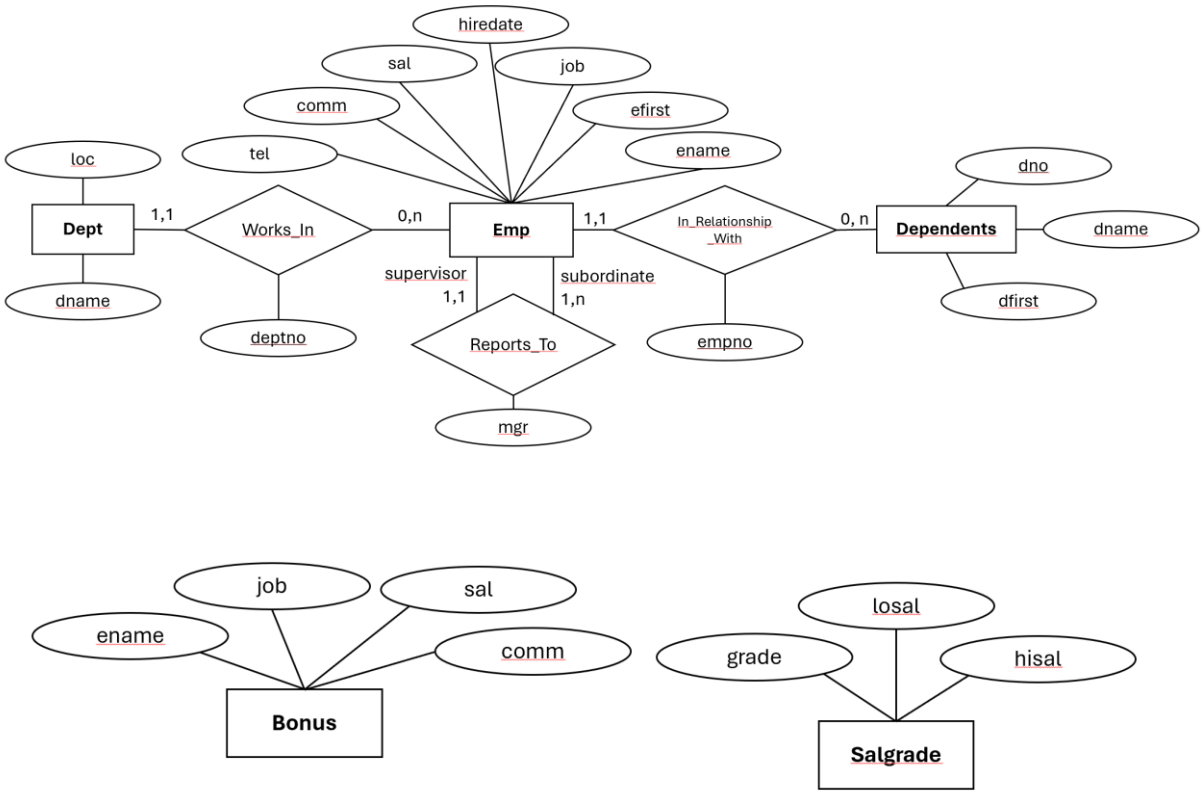
Edouard Yu

Advanced Database - TP 1 SQL

Exercise 1: LDD queries

2. E/R diagram:



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

ALTER TABLE emp ADD CONSTRAINT unique_name_tel UNIQUE (ename, efirst, tel);

4. Second integrity Constraint: Set up an integrity constraint in order to make sure that the entered mobile number begins with 06:

ALTER TABLE emp ADD CONSTRAINT check_mobile_phone_number CHECK (tel LIKE '06%');

5. Third integrity Constraint On Delete: In practice, we realize that when one wants to delete an employee, it is necessary to remove all his dependents. How is it possible to make such removal dynamic?

ALTER TABLE dependents DROP CONSTRAINT fk_dependent_emp, ADD CONSTRAINT fk_dependent_emp FOREIGN KEY (empno) REFERENCES emp (empno) ON DELETE CASCADE;

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

In question 4, we added a constraint which forces all mobile numbers to start with 06. However, the insertion script contains employees who have a mobile number which begins with 01. Therefore, we must replace their mobile numbers so that all the numbers in the script start with 06. In addition, the president doesn't have an mgr, however this column has the constraint of being not null, so let's assign it an mgr randomly, let's say the number 1 because it is the higher hierarchically.

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

CREATE SEQUENCE dependents_dno_sequence START WITH 8000 INCREMENT BY 1;

8. Use Sequence: Add some tuples to the DEPENDENT table using the previous sequence. Use ma_sequence.NEXTVAL to get the values from the sequence:

INSERT INTO dependents VALUES (dependents_dno_sequence.NEXTVAL, 'BLAKE', 'JENNY', 7698); doesn't work on PostgreSQL,
let's fix the syntax:
INSERT INTO dependents VALUES (nextval('dependents_dno_sequence'), 'BLAKE', 'JENNY', 7698);

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

Instead of using a SEQUENCE, PostgreSQL allows us to auto-increment our primary key with the functions SERIAL (old implementation) or GENERATED AS IDENTITY (new implementation). This can solve some problems such as forgetting to increment the sequence during an insertion, which can cause an error later. Let's make the modification:

ALTER TABLE dependents ALTER COLUMN dno ADD GENERATED BY DEFAULT AS IDENTITY (START WITH 8000);

INSERT INTO dependents (dname, dfirst, empno) VALUES ('JONES', 'DENNY', 7566);

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

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

SELECT table_name, column_name

FROM information_schema.columns

WHERE table_schema NOT IN ('pg_catalog', 'information_schema')

ORDER BY table_name, ordinal_position;

	table_name name	column_name name
1	bonus	ename
2	bonus	job
3	bonus	sal
4	bonus	comm
5	dependents	dno
6	dependents	dname
7	dependents	dfirst
8	dependents	empno
9	dept	deptno
10	dept	dname
11	dept	loc
12	emp	empno
13	emp	ename
14	emp	efirst
15	emp	job
16	emp	mgr
17	emp	hiredate
18	emp	sal
19	emp	comm
20	emp	tel
21	emp	deptno
22	salgrade	grade
23	salgrade	losal
24	salgrade	hisal

2. Select employees whose commission is higher than their salary

```
SELECT *
FROM emp
WHERE emp.comm > emp.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	deptno integer
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30

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

```
SELECT *
FROM emp
WHERE (COALESCE(emp.comm, 0) + emp.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	deptno integer
1	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30
2	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30
3	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10
4	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30

4. Select employees who are CLERK or ANALYST

```
SELECT *
FROM emp
WHERE emp.job IN ('CLERK', '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	deptno integer
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
2	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20
4	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
5	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
6	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10

5. Select employees whose name begins by M

```
SELECT *
FROM emp
WHERE emp.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	deptno integer
1	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10

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

```
SELECT *
FROM emp
WHERE emp.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	deptno integer
1	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
2	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
3	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30

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

```
SELECT *
FROM emp
WHERE emp.deptno = 10 AND emp.job IN ('CLERK', 'MANAGER') AND emp.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	deptno integer
1	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10

8. Select employees whose commission is NULL

```
SELECT *
FROM emp
WHERE emp.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	deptno integer
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
2	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
3	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
4	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
5	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
6	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
7	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20
8	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
9	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
10	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10

9. Select employees by ascending order (by hiredate)

```
SELECT *
FROM emp
ORDER BY emp.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	deptno integer
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
2	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30
3	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30
4	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
5	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
6	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
7	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30
8	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30
9	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
10	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
11	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
12	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10
13	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
14	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20

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

```
SELECT *
FROM emp
ORDER BY emp.job ASC, emp.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	deptno integer
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
2	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
3	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10
4	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20
5	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
6	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
7	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
8	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
9	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
10	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
11	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30
12	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30
13	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30
14	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30

11. Select departments without employees

```
SELECT *
FROM dept
LEFT JOIN emp ON dept.deptno = emp.deptno
WHERE emp.deptno IS NULL;
```

	deptno integer	dname character varying (14)	loc character varying (13)	empno 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	40	OPERATIONS	BOSTON	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]

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

