**DATABASE PROJECT**

**UNIVERSITIES**

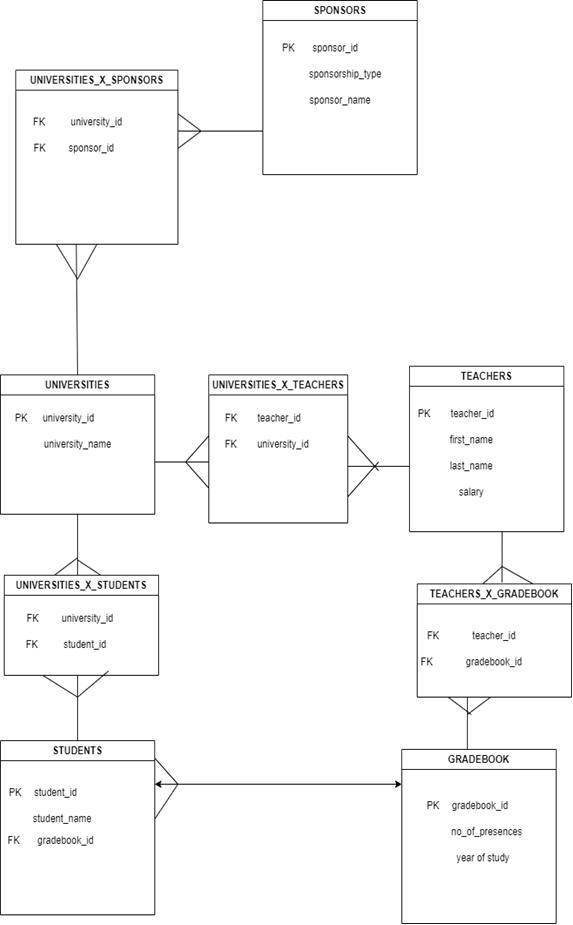
BY NAFORNITA DANIELA ANDREA

CSIE YEAR 2

The target of the project is to design a database in a database management system(named SQL Oracle Developer) specially for usage of managing data about different universities in Bucharest. This project contains different tables who are managed specially to show what are the relations between different institutions, employees, and students.With this database, we will find out which universities are sponsorised by which sponsors(like Vodafone, Orange, Accenture), and more details about different students, and their teachers.

In order to be all clear, I have created 5 main tables who are about: sponsors, teachers, universities,gradebook, and students. In my opinion, those are the main options who we can relate to when we talk about managing a database for universities in general. Because most of them are in relations „many-to-many”, I have created some intermediary tabels who are in relation with minimum 2 main tables, and I named them as a combination of two in order to be easy to recognize them.

Note: The schema is the same as in semester 1.



**From the table created below, we can conclude that: because most of the tables were in relation „many to many”, and it is forbidden to make a table like that and to make foreign keys and primary keys, I have created some „intermediary” tables who can give to the tables the relation „one to many” who is allowed to be made.**

To be more explicit, those are the main topics that we are going to talk about:

1. Interaction with the Oracle server through SQL commands (LDD and LMD): the use of execute immediate, particularities regarding the use of the select command as well as row and group level functions.
2. Alternate and repetitive structures.
3. Data collections (index by table, nested table, varray)
4. Exception handling. (minimum 3 implicit, 2 explicit)
5. Cursor handling: default and explicit (with and without parameters)
6. Functions, procedures, their inclusion in packages. (minimum 3 functions, 3 procedures and a package that includes other functions and procedures)
7. Statement-level and row-level triggers (minimum 2 of each)

**1. Interaction with the Oracle server through SQL commands (LDD and LMD): the use of execute immediate.**

*1.1.1 LDD (Data Definition Language) commands:* ***ALTER*** *- Alter a table using* ***EXECUTE IMMEDIATE****:*

DECLARE

v\_table\_name VARCHAR2(30) := 'teachers';

BEGIN

EXECUTE IMMEDIATE 'ALTER TABLE ' || v\_table\_name || ' ADD (experiences NUMBER)';

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' altered.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'teachers';

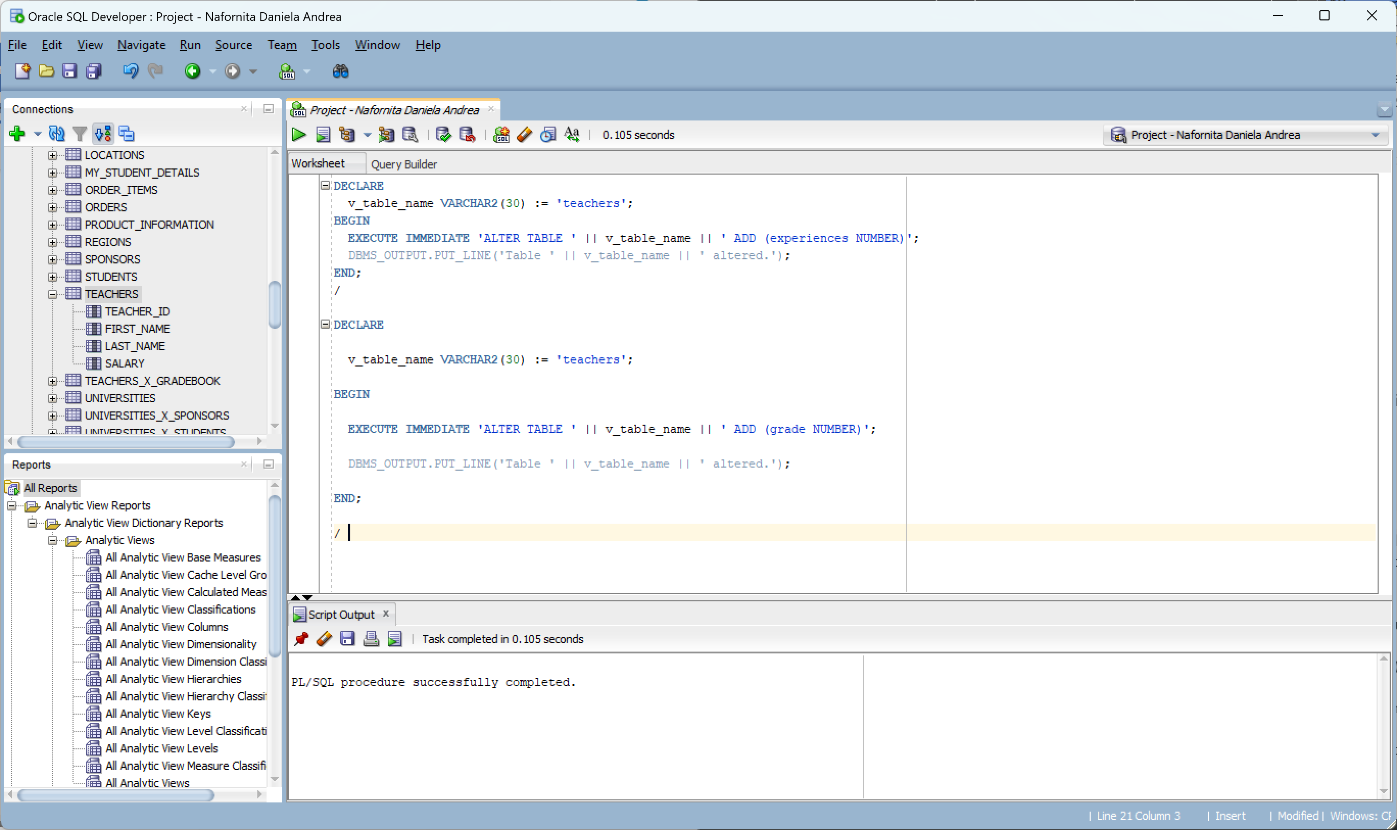
BEGIN

EXECUTE IMMEDIATE 'ALTER TABLE ' || v\_table\_name || ' ADD (grade NUMBER)';

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' altered.');

END;

/



*1.1.2 LDD (Data Definition Language) commands:* ***CREATE*** *- Create a table using* ***EXECUTE IMMEDIATE****:*

DECLARE

v\_table\_name VARCHAR2(30) := 'person';

BEGIN

EXECUTE IMMEDIATE 'CREATE TABLE ' || v\_table\_name || ' (id NUMBER, name VARCHAR2(30))';

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' created.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'guests';

