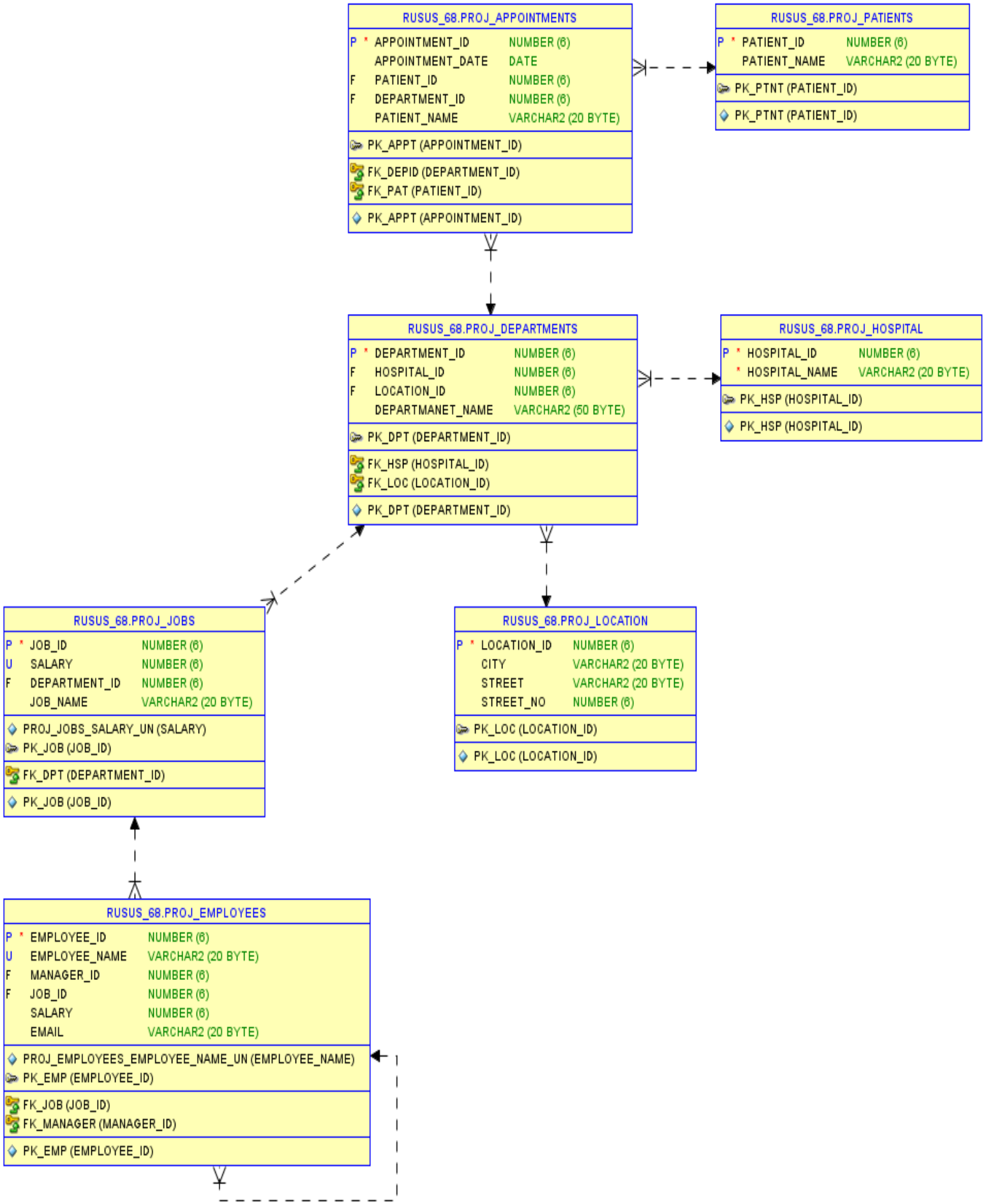

RUSUS_68.PROJ_APPOINTMENTS
P * APPOINTMENT_ID NUMBER (6)
APPOINTMENT_DATE DATE
F PATIENT_ID NUMBER (6)
F DEPARTMENT_ID NUMBER (6)
PATIENT_NAME VARCHAR2 (20 BYTE)
PK_APPT (APPOINTMENT_ID)
FK_DEPID (DEPARTMENT_ID)
FK_PAT (PATIENT_ID)
PK_APPT (APPOINTMENT_ID)
RUSUS_68.PROJ_PATIENTS
P * PATIENT_ID NUMBER (6)
PATIENT_NAME VARCHAR2 (20 BYTE)
PK_PTNT (PATIENT_ID)
PK_PTNT (PATIENT_ID)
RUSUS_68.PROJ_DEPARTMENTS
P * DEPARTMENT_ID NUMBER (6)
F HOSPITAL_ID NUMBER (6)
F LOCATION_ID NUMBER (6)
DEPARTMANET_NAME VARCHAR2 (50 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_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_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_EMPLOYEES
P * EMPLOYEE_ID NUMBER (6)
U EMPLOYEE_NAME VARCHAR2 (20 BYTE)
F MANAGER_ID NUMBER (6)
F JOB_ID NUMBER (6)
SALARY NUMBER (6)
EMAIL VARCHAR2 (20 BYTE)
PROJ_EMPLOYEES_EMPLOYEE_NAME_UN (EMPLOYEE_NAME)
PK_EMP (EMPLOYEE_ID)
FK_JOB (JOB_ID)
FK_MANAGER (MANAGER_ID)
PK_EMP (EMPLOYEE_ID)

Database Project – Healthcare

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.