```
SELECT e.empno, e.ename, e.efirst, CONCAT(m.efirst, ' ', m.ename) AS managername
FROM emp e
LEFT JOIN emp m ON e.mgr = m.empno;
```

	empno [PK] integer	ename character varying (10)	efirst character varying (10)	managername text
1	7369	SMITH	JOHN	MARIA FORD
2	7521	WARD	PETER	BOB BLAKE
3	7566	JONES	JOHN	GUY KING
4	7654	MARTIN	JOE	BOB BLAKE
5	7698	BLAKE	BOB	GUY KING
6	7782	CLARK	JOHN	GUY KING
7	7788	SCOTT	GUY	JOHN JONES
8	7839	KING	GUY	
9	7844	TURNER	PETER	BOB BLAKE
10	7876	ADAMS	JOSEPH	GUY SCOTT
11	7900	JAMES	ALAN	BOB BLAKE
12	7902	FORD	MARIA	JOHN JONES
13	7934	MILLER	ALICE	JOHN CLARK
14	7499	ALLEN	BOB	BOB BLAKE

Note: for lines 1 and 10, MARIA FORD and GUY SCOTT aren't managers but analysts. For lines 3, 5 and 6, the employees are themselves managers and are led by the president GUY KING (line 8) who has no hierarchical superior.

13. List employees earning more than JONES

```
SELECT *
FROM emp
WHERE emp.sal + COALESCE(emp.comm, 0) >
(SELECT emp.sal + COALESCE(emp.comm, 0)
FROM emp
WHERE emp.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	deptno integer
1	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
2	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
3	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20

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

```
SELECT CONCAT('salary: ', emp.sal, COALESCE(', commission: ' || emp.comm, '')) AS "salary&commission", *
FROM emp;
```

	salary&commission text	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	salary: 800	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
2	salary: 1250, commission: 500	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30
3	salary: 2975	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
4	salary: 1250, commission: 1400	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30
5	salary: 2850	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
6	salary: 2450	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
7	salary: 3000	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
8	salary: 5000	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
9	salary: 1500, commission: 0	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30
10	salary: 1100	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20
11	salary: 950	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
12	salary: 3000	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
13	salary: 1300	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10
14	salary: 1600, commission: 300	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30

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

```
SELECT DISTINCT dept.deptno
FROM dept
INNER JOIN emp ON dept.deptno = emp.deptno;
```

	deptno [PK] integer
1	10
2	20
3	30

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

```
SELECT e1.*
FROM emp e1
INNER JOIN dept ON e1.deptno = dept.deptno
WHERE dept.loc = 'CHICAGO'
AND e1.job = (SELECT e2.job
FROM emp e2
WHERE e2.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	deptno integer
1	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30

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

```
SELECT e.*
FROM emp e
LEFT JOIN emp m ON e.mgr = m.empno
WHERE e.deptno != m.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	deptno integer
1	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30

Following question 12, we can see that the managers JOHN JONES and BOB BLAKE don’t work in the same department as the president. Excluding managers, all employees work in the same department as their superiors

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

```
WITH deptclerk AS (
   SELECT DISTINCT emp.deptno
   FROM emp
   WHERE emp.job = 'CLERK'
)
SELECT emp.*
FROM emp
JOIN deptClerk ON deptclerk.deptno = emp.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	deptno integer
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20
2	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30
3	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20
4	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30
5	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
6	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
7	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20
8	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10
9	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30
10	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20
11	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30
12	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20
13	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10
14	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30

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

```
WITH salesdeptjob AS (
 SELECT DISTINCT job
    FROM emp
    JOIN dept ON emp.deptno = dept.deptno
    WHERE dept.dname = 'SALES'
)
SELECT e.*
FROM emp e
INNER JOIN salesdeptjob ON e.job = salesdeptjob.job
WHERE e.deptno = 10;
```

	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]	0649545245	10
2	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10

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

```
SELECT *
FROM emp
WHERE emp.job =
(SELECT emp.job
FROM emp
WHERE emp.ename = 'JONES')
OR emp.sal >
(SELECT emp.sal
FROM emp
WHERE emp.ename = 'FORD');
```

	empno [PK] integer	ename character va	efirst racter 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]	0649545456	20
2	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30
3	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10
4	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10

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

```
SELECT *
FROM emp
WHERE emp.sal >
ALL (SELECT emp.sal
FROM emp
WHERE emp.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	deptno integer
1	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10

Exercise 3: Join Table

1. **Create a Table for employee's Projects:**

```
CREATE TABLE project (
    projno INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    pname VARCHAR(255),
    startdate DATE,
    budget DECIMAL(10, 2)
);
```

2. **Create the Join Table and write some requests to insert elements in the table:**

```
CREATE TABLE project_emp (
    empno INT,
    projno INT,
    PRIMARY KEY (empno, projno),
    CONSTRAINT emp_fk FOREIGN KEY (empno) REFERENCES emp(empno),
    CONSTRAINT project_fk FOREIGN KEY (projno) REFERENCES project(projno)
);

INSERT INTO project (pname, startdate, budget) VALUES ('Projet A', '2023-01-01', 10000);
INSERT INTO project (pname, startdate, budget) VALUES ('Projet B', '2023-02-15', 15000);
INSERT INTO project (pname, startdate, budget) VALUES ('Projet C', '2023-02-20', 15000);

INSERT INTO project_emp (empno, projno) VALUES (7521, 1);
INSERT INTO project_emp (empno, projno) VALUES (7900, 1);
INSERT INTO project_emp (empno, projno) VALUES (7654, 2);
INSERT INTO project_emp (empno, projno) VALUES (7844, 2);
INSERT INTO project_emp (empno, projno) VALUES (7900, 2);
INSERT INTO project_emp (empno, projno) VALUES (7521, 3);
INSERT INTO project_emp (empno, projno) VALUES (7499, 3);
INSERT INTO project_emp (empno, projno) VALUES (7900, 3);
```

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

```
SELECT empno
FROM project_emp
GROUP BY empno
HAVING COUNT(DISTINCT projno) = (SELECT COUNT(*) FROM project);
```

	empno integer
1	7900

4. **Options on View creation:**

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

This instruction creates a view, a virtual table that facilitates access to specific data, named sales_staff which selects the employee numbers, names, and department numbers of employees in department 10. The WITH CHECK OPTION ensures that all changes contributing to this view, such as inserts or updates, adhere to the WHERE deptno = 10 condition, helping to preserve the integrity of the data within this view.

5. **View Insertion:**

```
ALTER TABLE emp ALTER COLUMN mgr DROP NOT NULL;
```
to avoid the error caused by the not null constraint.

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```
the operation was successful because this insertion meets the condition defined in the sales_staff view (deptno = 10). The employee 'OSTER' is also added to the emp table.

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```
the operation failed because this insert doesn’t meet the condition defined in the sales_staff view (deptno = 10).

Advanced Database - TP 2 Advanced SQL

Exercise 1: Analytics Queries. Window Queries

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

```
WITH deptemp AS (
SELECT *, ROW_NUMBER()
OVER (PARTITION BY emp.deptno ORDER BY emp.hiredate DESC) AS rankemp
FROM emp WHERE emp.hiredate IS NOT NULL // To delete the line with empno 7584, ename: OSTER due to TP1 exercice 3
SELECT *
FROM deptemp
WHERE deptemp.rankemp <= 2;
```

	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	rankemp bigint
1	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10	1
2	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10	2
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20	1
4	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20	2
5	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30	1
6	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30	2

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

```
SELECT *,
ROUND(avg(emp.sal) OVER
(ORDER BY emp.sal ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING )
) AS salavgwith2othersemp
FROM emp;
```

We display here the average salary of 3 people, between the employee row and the employee who earns just a little more than him and the employee who earns just a little less than him.

