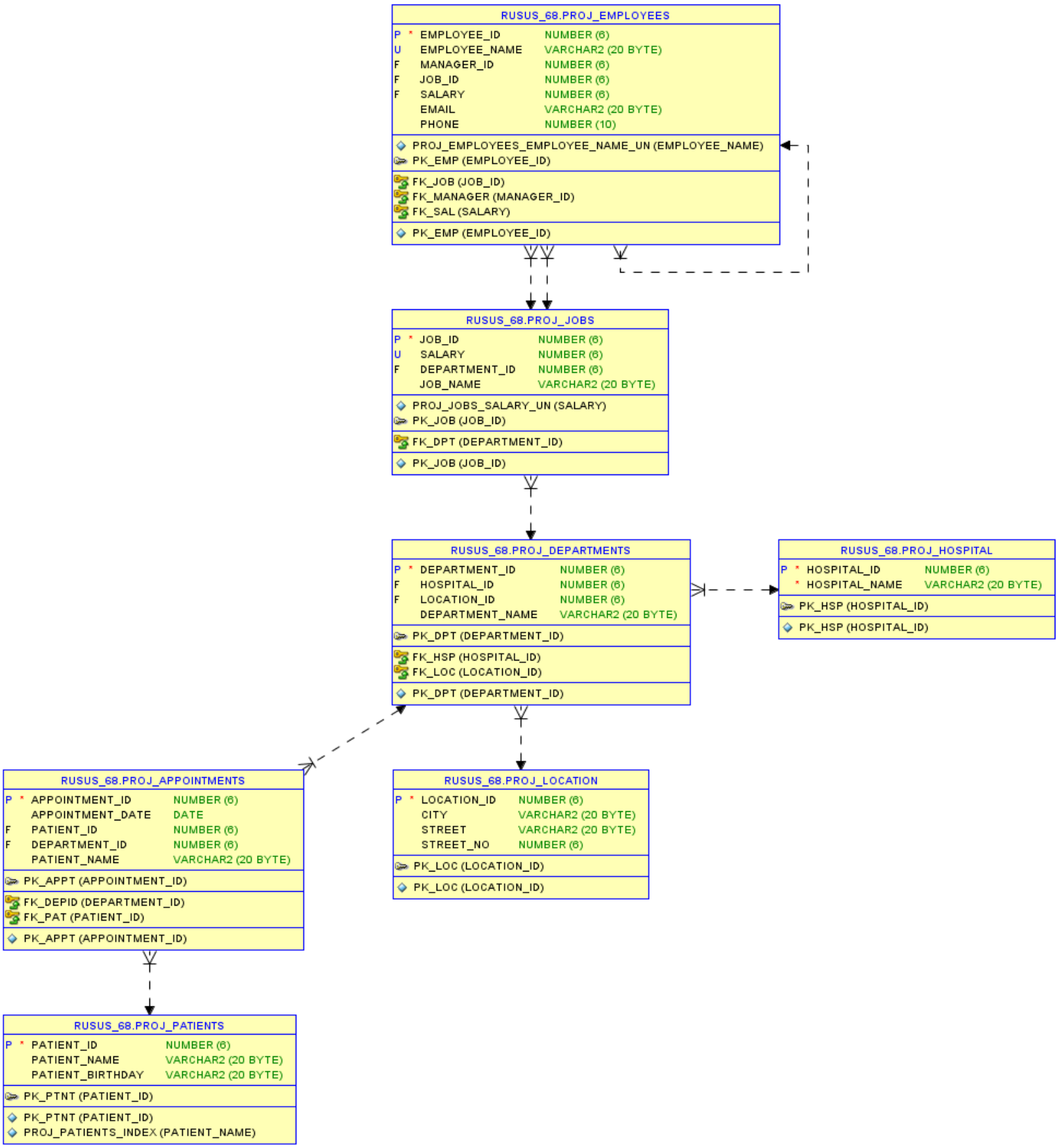

RUSUS_68.PROJ_EMPLOYEES
P * EMPLOYEE_ID NUMBER (6)
U EMPLOYEE_NAME VARCHAR2 (20 BYTE)
F MANAGER_ID NUMBER (6)
F JOB_ID NUMBER (6)
F SALARY NUMBER (6)
EMAIL VARCHAR2 (20 BYTE)
PHONE NUMBER (10)
PROJ_EMPLOYEES_EMPLOYEE_NAME_UN (EMPLOYEE_NAME)
PK_EMP (EMPLOYEE_ID)
FK_JOB (JOB_ID)
FK_MANAGER (MANAGER_ID)
FK_SAL (SALARY)
PK_EMP (EMPLOYEE_ID)
RUSUS_68.PROJ_JOBS
P * JOB_ID NUMBER (6)
U SALARY NUMBER (6)
F DEPARTMENT_ID NUMBER (6)
JOB_NAME VARCHAR2 (20 BYTE)
PROJ_JOBS_SALARY_UN (SALARY)
PK_JOB (JOB_ID)
FK_DPT (DEPARTMENT_ID)
PK_JOB (JOB_ID)
RUSUS_68.PROJ_DEPARTMENTS
P * DEPARTMENT_ID NUMBER (6)
F HOSPITAL_ID NUMBER (6)
F LOCATION_ID NUMBER (6)
DEPARTMENT_NAME VARCHAR2 (20 BYTE)
PK_DPT (DEPARTMENT_ID)
FK_HSP (HOSPITAL_ID)
FK_LOC (LOCATION_ID)
PK_DPT (DEPARTMENT_ID)
RUSUS_68.PROJ_HOSPITAL
P * HOSPITAL_ID NUMBER (6)
* HOSPITAL_NAME VARCHAR2 (20 BYTE)
PK_HSP (HOSPITAL_ID)
PK_HSP (HOSPITAL_ID)
RUSUS_68.PROJ_APPOINTMENTS
P * APPOINTMENT_ID NUMBER (6)
APPOINTMENT_DATE DATE
F PATIENT_ID NUMBER (6)
F DEPARTMENT_ID NUMBER (6)
PATIENT_NAME VARCHAR2 (20 BYTE)
PK_APPT (APPOINTMENT_ID)
FK_DEPID (DEPARTMENT_ID)
FK_PAT (PATIENT_ID)
PK_APPT (APPOINTMENT_ID)
RUSUS_68.PROJ_LOCATION
P * LOCATION_ID NUMBER (6)
CITY VARCHAR2 (20 BYTE)
STREET VARCHAR2 (20 BYTE)
STREET_NO NUMBER (6)
PK_LOC (LOCATION_ID)
PK_LOC (LOCATION_ID)
RUSUS_68.PROJ_PATIENTS
P * PATIENT_ID NUMBER (6)
PATIENT_NAME VARCHAR2 (20 BYTE)
PATIENT_BIRTHDAY VARCHAR2 (20 BYTE)
PK_PTNT (PATIENT_ID)
PK_PTNT (PATIENT_ID)
PROJ_PATIENTS_INDEX (PATIENT_NAME)

Database Project – Healthcare

I. Introduction

In this project, I am creating a database for healthcare institutions that tracks various aspects of hospital operations such as patient appointments, jobs, employees departments and so on. All of these are key elements that are necessary for the successful management of a hospital.

In this project, I am creating a database for healthcare institutions that tracks various aspects of hospital operations. The foundation of this database is the Hospital table, which tracks the location and name of each healthcare institution. This table is crucial as it provides a clear overview of the different hospitals that are part of the system, and it makes it easy to access specific information about a particular hospital.