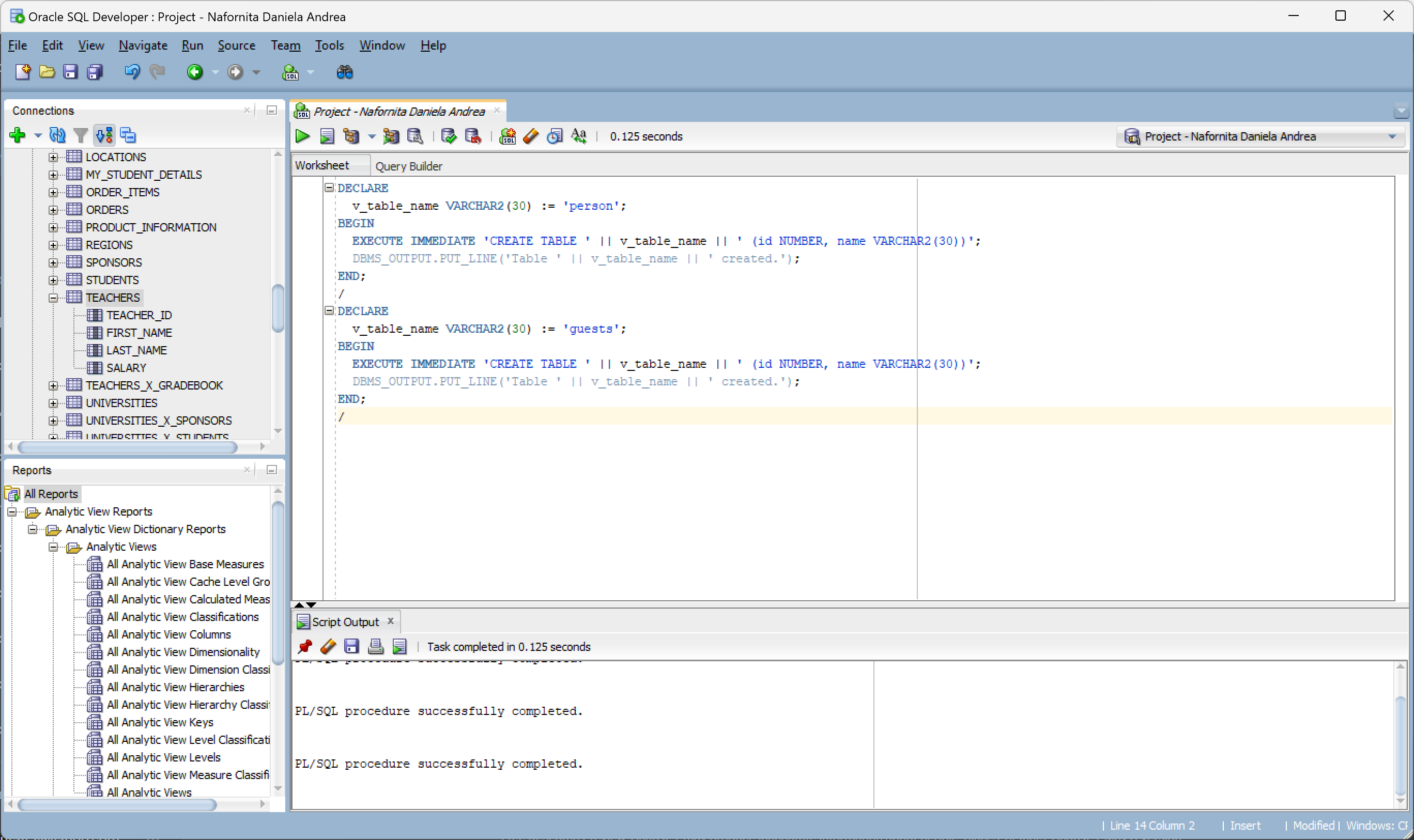
BEGIN

EXECUTE IMMEDIATE 'CREATE TABLE ' || v\_table\_name || ' (id NUMBER, name VARCHAR2(30))';

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' created.');

END;

/



*1.1.3 LDD (Data Definition Language) commands:* ***DROP*** *- Drop a table using* ***EXECUTE IMMEDIATE****:*

DECLARE

v\_table\_name VARCHAR2(30) := 'person';

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ' || v\_table\_name;

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' dropped.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'guests';

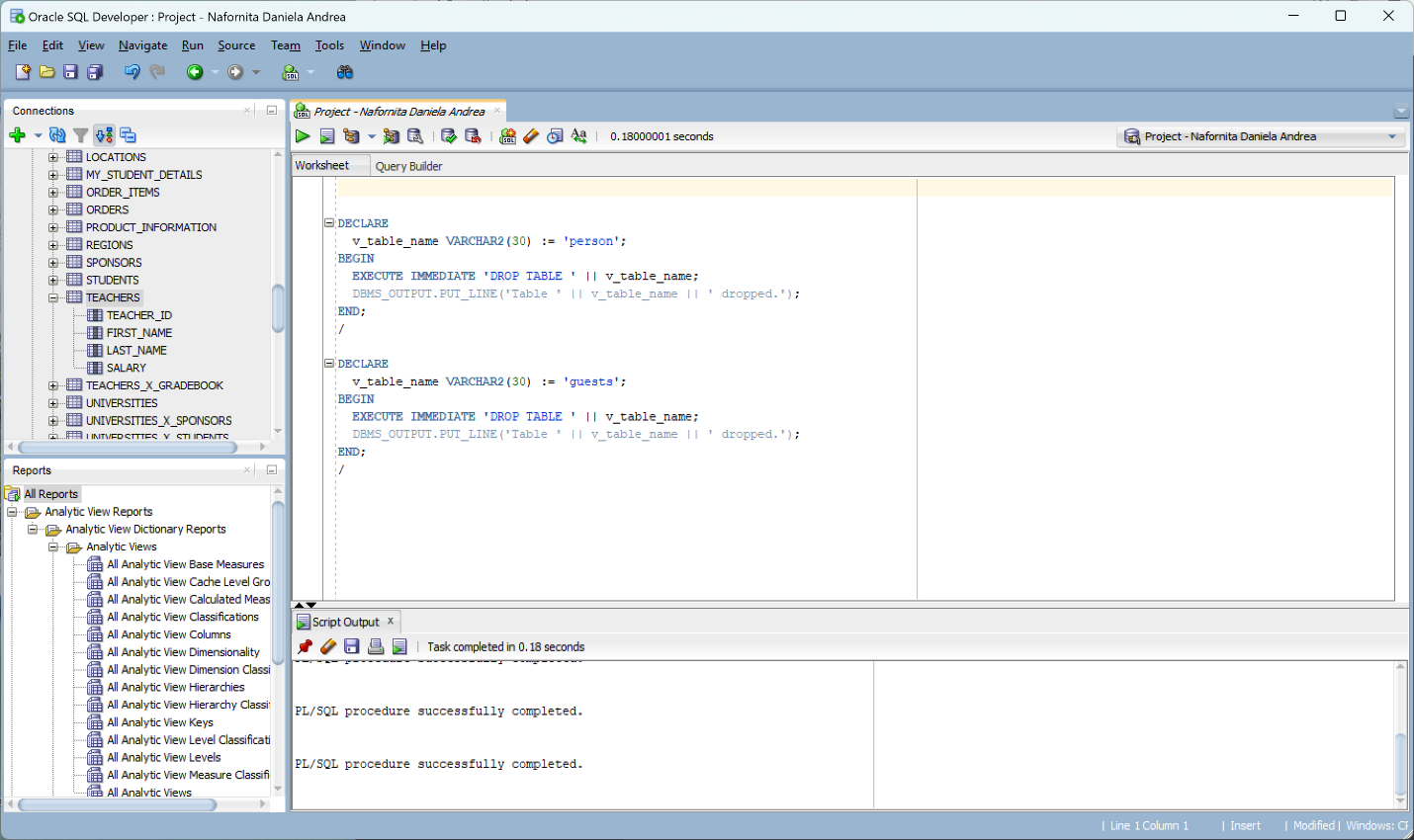
BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ' || v\_table\_name;

DBMS\_OUTPUT.PUT\_LINE('Table ' || v\_table\_name || ' dropped.');

END;

/



*1.2.1 LMD (Data Manipulation Language) commands: SELECT statement using EXECUTE command:*

DECLARE

v\_sql\_stmt VARCHAR2(1000);

teacher\_id NUMBER(6) := 100;

last\_name VARCHAR2(15);

first\_name VARCHAR2(15);

BEGIN

v\_sql\_stmt := 'SELECT teacher\_id, last\_name, first\_name FROM teachers WHERE teacher\_id = :id';

EXECUTE IMMEDIATE v\_sql\_stmt INTO teacher\_id, last\_name, first\_name USING teacher\_id;

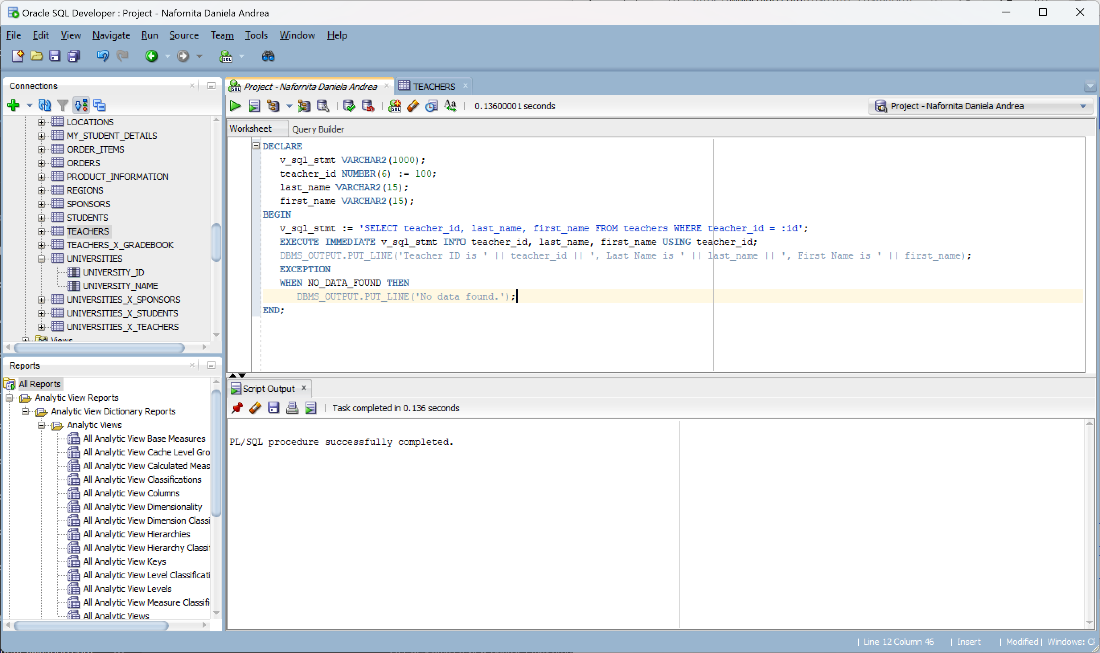
DBMS\_OUTPUT.PUT\_LINE('Teacher ID is ' || teacher\_id || ', Last Name is ' || last\_name || ', First Name is ' || first\_name);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found.');

END;



*1.2.2 LMD (Data Manipulation Language) commands: UPDATE statement using EXECUTE command:*