Query Query History Scratch Pad ×

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	salavgwith2othersemp numeric
1	7369	SMITH	JOHN	CLERK	7902	1980-12-17	800	[null]	0649545243	20	875
2	7900	JAMES	ALAN	CLERK	7698	1981-12-03	950	[null]	0649545564	30	950
3	7876	ADAMS	JOSEPH	CLERK	7788	1983-01-12	1100	[null]	0649565243	20	1100
4	7654	MARTIN	JOE	SALESMAN	7698	1981-09-28	1250	1400	0649545784	30	1200
5	7521	WARD	PETER	SALESMAN	7698	1981-02-22	1250	500	0649545247	30	1267
6	7934	MILLER	ALICE	CLERK	7782	1982-01-23	1300	[null]	0699545243	10	1350
7	7844	TURNER	PETER	SALESMAN	7698	1981-09-08	1500	0	0649548243	30	1467
8	7499	ALLEN	BOB	SALESMAN	7698	1981-02-20	1600	300	0649547243	30	1850
9	7782	CLARK	JOHN	MANAGER	7839	1981-06-09	2450	[null]	0649545245	10	2300
10	7698	BLAKE	BOB	MANAGER	7839	1981-05-01	2850	[null]	0649545254	30	2758
11	7566	JONES	JOHN	MANAGER	7839	1981-04-02	2975	[null]	0649545456	20	2942
12	7788	SCOTT	GUY	ANALYST	7566	1982-12-09	3000	[null]	0649545249	20	2992
13	7902	FORD	MARIA	ANALYST	7566	1981-12-03	3000	[null]	0649785243	20	3667
14	7839	KING	GUY	PRESIDENT	1	1981-11-17	5000	[null]	0649545241	10	4000
15	7584	OSTER	[null]	[null]	[null]	[null]	[null]	[null]	[null]	10	5000

Exercise 2: Index & Explain Plan

1. Now let's create a new schema in the database and go over it with ‘SET search_path TO’ command line to do exercises 2 and 3. We create the new tables with values with the script.

2. **Tune the following query:**

SELECT gender, COUNT(*) FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

	gender character varying (2)	count bigint
1	M	31
2	F	28

Total rows: 2 of 2 Query complete 00:00:00.122 = 122 ms

3.1. Use EXPLAIN plan to analyze the query:

EXPLAIN SELECT gender, COUNT(*) FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

	QUERY PLAN text
1	HashAggregate (cost=114.79..114.81 rows=2 width=10)
2	Group Key: gender
3	-> Seq Scan on emp_medium_table (cost=0.00..114.50 rows=59 width=...
4	Filter: (manager_id = 7)

Total rows: 4 of 4 Query complete 00:00:00.102

To get the Planning Time and Execution Time:

EXPLAIN ANALYZE SELECT gender, COUNT(*) FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

	QUERY PLAN text
1	HashAggregate (cost=114.79..114.81 rows=2 width=10) (actual time=0.313..0.314 rows=2 loops=1)
2	Group Key: gender
3	Batches: 1 Memory Usage: 24kB
4	-> Seq Scan on emp_medium_table (cost=0.00..114.50 rows=59 width=2) (actual time=0.022..0.287 rows=59 loops...
5	Filter: (manager_id = 7)
6	Rows Removed by Filter: 4941
7	Planning Time: 0.113 ms
8	Execution Time: 0.350 ms

Total rows: 8 of 8 | Query complete 00:00:00.082 | Ln 1, Col 17

3.2. Activate stats:

CREATE EXTENSION pg_stat_statements;

We execute the following code 10 times:

SELECT gender, COUNT(*) from FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

By running the following code, we obtain the values below:

SELECT * FROM pg_stat_statements

WHERE QUERY LIKE '%group by gender' AND QUERY NOT LIKE 'EXPLAIN%';

	calls bigint	total_exec_time double precision	min_exec_time double precision	max_exec_time double precision	mean_exec_time double precision	stddev_exec_time double precision
1	10	3.6307	0.25549999999999995	0.7441	0.36306999999999995	0.13699645287378792

4. Add a covering index on both columns fetched:

CREATE INDEX manager_id_gender_index ON emp_medium_table(manager_id, gender);

5. Reset the stats:

SELECT pg_stat_statements_reset();

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

SELECT gender, COUNT(*) from FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

SELECT * FROM pg_stat_statements;

WHERE QUERY LIKE '%group by gender' AND QUERY NOT LIKE 'EXPLAIN%';

	calls bigint	total_exec_time double precision	min_exec_time double precision	max_exec_time double precision	mean_exec_time double precision	stddev_exec_time double precision
1	10	0.7263	0.0317	0.3234	0.07262999999999999	0.08427062418185831

We can see that it is 5 times faster now.

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

EXPLAIN ANALYZE SELECT gender, COUNT(*) from FROM emp_medium_table WHERE manager_id = 7 GROUP BY gender;

	QUERY PLAN text
1	GroupAggregate (cost=0.28..5.63 rows=2 width=10) (actual time=0.034..0.038 rows=2 loops=1)
2	Group Key: gender
3	-> Index Only Scan using manager_id_gender_index on emp_medium_table (cost=0.28..5.31 rows=59 width=2) (actual time=0.026..0.029 rows=59 loop…
4	Index Cond: (manager_id = 7)
5	Heap Fetches: 0
6	Planning Time: 0.113 ms
7	Execution Time: 0.061 ms

Planning time remains the same, but the execution time is faster after indexing because the index can quickly locate data without having to scan the entire table.

Exercise 3: Data Dictionary

In our diagram, we have no constraints, let's add some first to answer this question:

```
ALTER TABLE emp_medium_table
ADD CONSTRAINT pk_emp PRIMARY KEY (empno);
ALTER TABLE project_medium_table
ADD CONSTRAINT pk_project PRIMARY KEY (projectno);
```

Use these views to create a table called MY_OBJECTS with 2 columns: Object (Name of your Object) / Type (Table, column, constraint …):

```
CREATE TABLE MY_OBJECTS (
    Object VARCHAR(255),
    Type VARCHAR(255)
);

// Insert Tables
INSERT INTO MY_OBJECTS (Object, Type)
SELECT table_name, 'Table'
FROM information_schema.tables
WHERE table_schema = 'schema_name';

// Insert Columns
INSERT INTO MY_OBJECTS (Object, Type)
SELECT column_name, 'Column'
FROM information_schema.columns
WHERE table_schema = 'schema_name';
```

// Insert Constraints

```
INSERT INTO MY_OBJECTS (Object, Type)
SELECT conname, 'Constraint'
FROM pg_catalog.pg_constraint
JOIN pg_catalog.pg_namespace ON pg_constraint.connamespace = pg_namespace.oid
WHERE nspname = 'schema_name';
```

// Insert Views

```
INSERT INTO MY_OBJECTS (Object, Type)
SELECT table_name, 'View'
FROM information_schema.views
WHERE table_schema = 'schema_name';
```

SELECT * FROM MY_OBJECTS;

	object character varying (255)	type character varying (255)
1	pg_stat_statements_info	Table
2	pg_stat_statements	Table
3	emp_medium_table	Table
4	project_medium_table	Table
5	project_emp_medium_table	Table
6	my_objects	Table
7	userid	Column
8	shared_blks_written	Column
9	type	Column
10	mean_exec_time	Column
11	queryid	Column
12	jit_functions	Column
13	manager_id	Column
14	local_blks_hit	Column
15	name_hashed	Column
16	max_plan_time	Column
17	total_plan_time	Column
18	jit_emission_time	Column
19	toplevel	Column
20	min_exec_time	Column
21	wal_bytes	Column

22	empno	Column
23	min_plan_time	Column
24	stats_reset	Column
25	blk_write_time	Column
26	temp_blks_written	Column
27	jit_emission_count	Column
28	jit_optimization_time	Column
29	blk_read_time	Column
30	plans	Column
31	name	Column
32	total_exec_time	Column
33	query	Column
34	shared_blks_dirtied	Column
35	max_exec_time	Column
36	jit_optimization_count	Column
37	temp_blk_write_time	Column
38	local_blks_read	Column
39	jit_inlining_count	Column
40	jit_generation_time	Column
41	projectno	Column

42	shared_blks_hit	Column
43	dealloc	Column
44	local_blks_written	Column
45	local_blks_dirtied	Column
46	dept_id	Column
47	temp_blks_read	Column
48	projectno	Column
49	shared_blks_read	Column
50	rows	Column
51	mean_plan_time	Column
52	calls	Column
53	wal_fpi	Column
54	object	Column
55	gender	Column
56	dbid	Column
57	jit_inlining_time	Column
58	empno	Column
59	stddev_plan_time	Column
60	wal_records	Column

61	stddev_exec_time	Column
62	temp_blk_read_time	Column
63	pk_emp	Constraint
64	pk_project	Constraint
65	pg_stat_statements_info	View
66	pg_stat_statements	View