The Employees table is also an essential part of the database. This table contains information about the individuals working at the hospital, including their names, positions, and salaries. This table is important because it gives a clear picture of the staff working at each hospital and their roles, which is crucial for the effective management of the hospital. The table also includes a hierarchical structure, which makes it easy to identify the managers and their employees.

Another important table is the Departments table, which keeps track of the different departments within the hospital, including their location and department head. This table is essential because it helps to identify the different services offered by the hospital and the staff responsible for each department. It goes hand in hand with the Locations table, which shows the location names, the city, for each department.

Patient appointments are recorded in the Appointments table, which includes the date and patient name. This table is important because it helps to keep track of patient appointments and ensures that they are scheduled in a timely and efficient manner. Additionally, the Patients table stores information about each patient, including their name and birthdate. This table is important because it provides a clear record of the patients that have been seen at the hospital.

For job positions, we have the Jobs table, where you can find data about the different job positions available in the hospitals and the salaries. This table is important because it helps to identify the different job positions available in the hospital and the qualifications and requirements for each job.

In conclusion, all these tables are key elements that are necessary for the successful management of a hospital. They provide a comprehensive view of the different aspects of hospital operations, make it easy to access, and manage the data. This database will help to improve the efficiency and effectiveness of healthcare institutions and provide a better experience for both the employees and patients.

II. Creating the tables

As mentioned before, our first table created was the Hospital table, the main table of our project where we store the hospital id and name. A simple but important table I might say.

```
CREATE TABLE PROJ_Hospital
(
hospital_id NUMBER(6) CONSTRAINT pk_hsp PRIMARY KEY
);
ALTER TABLE PROJ_Hospital ADD
(
hospital_name VARCHAR2(20) NOT NULL
);
```

Script Output | Task completed in 0.175 seconds

```
(
hospital_id NUMBER(6) CONSTRAINT pk_hsp PRIMARY KEY,
)
Error report -
ORA-00904: : invalid identifier
00904. 00000 -  "%s: invalid identifier"
*Cause:
*Action:

Table PROJ_HOSPITAL created.


Table PROJ_HOSPITAL altered.
```

When creating the table, I remembered I wanted to have a column for the hospital name so I used ALTER TABLE in order to do that.

```
);
CREATE TABLE PROJ_Location
(
location_id NUMBER(6) CONSTRAINT pk_loc PRIMARY KEY,
city VARCHAR2(20),
street VARCHAR2(20)
);

ALTER TABLE PROJ_Location ADD
(street_no NUMBER(6));
```

Script Output | Task completed in 0.062 seconds

```
Table PROJ_LOCATION created.


Table PROJ_LOCATION altered.
```

The next table was the location table. It is the table where I store the data about the city, the street and the number where the hospital is situated.

I keep on using alter table to practice and to add new data to my table

The table that goes hand in hand with the Location table is the Department table.

Here is the place where the foreign keys are the location and the hospital ids. I do that to see in which city/hospital each department is found.

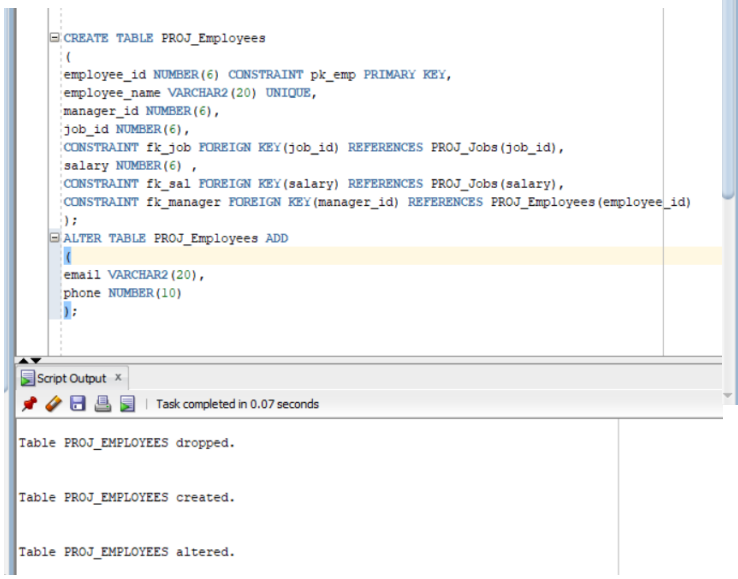
```
CREATE TABLE PROJ_Departments
(
department_id NUMBER(6) CONSTRAINT pk_dpt PRIMARY KEY,
 hospital_id NUMBER(6),
 CONSTRAINT fk_hsp FOREIGN KEY(hospital_id) REFERENCES PROJ_Hospital(hospital_id),
 location_id NUMBER(6),
 CONSTRAINT fk_loc FOREIGN KEY(location_id) REFERENCES PROJ_Location(location_id)
);
ALTER TABLE PROJ_Departments ADD
( departmanet_name VARCHAR2(20));
```

Script Output | Task completed in 0.159 seconds

```
Table PROJ_DEPARTMENTS created.


Table PROJ_DEPARTMENTS altered.
```

The next two tables that also go hand in hand are the Employees and Jobs table, where we can see personal details about the employees, their salaries, their ids etc.



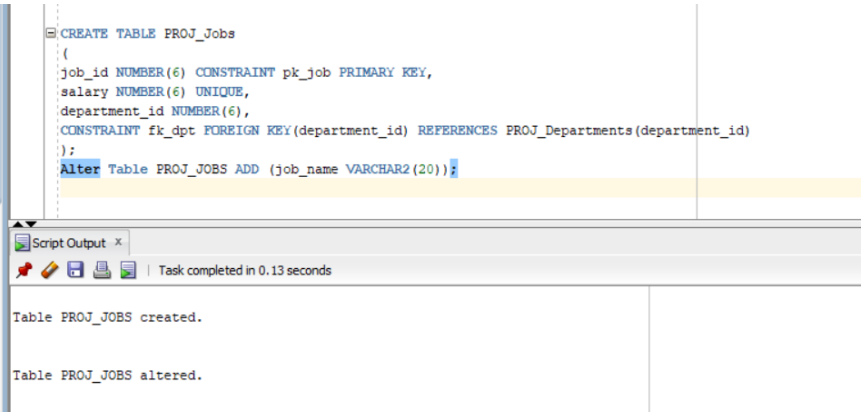
```
CREATE TABLE PROJ_Employees
(
employee_id NUMBER(6) CONSTRAINT pk_emp PRIMARY KEY,
employee_name VARCHAR2(20) UNIQUE,
manager_id NUMBER(6),
job_id NUMBER(6),
CONSTRAINT fk_job FOREIGN KEY(job_id) REFERENCES PROJ_Jobs(job_id),
salary NUMBER(6) ,
CONSTRAINT fk_sal FOREIGN KEY(salary) REFERENCES PROJ_Jobs(salary),
CONSTRAINT fk_manager FOREIGN KEY(manager_id) REFERENCES PROJ_Employees(employee_id)
);
ALTER TABLE PROJ_Employees ADD
(
email VARCHAR2(20),
phone NUMBER(10)
);
```

```
Script Output
Task completed in 0.07 seconds

Table PROJ_EMPLOYEES dropped.

Table PROJ_EMPLOYEES created.

Table PROJ_EMPLOYEES altered.
```



```
CREATE TABLE PROJ_Jobs
(
job_id NUMBER(6) CONSTRAINT pk_job PRIMARY KEY,
salary NUMBER(6) UNIQUE,
department_id NUMBER(6),
CONSTRAINT fk_dpt FOREIGN KEY(department_id) REFERENCES PROJ_Departments(department_id)
);
Alter Table PROJ_JOBS ADD (job_name VARCHAR2(20));
```