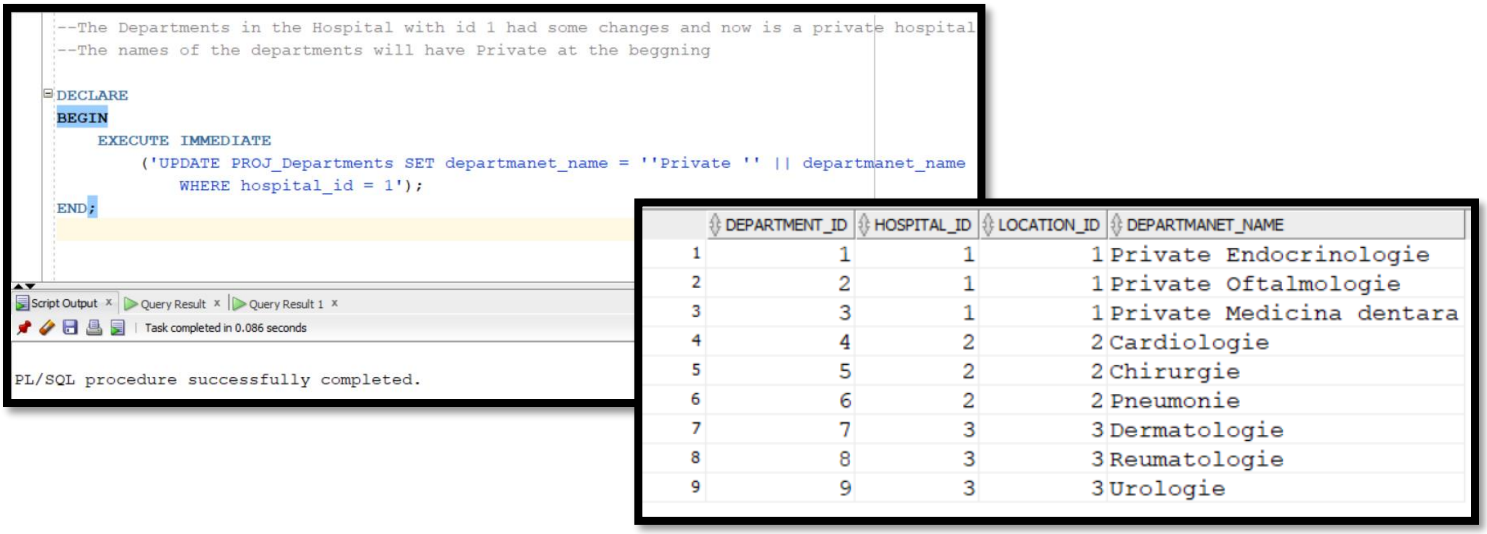
In conclusion, all these tables are key elements that are necessary for the successful management of a hospital. They provide a comprehensive view of the different aspects of hospital operations, make it easy to access, and manage the data. This database will help to improve the efficiency and effectiveness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

1. **I used the DDL command ALTER and execute immediate to remove the column “Phone” from my table because it was no longer needed.**

```
DECLARE
BEGIN
   EXECUTE IMMEDIATE 'ALTER TABLE PROJ_EMPLOYEES DROP COLUMN PHONE';
END;
```

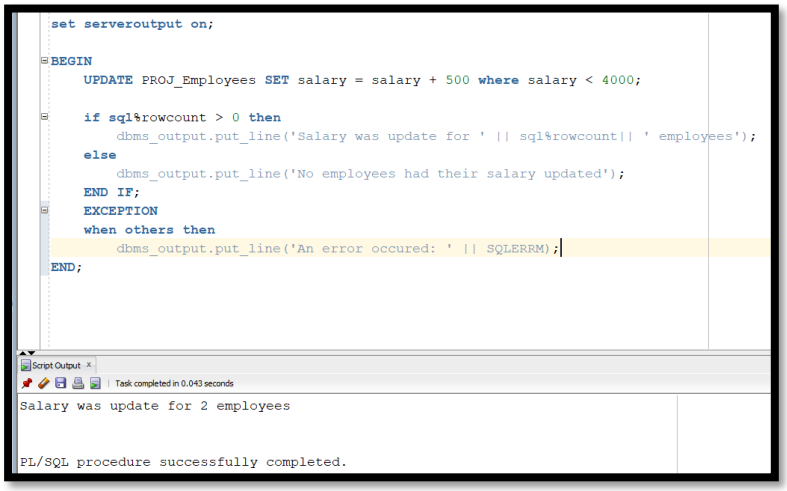
2. **Used a DML command to update the name of all the departments in the hospital with id 1, because now, the hospital is private, so I added ‘Private’ in front of the names.**

```
DECLARE
BEGIN
   EXECUTE IMMEDIATE
      ('UPDATE PROJ_Departments SET department_name = "Private " || department _name
         WHERE hospital_id = 1');
END;
```



3. **I decided that due to inflation, my employees should have salary bigger than 4000. I updated their salaries and I used the implicit cursor SQL%ROWCOUNT inside an IF statement to see how many employees got their salaries increased. I also used an implicit error SQLERRM in case there would have been an error.**

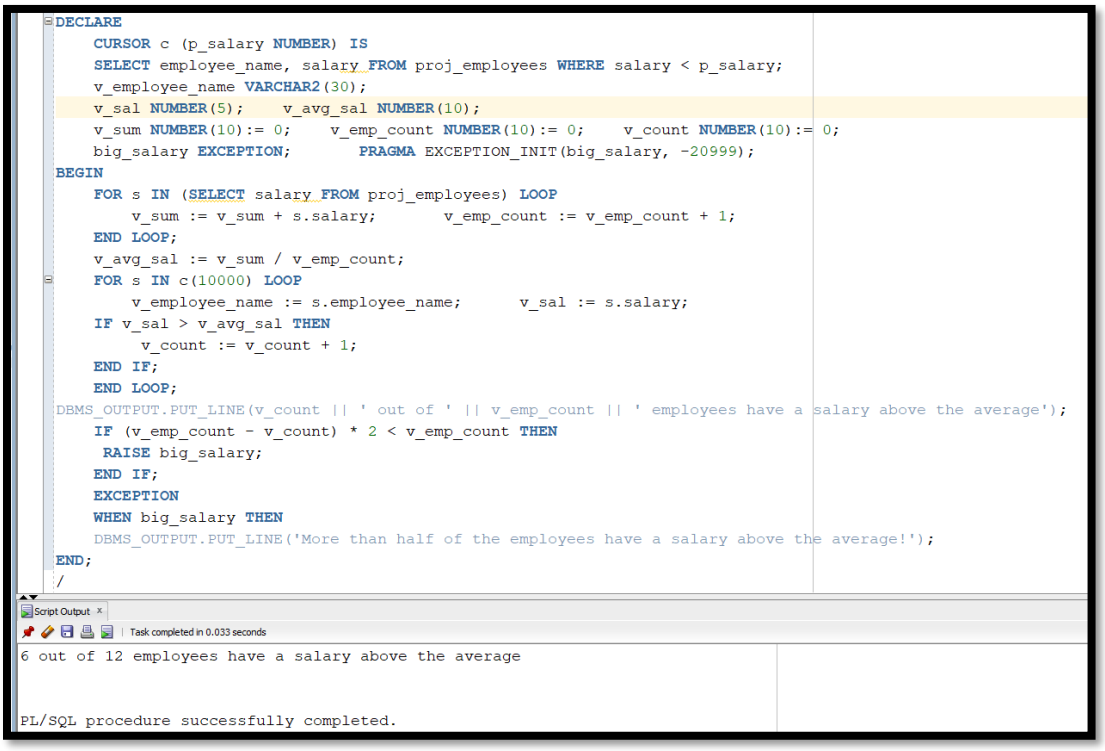
```
BEGIN
   UPDATE PROJ_Employees SET salary = salary + 500 where salary < 4000;

   if sql%rowcount > 0 then
      dbms_output.put_line('Salary was update for ' || sql%rowcount|| ' employees');
   else
      dbms_output.put_line('No employees had their salary updated');
   END IF;
   EXCEPTION
   when others then
      dbms_output.put_line('An error occured: ' || SQLERRM);
END;
```