Exercise 4: Use Postgres via CLI

For exercises 4 and let's return to the schema developed during TP1

```
postgres=# SELECT * FROM EMP;
 empno | ename  | efirst |    job    | mgr  |  hiredate  | sal  | comm |    tel     | deptno
-------+--------+--------+-----------+------+------------+------+------+------------+--------
  7369 | SMITH  | JOHN   | CLERK     | 7902 | 1980-12-17 |  800 |      | 0649545243 |     20
  7521 | WARD   | PETER  | SALESMAN  | 7698 | 1981-02-22 | 1250 |  500 | 0649545247 |     30
  7566 | JONES  | JOHN   | MANAGER   | 7839 | 1981-04-02 | 2975 |      | 0649545456 |     20
  7654 | MARTIN | JOE    | SALESMAN  | 7698 | 1981-09-28 | 1250 | 1400 | 0649545784 |     30
  7698 | BLAKE  | BOB    | MANAGER   | 7839 | 1981-05-01 | 2850 |      | 0649545254 |     30
  7782 | CLARK  | JOHN   | MANAGER   | 7839 | 1981-06-09 | 2450 |      | 0649545245 |     10
  7788 | SCOTT  | GUY    | ANALYST   | 7566 | 1982-12-09 | 3000 |      | 0649545249 |     20
  7839 | KING   | GUY    | PRESIDENT |    1 | 1981-11-17 | 5000 |      | 0649545241 |     10
  7844 | TURNER | PETER  | SALESMAN  | 7698 | 1981-09-08 | 1500 |    0 | 0649548243 |     30
  7876 | ADAMS  | JOSEPH | CLERK     | 7788 | 1983-01-12 | 1100 |      | 0649565243 |     20
  7900 | JAMES  | ALAN   | CLERK     | 7698 | 1981-12-03 |  950 |      | 0649545564 |     30
  7902 | FORD   | MARIA  | ANALYST   | 7566 | 1981-12-03 | 3000 |      | 0649785243 |     20
  7934 | MILLER | ALICE  | CLERK     | 7782 | 1982-01-23 | 1300 |      | 0699545243 |     10
  7499 | ALLEN  | BOB    | SALESMAN  | 7698 | 1981-02-20 | 1600 |  300 | 0649547243 |     30
  7584 | OSTER  |        |           |      |            |      |      |            |     10
(15 lignes)
```

Exercise 5: Transaction Part 1 – Beginner

1. **On the CLI, run the following code, and on the pgAdmin UI, uncheck the auto-commit option:**

\set AUTOCOMMIT off

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

UPDATE EMP SET SAL = 5000 WHERE EMPNO = 7369;

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

UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;

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

Client 1:

SELECT emp.empno, emp.sal FROM emp WHERE emp.empno = 7369;

	empno [PK] integer	sal integer
1	7369	7000

Client 2:

```
postgres=*# SELECT emp.empno, emp.sal FROM emp WHERE emp.empno = 7369;
 empno | sal
-------+-----
  7369 | 800
(1 ligne)
```

Yes and no, the salary is 7000 on client 1 but on client 2, salary always remains 800 (salary before the two updates of client 1).

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

UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;

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

No because the second client is waiting the first session (the first client) to finish the transaction. The query is currently running indefinitely on client second's session. This is due to the ACID properties of the transactions. Transaction isolation ensures that operations taking place in an uncommitted transaction are not visible to other transactions. We need to commit (or rollback) to transaction (du premier client) to unlock the entry in the row.

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

UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;

What is happening and why?

It seems that for this question, we want to cause a dead lock, but client 1 can still make these changes on the row. To cause a dead lock, client 2 needs to execute a command (update, delete) on a different resource which will lock the resource, this can be another row in the emp table. Then we must execute the command from question 4 which will make client 2 wait because client 1 hasn't committed or rollbacked its transaction. And if client 1 executes a command on the resource that client 2 has locked, there we will have a dead lock.

What is the name of this mechanism? (hint: *ea*lk)**

The mechanism is called a deadlock and occurs when client 1 is waiting for client 2's transaction to release the lock on one resource, while client 2 is waiting for client 1 to release the lock on another resource. None of the transactions can progress, creating a deadlock.

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

COMMIT;

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

```
postgres=*# UPDATE EMP SET SAL = 7000 WHERE EMPNO = 7369;
UPDATE 1
postgres=*# SELECT emp.empno, emp.sal FROM emp WHERE emp.empno = 7369;
 empno | sal
-------+------
  7369 | 7000
(1 ligne)
```

What happened and why?

Since client 1 has unlocked the row, client 2 can now update the row.

TP 3 - A Scene Of Information And Flexibility Java and DataBase Connectivity (JDBC)

Github:

- **Part I and II:** https://github.com/EdouardYu/SQL-TP/tree/part1and2
- **Exercise 7:** https://github.com/EdouardYu/SQL-TP/tree/exercise7
- **Part III :** https://github.com/EdouardYu/SQL-TP/tree/part3

Part I: A Graphic Of Table: communication with database

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

    try {
        String url = "jdbc:postgresql://localhost:5432/postgres";
        String user = "postgres";
        String pass = "Xinyi.xian2509";
        Connection connection = DriverManager
                                        .getConnection( url, user, pass );
        /* Requests to bdd will be here */
        System.out.println("DB Connected");
        Utils.displayDepartment(connection);

        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter the empno of the employee you want
                        to move from department? ");
        int empno = sc.nextInt();
        System.out.print("To which new department do you want to move it?
                        ");
        int newDeptno = sc.nextInt();
        sc.close();
        Utils.moveDepartment(connection, empno, newDeptno);

        Utils.displayTable(connection, "emp");

        connection.close();
    } catch ( SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Exercise 1:

```
public static void displayDepartment(Connection connection) throws
SQLException {
    Statement statement = connection.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();
    statement.close();
}
```

```
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:

```
public static void moveDepartment(Connection connection,
                                  int empno,
                                  int newDeptno
) throws SQLException {
    Statement statement = connection.createStatement();

    int affectedEmp = statement.
            executeUpdate( "UPDATE emp SET deptno = " + newDeptno + " WHERE
                    empno = " + empno);

    if (affectedEmp > 0) {
        System.out.println("Update successful, employee with empno " +
                empno + " moves to department " + newDeptno + ".");
    } else {
        System.out.println("Update failed.");
    }

    statement.close();
}
```

```
Please enter the empno of the employee you want to move from department? 7521
To which new department do you want to move it? 20
Update successful, employee with empno 7521 moves to department 20.
```

Exercise 3:

```java
public static void displayTable(Connection connection, String tableName)
throws SQLException {
    Statement statement = connection.createStatement();

    ResultSet result = statement.executeQuery("SELECT * FROM " +
                                              tableName);
    ResultSetMetaData rsmd = result.getMetaData();
    int columnsNumber = rsmd.getColumnCount();


    // Print column names
    for (int i = 1; i <= columnsNumber; i++) {
        System.out.print(rsmd.getColumnName(i).toUpperCase() + " | ");
    }
    System.out.println();

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

    statement.close();
}
```

```
EMPNO | ENAME | EFIRST | JOB | MGR | HIREDATE | SAL | COMM | TEL | DEPTNO |
7566 | JONES | JOHN | MANAGER | 7839 | 1981-04-02 | 2975 | null | 0649545456 | 20 |
7654 | MARTIN | JOE | SALESMAN | 7698 | 1981-09-28 | 1250 | 1400 | 0649545784 | 30 |
7698 | BLAKE | BOB | MANAGER | 7839 | 1981-05-01 | 2850 | null | 0649545254 | 30 |
7782 | CLARK | JOHN | MANAGER | 7839 | 1981-06-09 | 2450 | null | 0649545245 | 10 |
7788 | SCOTT | GUY | ANALYST | 7566 | 1982-12-09 | 3000 | null | 0649545249 | 20 |
7839 | KING | GUY | PRESIDENT | 1 | 1981-11-17 | 5000 | null | 0649545241 | 10 |
7844 | TURNER | PETER | SALESMAN | 7698 | 1981-09-08 | 1500 | 0 | 0649548243 | 30 |
7876 | ADAMS | JOSEPH | CLERK | 7788 | 1983-01-12 | 1100 | null | 0649565243 | 20 |
7900 | JAMES | ALAN | CLERK | 7698 | 1981-12-03 | 950 | null | 0649545564 | 30 |
7902 | FORD | MARIA | ANALYST | 7566 | 1981-12-03 | 3000 | null | 0649785243 | 20 |
7934 | MILLER | ALICE | CLERK | 7782 | 1982-01-23 | 1300 | null | 0699545243 | 10 |
7499 | ALLEN | BOB | SALESMAN | 7698 | 1981-02-20 | 1600 | 300 | 0649547243 | 30 |
7584 | OSTER | null | null | null | null | null | null | 10 |
7369 | SMITH | JOHN | CLERK | 7902 | 1980-12-17 | 7000 | null | 0649545243 | 20 |
7521 | WARD | PETER | SALESMAN | 7698 | 1981-02-22 | 1250 | 500 | 0649545247 | 30 |
```

Exercise 4:

The major consequence of using native SQL methods is security, we risk potential cyber-attacks on our database by SQL code injection by a malicious person. The other disadvantages are that we lose performance and make the code less readable by mixing SQL and Java, at the same time we will have a high chance of having certain syntax and formatting problems such as the manipulation of dates.

Exercise 5:

```
public static void moveDepartment(Connection connection,
                                  int empno,
                                  int newDeptno
) throws SQLException {
    PreparedStatement preparedStatement = connection.prepareStatement(
            "UPDATE emp SET deptno = ? WHERE empno = ?"
    );

    preparedStatement.setInt(1, newDeptno);
    preparedStatement.setInt(2, empno);

    int affectedEmp = preparedStatement.executeUpdate();


    if (affectedEmp > 0) {
        System.out.println("Update successful, employee with empno " +
                           empno + " moves to department " + newDeptno + ".");
    } else {
        System.out.println("Update failed.");
    }

    preparedStatement.close();
}
```

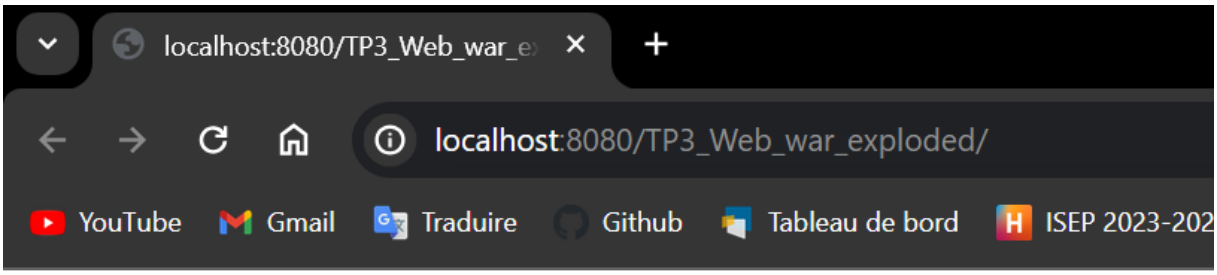
```
Please enter the empno of the employee you want to move from department? 7521
To which new department do you want to move it? 30
Update successful, employee with empno 7521 moves to department 30.
```

Exercise 6:

We cannot configure table or column names with prepare queries. Ultimately, we can do a manual check before executing the query.

Exercise 7:

```
@WebServlet(name = "deptServlet", value = "/dept-servlet")
public class DeptServlet extends HttpServlet {
    private Connection connection;
    @Override
    public void init() {
        try {
            Class.forName( "org.postgresql.Driver" );
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }

        String url = "jdbc:postgresql://localhost:5432/postgres";
        String user = "postgres";
        String pass = "Xinyi.xian2509";
        try {
            this.connection = DriverManager.getConnection(url, user, pass);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();

        out.println("<h1>Display departments in a web browser</h1>");

        try {
            Statement statement = this.connection.createStatement();
            ResultSet result = statement
                                    .executeQuery("SELECT * FROM dept");

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

                out.println("<p>Department " + deptno + " is for "
                        + dname + " and located in " + loc + ".</p>");
            }

            result.close();
            statement.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        out.close();
    }
    @Override
    public void destroy() {
        try {
            this.connection.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```



Display departments in a web browser

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.

Part II: DAO

Exercise 8:

The inconvenience of JDBC is that we have persistence logic that is scattered throughout our business logic making it extremely difficult to maintain over the long term as our project grows we will then have spaghetti code. Thus, Data Access Objects allow on the one hand to have more flexibility and control of the code thanks to the possibility of putting several layers of abstraction and of encapsulation. On the other hand, we centralize persistence logic and separate it from business logic. Additionally, we will write less SQL code and more Java code for data manipulation and thus standardize the language. All of this makes code reuse, modification, and maintenance much easier.

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

    try {
        String url = "jdbc:postgresql://localhost:5432/postgres";
        String user = "postgres";
        String pass = "password";
        Connection connection = DriverManager.getConnection(url, user,
                                                            pass);
        // Requests to bdd will be here
        System.out.println("DB Connected");

        DAOFactory daoFactory = new DAOFactory(connection);
```

```
        DAO<Dept> departmentDao = daoFactory.getDeptDAO();
        Dept dept20 = departmentDao.find(20);
        System.out.println(dept20);
        DAO<Emp> empDAO = daoFactory.getEmpDAO();
        Emp emp = empDAO.find(7369);
        System.out.println(emp);

        connection.close();
    } catch ( SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Exercise 9:

```
public class Dept {
    private int deptno;
    private String dname;
    private String loc;

    public Dept() {}

    public Dept(int deptno, String dname, String loc) {
        this.deptno = deptno;
        this.dname = dname;
        this.loc = loc;
    }

    public int getDeptno() {
        return this.deptno;
    }

    public void setDeptno(int deptno) {
        this.deptno = deptno;
    }

    public String getDname() {
        return this.dname;
    }

    public void setDname(String dname) {
        this.dname = dname;
    }

    public String getLoc() {
        return this.loc;
    }

    public void setLoc(String loc) {
        this.loc = loc;
    }

    @Override
    public String toString() {
        return "Dept{" +
                "\n\tdeptno=" + deptno +
                ",\n\tdname='" + dname + '\'' +
                ",\n\tloc='" + loc + '\'' + "\n}";
    }
}
```

Exercise 10:

```
public abstract class DAO<T> {
    protected Connection connect;
    public DAO(Connection connect) {
        this.connect = connect;
    }
    public abstract T find(int id) throws SQLException;
    public abstract boolean create(T object);
    public abstract boolean update(T object);
    public abstract boolean delete(T object);
}
public class DeptDAO extends DAO<Dept> {
    public DeptDAO(Connection connect) {
        super(connect);
    }

    @Override
    public Dept find(int id) throws SQLException {
        PreparedStatement pstmt = this.connect
                .prepareStatement("SELECT * FROM dept WHERE deptno = ?");
        pstmt.setInt(1, id);
        ResultSet result = pstmt.executeQuery();
        result.next();

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

        return new Dept(deptno, dname, loc);
    }

    @Override
    public boolean create(Dept object) {
        return false;
    }

    @Override
    public boolean update(Dept object) {
        return false;
    }

    @Override
    public boolean delete(Dept object) {
        return false;
    }
}
```