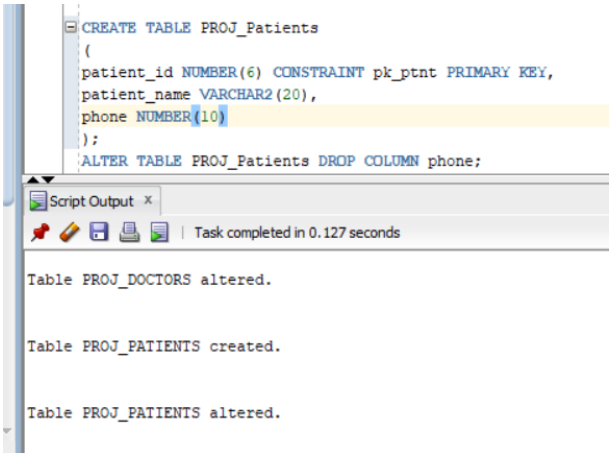
```
Script Output
Task completed in 0.13 seconds

Table PROJ_JOBS created.

Table PROJ_JOBS altered.
```

In here we can also see that the Employee table has a column called manager id for when we will us hierarchical query.

The last two tables are the Patients and Appointments. In these tables, we see the details about the patients and when their appointments are.



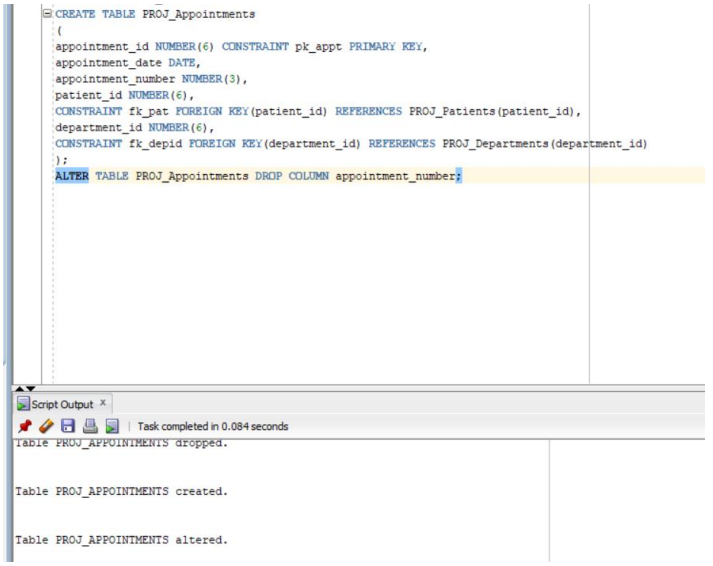
```
CREATE TABLE PROJ_Patients
(
patient_id NUMBER(6) CONSTRAINT pk_ptnt PRIMARY KEY,
patient_name VARCHAR2(20),
phone NUMBER(10)
);
ALTER TABLE PROJ_Patients DROP COLUMN phone;
```

```
Script Output
Task completed in 0.127 seconds

Table PROJ_DOCTORS altered.

Table PROJ_PATIENTS created.

Table PROJ_PATIENTS altered.
```



```
CREATE TABLE PROJ_Appointments
(
appointment_id NUMBER(6) CONSTRAINT pk_appt PRIMARY KEY,
appointment_date DATE,
appointment_number NUMBER(3),
patient_id NUMBER(6),
CONSTRAINT fk_pat FOREIGN KEY(patient_id) REFERENCES PROJ_Patients(patient_id),
department_id NUMBER(6),
CONSTRAINT fk_depid FOREIGN KEY(department_id) REFERENCES PROJ_Departments(department_id)
);
ALTER TABLE PROJ_Appointments DROP COLUMN appointment_number;
```

```
Script Output
Task completed in 0.084 seconds

Table PROJ_APPOINTMENTS dropped.

Table PROJ_APPOINTMENTS created.

Table PROJ_APPOINTMENTS altered.
```

I also added a section where I used DROP to remove all the tables, if someone would like to try the code themselves.

```
DROP TABLE PROJ_Appointments;
DROP TABLE PROJ_Patients;
DROP TABLE PROJ_Employees;
DROP TABLE PROJ_Jobs;
DROP TABLE PROJ_Departments;
DROP TABLE PROJ_Location;
DROP TABLE PROJ_Hospital;
```

III. INSERTING DATA INTO THE TABLES

The next step in creating a database is inserting data into the tables / or the SELECTS. Below I will attach screenshots of every insert I have done:

```
INSERT INTO PROJ_Appointments values(1, '12-JAN-2023', 1 , 1);
INSERT INTO PROJ_Appointments values(2, '21-JAN-2023', 3 , 1);
INSERT INTO PROJ_Appointments values(3, '17-FEB-2023', 4 , 3);
INSERT INTO PROJ_Appointments values(4, '25-MAR-2023', 2 , 2);

SELECT * FROM PROJ_Appointments;
```

Script Output | Query Result 2 | Query Result 3 | Query Result 4

All Rows Fetched: 4 in 0.015 seconds

	APPOINTMENT_ID	APPOINTMENT_DATE	PATIENT_ID	DEPARTMENT_ID
1	1	12-JAN-23	1	1
2	2	21-JAN-23	3	1
3	3	17-FEB-23	4	3
4	4	25-MAR-23	2	2

```
INSERT INTO PROJ_Patients values(1, 'Bogdan DLP');
INSERT INTO PROJ_Patients values(2, 'Mihaita Piticu');
INSERT INTO PROJ_Patients values(3, 'Florin Salam');
INSERT INTO PROJ_Patients values(4, 'Uraganul Tzanca');

SELECT * FROM PROJ_Patients;
```

Script Output | Query Result 2 | Query Result 3 | Query Result 4

All Rows Fetched: 4 in 0.016 seconds

	PATIENT_ID	PATIENT_NAME
1	1	Bogdan DLP
2	2	Mihaita Piticu
3	3	Florin Salam
4	4	Uraganul Tzanca

```
INSERT INTO PROJ_Jobs values(1, 5700, 1, 'Doctor');
INSERT INTO PROJ_Jobs values(2, 4300, 2, 'Asistent');
INSERT INTO PROJ_Jobs values(3, 5000, 3, 'Doctor');
INSERT INTO PROJ_Jobs values(4, 4000, 4, 'Asistent');
INSERT INTO PROJ_Jobs values(5, 5200, 5, 'Doctor');
INSERT INTO PROJ_Jobs values(6, 3900, 6, 'Asistent');
INSERT INTO PROJ_Jobs values(7, 4100, 7, 'Asistent');
INSERT INTO PROJ_Jobs values(8, 6000, 8, 'Doctor');
INSERT INTO PROJ_Jobs values(9, 3800, 9, 'Asistent');

SELECT* FROM PROJ_Jobs ORDER BY job_name;
```