4. **Now, after I raised the salaries of some employees, I decided that I should check how many of them have salaries above average. If more than half of my employees have salary above the average, I raise an explicit exception so I know I should lower some salaries. I use an explicit cursor with parameters and a for loop to look into all my employees and their salaries.**

```
DECLARE
   CURSOR c (p_salary NUMBER) IS
   SELECT employee_name, salary FROM proj_employees WHERE salary < p_salary;
   v_employee_name VARCHAR2(30);
   v_sal NUMBER(5);    v_avg_sal NUMBER(10);
   v_sum NUMBER(10):= 0;    v_emp_count NUMBER(10):= 0;    v_count NUMBER(10):= 0;
   big_salary EXCEPTION;    PRAGMA EXCEPTION_INIT(big_salary, -20999);
BEGIN
   FOR s IN (SELECT salary FROM proj_employees) LOOP
      v_sum := v_sum + s.salary;    v_emp_count := v_emp_count + 1;
   END LOOP;
   v_avg_sal := v_sum / v_emp_count;
   FOR s IN c(10000) LOOP
      v_employee_name := s.employee_name;    v_sal := s.salary;
```

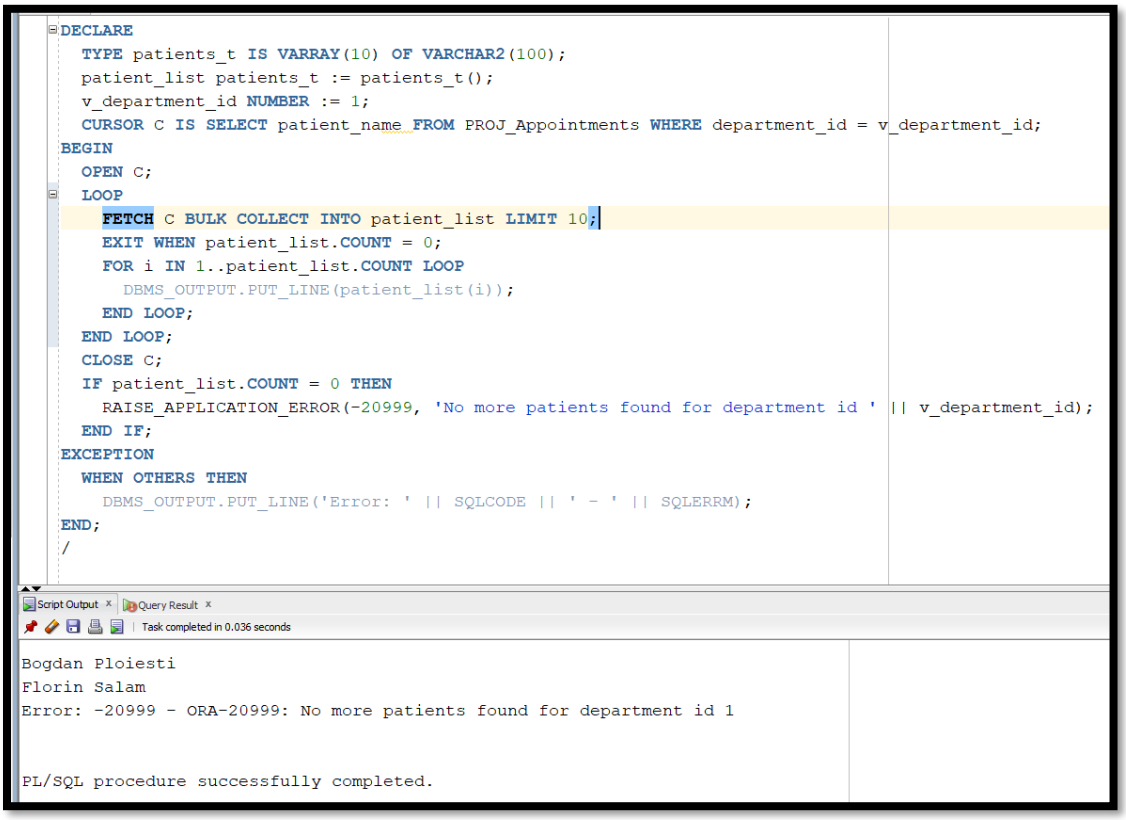
```
    IF v_sal > v_avg_sal THEN
        v_count := v_count + 1;
    END IF;
    END LOOP;
DBMS_OUTPUT.PUT_LINE(v_count || ' out of ' || v_emp_count || ' employees have a salary above the average');
    IF (v_emp_count - v_count) * 2 < v_emp_count THEN
     RAISE big_salary;
    END IF;
    EXCEPTION
    WHEN big_salary THEN
    DBMS_OUTPUT.PUT_LINE('More than half of the employees have a salary above the average!');
END;
```



5. **For the next exercise, I used a VARRAY, an explicit cursor without parameters and a LOOP to make a list of the patients in a specific department. I also raised an explicit exception when there were no more patients in that department.**

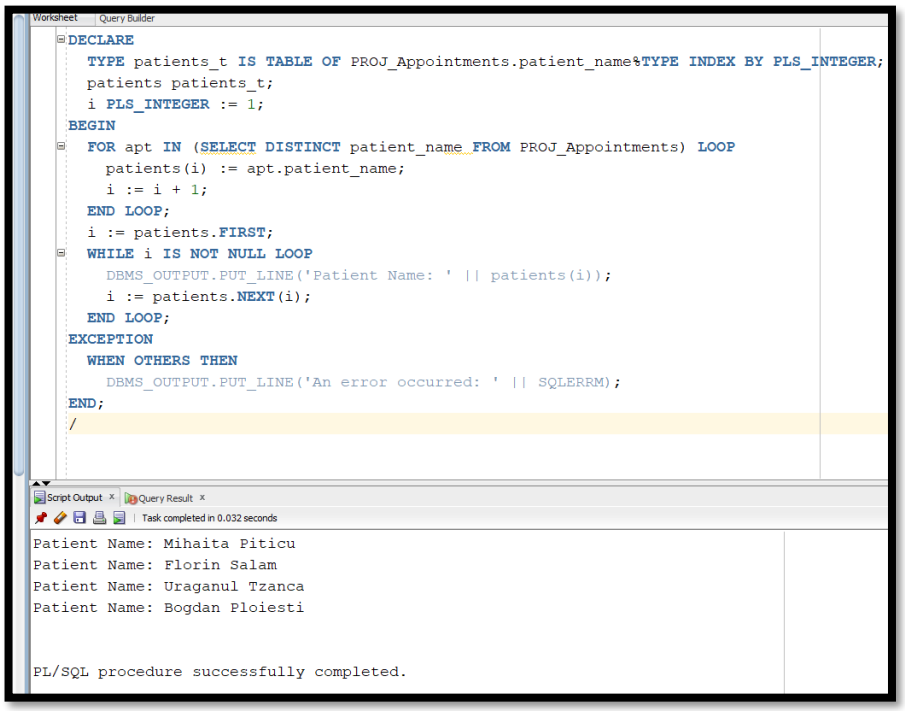
```
DECLARE
  TYPE patients_t IS VARRAY(10) OF VARCHAR2(100);
  patient_list patients_t := patients_t();
  v_department_id NUMBER := 1;
  CURSOR C IS SELECT patient_name FROM PROJ_Appointments WHERE department_id = v_department_id;
```

```
BEGIN
  OPEN C;
  LOOP
    FETCH C BULK COLLECT INTO patient_list LIMIT 10;
    EXIT WHEN patient_list.COUNT = 0;
    FOR i IN 1..patient_list.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE(patient_list(i));
    END LOOP;
  END LOOP;
  CLOSE C;
  IF patient_list.COUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20999, 'No more patients found for department id
' || v_department_id);
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);
END;
```