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

Exercise 11:

```
public class Emp {
    private int empno;
    private String ename;
    private String efirst;
    private String job;
    private Emp mgr;
    private Date hiredate;
    private int sal;
    private int comm;
    private String tel;
    private Dept department;

    public Emp() {}

    public Emp(
            int empno,
            String ename,
            String efirst,
            String job,
            Emp mgr,
            Date hiredate,
            int sal,
            int comm,
            String tel,
            Dept department
    ) {
        this.empno = empno;
        this.ename = ename;
        this.efirst = efirst;
        this.job = job;
        this.mgr = mgr;
        this.hiredate = hiredate;
        this.sal = sal;
        this.comm = comm;
        this.tel = tel;
        this.department = department;
    }

    public int getEmpno() {
        return this.empno;
    }

    public void setEmpno(int empno) {
        this.empno = empno;
    }

    public String getEname() {
        return this.ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public String getEfirst() {
        return this.efirst;
    }
```

```
    public void setEfirst(String efirst) {
        this.efirst = efirst;
    }

    public String getJob() {
        return this.job;
    }

    public void setJob(String job) {
        this.job = job;
    }

    public Emp getMgr() {
        return this.mgr;
    }

    public void setMgr(Emp mgr) {
        this.mgr = mgr;
    }

    public Date getHiredate() {
        return this.hiredate;
    }

    public void setHiredate(Date hiredate) {
        this.hiredate = hiredate;
    }

    public int getSal() {
        return this.sal;
    }

    public void setSal(int sal) {
        this.sal = sal;
    }

    public int getComm() {
        return this.comm;
    }

    public void setComm(int comm) {
        this.comm = comm;
    }

    public String getTel() {
        return this.tel;
    }

    public void setTel(String tel) {
        this.tel = tel;
    }

    public Dept getDepartment() {
        return this.department;
    }

    public void setDepartment(Dept department) {
        this.department = department;
    }
```

```
    @Override
    public String toString() {
        return "Emp{" +
                "\n\tempno=" + empno +
                ",\n\tename='" + ename + '\'' +
                ",\n\tefirst='" + efirst + '\'' +
                ",\n\tjob='" + job + '\'' +
                ",\n\tmgr=" + mgr +
                ",\n\thiredate=" + hiredate +
                ",\n\tsal=" + sal +
                ",\n\tcomm=" + comm +
                ",\n\ttel='" + tel + '\'' +
                ",\n\tdepartment=" + department +
                "\n}";
    }
}
```

```
public class EmpDAO extends DAO<Emp> {
    public EmpDAO(Connection connect) {
        super(connect);
    }

    @Override
    public Emp find(int id) throws SQLException {
        PreparedStatement pstmt = this.connect
                .prepareStatement("SELECT * FROM emp WHERE empno = ?");
        pstmt.setInt(1, id);

        ResultSet result = pstmt.executeQuery();
        if(!result.next()) return null;

        int empno = result.getInt("empno");
        String ename = result.getString("ename");
        String efirst = result.getString("efirst");
        String job = result.getString("job");

        Emp mgr = this.find(result.getInt("mgr"));

        Date hiredate = result.getDate("hiredate");
        int sal = result.getInt("sal");
        int comm = result.getInt("comm");
        String tel = result.getString("tel");

        DAO<Dept> departmentDao = new DeptDAO(this.connect);
        Dept department = departmentDao.find(result.getInt("deptno"));

        return new Emp(empno, ename, efirst, job, mgr, hiredate, sal, comm,
                       tel, department);

    }

    @Override
    public boolean create(Emp object) {
        return false;
    }

    @Override
    public boolean update(Emp object) {
        return false;
    }
```

```
    @Override
    public boolean delete(Emp object) {
        return false;
    }
}
```

```
Emp{
    empno=7369,
    ename='SMITH',
    efirst='JOHN',
    job='CLERK',
    mgr=Emp{
    empno=7902,
    ename='FORD',
    efirst='MARIA',
    job='ANALYST',
    mgr=Emp{
    empno=7566,
    ename='JONES',
    efirst='JOHN',
    job='MANAGER',
    mgr=Emp{
    empno=7839,
    ename='KING',
    efirst='GUY',
    job='PRESIDENT',
    mgr=null,
    hiredate=1981-11-17,
    sal=5000,
    comm=0,
    tel='0649545241',
    department=Dept{
    deptno=10,
    dname='ACCOUNTING',
    loc='NEW YORK'
}
},
    hiredate=1981-04-02,
    sal=2975,
    comm=0,
    tel='0649545456',
    department=Dept{
    deptno=20,
    dname='RESEARCH',
    loc='DALLAS'
}
},
    hiredate=1981-12-03,
    sal=3000,
    comm=0,
    tel='0649785243',
    department=Dept{
    deptno=20,
    dname='RESEARCH',
    loc='DALLAS'
}
},
    hiredate=1980-12-17,
    sal=7000,
    comm=0,
    tel='0649545243',
    department=Dept{
    deptno=20,
    dname='RESEARCH',
    loc='DALLAS'
}
}
```

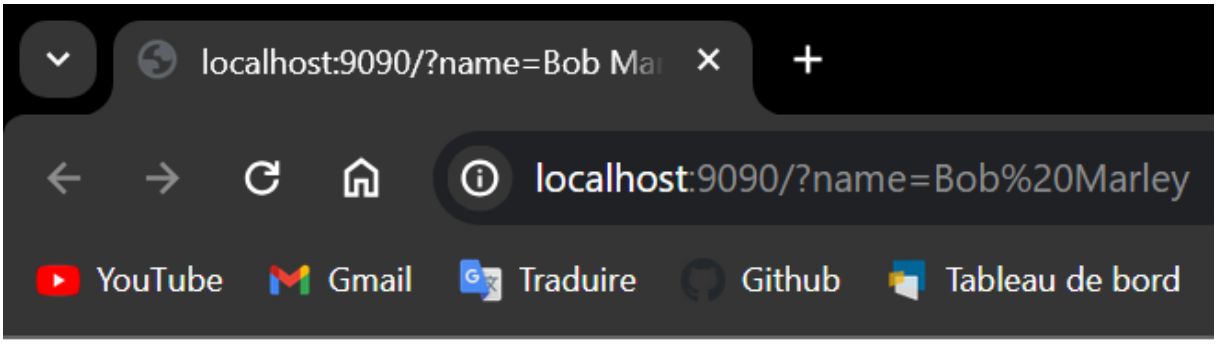
Exercise 12:

```
public class DAOFactory {
    private final Connection connect;

    public DAOFactory(Connection connect) {
        this.connect = connect;
    }

    public DeptDAO getDeptDAO() {
        return new DeptDAO(this.connect);
    }

    public EmpDAO getEmpDAO() {
        return new EmpDAO(this.connect);
    }
}
```

Part III: Spring Boot & JPA

Exercise III.1:

```
@GetMapping(produces = MediaType.TEXT_HTML_VALUE)
public String hello(@RequestParam(value = "name", required = false) String
name) {
    if(name == null) name = "";
    return "<p>Hello " + name + "</p>";
}
```



Hello Bob Marley

Exercise III.2:

```
spring.application.name=TP3-JPA
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.open-in-view=true
```

Exercise III.3:

```
@AllArgsConstructor
@RestController
public class SimpleController {
    private final EmpRepository empRepository;

    @GetMapping(value="/employees", produces =
                MediaType.APPLICATION_JSON_VALUE)
    public List<Emp> getEmployees() {
        return this.empRepository.findAll();
    }
}
```

```
@Repository
public interface EmpRepository extends JpaRepository<Emp, Integer> {
}
```

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "emp")
public class Emp {
    @Id
    private Integer empno;
    @Column(name = "ename")
    private String ename;
    @Column(name = "efirst")
    private String efirst;
    @Column(name = "job")
    private String job;
    @Column(name = "mgr")
    private Integer mgr;
    @Column(name = "hiredate")
    private Date hiredate;
    @Column(name = "sal")
    private Integer sal;
    @Column(name = "comm")
    private Integer comm;
    @Column(name = "tel")
    private String tel;
    @Column(name = "deptno")
    private Integer deptno;
}
```