Script Output | Query Result | Query Result 1 | Query Result 2

All Rows Fetched: 9 in 0.017 seconds

	JOB_ID	SALARY	DEPARTMENT_ID	JOB_NAME
1	2	4300	2	Asistent
2	7	4100	7	Asistent
3	6	3900	6	Asistent
4	4	4000	4	Asistent
5	9	3800	9	Asistent
6	8	6000	8	Doctor
7	3	5000	3	Doctor
8	1	5700	1	Doctor
9	5	5200	5	Doctor

```
INSERT INTO PROJ_Departments values(1, 1, 1 , 'Endocrinologie');
INSERT INTO PROJ_Departments values(2, 1, 1 , 'Oftalmologie');
INSERT INTO PROJ_Departments values(3, 1, 1 , 'Medicina dentara');

INSERT INTO PROJ_Departments values(4, 2, 2, 'Cardiologie');
INSERT INTO PROJ_Departments values(5, 2, 2, 'Chirurgie');
INSERT INTO PROJ_Departments values(6, 2, 2, 'Pneumonie');

INSERT INTO PROJ_Departments values(7, 3, 3 , 'Dermatologie');
INSERT INTO PROJ_Departments values(8, 3, 3 , 'Reumatologie');
INSERT INTO PROJ_Departments values(9, 3, 3 , 'Urologie');

SELECT * FROM PROJ_Departments;
```

Script Output | Query Result | Query Result 1 | Query Result 2

All Rows Fetched: 9 in 0.02 seconds

	DEPARTMENT_ID	HOSPITAL_ID	LOCATION_ID	DEPARTMANET_NAME
1	1	1	1	Endocrinologie
2	2	1	1	Oftalmologie
3	3	1	1	Medicina dentara
4	4	2	2	Cardiologie
5	5	2	2	Chirurgie
6	6	2	2	Pneumonie
7	7	3	3	Dermatologie
8	8	3	3	Reumatologie
9	9	3	3	Urologie

```
INSERT INTO PROJ_Employees values(1, 'Rusu Sergiu-Ioan', NULL, 1, 5700, 'rususergiu21@ase.ro', 0722232323);
INSERT INTO PROJ_Employees values(2, 'Licxandru Teodora', NULL, 5, 5200, 'licxthea@ase.ro', 0712332123);
INSERT INTO PROJ_Employees values(3, 'Nistor Stefana', NULL, 8, 6000, 'nistorstefana@ase.ro', 0765757575);
INSERT INTO PROJ_Employees values(4, 'Petrescu Roxana', 2, 4, 4000, 'petrescurox@ase.ro', 0765123456);
INSERT INTO PROJ_Employees values(5, 'Milea Robert',2, 5, 5200, 'mileaboss@ase.ro', 0778995522);
INSERT INTO PROJ_Employees values(6, 'Rata Stefania', 2, 6, 3900, 'ratastef@macmac.ro', 0777888777);
INSERT INTO PROJ_Employees values(7, 'Rosu Ionut', 1, 1, 5700, 'rosuiout@poli.ro', 0732334455);
INSERT INTO PROJ_Employees values(8, 'Onea Maria', 1, 2, 4300, 'mariaonea@upb.ro', 0765655656);
INSERT INTO PROJ_Employees values(9, 'Tanase Theodor',1, 3, 5000, 'theodortns@upb.ro', 0733445566);
INSERT INTO PROJ_Employees values(10, 'Popa Madalin',3,7,4100,'popamada@ase.ro',0733441122);
INSERT INTO PROJ_Employees values(11, 'Nistor Andrei', 3, 8, 6000, 'andrei_nist@uni.ro', 0745055932);
INSERT INTO PROJ_Employees values(12, 'Mihalovici Tudor',3,9,3800, 'mihalovicit@ase.ro', 0745010622);
SELECT * FROM PROJ_Employees order by MANAGER_ID;
```

Script Output | Query Result | Query Result 1

All Rows Fetched: 12 in 0.015 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	MANAGER_ID	JOB_ID	SALARY	EMAIL	PHONE
1	8	Onea Maria	1	2	4300	mariaonea@upb.ro	765655656
2	9	Tanase Theodor	1	3	5000	theodortns@upb.ro	733445566
3	7	Rosu Ionut	1	1	5700	rosuiout@poli.ro	732334455
4	5	Milea Robert	2	5	5200	mileaboss@ase.ro	778995522
5	4	Petrescu Roxana	2	4	4000	petrescurox@ase.ro	765123456
6	6	Rata Stefania	2	6	3900	ratastef@macmac.ro	777888777
7	10	Popa Madalin	3	7	4100	popamada@ase.ro	733441122
8	11	Nistor Andrei	3	8	6000	andrei_nist@uni.ro	745055932
9	12	Mihalovici Tudor	3	9	3800	mihalovicit@ase.ro	745010622
10	3	Nistor Stefana	(null)	8	6000	nistorstefana@ase.ro	765757575
11	2	Licxandru Teodora	(null)	5	5200	licxthea@ase.ro	712332123
12	1	Rusu Sergiu-Ioan	(null)	1	5700	rususergiu21@ase.ro	722232323

```
SELECT * FROM PROJ_Location;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.018 seconds

	LOCATION_ID	CITY	STREET	STREET_NO
1	1	Constanta	Str. Sanatatii	28
2	2	Bucuresti	Str. Vietii	12
3	3	Valcea	Soseaua Valcea	1

```
SELECT * FROM PROJ_Hospital;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.089 seconds

	HOSPITAL_ID	HOSPITAL_NAME
1	1	Spitalul Judetean
2	2	Spitalul Municipal
3	3	Spitalul Sf. Ana

Here I also inserted myself as one of the employees in the table:

```
INSERT INTO PROJ_Employees values(1, 'Rusu Sergiu-Ioan', NULL, 8, 6000, 'rususergiu2lase.ro', 0722232323);
SELECT * FROM PROJ_Employees;
```

Script Output | Query Result | Query Result 2

Task completed in 0.046 seconds

```
1 row inserted.

Error starting at line : 27 in command -
INSERT INTO PROJ_Employees values(1, 'Rusu Sergiu-Ioan', NULL, 8, 6000, 'rususergiu21@stud.ase.ro', 0722232323)
r column "RUSUS_68"."PROJ_EMPLOYEES"."EMAIL" (actual: 24, maximum: 20)
```

```
INSERT INTO PROJ_Employees values(1, 'Rusu Sergiu-Ioan', NULL, 8, 6000, 'rususergiu2lase.ro', 0722232323);
SELECT * FROM PROJ_Employees;
```

Script Output | Query Result | Query Result 2

SQL | All Rows Fetched: 1 in 0.023 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	MANAGER_ID	JOB_ID	SALARY	EMAIL	PHONE
1	1	Rusu Sergiu-Ioan	(null)	8	6000	rususergiu2lase.ro	722232323

Next, we "played" with the data, updating, merging or deleting some data:

```
UPDATE PROJ_Patients
set patient_name = 'Bogdan Ploiesti' where patient_id = 1;

INSERT INTO PROJ_Appointments values();
```

Script Output | Query Result 2 | Query Result 3 | Query Result 4

Task completed in 0.045 seconds

```
*Action:

1 row updated.
```

```
INSERT INTO PROJ_Hospital values(1, 'Spitalul Judetean');
INSERT INTO PROJ_Hospital values(2, 'Spitalul Municipal');
INSERT INTO PROJ_Hospital values(3, 'Spitalul Sf. Ana');
INSERT INTO PROJ_Hospital values(4, 'Spitalul de nebuni');

DELETE FROM PROJ_Hospital where hospital_id = 4;
```

Script Output

Task completed in 0.093 seconds

```
1 row inserted.

1 row inserted.

1 row inserted.

1 row deleted.
```

```
--We added the patient name and merged with the values in patient
ALTER TABLE PROJ_Appointments ADD (patient_name VARCHAR2(20));
MERGE INTO PROJ_Appointments a USING PROJ_Patients p
on(p.patient_id = a.patient_id)
WHEN MATCHED THEN UPDATE SET a.patient_name = p.patient_name;
SELECT * FROM PROJ_Appointments;
```