6. For the next exercise, I made an index by table, with an implicit exception and a while loop to see all the patient names that have an appointment.

```
DECLARE
  TYPE patients_t IS TABLE OF PROJ_Appointments.patient_name%TYPE INDEX BY
PLS_INTEGER;
  patients patients_t;
  i PLS_INTEGER := 1;
BEGIN
  FOR apt IN (SELECT DISTINCT patient_name FROM PROJ_Appointments) LOOP
    patients(i) := apt.patient_name;
    i := i + 1;
  END LOOP;
  i := patients.FIRST;
  WHILE i IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE('Patient Name: ' || patients(i));
    i := patients.NEXT(i);
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```



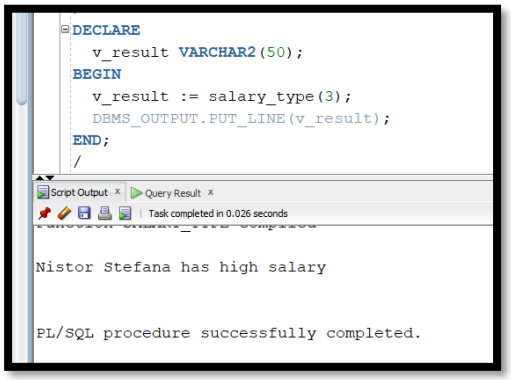
7. **Now I tried to do the same thing with a nested table and an implicit exception. This time I also printed the numbers in the list :**

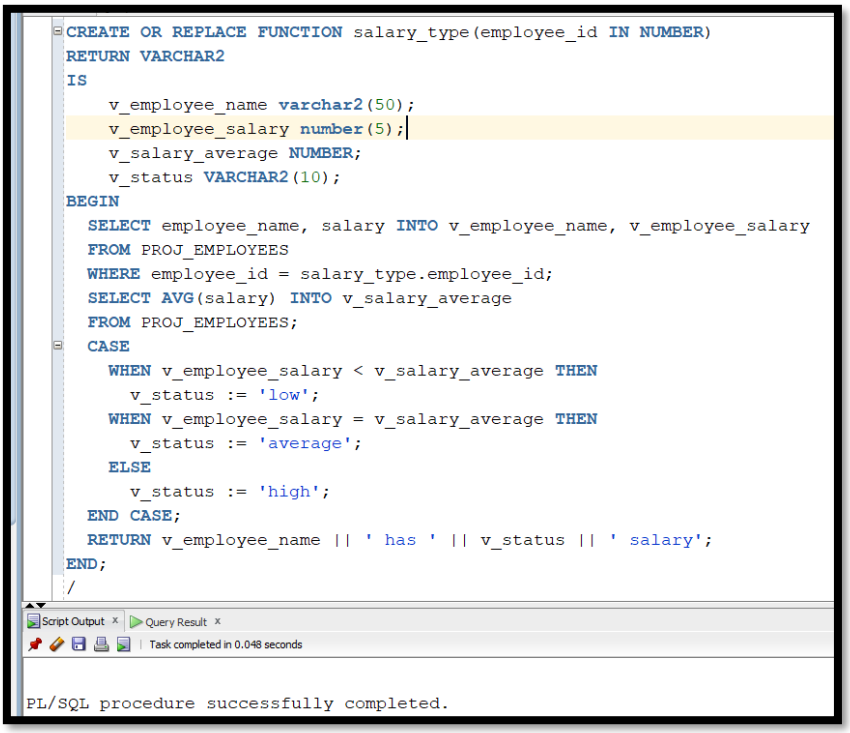
```
DECLARE
  TYPE patient_list IS TABLE OF PROJ_Patients.patient_name%TYPE;
  patients patient_list := patient_list('Bogdan Ploiesti', 'Mihaita Piticu', 'Florin Salam',
'Uraganul Tzanca');
BEGIN
  FOR i IN 1..patients.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Patient ' || i || ': ' || patients(i));
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

8. **Here I made a function that tells what kind of salary an employee has using CASE.**

```
CREATE OR REPLACE FUNCTION salary_type(employee_id IN NUMBER)
RETURN VARCHAR2
IS
   v_employee_name varchar2(50);
   v_employee_salary number(5);
   v_salary_average NUMBER;
   v_status VARCHAR2(10);
BEGIN
  SELECT employee_name, salary INTO v_employee_name, v_employee_salary
  FROM PROJ_EMPLOYEES
  WHERE employee_id = salary_type.employee_id;
  SELECT AVG(salary) INTO v_salary_average
  FROM PROJ_EMPLOYEES;
```

```
  CASE
    WHEN v_employee_salary < v_salary_average THEN
      v_status := 'low';
    WHEN v_employee_salary = v_salary_average THEN
      v_status := 'average';
    ELSE
      v_status := 'high';
  END CASE;
  RETURN v_employee_name || ' has ' || v_status || '
salary';
END;
/
DECLARE
  v_result VARCHAR2(50);
BEGIN
  v_result := salary_type(3);
  DBMS_OUTPUT.PUT_LINE(v_result);
END;
```