DECLARE

v\_table\_name VARCHAR2(30) := 'teachers';

BEGIN

EXECUTE IMMEDIATE 'UPDATE ' || v\_table\_name || ' SET FIRST\_NAME = ''Jane Smith'' WHERE TEACHER\_ID = 1';

DBMS\_OUTPUT.PUT\_LINE('Data updated in table ' || v\_table\_name || '.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'sponsors';

BEGIN

EXECUTE IMMEDIATE 'UPDATE ' || v\_table\_name || ' SET SPONSOR\_NAME = ''Jane Smith'' WHERE SPONSOR\_ID = 1';

DBMS\_OUTPUT.PUT\_LINE('Data updated in table ' || v\_table\_name || '.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'Students';

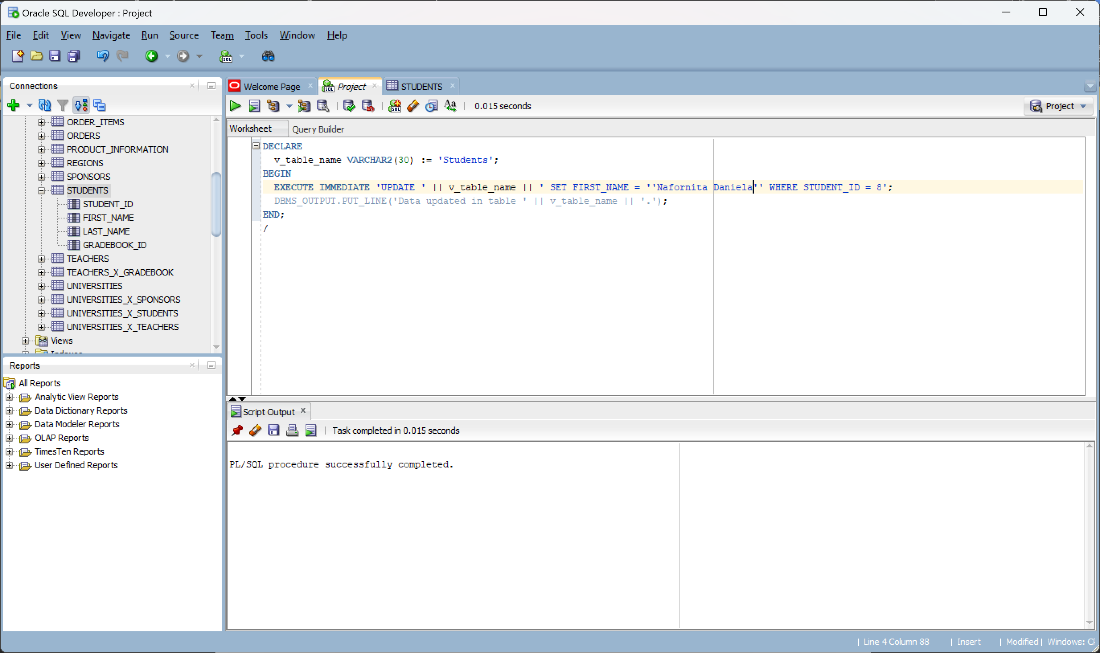
BEGIN

EXECUTE IMMEDIATE 'UPDATE ' || v\_table\_name || ' SET FIRST\_NAME = ''Nafornita Daniela'' WHERE STUDENT\_ID = 8';

DBMS\_OUTPUT.PUT\_LINE('Data updated in table ' || v\_table\_name || '.');

END;

/



*1.2.2 LMD (Data Manipulation Language) commands: DELETE statement using EXECUTE command:*

DECLARE

v\_table\_name VARCHAR2(30) := 'SPONSORS';

BEGIN

EXECUTE IMMEDIATE 'DELETE FROM ' || v\_table\_name || ' WHERE SPONSOR\_ID = 1';

DBMS\_OUTPUT.PUT\_LINE('Data deleted from table ' || v\_table\_name || '.');

END;

/

DECLARE

v\_table\_name VARCHAR2(30) := 'TEACHERS';

BEGIN

EXECUTE IMMEDIATE 'DELETE FROM ' || v\_table\_name || ' WHERE TEACHER\_ID = 1';

DBMS\_OUTPUT.PUT\_LINE('Data deleted from table ' || v\_table\_name || '.');

END;

/

*1.3.1 Interaction with the Oracle server through SQL commands (LDD and LMD): the use of execute immediate, particularities regarding the use of the select command as well as row and group level functions.*

Using row-level functions in a SELECT query:

SELECT FIRST\_NAME, LENGTH(FIRST\_NAME) AS name\_length FROM TEACHERS;

Using group-level functions in a SELECT query:

SELECT TEACHER\_ID, AVG(SALARY) AS avg\_salary FROM TEACHERS GROUP BY TEACHER\_ID;

***2. Alternate and repetitive structures.***

*IF-THEN-ELSE statement:*

DECLARE

SALARY NUMBER;

BEGIN

SELECT SALARY INTO SALARY FROM teachers WHERE teacher\_id = 5;

IF SALARY >= 2000 THEN

DBMS\_OUTPUT.PUT\_LINE('Teacher has not minimum per economy.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Teacher has minimum per economy.');

END IF;

END;

/

*“CASE” statement:*

DECLARE

v\_university\_id NUMBER := 1;

v\_university\_name VARCHAR2(100);

BEGIN

SELECT university\_name INTO v\_university\_name FROM universities WHERE university\_id = v\_university\_id;

CASE v\_university\_name

WHEN 'UAIUM' THEN

DBMS\_OUTPUT.PUT\_LINE('Universitatea Ion Mincu.');

WHEN 'ASE' THEN

DBMS\_OUTPUT.PUT\_LINE('It is Academia de Studii Economice.');

WHEN 'UTM' THEN

DBMS\_OUTPUT.PUT\_LINE('It is Universitatea Titu Maiorescu.');

WHEN 'UMFCD' THEN

DBMS\_OUTPUT.PUT\_LINE('It is Universitatea de medicina si farmaice Carol Davilla.');

WHEN 'UniBuc' THEN

DBMS\_OUTPUT.PUT\_LINE('It is Universitatea Bucuresti.');

WHEN 'UNATC' THEN

DBMS\_OUTPUT.PUT\_LINE('It is Universitatea de Actorie.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Invalid University.');

END CASE;

END;

/

*FOR LOOP statement:*

DECLARE

v\_sum NUMBER := 0;

BEGIN

FOR i IN 2..10 LOOP

SELECT sponsor\_id INTO v\_sum FROM sponsors WHERE sponsor\_id = i;

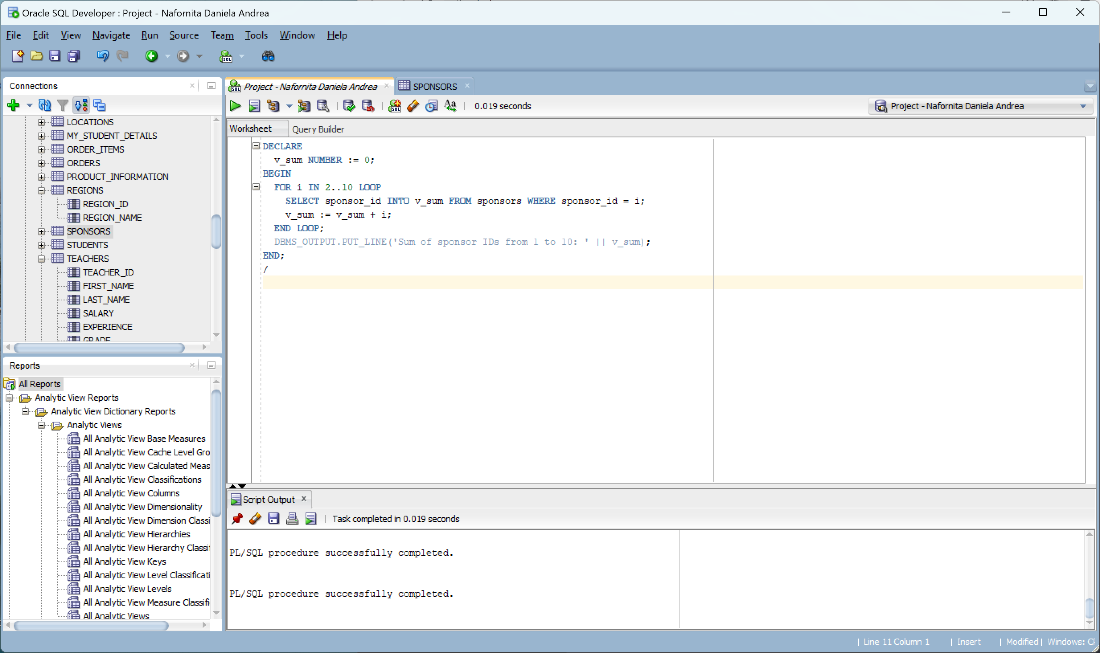
v\_sum := v\_sum + i;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Sum of sponsor IDs from 1 to 10: ' || v\_sum);

END;

/



*WHILE LOOP statement:*

DECLARE

v\_count NUMBER := 1;

v\_student\_id NUMBER;

v\_first\_name VARCHAR2(100);

v\_last\_name VARCHAR2(100);

BEGIN

WHILE v\_count <= 5 LOOP

SELECT student\_id, first\_name, last\_name INTO v\_student\_id, v\_first\_name, v\_last\_name

FROM students WHERE student\_id = v\_count;

DBMS\_OUTPUT.PUT\_LINE('Student ID: ' || v\_student\_id);