```
// 20240401205208
// http://localhost:9090/employees

[
  {
    "empno": 7566,
    "ename": "JONES",
    "efirst": "JOHN",
    "job": "MANAGER",
    "mgr": 7839,
    "hiredate": "1981-04-01T22:00:00.000+00:00",
    "sal": 2975,
    "comm": null,
    "tel": "0649545456",
    "deptno": 20
  },
  {
    "empno": 7654,
    "ename": "MARTIN",
    "efirst": "JOE",
    "job": "SALESMAN",
    "mgr": 7698,
    "hiredate": "1981-09-27T23:00:00.000+00:00",
    "sal": 1250,
    "comm": 1400,
    "tel": "0649545784",
    "deptno": 30
  },
  {
    "empno": 7698,
    "ename": "BLAKE",
    "efirst": "BOB",
    "job": "MANAGER",
    "mgr": 7839,
    "hiredate": "1981-04-30T22:00:00.000+00:00",
    "sal": 2850,
    "comm": null,
    "tel": "0649545254",
    "deptno": 30
  },
  {
    "empno": 7782,
    "ename": "CLARK",
    "efirst": "JOHN",
    "job": "MANAGER",
    "mgr": 7839,
    "hiredate": "1981-06-08T22:00:00.000+00:00",
    "sal": 2450,
    "comm": null,
    "tel": "0649545245",
    "deptno": 10
  },
  {
    "empno": 7788,
    "ename": "SCOTT",
    "efirst": "GUY",
    "job": "ANALYST",
    "mgr": 7566,
    "hiredate": "1982-12-08T23:00:00.000+00:00",
    "sal": 3000,
    "comm": null,
    "tel": "0649545249",
    "deptno": 20
  },
  {
    "empno": 7839,
    "ename": "KING",
    "efirst": "GUY",
    "job": "PRESIDENT",
    "mgr": 1,
    "hiredate": "1981-11-16T23:00:00.000+00:00",
    "sal": 5000,
    "comm": null,
    "tel": "0649545241",
    "deptno": 10
  },
  {
    "empno": 7844,
    "ename": "TURNER",
    "efirst": "PETER",
    "job": "SALESMAN",
    "mgr": 7698,
    "hiredate": "1981-09-07T22:00:00.000+00:00",
    "sal": 1500,
    "comm": 0,
    "tel": "0649548243",
    "deptno": 30
  },
  {
    "empno": 7876,
    "ename": "ADAMS",
    "efirst": "JOSEPH",
    "job": "CLERK",
    "mgr": 7788,
    "hiredate": "1983-01-11T23:00:00.000+00:00",
    "sal": 1100,
    "comm": null,
    "tel": "0649565243",
    "deptno": 20
  },
```

```
{
  "empno": 7900,
  "ename": "JAMES",
  "efirst": "ALAN",
  "job": "CLERK",
  "mgr": 7698,
  "hiredate": "1981-12-02T23:00:00.000+00:00",
  "sal": 950,
  "comm": null,
  "tel": "0649545564",
  "deptno": 30
},
{
  "empno": 7902,
  "ename": "FORD",
  "efirst": "MARIA",
  "job": "ANALYST",
  "mgr": 7566,
  "hiredate": "1981-12-02T23:00:00.000+00:00",
  "sal": 3000,
  "comm": null,
  "tel": "0649785243",
  "deptno": 20
},
{
  "empno": 7934,
  "ename": "MILLER",
  "efirst": "ALICE",
  "job": "CLERK",
  "mgr": 7782,
  "hiredate": "1982-01-22T23:00:00.000+00:00",
  "sal": 1300,
  "comm": null,
  "tel": "0699545243",
  "deptno": 10
},
{
  "empno": 7499,
  "ename": "ALLEN",
  "efirst": "BOB",
  "job": "SALESMAN",
  "mgr": 7698,
  "hiredate": "1981-02-19T23:00:00.000+00:00",
  "sal": 1600,
  "comm": 300,
  "tel": "0649547243",
  "deptno": 30
},
{
  "empno": 7584,
  "ename": "OSTER",
  "efirst": null,
  "job": null,
  "mgr": null,
  "hiredate": null,
  "sal": null,
  "comm": null,
  "tel": null,
  "deptno": 10
},
{
  "empno": 7369,
  "ename": "SMITH",
  "efirst": "JOHN",
  "job": "CLERK",
  "mgr": 7902,
  "hiredate": "1980-12-16T23:00:00.000+00:00",
  "sal": 7000,
  "comm": null,
  "tel": "0649545243",
  "deptno": 20
},
{
  "empno": 7521,
  "ename": "WARD",
  "efirst": "PETER",
  "job": "SALESMAN",
  "mgr": 7698,
  "hiredate": "1981-02-21T23:00:00.000+00:00",
  "sal": 1250,
  "comm": 500,
  "tel": "0649545247",
  "deptno": 30
}
]
```

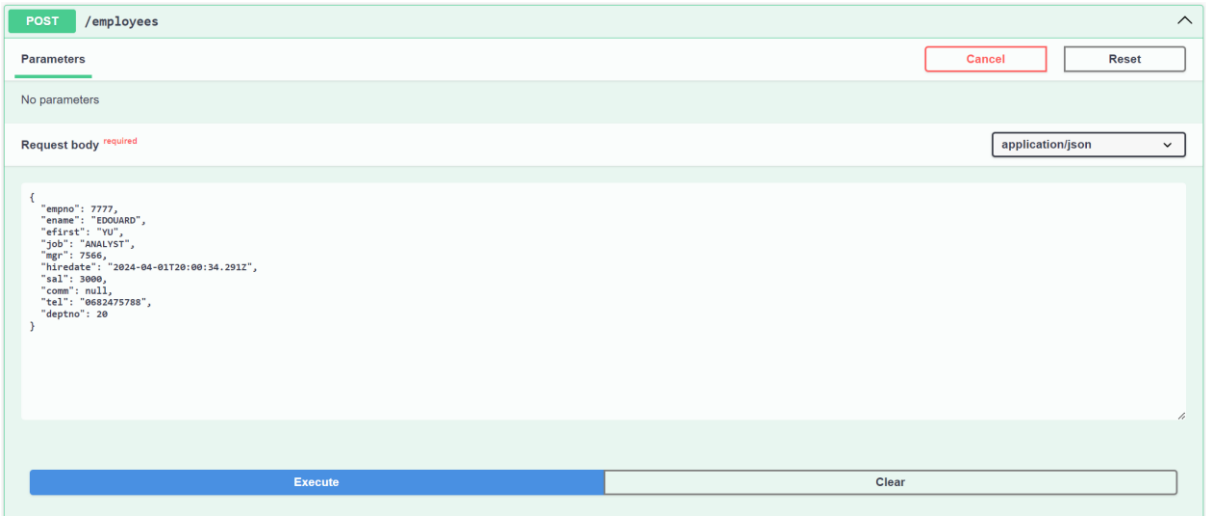

Exercice III.4:

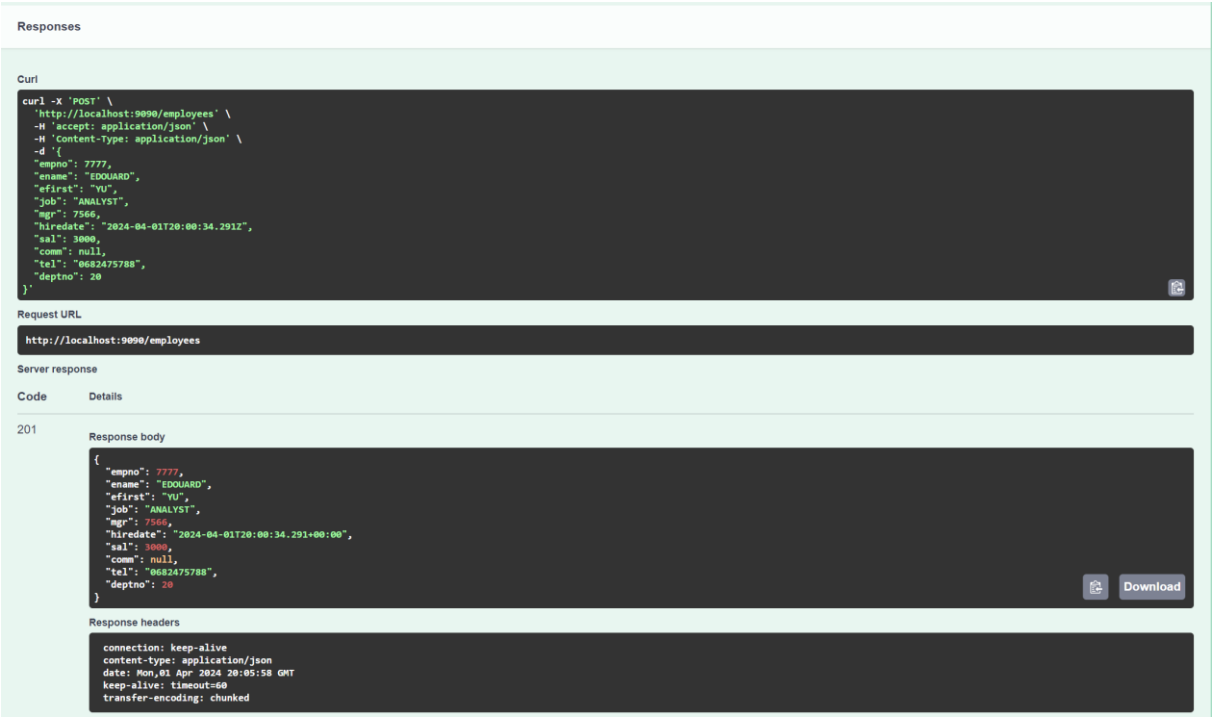
```java
@ResponseStatus(HttpStatus.CREATED)
@PostMapping(value="/employees",
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_VALUE
)
public Emp addEmployee(@RequestBody Emp emp) {
    return this.empRepository.save(emp);
}
```

Since springfox is deprecated on spring boot 3, I instead used pringdoc-openapi-starter-webmvc-ui to build the swagger and api-docs.

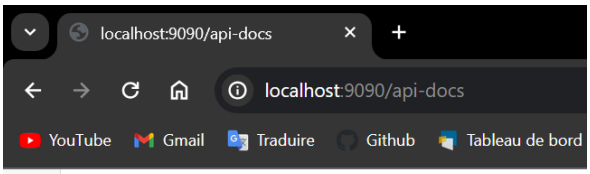
Swagger:

http://localhost:9090/swagger-ui/index.html#/simple-controller/addEmployee





API-Docs:



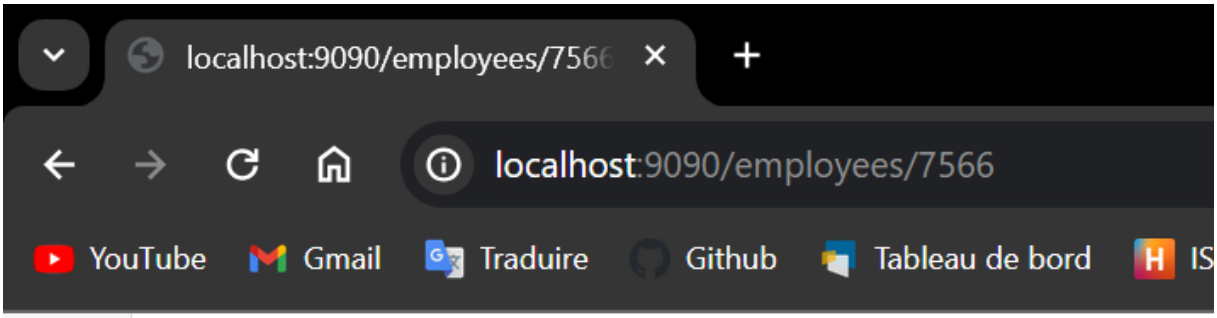