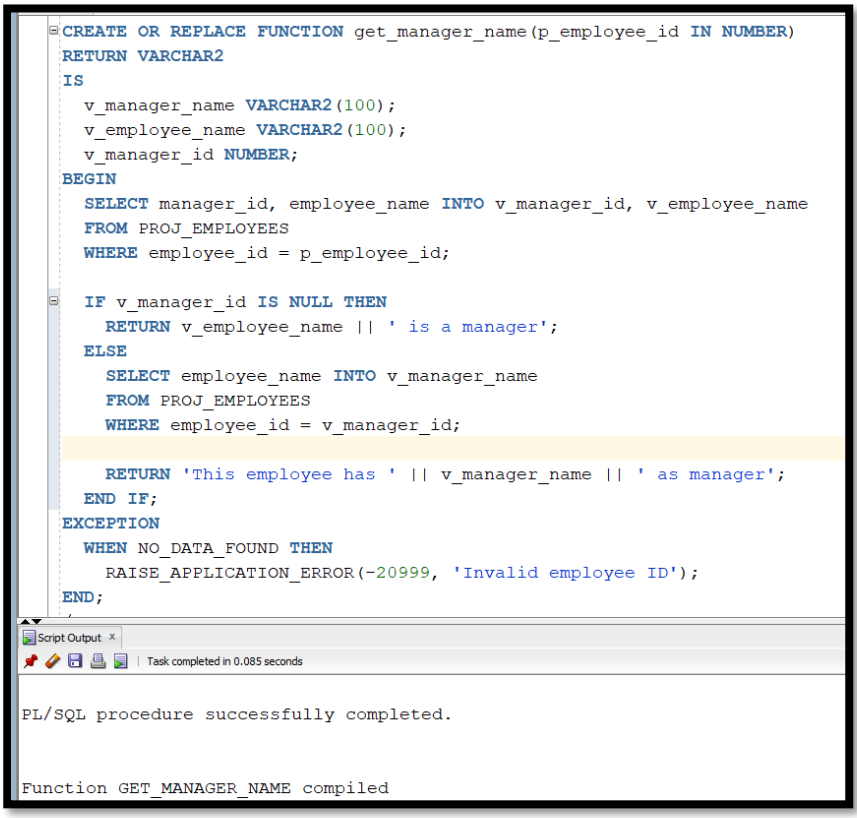




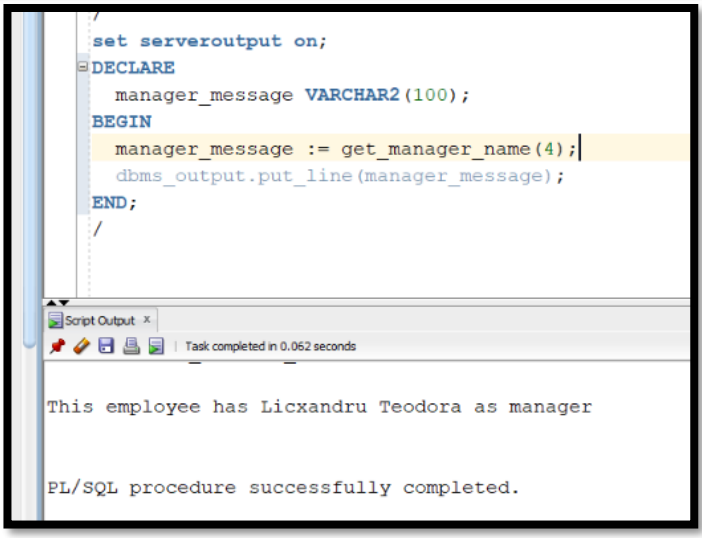
9. **Next, I made a function that returns the manager name by giving it the employee id so I can track which employee has which manager. If the v_manager_id would be null, I will return that he/she is a manager. If an invalid id would be inserted, I raise an exception.**

```
CREATE OR REPLACE FUNCTION get_manager_name(p_employee_id IN NUMBER)
RETURN VARCHAR2
```

```
IS
  v_manager_name VARCHAR2(100);
  v_employee_name VARCHAR2(100);
  v_manager_id NUMBER;
BEGIN
  SELECT manager_id, employee_name INTO v_manager_id, v_employee_name
  FROM PROJ_EMPLOYEES
  WHERE employee_id = p_employee_id;

  IF v_manager_id IS NULL THEN
    RETURN v_employee_name || ' is a manager';
  ELSE
    SELECT employee_name INTO v_manager_name
    FROM PROJ_EMPLOYEES
    WHERE employee_id = v_manager_id;

    RETURN 'This employee has ' || v_manager_name || ' as manager';
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid employee ID');
END;
/
```



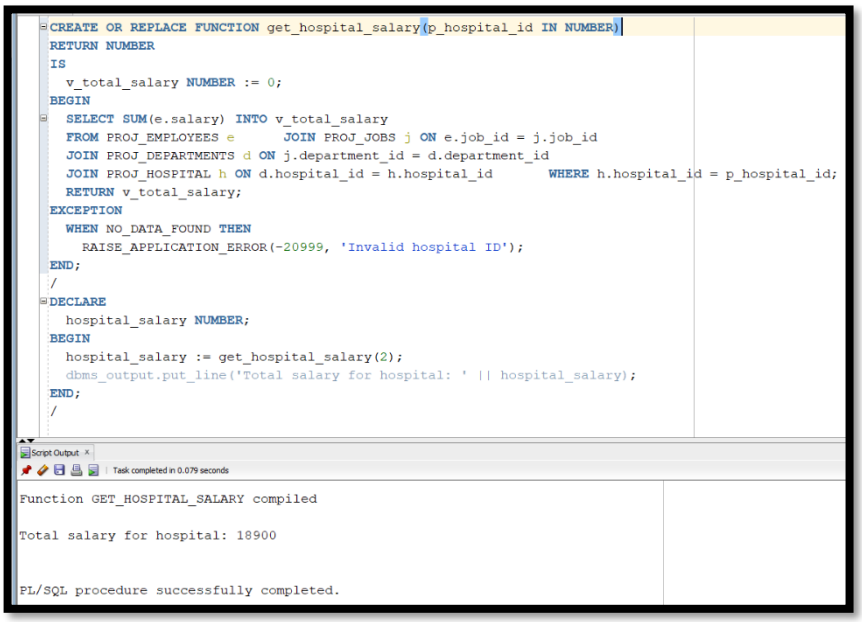

```
set serveroutput on;
DECLARE
  manager_message VARCHAR2(100);
BEGIN
  manager_message := get_manager_name(4);
  dbms_output.put_line(manager_message);
END;
/
```



10. My last function is a function that takes the hospital id and returns the sum of salaries of the employees in that department by doing 3 joins.