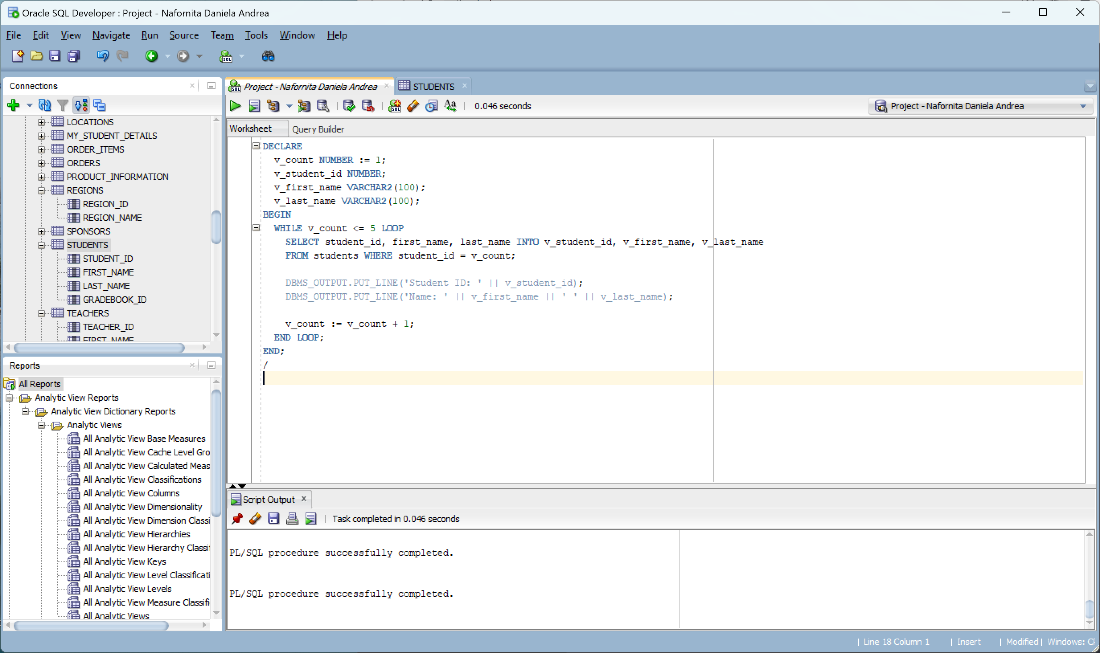
DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_first\_name || ' ' || v\_last\_name);

v\_count := v\_count + 1;

END LOOP;

END;

/



*EXCEPTION statement:*

DECLARE

v\_num1 NUMBER;

v\_num2 NUMBER;

v\_result NUMBER;

BEGIN

SELECT no\_of\_presences, no\_of\_restances

INTO v\_num1, v\_num2

FROM gradebook

WHERE gradebook\_id = 5;

BEGIN

v\_result := v\_num1 / v\_num2;

EXCEPTION

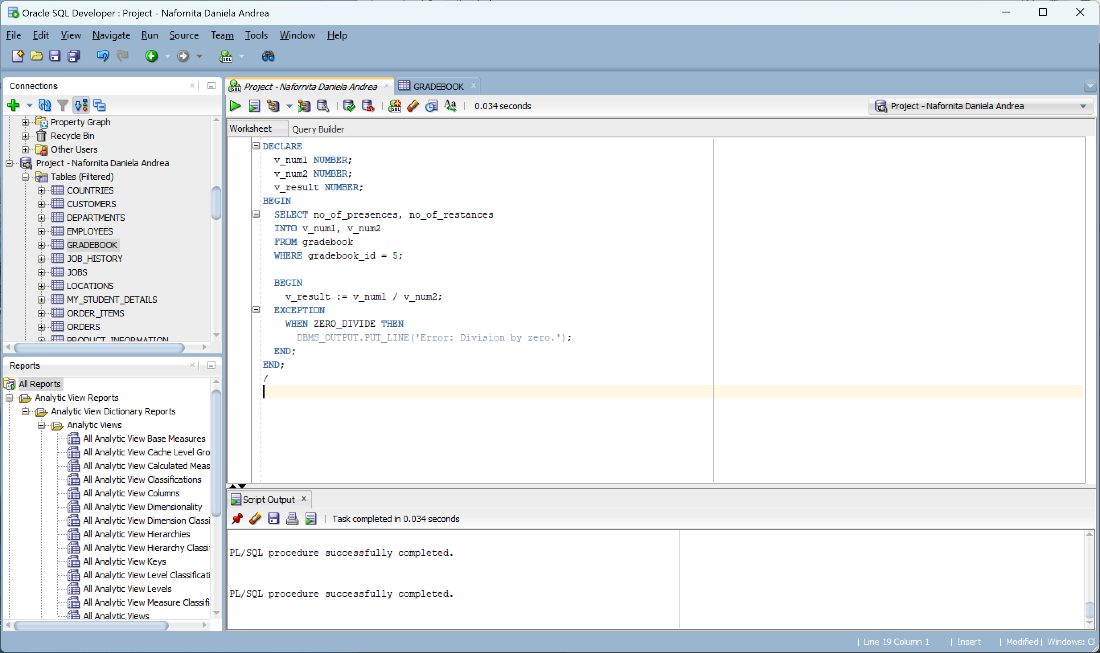
WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division by zero.');

END;

END;

/



*Nested Blocks:*

DECLARE

v\_num1 NUMBER := 10;

v\_num2 NUMBER := 5;

BEGIN

DECLARE

v\_result NUMBER;

v\_teacher\_id NUMBER := 5; -- Specify the teacher ID you want to retrieve data for

BEGIN

SELECT salary, teacher\_id -- Replace column1 and column2 with the actual column names from the teachers table

INTO v\_num1, v\_num2

FROM teachers

WHERE teacher\_id = v\_teacher\_id;

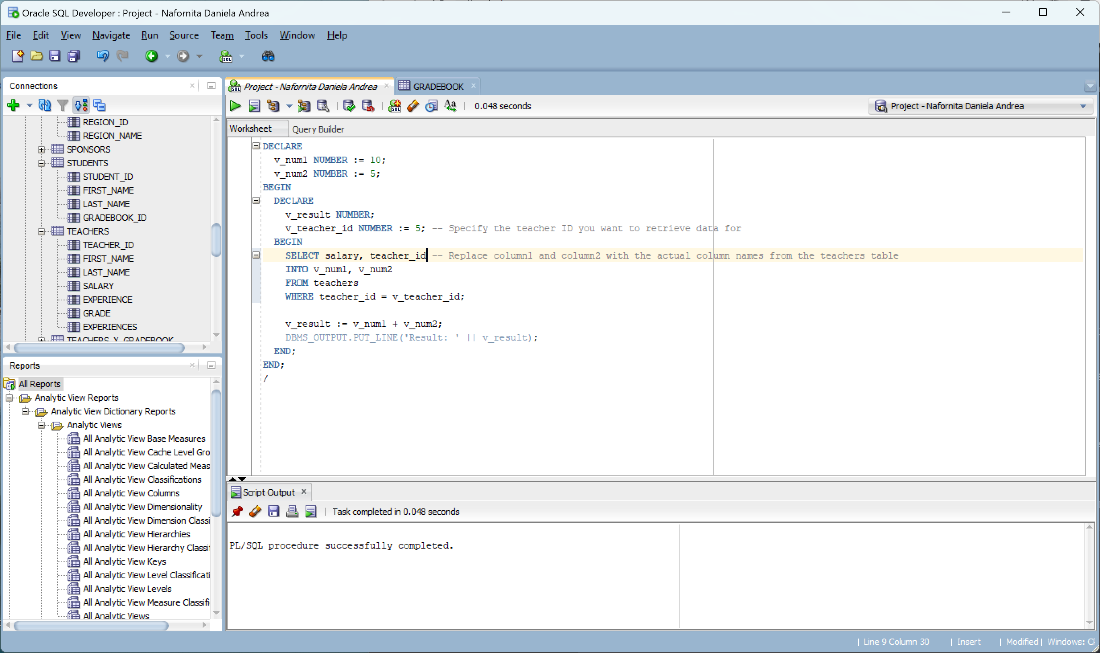
v\_result := v\_num1 + v\_num2;

DBMS\_OUTPUT.PUT\_LINE('Result: ' || v\_result);

END;

END;

/



*GO TO statement:*

DECLARE

v\_count NUMBER := 1;

BEGIN

<<my\_label>>

LOOP

DBMS\_OUTPUT.PUT\_LINE('Count: ' || v\_count);

v\_count := v\_count + 1;

IF v\_count > 5 THEN

EXIT my\_label;

END IF;

END LOOP my\_label;

END;

/

*EXIT statement:*

DECLARE

v\_sum NUMBER := 0;

BEGIN

FOR i IN 1..10 LOOP

v\_sum := v\_sum + i;

IF v\_sum > 20 THEN

EXIT;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Sum: ' || v\_sum);

END;

/

*CONTINUE statement:*

DECLARE

v\_sum NUMBER := 0;

BEGIN

FOR i IN 1..10 LOOP

IF i = 5 THEN

CONTINUE;

END IF;

v\_sum := v\_sum + i;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Sum: ' || v\_sum);

END;

/

*CONTINUE statement:*

DECLARE

CURSOR c\_employees IS

SELECT \* FROM employees;

BEGIN

FOR emp IN c\_employees LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp.name);

END LOOP;

END;

/

***3. Data collections (index by table, nested table, varray)***

*3.1 Index by table (Associative Array) based on the "teachers" table:*

DECLARE

TYPE teacher\_info IS TABLE OF teachers%ROWTYPE INDEX BY PLS\_INTEGER;

v\_teacher\_info teacher\_info;

BEGIN

-- Populating the index by table

FOR rec IN (SELECT \* FROM teachers) LOOP

v\_teacher\_info(rec.teacher\_id) := rec;

END LOOP;