Script Output | Query Result

Task completed in 0.052 seconds

```
Error report -
SQL Error: ORA-38104: Columns referenced in the ON Clause cannot be up
38104. 00000 -  "Columns referenced in the ON Clause cannot be updated
*Cause:    LHS of UPDATE SET contains the columns referenced in the ON
*Action:

Table PROJ_APPOINTMENTS altered.


4 rows merged.
```

IV. SELECTS

The last part of the project is the SELECT functions where I used some of the most important tools SQL Developer has to offer in order to take some important data from my database. I started with some basic ones, working my way up to more complicated things:

1. First I used <= and != to see the salaries <= than the salary of employee with id = 8.

```
--Employees that have a salary <= than employee_id 8, except him
SELECT * from PROJ_Employees
where salary <= (Select salary from PROJ_Employees where employee_id = 8) and employee_id != 8;

--Show all the information about the departments in Constanta(id 1) and Valcea(id 3)
SELECT * from PROJ_Departments
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6 | Script Output | Query Result 7

SQL | All Rows Fetched: 4 in 0.046 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	MANAGER_ID	JOB_ID	SALARY	EMAIL	PHONE
1	4	Petrescu Roxana	2	4	4000	petrescurox@ase.ro	765123456
2	6	Rata Stefania	2	6	3900	ratastef@macmac.ro	777888777
3	10	Popa Madalin	3	7	4100	popamada@ase.ro	733441122
4	12	Mihalovici Tudor	3	9	3800	mihalovicit@ase.ro	745010622

2. Next, almost the same thing, I used > to see the employees with a salary higher than 5500.

```
--a select that shows us the salaries > 5500

SELECT employee_name, salary from PROJ_Employees
where salary > 5500 order by employee_name;
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4

SQL | All Rows Fetched: 4 in 0.081 seconds

	EMPLOYEE_NAME	SALARY
1	Nistor Andrei	6000
2	Nistor Stefana	6000
3	Rosu Ionut	5700
4	Rusu Sergiu-Ioan	5700

```
--The salary of the employees with "Nistor" in their name

SELECT employee_name, salary from PROJ_Employees
where employee_name like 'Nistor%';

--Display the managers
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query R

SQL | All Rows Fetched: 2 in 0.063 seconds

	EMPLOYEE_NAME	SALARY
1	Nistor Andrei	6000
2	Nistor Stefana	6000

3. Here I used LIKE to see the salaries of the employees that have “Nistor” in their names.

4. Next I use IS NULL to see the managers in the employees table.

```
SELECT employee_name as Managers from PROJ_Employees
where manager_id is null;
```

Query Result

SQL | All Rows Fetched: 3 in 0.111 seconds

	MANAGERS
1	Rusu Sergiu-Ioan
2	Licxandru Teodora
3	Nistor Stefana

5. Here I used BETWEEN:

```
--Select that shows all the appointments in january
SELECT * from PROJ_Appointments
where appointment_date between '01-jan-2023' and '30-jan-2023';
```

Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5

SQL | All Rows Fetched: 2 in 0.056 seconds

	APPOINTMENT_ID	APPOINTMENT_DATE	PATIENT_ID	DEPARTMENT_ID
1	1	12-JAN-23	1	1
2	2	21-JAN-23	3	1

6. The next things I used are ALL, ANY and IN.

```
--Show the info about the appointments of the patients that have an id diffrent than 1
SELECT * FROM PROJ_Appointments
where appointment_id = ANY(SELECT patient_id from PROJ_Patients where patient_id != 1);

--How many employees does manacher 1 have
SELECT Count(employee_id) as number_of_emp, manager_id
FROM PROJ_Employees
group by manager_id
```

Query Result | Query Result 1

SQL | All Rows Fetched: 3 in 0.034 seconds

	APPOINTMENT_ID	APPOINTMENT_DATE	PATIENT_ID	DEPARTMENT_ID
1	2	21-JAN-23	3	1
2	3	17-FEB-23	4	3
3	4	25-MAR-23	2	2

```
--Show the info of the departments where the hospital_id is equal to the 2nd location id
SELECT * FROM PROJ_Departments
where hospital_id =  ALL (SELECT location_id from PROJ_Departments where location_id = 2);

--Show the info about the appointments of the patients that have an id diffrent than 1
SELECT * FROM PROJ_Appointments
where appointment_id = ANY(SELECT patient_id from PROJ_Patients where patient_id != 1);
```

Query Result

SQL | All Rows Fetched: 3 in 0.046 seconds

	DEPARTMENT_ID	HOSPITAL_ID	LOCATION_ID	DEPARTMENT_NAME
1	4	2	2	Cardiologie
2	5	2	2	Chirurgie
3	6	2	2	Pneumonie

```
--Show all the information about the departments in Constanta(id 1) and Valcea(id 3)
SELECT * from PROJ_Departments
where location_id IN (1 , 3);
```

Query Result 11

SQL | All Rows Fetched: 6 in 0.017 seconds

	DEPARTMENT_ID	HOSPITAL_ID	LOCATION_ID	DEPARTMENT_NAME
1	1	1	1	Endocrinologie
2	2	1	1	Oftalmologie
3	3	1	1	Medicina dentara
4	7	3	3	Dermatologie
5	8	3	3	Reumatologie
6	9	3	3	Urologie

7. Next, I used to_date and to_char, but in order to use to date, I added a column with the patient birthdate that was written somewhat ambiguous.

```
ALTER TABLE PROJ_Patients ADD (
patient_birthday VARCHAR2(20)
);

UPDATE PROJ_Patients set patient_birthday = '120102' where patient_id = 1;
UPDATE PROJ_Patients set patient_birthday = '111100' where patient_id = 2;
UPDATE PROJ_Patients set patient_birthday = '080401' where patient_id = 3;
UPDATE PROJ_Patients set patient_birthday = '160202' where patient_id = 4;
SELECT * FROM PROJ_Patients;
```

Query Result | Query Result 1 | All Rows Fetched: 4 in 0.011 seconds

	PATIENT_NAME	PATIENT_BIRTHDAY
1	Bogdan Ploiesti	120102
2	Mihaita Piticu	111100
3	Florin Salam	080401
4	Uraganul Tzanca	160202

```
SELECT patient_id, patient_name, TO_DATE(patient_birthday, 'DDMMYY') FROM PROJ_Patients;
```

Script Output | Query Result | Query Result 1 | SQL | All Rows Fetched: 4 in 0.017 seconds

	PATIENT_ID	PATIENT_NAME	TO_DATE(PATIENT_BIRTHDAY,'DDMMYY')
1	1	Bogdan Ploiesti	12-JAN-02
2	2	Mihaita Piticu	11-NOV-00
3	3	Florin Salam	08-APR-01
4	4	Uraganul Tzanca	16-FEB-02

```
--WE want to see the month of the patient name Florin Salam
SELECT patient_name, TO_CHAR(appointment_date, 'Month') as MONTH
from PROJ_Appointments
where patient_name = 'Florin Salam';
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.018 seconds