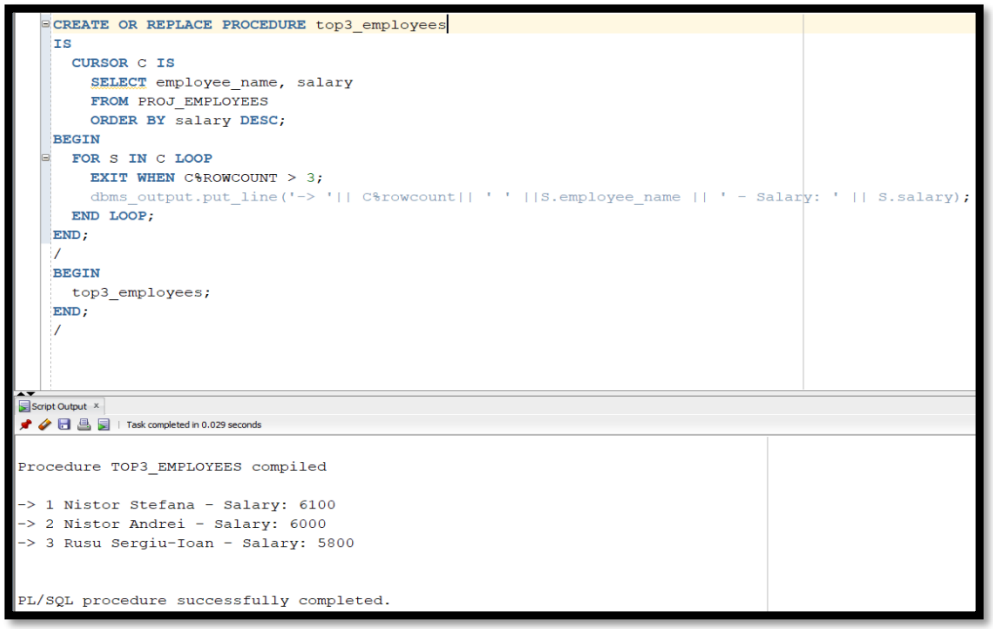
```
CREATE OR REPLACE FUNCTION get_hospital_salary(p_hospital_id IN NUMBER)
RETURN NUMBER
IS
  v_total_salary NUMBER := 0;
BEGIN
  SELECT SUM(e.salary) INTO v_total_salary
```

```
  FROM PROJ_EMPLOYEES e      JOIN PROJ_JOBS j ON e.job_id = j.job_id
  JOIN PROJ_DEPARTMENTS d ON j.department_id = d.department_id
  JOIN      PROJ_HOSPITAL      h      ON
d.hospital_id = h.hospital_id      WHERE
h.hospital_id = p_hospital_id;
  RETURN v_total_salary;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-
20999, 'Invalid hospital ID');
END;
/
DECLARE
  hospital_salary NUMBER;
BEGIN
  hospital_salary := get_hospital_salary(2);
  dbms_output.put_line('Total   salary   for
hospital: ' || hospital_salary);
END;
```



11. For my first procedure, I decided that I want to print the employees with the three highest salaries.

```
CREATE OR REPLACE PROCEDURE top3_employees
IS
  CURSOR C IS
    SELECT employee_name, salary
    FROM PROJ_EMPLOYEES
    ORDER BY salary DESC;
BEGIN
  FOR S IN C LOOP
    EXIT WHEN
C%ROWCOUNT > 3;

dbms_output.put_line('->
'||  C%rowcount||  '  '
||S.employee_name  ||  '  -
Salary: ' || S.salary);
  END LOOP;
END;
/
BEGIN
  top3_employees;
END;
```



12. In this procedure, I decided to make it easy for the managers to update their employees' salaries so I my procedure takes as parameters the manager_id and raises the salaries of all his employees. You can also see MY NAME HERE.

```
CREATE OR REPLACE PROCEDURE manager_raise(p_manager_id IN NUMBER, p_raise IN NUMBER)
IS
  v_manager_name VARCHAR2(50);
BEGIN
  SELECT employee_name INTO v_manager_name
  FROM proj_employees
  WHERE employee_id = p_manager_id;
  IF v_manager_name IS NULL THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid manager_id');
  END IF;

  UPDATE proj_employees
  SET salary = salary + p_raise
  WHERE manager_id = p_manager_id;

  DBMS_OUTPUT.PUT_LINE('Manager ' || v_manager_name || ' increased the salaries of their employees by: ' || p_raise);
  DBMS_OUTPUT.PUT_LINE('Their salaries are now: ');
  FOR emp IN (SELECT employee_name, salary FROM proj_employees WHERE manager_id = p_manager_id)
  LOOP
    DBMS_OUTPUT.PUT_LINE(emp.employee_name || ': ' || emp.salary);
  END LOOP;
END;
/
BEGIN
  manager_raise(1, 1000);
END;
```

```
BEGIN
  manager_raise(1, 1000);
END;
```

```
Script Output
Task completed in 0.033 seconds
Manager Rusu Sergiu-Ioan increased the salaries of their employees by: 1000
Their salaries are now:
Rosu Ionut: 6700
Onea Maria: 5300
Tanase Theodor: 6000


PL/SQL procedure successfully completed.
```

13. For my third and last procedure, I want to print all the departments inside a hospital so I can better manage it. I used a cursor to loop through the departments and threw an error if I inserted an invalid id.