-- Accessing teacher information by teacher ID

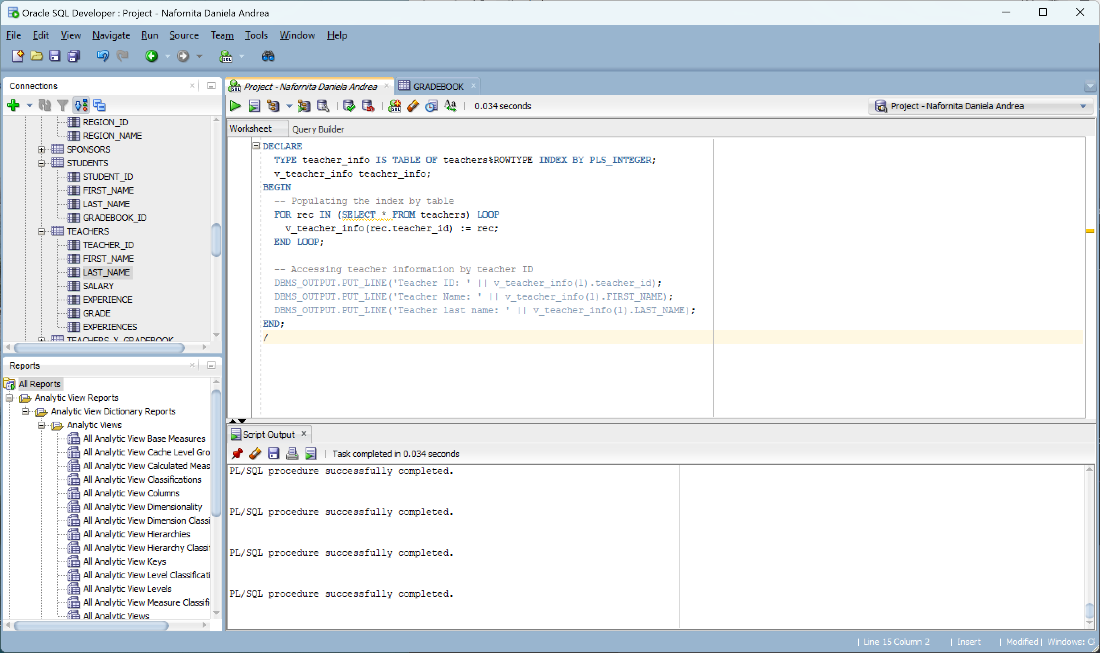
DBMS\_OUTPUT.PUT\_LINE('Teacher ID: ' || v\_teacher\_info(1).teacher\_id);

DBMS\_OUTPUT.PUT\_LINE('Teacher Name: ' || v\_teacher\_info(1).FIRST\_NAME);

DBMS\_OUTPUT.PUT\_LINE('Teacher last name: ' || v\_teacher\_info(1).LAST\_NAME);

END;

/



*3.2 Nested table based on the "Universities" table:*

DECLARE

TYPE university\_list IS TABLE OF universities%ROWTYPE;

v\_university\_list university\_list := university\_list();

-- Populating the nested table

CURSOR c\_universities IS

SELECT \* FROM universities;

BEGIN

FOR rec IN c\_universities LOOP

v\_university\_list.EXTEND;

v\_university\_list(v\_university\_list.LAST) := rec;

END LOOP;

-- Looping through the nested table

FOR i IN v\_university\_list.FIRST..v\_university\_list.LAST LOOP

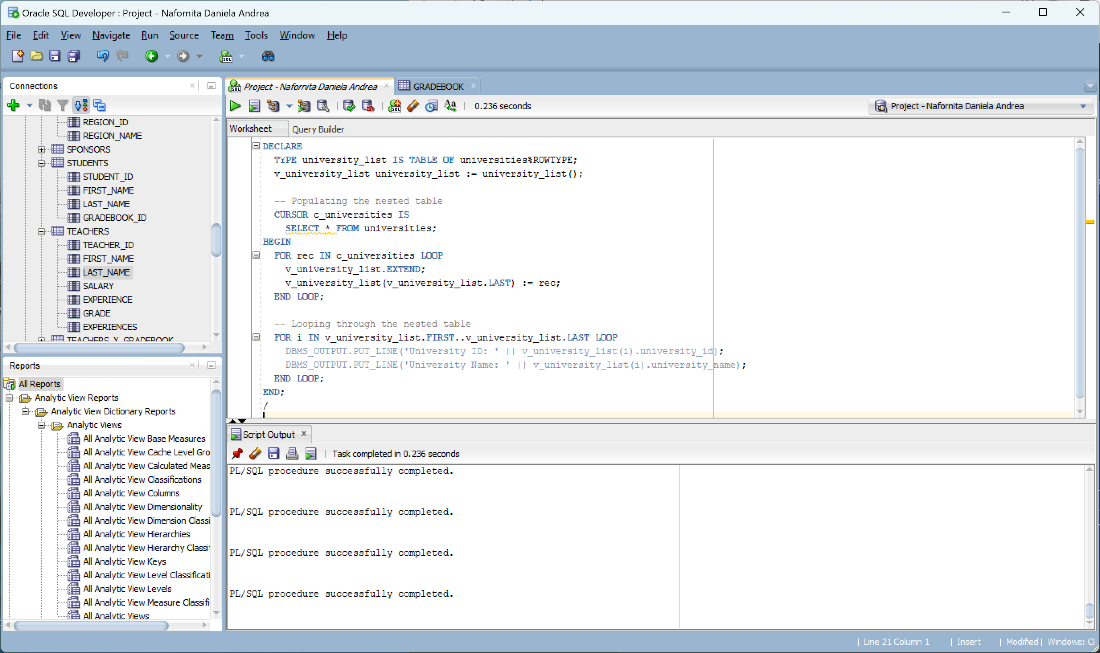
DBMS\_OUTPUT.PUT\_LINE('University ID: ' || v\_university\_list(i).university\_id);

DBMS\_OUTPUT.PUT\_LINE('University Name: ' || v\_university\_list(i).university\_name);

END LOOP;

END;

/



*3.3 Varray (Variable-size Array) based on the "students" table:*

SET SERVEROUTPUT ON

DECLARE

TYPE student\_names IS VARRAY(10) OF students.first\_name%TYPE;

v\_student\_names student\_names := student\_names();

-- Populating the varray

CURSOR c\_students IS

SELECT first\_name FROM students;

BEGIN

FOR rec IN c\_students LOOP

v\_student\_names.EXTEND;

v\_student\_names(v\_student\_names.COUNT) := rec.first\_name;

END LOOP;

-- Looping through the varray

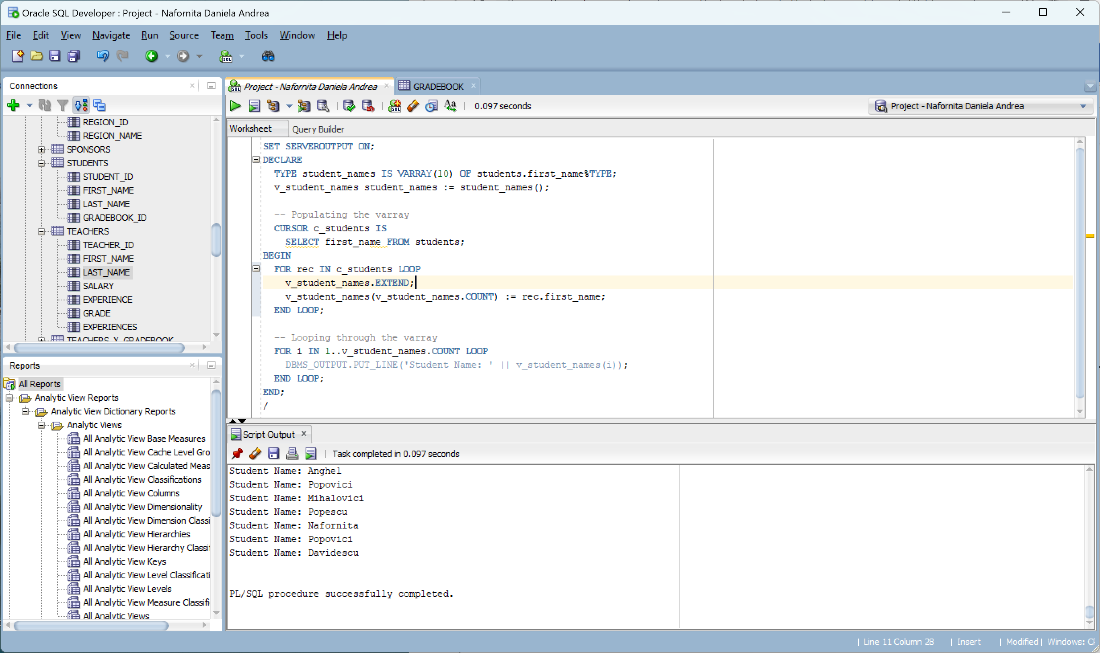
FOR i IN 1..v\_student\_names.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_student\_names(i));

END LOOP;

END;