```
// 20240401221142
// http://localhost:9090/api-docs

{
  "openapi": "3.0.1",
  "info": {
    "title": "OpenAPI definition",
    "version": "v0"
  },
  "servers": [
    {
      "url": "http://localhost:9090",
      "description": "Generated server url"
    }
  ],
  "paths": {
    "/employees": {
      "get": {
        "tags": [
          "simple-controller"
        ],
        "operationId": "getEmployees",
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/Emp"
                  }
                }
              }
            }
          }
        }
      },
      "post": {
        "tags": [
          "simple-controller"
        ],
        "operationId": "addEmployee",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Emp"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "Created",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Emp"
                }
              }
            }
          }
        }
      }
    },
    "/": {
      "get": {
        "tags": [
          "simple-controller"
        ],
        "operationId": "hello_1",
        "parameters": [
          {
            "name": "name",
            "in": "query",
            "required": false,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "text/html": {
                "schema": {
                  "type": "string"
                }
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "Emp": {
        "type": "object",
        "properties": {
          "empno": {
            "type": "integer",
            "format": "int32"
          },
          "ename": {
            "type": "string"
          },
          "efirst": {
            "type": "string"
          },
          "job": {
            "type": "string"
          },
          "mgr": {
            "type": "integer",
            "format": "int32"
          },
          "hiredate": {
            "type": "string",
            "format": "date-time"
          },
          "sal": {
            "type": "integer",
            "format": "int32"
          },
          "comm": {
            "type": "integer",
            "format": "int32"
          },
          "tel": {
            "type": "string"
          },
          "deptno": {
            "type": "integer",
            "format": "int32"
          }
        }
      }
    }
  }
}
```

Exercice III.5:

Get by ID:

```
@GetMapping(value="/employees/{empno}", produces =
MediaType.APPLICATION_JSON_VALUE)
public Emp getEmployeeByID(@PathVariable Integer empno) {
    return this.empService.getEmployeeByID(empno);
}
```

```
public Emp getEmployeeByID(Integer empno) {
    return this.empRepository.findById(empno)
            .orElseThrow(() -> new RuntimeException("Unknown employee"));
}
```



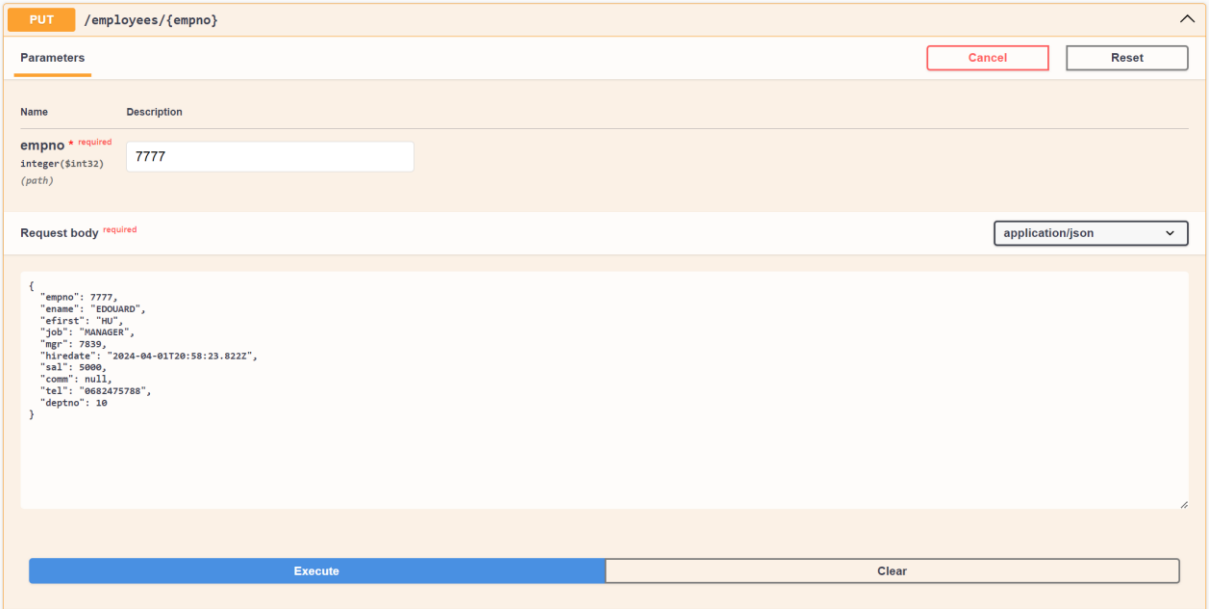

```
// 20240401223542
// http://localhost:9090/employees/7566

{
  "empno": 7566,
  "ename": "JONES",
  "efirst": "JOHN",
  "job": "MANAGER",
  "mgr": 7839,
  "hiredate": "1981-04-01T22:00:00.000+00:00",
  "sal": 2975,
  "comm": null,
  "tel": "0649545456",
  "deptno": 20
}
```

Get All: Exercise III.3

Create: Exercise III.4

Update:

```java
@PutMapping(value="/employees/{empno}",
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_VALUE
)
public Emp updateEmployee(@PathVariable Integer empno, @RequestBody Emp
emp) {
    return this.empService.updateEmployee(empno, emp);
}

public Emp updateEmployee(Integer empno, Emp emp) {
    Emp employee = this.empRepository.findById(empno)
            .orElseThrow(() -> new RuntimeException("Unknown employee"));

    if(employee.getEmpno().equals(emp.getEmpno())) {
        return this.empRepository.save(emp);
    }

    throw new RuntimeException("Empno of the body doesn't correspond to
                               that of the path variable");
}
```



Delete:

```
@ResponseStatus(value = HttpStatus.NO_CONTENT)
@DeleteMapping(path = "/employees/{empno}")
public void deleteEmployee(@PathVariable Integer empno) {
    this.empService.deleteEmployee(empno);
}
```

```
public void deleteEmployee(Integer empno) {
    this.empRepository.deleteById(empno);
}
```