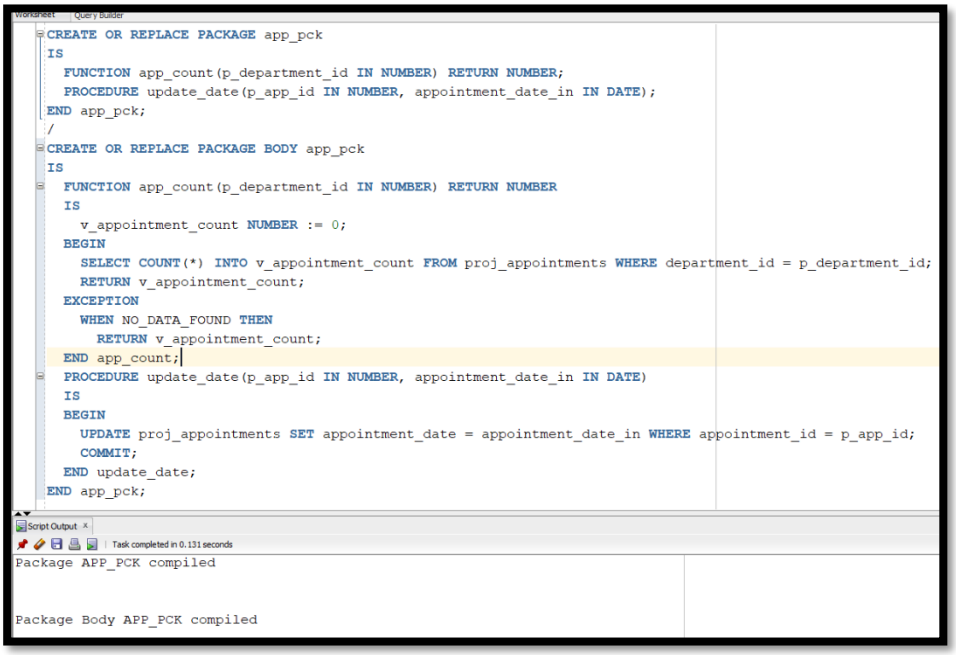
```
CREATE OR REPLACE PROCEDURE departments_list(p_hospital_id IN NUMBER)
IS
  CURSOR c IS
    SELECT department_name FROM proj_departments WHERE hospital_id = p_hospital_id;
  v_hospital_id NUMBER;
  v_hospital_name VARCHAR2(50);
BEGIN
  SELECT hospital_id, hospital_name INTO v_hospital_id, v_hospital_name
  FROM proj_hospital
  WHERE hospital_id = p_hospital_id;
  IF v_hospital_id IS NULL THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid hospital ID');
  END IF;
  DBMS_OUTPUT.PUT_LINE(v_hospital_name || ' has the following departments:');
  FOR s IN c LOOP

DBMS_OUTPUT.PUT_LINE(s.department_name);
  END LOOP;
END;
/
BEGIN
  departments_list(1);
END;
```

```
CREATE OR REPLACE PROCEDURE departments_list(p_hospital_id IN NUMBER)
IS
  CURSOR c IS
    SELECT department_name  FROM proj_departments  WHERE hospital_id = p_hospital_id;
  v_hospital_id NUMBER;
  v_hospital_name VARCHAR2(50);
BEGIN
  SELECT hospital_id, hospital_name INTO v_hospital_id, v_hospital_name
  FROM proj_hospital
  WHERE hospital_id = p_hospital_id;
  IF v_hospital_id IS NULL THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid hospital ID');
  END IF;
  DBMS_OUTPUT.PUT_LINE(v_hospital_name || ' has the following departments:');
  FOR s IN c LOOP
    DBMS_OUTPUT.PUT_LINE(s.department_name);
  END LOOP;
END;
/
BEGIN
  departments_list(1);
END;
```

```
Script Output
Task completed in 0.04 seconds
Procedure DEPARTMENTS_LIST compiled

Spitalul Judetean has the following departments:
Endocrinologie
Oftalmologie
Medicina dentara


PL/SQL procedure successfully completed.
```

14. Not knowing so many things about packages, I made a simple one with one function and one procedure. The function returns how many appointments are in a department and the procedure updates the date of a specified appointment to a new date.

```
CREATE OR REPLACE PACKAGE app_pck
IS
  FUNCTION app_count(p_department_id IN NUMBER) RETURN NUMBER;
  PROCEDURE update_date(p_app_id IN NUMBER, appointment_date_in IN DATE);
END app_pck;
/
CREATE OR REPLACE PACKAGE BODY app_pck
IS
  FUNCTION app_count(p_department_id IN NUMBER) RETURN NUMBER
  IS
    v_appointment_count NUMBER := 0;
  BEGIN
    SELECT COUNT(*) INTO v_appointment_count FROM proj_appointments WHERE
department_id = p_department_id;
    RETURN v_appointment_count;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN v_appointment_count;
  END app_count;
  PROCEDURE update_date(p_app_id IN NUMBER, appointment_date_in IN DATE)
  IS
  BEGIN
    UPDATE proj_appointments
SET appointment_date =
appointment_date_in WHERE
appointment_id = p_app_id;
    COMMIT;
  END update_date;
END app_pck;
/

DECLARE
  v_count NUMBER;
BEGIN
  v_count :=
app_pck.app_count(1);
```



```
  DBMS_OUTPUT.PUT_LINE('Appointment count for department_id 1: ' || v_count);
  app_pck.update_date(2, TO_DATE('2023-05-21', 'YYYY-MM-DD'));
  DBMS_OUTPUT.PUT_LINE('Appointment date updated for appointment_id 100.');
END;
```

```
DECLARE
  v_count NUMBER;
BEGIN
  v_count := app_pck.app_count(1);
  DBMS_OUTPUT.PUT_LINE('Appointment count for department_id 1: ' || v_count);
  app_pck.update_date(2, TO_DATE('2023-05-21', 'YYYY-MM-DD'));
  DBMS_OUTPUT.PUT_LINE('Appointment date updated for appointment_id 100.');
END;
```

Script Output | Task completed in 0.056 seconds

```
Package Body APP_PCK compiled

Appointment count for department_id 1: 2
Appointment date updated for appointment_id 100.


PL/SQL procedure successfully completed.
```

15. Another new chapter for me are trigger, so I decided to keep it simple as well. My first row level trigger raises an exception if the salary update goes over 10000.

```
CREATE OR REPLACE TRIGGER salary_too_high
BEFORE UPDATE ON proj_employees
FOR EACH ROW
BEGIN
  IF :NEW.salary > 10000 THEN

RAISE_APPLICATION_ERROR(-20999, 'Salary is too high for updates!');
  END IF;
END;
/
DECLARE
  v_employee_id NUMBER := 1;
BEGIN
  UPDATE proj_employees SET salary = 150000 WHERE employee_id = v_employee_id;
END;
/
```

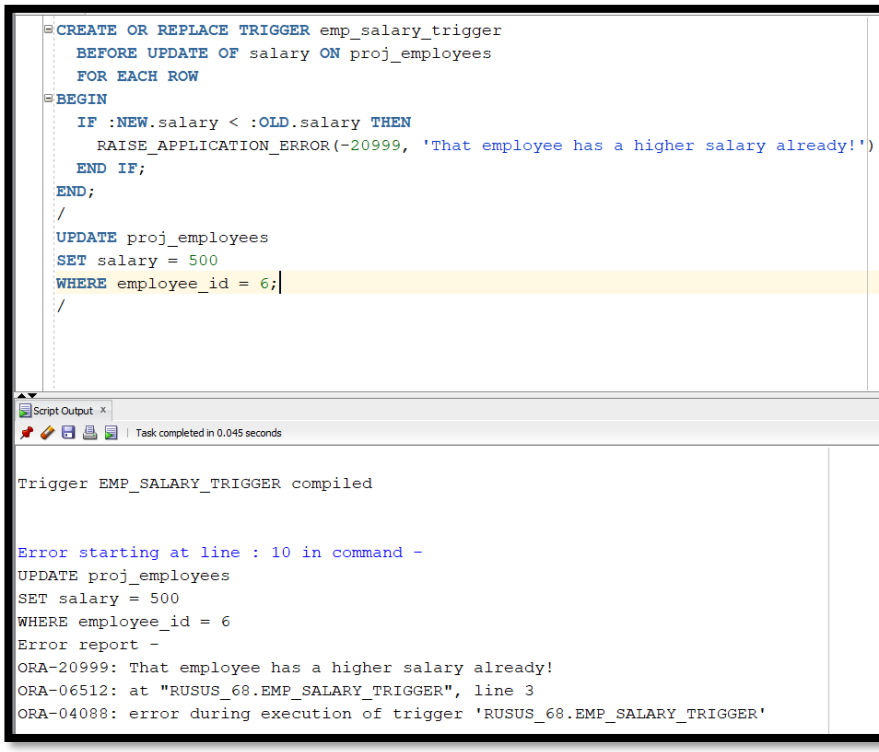
```
CREATE OR REPLACE TRIGGER salary_too_high
BEFORE UPDATE ON proj_employees
FOR EACH ROW
BEGIN
  IF :NEW.salary > 10000 THEN
    RAISE_APPLICATION_ERROR(-20999, 'Salary is too high for updates!');
  END IF;
END;
/
DECLARE
  v_employee_id NUMBER := 1;
BEGIN
  UPDATE proj_employees SET salary = 150000 WHERE employee_id = v_employee_id;
END;
/
```