/



***4. Handling exceptions. (at least 3 implicit, 2 explicit)***

*4.1.1 Implicit exceptions - NO\_DATA\_FOUND Exception:*

BEGIN

DELETE FROM students

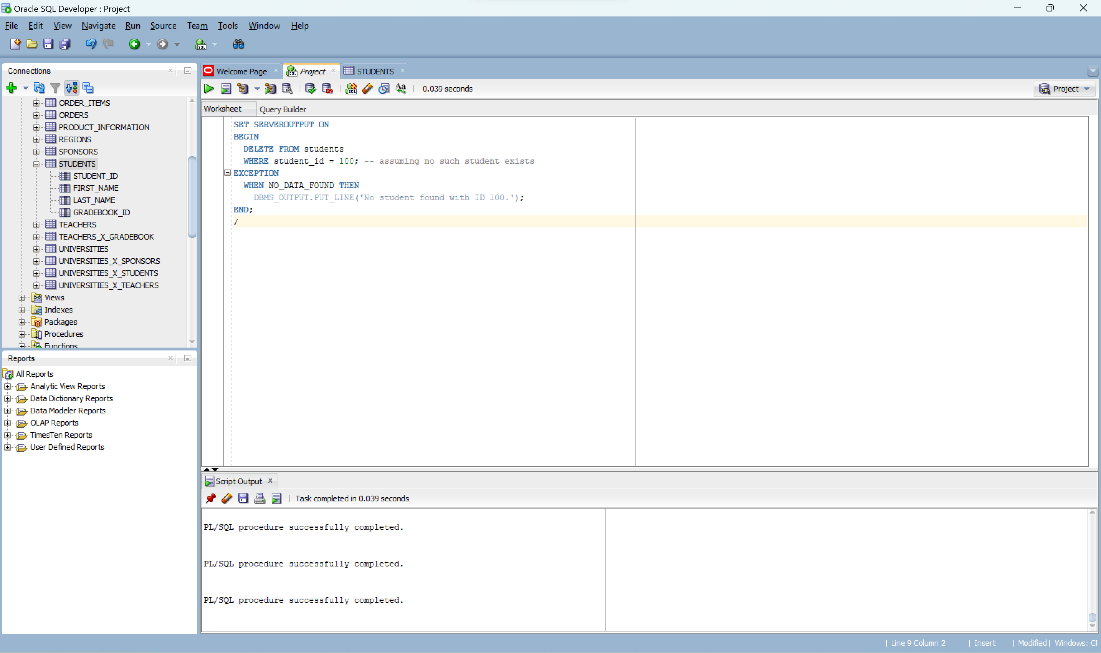
WHERE student\_id = 100; -- assuming no such student exists

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No student found with ID 100.');

END;



*4.1.2 Implicit exceptions - ZERO\_DIVIDE Exception:*

Assume we have a procedure to calculate the average salary of teachers and we're dividing the total salary by the number of teachers. If there are no teachers, this would cause a divide by zero error.

DECLARE

total\_salary NUMBER(8,2);

num\_teachers NUMBER(6);

avg\_salary NUMBER(8,2);

BEGIN

SELECT COUNT(\*) INTO num\_teachers FROM teachers;

SELECT SUM(salary) INTO total\_salary FROM teachers;

avg\_salary := total\_salary / num\_teachers;

DBMS\_OUTPUT.PUT\_LINE('Average Salary: ' || avg\_salary);

EXCEPTION

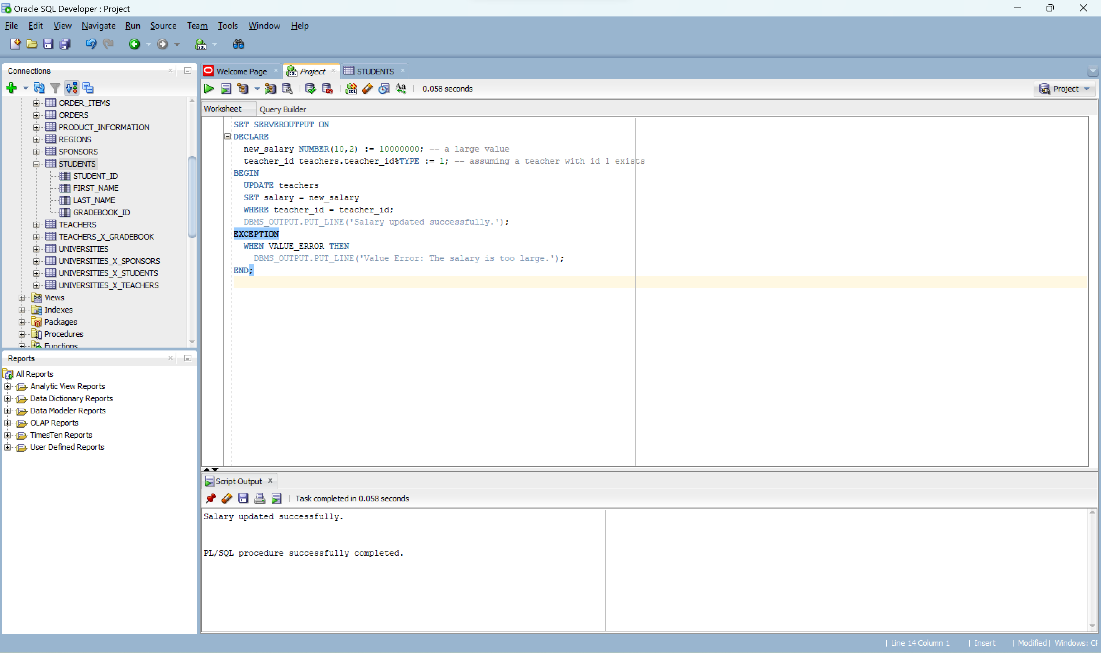
WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Cannot divide by zero. No teachers found.');

END;

*4.1.3 Implicit exceptions - VALUE\_ERROR Exception:*

Assume we're creating a procedure to update the teacher's salary. If the salary value exceeds the maximum value allowed by the **salary** field in the **teachers** table, it would cause a **VALUE\_ERROR**.



*4.2.1 Explicit exceptions -* Explicitly raising an exception:

we are checking if the **salary** of a teacher exceeds a certain limit, and if it does, we raise an exception.

DECLARE

max\_salary\_allowed NUMBER(8,2) := 10000;

high\_salary EXCEPTION;

teacher\_salary teachers.salary%TYPE;

teacher\_id teachers.teacher\_id%TYPE := 1; -- assuming a teacher with id 1 exists

BEGIN

SELECT MAX(salary) INTO teacher\_salary FROM teachers WHERE teacher\_id = teacher\_id;

IF teacher\_salary > max\_salary\_allowed THEN

RAISE high\_salary;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Teacher salary is within the allowed limit.');

EXCEPTION

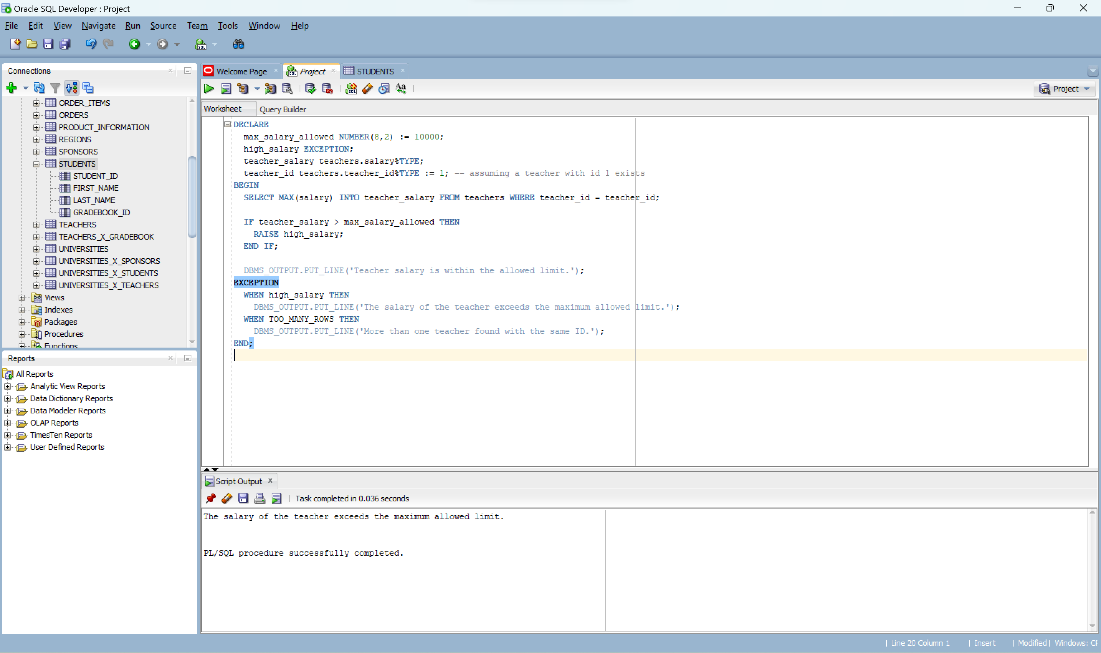
WHEN high\_salary THEN

DBMS\_OUTPUT.PUT\_LINE('The salary of the teacher exceeds the maximum allowed limit.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('More than one teacher found with the same ID.');

END;



*4.2.2 Explicit exceptions - Raising a predefined exception explicitly:*

For instance, let's say we want to create a procedure to increase the salary of a teacher, but we want to put a cap on the maximum allowable salary. If the new salary exceeds this cap, we want to raise the **VALUE\_ERROR** exception.