	PATIENT_NAME	MONTH
1	Florin Salam	January

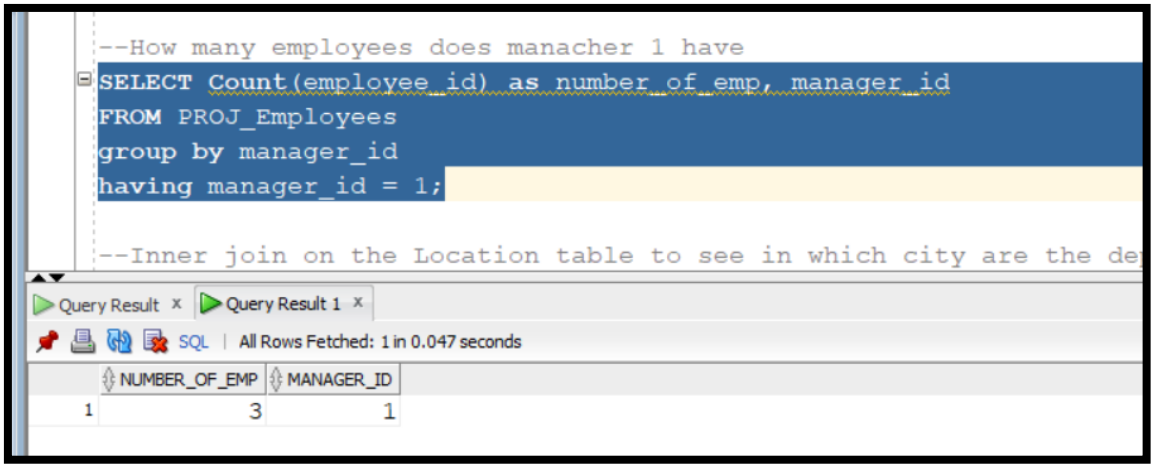
8. Here, I have an example of correlated subquery where I also used AVG function.

```
--Corealted subquery salaries > avg salaries
SELECT employee_name, salary from PROJ_Employees
where salary > (Select AVG(salary)
from Proj_Employees);
```

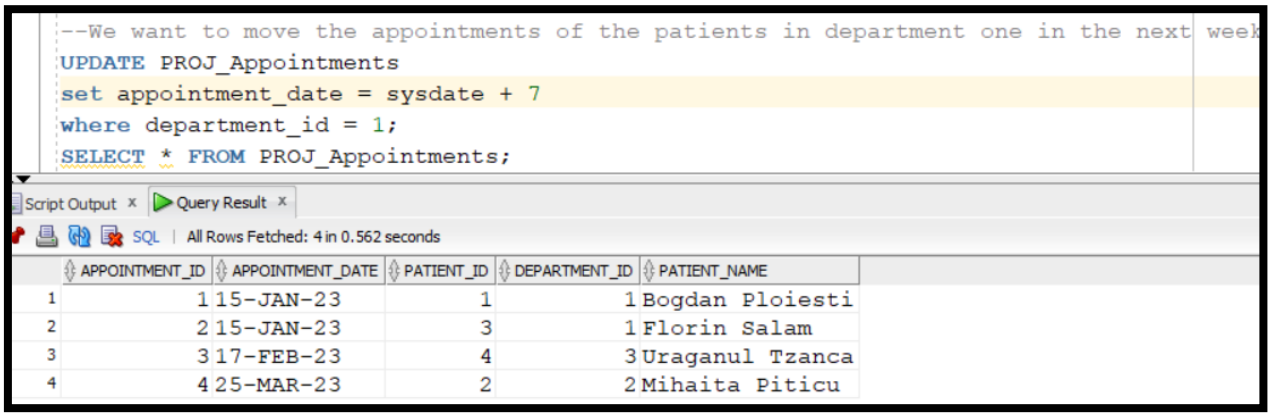
Query Result | Query Result 1 | SQL | All Rows Fetched: 7 in 0.075 seconds

	EMPLOYEE_NAME	SALARY
1	Rusu Sergiu-Ioan	5700
2	Licxandru Teodora	5200
3	Nistor Stefana	6000
4	Milea Robert	5200
5	Rosu Ionut	5700
6	Tanase Theodor	5000
7	Nistor Andrei	6000

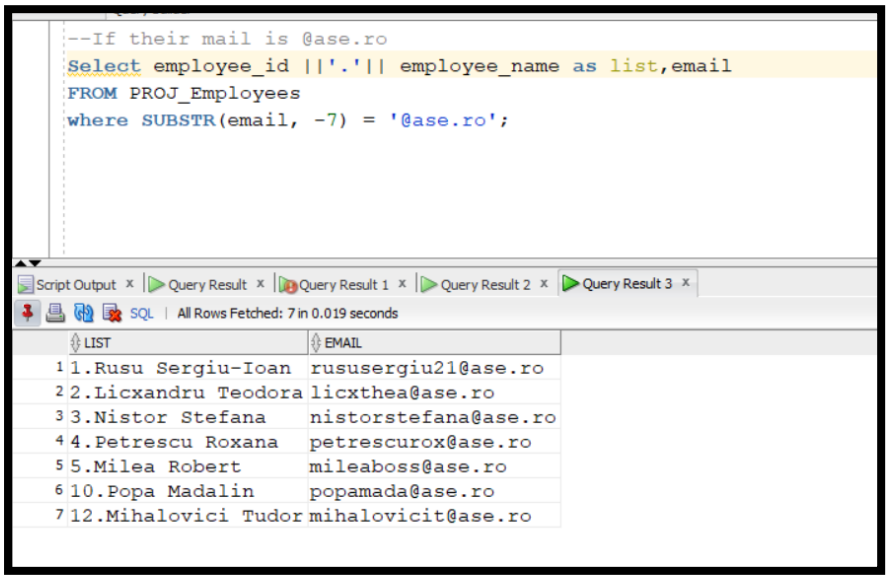
9. Here, I used having, group by and COUNT in order to count the number of subordinates the manager 1 has:



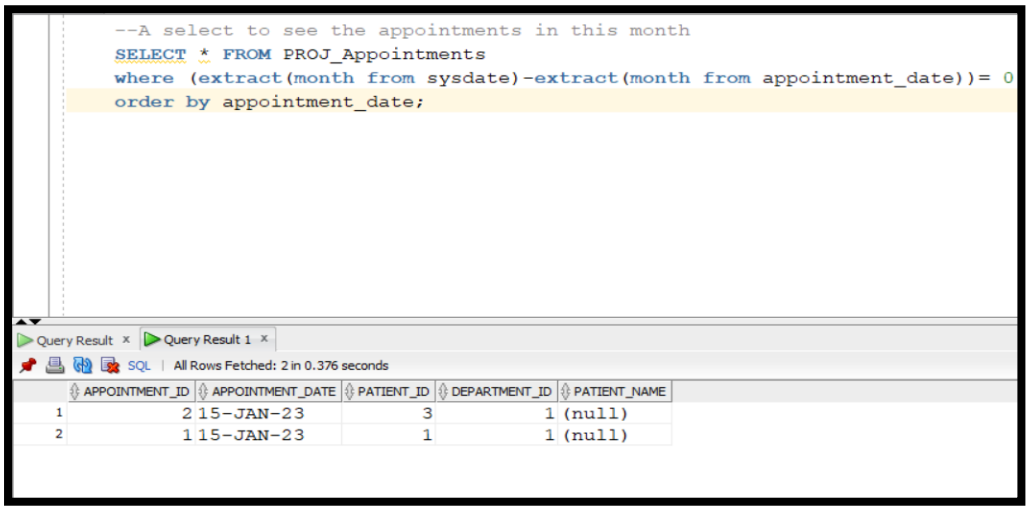
10. In the next exercise, I used sysdate and I modified the appointments of the employees that were supposed to be treated in the department 1 to be next week.