Script Output | Task completed in 0.188 seconds

```
Trigger SALARY_TOO_HIGH compiled


Error starting at line : 11 in command -
DECLARE
  v_employee_id NUMBER := 1;
BEGIN
  UPDATE proj_employees SET salary = 150000 WHERE employee_id = v_employee_id;
END;
Error report -
ORA-20999: Salary is too high for updates!
ORA-06512: at "RUSUS_68.SALARY_TOO_HIGH", line 3
ORA-04088: error during execution of trigger 'RUSUS_68.SALARY_TOO_HIGH'
ORA-06512: at line 4
```

16. Next, another simple row level trigger that prevents the update of the salary to give the actual employee salary that is lower than the one it already has.

```
CREATE OR REPLACE TRIGGER emp_salary_trigger
  BEFORE UPDATE OF salary ON proj_employees
  FOR EACH ROW
BEGIN
  IF :NEW.salary < :OLD.salary THEN
    RAISE_APPLICATION_ERROR(-20999, 'That employee has a higher salary already!');
  END IF;
END;
/
UPDATE proj_employees
SET salary = 500
WHERE employee_id = 6;
/
```



17. For my first statement level trigger, again, I kept it simple and I made a trigger that raises an exception when trying to delete a patient that still exists in my appointments.

```
CREATE OR REPLACE TRIGGER remove_patient
AFTER DELETE ON PROJ_PATIENTS
FOR EACH ROW
DECLARE
  v_patient_id NUMBER;
 v_appointment_count NUMBER;
BEGIN
  v_patient_id := :old.patient_id;
  SELECT COUNT(*) INTO v_appointment_count FROM PROJ_APPOINTMENTS WHERE patient_id = v_patient_id;
  IF v_appointment_count > 0 THEN
  RAISE_APPLICATION_ERROR(-20999, 'Cannot delete patient with active appointments');
  END IF;
END;
/
```

```
DECLARE
  v_patient_id NUMBER := 3;
BEGIN
  DELETE FROM PROJ_PATIENTS WHERE patient_id = v_patient_id;
END;
```

```
CREATE OR REPLACE TRIGGER remove_patient
AFTER DELETE ON PROJ_PATIENTS
FOR EACH ROW
DECLARE
  v_patient_id NUMBER;   v_appointment_count NUMBER;
BEGIN
  v_patient_id := :old.patient_id;
  SELECT COUNT(*) INTO v_appointment_count FROM PROJ_APPOINTMENTS WHERE patient_id = v_patient_id;
  IF v_appointment_count > 0 THEN
  RAISE_APPLICATION_ERROR(-20999, 'Cannot delete patient with active appointments');
  END IF;
END;
/
DECLARE
  v_patient_id NUMBER := 3;
BEGIN
  DELETE FROM PROJ_PATIENTS WHERE patient_id = v_patient_id;
END;
```

```
Trigger REMOVE_PATIENT compiled

PL/SQL procedure successfully completed.

Error starting at line : 15 in command -
DECLARE
  v_patient_id NUMBER := 3;
BEGIN
  DELETE FROM PROJ_PATIENTS WHERE patient_id = v_patient_id;
END;
Error report -
ORA-20999: Cannot delete patient with active appointments
```

18. For my last statement-level-trigger, I made it so you cannot add an appointment in the past; all the appointments should be in the future.

```
CREATE OR REPLACE TRIGGER new_appointment_check
BEFORE INSERT ON PROJ_APPOINTMENTS
FOR EACH ROW
BEGIN
  IF :new.appointment_date < SYSDATE THEN
    RAISE_APPLICATION_ERROR(-20999, 'You cannot insert an appointment in the past.');
  END IF;
END;
/
DECLARE
  v_patient_id NUMBER := 5;
  v_department_id NUMBER := 2;
  v_appointment_date DATE := TO_DATE('2022-06-01', 'YYYY-MM-DD');
BEGIN
  INSERT INTO PROJ_APPOINTMENTS VALUES (1, v_appointment_date, v_patient_id, v_department_id, 'Guta Nicolae');
END;
```

```
CREATE OR REPLACE TRIGGER new_appointment_check
BEFORE INSERT ON PROJ_APPOINTMENTS
FOR EACH ROW
BEGIN
  IF :new.appointment_date < SYSDATE THEN
    RAISE_APPLICATION_ERROR(-20999, 'You cannot insert an appointment in the past.');
  END IF;
END;
/
DECLARE
  v_patient_id NUMBER := 5;
  v_department_id NUMBER := 2;
  v_appointment_date DATE := TO_DATE('2022-06-01', 'YYYY-MM-DD');
BEGIN
  INSERT INTO PROJ_APPOINTMENTS VALUES (1, v_appointment_date, v_patient_id, v_department_id, 'Guta Nicolae');
END;
```

```
Trigger NEW_APPOINTMENT_CHECK compiled

Error starting at line : 10 in command -
DECLARE
  v_patient_id NUMBER := 5;
  v_department_id NUMBER := 2;
  v_appointment_date DATE := TO_DATE('2022-06-01', 'YYYY-MM-DD');
BEGIN
  INSERT INTO PROJ_APPOINTMENTS VALUES (1, v_appointment_date, v_patient_id, v_department_id, 'Guta Nicolae');
END;
Error report -
ORA-20999: You cannot insert an appointment in the past.
```

CONCLUSION

In conclusion, using PL/SQL in my hospital database project has greatly improved my capacity to manage a database more effectively. Throughout the project, I had the opportunity to see the advantages of PL/SQL over SQL. I was able to develop effective procedures and functions, as well as reliable error handling techniques dues to PL/SQL's features. All in all, the experience has shown that PL/SQL is a useful tool for handling databases and can increase productivity and efficiency.