DECLARE

max\_salary\_allowed NUMBER(8,2) := 10000;

teacher\_salary teachers.salary%TYPE;

teacher\_id teachers.teacher\_id%TYPE := 1; -- assuming a teacher with id 1 exists

BEGIN

SELECT MAX(salary) INTO teacher\_salary FROM teachers WHERE teacher\_id = teacher\_id;

teacher\_salary := teacher\_salary \* 1.1; -- giving a 10% raise

IF teacher\_salary > max\_salary\_allowed THEN

RAISE VALUE\_ERROR;

ELSE

UPDATE teachers SET salary = teacher\_salary WHERE teacher\_id = teacher\_id;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

END IF;

EXCEPTION

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE('The new salary exceeds the maximum allowed limit.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('More than one teacher found with the same ID.');

END;

***5.* Managing cursors: implicit and explicit (with and without parameters)**

**Implicit Cursors**:

They are created automatically when an SQL statement is executed. Here is an example of an implicit cursor using your provided tables. The following block of code retrieves the maximum salary of a teacher and prints it:

DECLARE

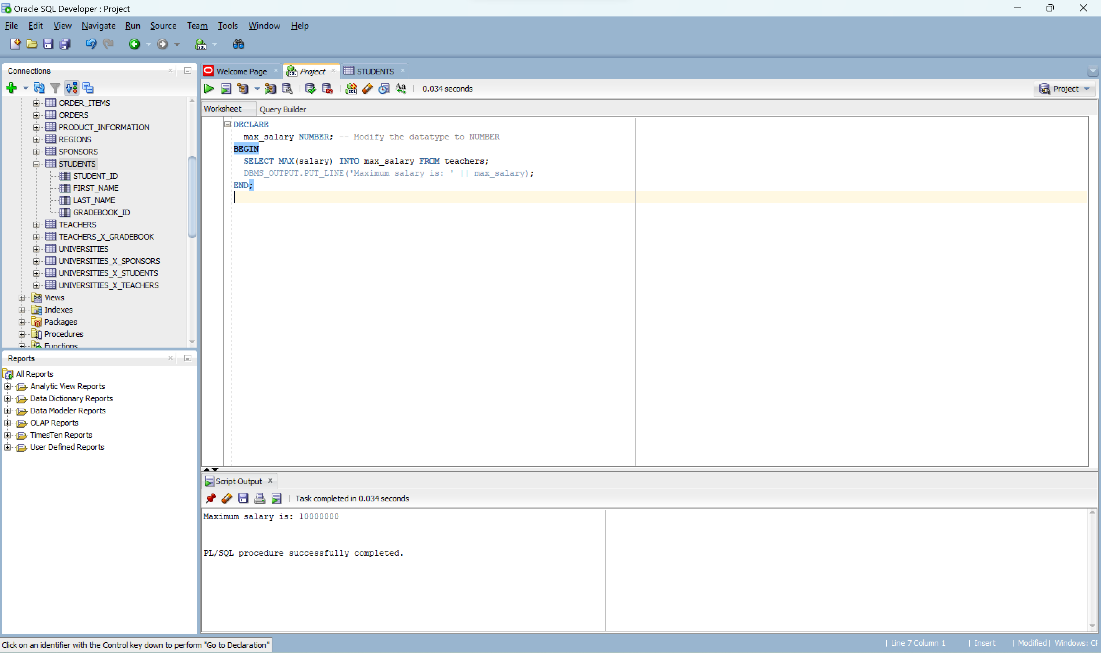
max\_salary NUMBER; -- Modify the datatype to NUMBER

BEGIN

SELECT MAX(salary) INTO max\_salary FROM teachers;

DBMS\_OUTPUT.PUT\_LINE('Maximum salary is: ' || max\_salary);

END;



**Explicit Cursors**:

Explicit cursors are created by the programmers to handle queries that return more than one row. They give more control over the context area than implicit cursors.

E*xample without parameters:*

DECLARE

CURSOR teacher\_cursor IS

SELECT teacher\_id, first\_name, last\_name, salary

FROM teachers;

teacher\_rec teacher\_cursor%ROWTYPE;

BEGIN

OPEN teacher\_cursor;

LOOP

FETCH teacher\_cursor INTO teacher\_rec;

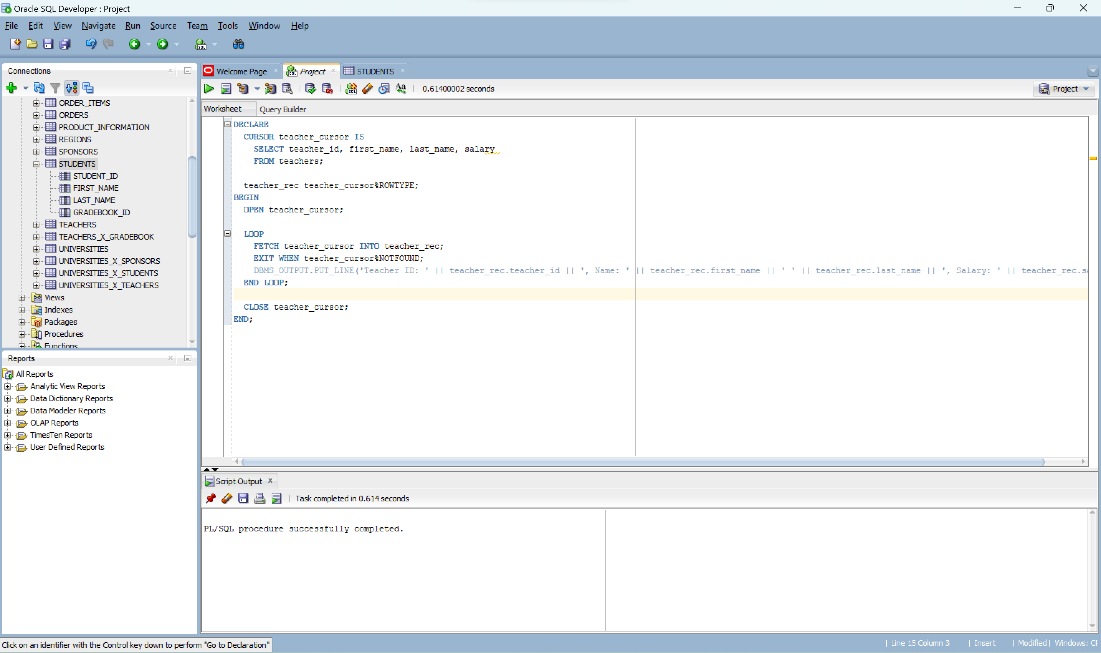
EXIT WHEN teacher\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Teacher ID: ' || teacher\_rec.teacher\_id || ', Name: ' || teacher\_rec.first\_name || ' ' || teacher\_rec.last\_name || ', Salary: ' || teacher\_rec.salary);

END LOOP;

CLOSE teacher\_cursor;

END;



*Example with parameters:*

DECLARE

CURSOR teacher\_cursor (p\_teacher\_id teachers.teacher\_id%TYPE) IS

SELECT teacher\_id, first\_name, last\_name, salary

FROM teachers

WHERE teacher\_id = p\_teacher\_id;

teacher\_rec teacher\_cursor%ROWTYPE;

BEGIN

OPEN teacher\_cursor(1); -- let's assume we're fetching the teacher with id 1

FETCH teacher\_cursor INTO teacher\_rec;

IF teacher\_cursor%FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Teacher ID: ' || teacher\_rec.teacher\_id || ', Name: ' || teacher\_rec.first\_name || ' ' || teacher\_rec.last\_name || ', Salary: ' || teacher\_rec.salary);

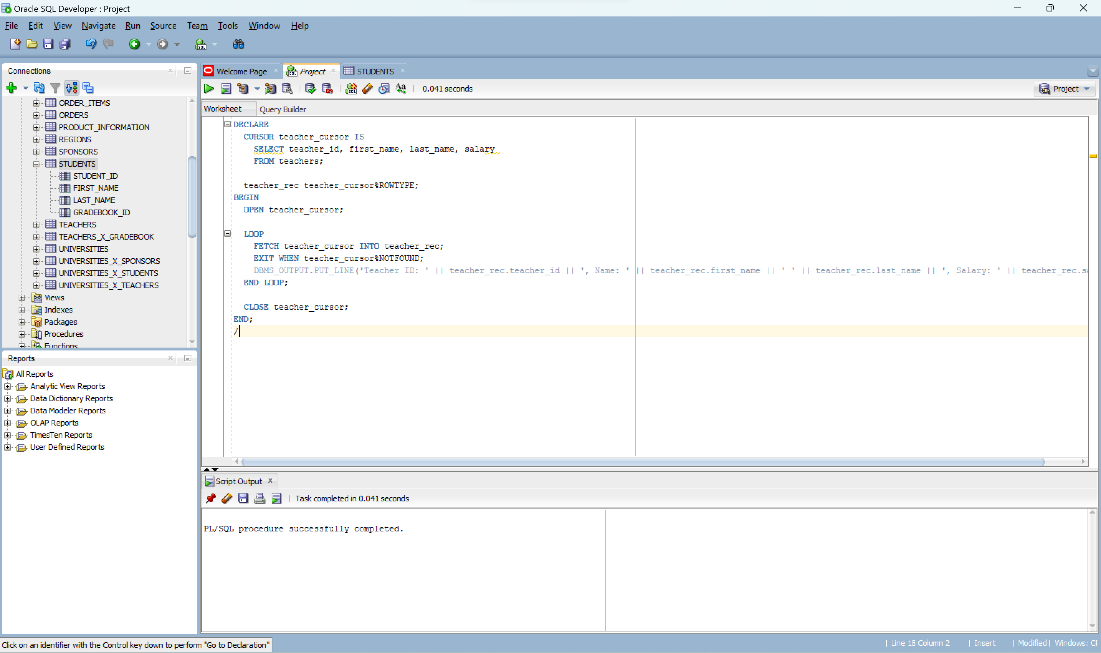
ELSE

DBMS\_OUTPUT.PUT\_LINE('No teacher found with the provided ID.');

END IF;

CLOSE teacher\_cursor;

END;



***6.* Functions, procedures, including these in packages. (at least 3 functions, 3 procedures, and a package that includes other functions and procedures)**

**1. Function to get Total Number of Sponsors**

CREATE OR REPLACE FUNCTION get\_total\_sponsors

RETURN INT

AS

total\_sponsors INT;