11. I used the subtract function to see the employees that have an email address like @ase.ro.



12. After that, I used the extract function to see the appointments that are in this month.



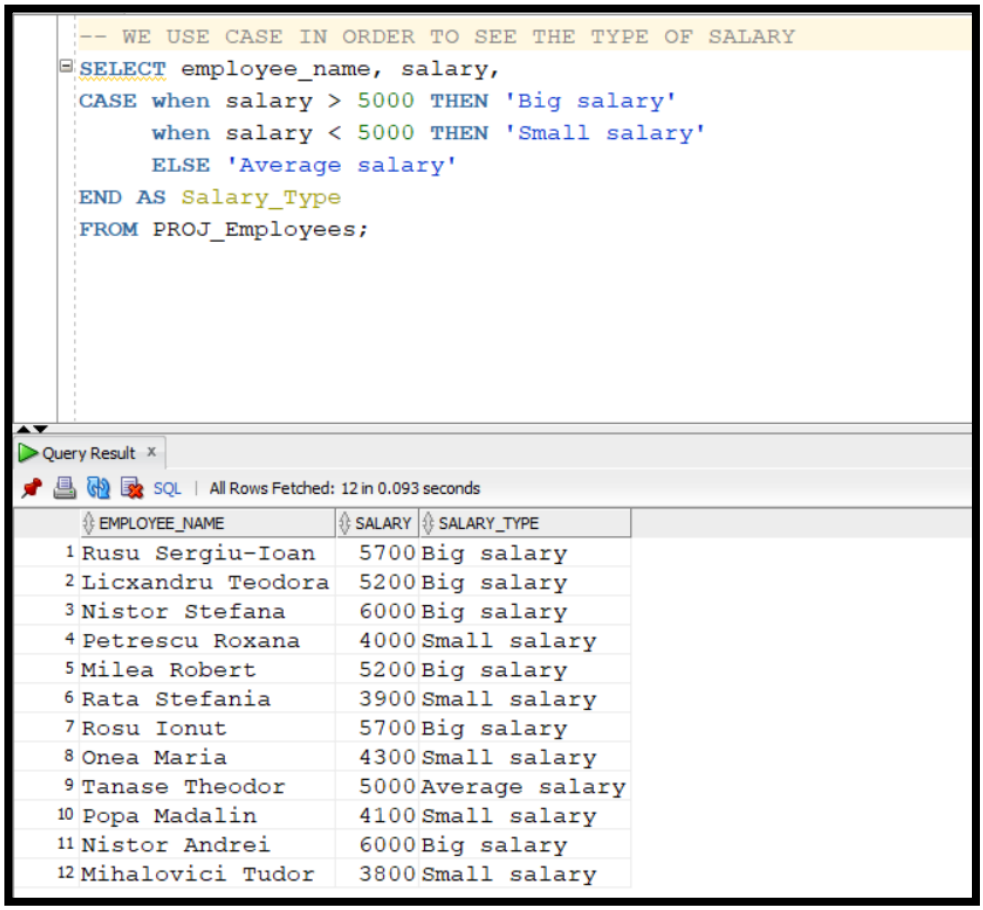
```
--A select to see the appointments in this month
SELECT * FROM PROJ_Appointments
where (extract(month from sysdate)-extract(month from appointment_date))= 0
order by appointment_date;
```

Query Result | Query Result 1

SQL | All Rows Fetched: 2 in 0.376 seconds

	APPOINTMENT_ID	APPOINTMENT_DATE	PATIENT_ID	DEPARTMENT_ID	PATIENT_NAME
1	2	15-JAN-23	3	1	(null)
2	1	15-JAN-23	1	1	(null)

13. Next, I used the CASE function to compare the salaries of all employees.



```
-- WE USE CASE IN ORDER TO SEE THE TYPE OF SALARY
SELECT employee_name, salary,
CASE when salary > 5000 THEN 'Big salary'
     when salary < 5000 THEN 'Small salary'
     ELSE 'Average salary'
END AS Salary_Type
FROM PROJ_Employees;
```

Query Result

SQL | All Rows Fetched: 12 in 0.093 seconds

	EMPLOYEE_NAME	SALARY	SALARY_TYPE
1	Rusu Sergiu-Ioan	5700	Big salary
2	Licxandru Teodora	5200	Big salary
3	Nistor Stefana	6000	Big salary
4	Petrescu Roxana	4000	Small salary
5	Milea Robert	5200	Big salary
6	Rata Stefania	3900	Small salary
7	Rosu Ionut	5700	Big salary
8	Onea Maria	4300	Small salary
9	Tanase Theodor	5000	Average salary
10	Popa Madalin	4100	Small salary
11	Nistor Andrei	6000	Big salary
12	Mihalovici Tudor	3800	Small salary

14. Similar to the CASE function I used decode to see the employees and the managers where the salary is over 5000.

```
-- We wanted to see if the people with a big salary, over 5000, are managers or employees

SELECT employee_id, employee_name,salary,
DECODE(nvl(manager_id,0), 0, 'Manager', 'Employee') as Manager_Or_Employee
FROM PROJ_Employees
where salary > 5000;
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3

All Rows Fetched: 6 in 0.02 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	MANAGER_OR_EMPLOYEE
1	1	Rusu Sergiu-Ioan	5700	Manager
2	2	Licxandru Teodora	5200	Manager
3	3	Nistor Stefana	6000	Manager
4	5	Milea Robert	5200	Employee
5	7	Rosu Ionut	5700	Employee
6	11	Nistor Andrei	6000	Employee

15. Next, we have a left outer join that shows us the patient name and the appointment date based on their id.

```
--Outter join that shows the names of the patients and each date based on the ids
SELECT p.patient_name, a.appointment_date
FROM PROJ_Patients p
left JOIN PROJ_Appointments a ON p.patient_id=a.patient_id;
```

Query Result | Query Result 1

All Rows Fetched: 4 in 0.056 seconds

	PATIENT_NAME	APPOINTMENT_DATE
1	Bogdan Ploiesti	12-JAN-23
2	Mihaita Piticu	25-MAR-23
3	Florin Salam	21-JAN-23
4	Uraganul Tzanca	17-FEB-23

16. Continuing with the joins, we have an inner join where we listed the department names and the cities where each department is situated.

```
SELECT d.department_name, l.city
FROM PROJ_Departments d
Inner join PROJ_Location l on d.location_id=l.location_id;
```

Query Result | Query Result 1

All Rows Fetched: 9 in 0.102 seconds

	DEPARTMENT_NAME	CITY
1	Endocrinologie	Constanta
2	Oftalmologie	Constanta
3	Medicina dentara	Constanta
4	Cardiologie	Bucuresti
5	Chirurgie	Bucuresti
6	Pneumonie	Bucuresti
7	Dermatologie	Valcea
8	Reumatologie	Valcea
9	Urologie	Valcea

17. Here, I used a left and a right joins with intersect:

```
--we used interset to show how we can make ourselves an inner join with left and right joins
SELECT e.employee_name, e.salary, j.department_id
from PROJ_Employees e
LEFT JOIN  PROJ_Jobs j on e.job_id=j.job_id
where department_id = 1
INTERSECT
SELECT e.employee_name, e.salary, j.department_id
from PROJ_Employees e
RIGHT JOIN  PROJ_Jobs j on e.job_id=j.job_id
where department_id = 1;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5 | Query Result 6 | Query Result 7

SQL | All Rows Fetched: 2 in 0.024 seconds

	EMPLOYEE_NAME	SALARY	DEPARTMENT_ID
1	Rosu Ionut	5700	1
2	Rusu Sergiu-Ioan	5700	1

18. Next, I used an UNION where I gave a bonus to the employees based on their managers.

```
--Manager 1 gives a 15% bonues , manager 2 a 10% bonus and manager 3 does not give a bonus
SELECT employee_id, employee_name,manager_id, salary*0.15 as Bonus, salary *1.15 as New_salary
FROM PROJ_EMployees
where manager_id = 1
UNION
SELECT employee_id, employee_name,manager_id, salary*0.1 as Bonus, salary *1.1 as New_salary
FROM PROJ_EMployees
where manager_id = 2
UNION
SELECT employee_id, employee_name,manager_id, salary*0 as Bonus, salary  as New_salary
FROM PROJ_EMployees
where manager_id = 3
order by manager_id;
```

Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 9 in 0.021 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	MANAGER_ID	BONUS	NEW_SALARY
1	7	Rosu Ionut	1	855	6555
2	8	Onea Maria	1	645	4945
3	9	Tanase Theodor	1	750	5750
4	4	Petrescu Roxana	2	400	4400
5	5	Milea Robert	2	520	5720
6	6	Rata Stefania	2	390	4290
7	10	Popa Madalin	3	0	4100
8	11	Nistor Andrei	3	0	6000
9	12	Mihalovici Tudor	3	0	3800

19. Next I used MINUS to decrease the salary of the assistants by 10%, due to the increased of the salaries that I did previously, but not for the ones where their manager is 3rd, because she did not gave any raise.

```
--We want to see the employees that are asistants and lower their salary by 10%
--except the asistant that work for manager 3
select  e.employee_name,e.manager_id, j.job_name, e.salary*0.9 as New_Salary
FROM PROJ_Employees e
INNER JOIN PROJ_Jobs j on j.job_id=e.job_id
where j.job_name = 'Asistent'
MINUS
select e.employee_name,e.manager_id, j.job_name, e.salary*0.9 as New_Salary
FROM PROJ_Employees e
INNER JOIN PROJ_Jobs j on j.job_id=e.job_id
where e.manager_id = 3;
```

Query Result | Script Output | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.02 seconds

	EMPLOYEE_NAME	MANAGER_ID	JOB_NAME	NEW_SALARY
1	Onea Maria	1	Asistent	3870
2	Petrescu Roxana	2	Asistent	3600
3	Rata Stefania	2	Asistent	3510

20. Here, I made a hierarchical subquery to enlist the employees, their managers and the number of subordinates.

```
--Since we only have 2 ranks, manager and employee, i wanted to see a list order by their level
--of authority ordering it by the manager_id
SELECT employee_name, CONNECT_BY_ROOT employee_name ManagerName,
level -1 Number_of_Superiors, SYS_CONNECT_BY_PATH(employee_name, '*') ManagerName
FROM PROJ_Employees
CONNECT BY PRIOR employee_id=manager_id
Start with NVL(manager_id,0) =0;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query R...

All Rows Fetched: 12 in 0.028 seconds

	EMPLOYEE_NAME	MANAGERNAME	NUMBER_OF_SUPERIORS	MANAGERNAME_1
1	Rusu Sergiu-Ioan	Rusu Sergiu-Ioan	0	*Rusu Sergiu-Ioan
2	Rosu Ionut	Rusu Sergiu-Ioan	1	*Rusu Sergiu-Ioan*Rosu Ionut
3	Onea Maria	Rusu Sergiu-Ioan	1	*Rusu Sergiu-Ioan*Onea Maria
4	Tanase Theodor	Rusu Sergiu-Ioan	1	*Rusu Sergiu-Ioan*Tanase Theodor
5	Licxandru Teodora	Licxandru Teodora	0	*Licxandru Teodora
6	Petrescu Roxana	Licxandru Teodora	1	*Licxandru Teodora*Petrescu Roxana
7	Milea Robert	Licxandru Teodora	1	*Licxandru Teodora*Milea Robert
8	Rata Stefania	Licxandru Teodora	1	*Licxandru Teodora*Rata Stefania
9	Nistor Stefana	Nistor Stefana	0	*Nistor Stefana
10	Popa Madalin	Nistor Stefana	1	*Nistor Stefana*Popa Madalin
11	Nistor Andrei	Nistor Stefana	1	*Nistor Stefana*Nistor Andrei
12	Mihalovici Tudor	Nistor Stefana	1	*Nistor Stefana*Mihalovici Tudor

21. Next, I created a synonym, an index and a sequence to help manage the data easier:

```
--A sequence to count our number of patients, starting with 5, because we already
CREATE Sequence PROJ_COUNT_PAITENTS
START WITH 5
increment by 1;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5

Task completed in 0.046 seconds

```
Error starting at line : 7 in command -
MINUS
Error report -
Unknown Command

>>Query Run In:Query Result 1

Sequence PROJ_COUNT_PAITENTS created.
```

```
CREATE SYNONYM PROJ_Patients_No for PROJ_COUNT_Patients;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4

Task completed in 0.064 seconds

```
Error report -
Unknown Command
```

```
CREATE INDEX PROJ_Patients_Index on PROJ_Patients(patient_name);
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5

Task completed in 0.086 seconds

```
*Cause:
*Action:

Index PROJ_PATIENTS_INDEX created.
```

22. Last, I created a view with all the details about the employees.

```
CREATE VIEW Details_about_employees as
SELECT e.*, j.job_name, j.department_id
FROM PROJ_Employees e
INNER JOIN
PROJ_Jobs j on e.job_id=j.job_id;
```

Query Result | Script Output | Query Result 1 | Query Result 2 | Query Result 3

Task completed in 0.174 seconds

```
Error report -
ORA-01408: such column list already indexed
01408. 00000 -  "such column list already indexed"
*Cause:
*Action:

Index PROJ_PATIENTS_INDEX created.


View DETAILS_ABOUT_EMPLOYEES created.
```

	EMPLOYEE_ID	EMPLOYEE_NAME	MANAGER_ID	JOB_ID	SALARY	EMAIL	PHONE	JOB_NAME	DEPARTMENT_ID
1	1	Rusu Sergiu-Ioan	(null)	1	5700	rususergiu21@ase.ro	722232323	Doctor	1
2	7	Rosu Ionut	1	1	5700	rosuiout@poli.ro	732334455	Doctor	1
3	8	Onea Maria	1	2	4300	mariaonea@upb.ro	765655656	Asistent	2
4	9	Tanase Theodor	1	3	5000	theodortns@upb.ro	733445566	Doctor	3
5	4	Petrescu Roxana	2	4	4000	petrescurox@ase.ro	765123456	Asistent	4
6	2	Licxandru Teodora	(null)	5	5200	licxthea@ase.ro	712332123	Doctor	5
7	5	Milea Robert	2	5	5200	mileaboss@ase.ro	778995522	Doctor	5
8	6	Rata Stefania	2	6	3900	ratastef@macmac.ro	777888777	Asistent	6
9	10	Popa Madalin	3	7	4100	popamada@ase.ro	733441122	Asistent	7
10	3	Nistor Stefana	(null)	8	6000	nistorstefana@ase.ro	765757575	Doctor	8
11	11	Nistor Andrei	3	8	6000	andrei_nist@uni.ro	745055932	Doctor	8
12	12	Mihalovici Tudor	3	9	3800	mihalovicit@ase.ro	745010622	Asistent	9

V. CONCLUSION

In conclusion, the hospital database has proven to be an essential tool in efficiently managing and organizing the medical operations of the hospital. Its effectiveness has been demonstrated through various relevant and diverse queries, and it is highly adaptable to different uses and improvements.