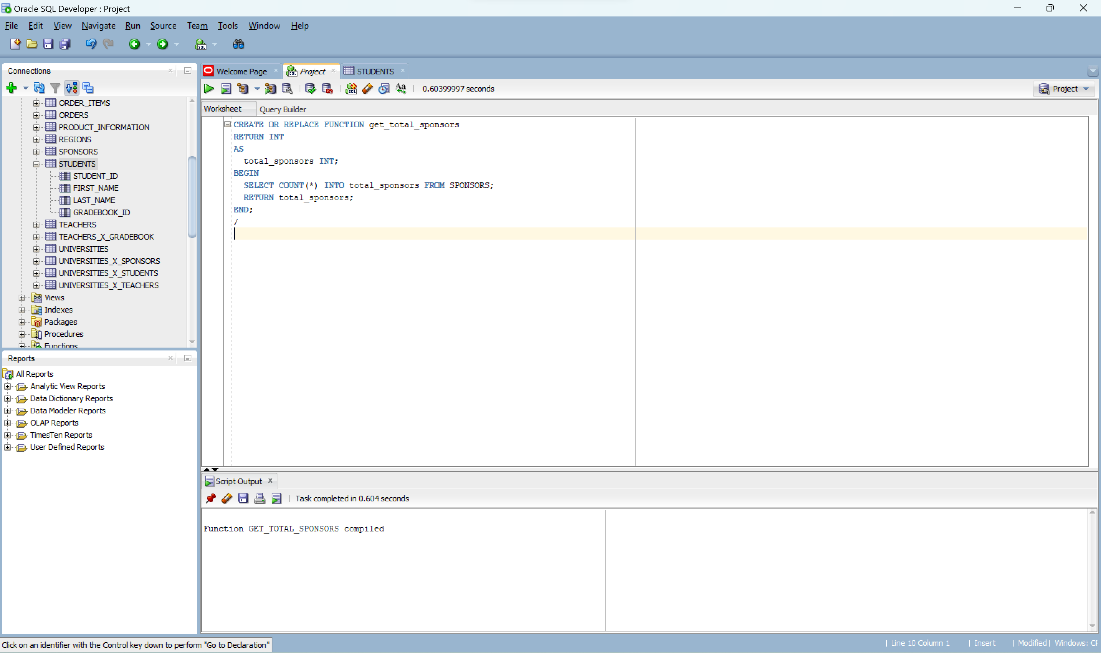
BEGIN

SELECT COUNT(\*) INTO total\_sponsors FROM SPONSORS;

RETURN total\_sponsors;

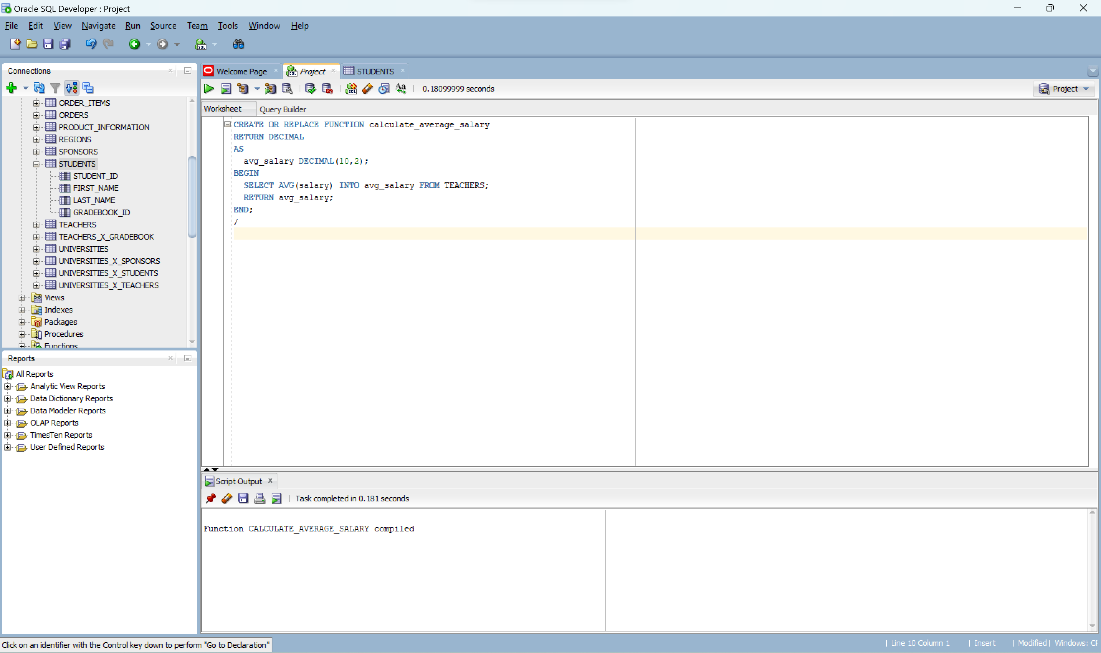
END;

/



**2. Procedure to display and Calculate Average Salary of Teachers**

CREATE OR REPLACE FUNCTION calculate\_average\_salary

RETURN DECIMAL

AS

avg\_salary DECIMAL(10,2);

BEGIN

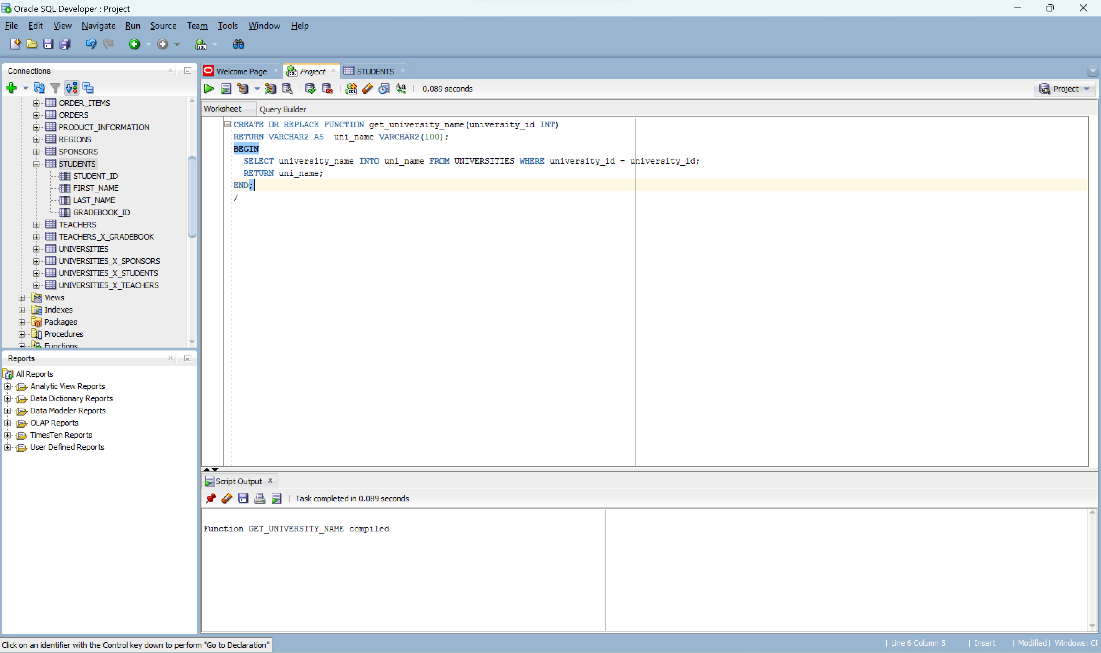
SELECT AVG(salary) INTO avg\_salary FROM TEACHERS;

RETURN avg\_salary;

END;

/

**3. This function retrieves the name of a university based on the given university ID**



CREATE OR REPLACE FUNCTION get\_university\_name(university\_id INT)

RETURN VARCHAR2 AS uni\_name VARCHAR2(100);

BEGIN

SELECT university\_name INTO uni\_name FROM UNIVERSITIES WHERE university\_id = university\_id;

RETURN uni\_name;

END;

/

**4. This procedure adds a new sponsor to the database.**

CREATE OR REPLACE PROCEDURE add\_new\_sponsor(

p\_sponsor\_id INT,

p\_sponsorship\_type VARCHAR,

p\_sponsor\_name VARCHAR

)

AS

BEGIN

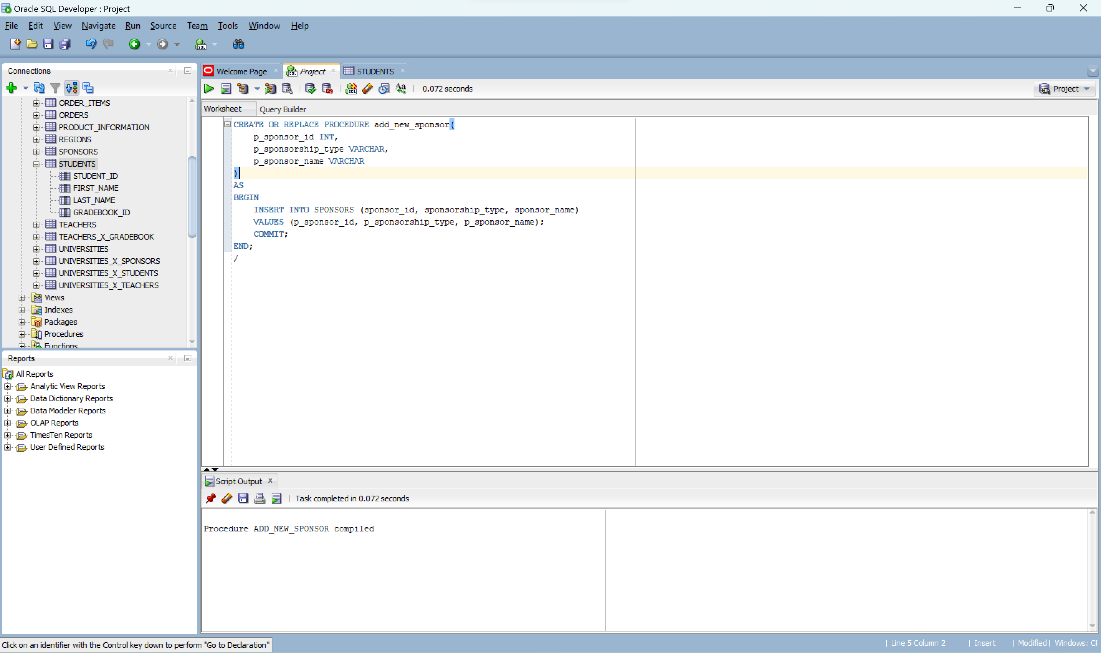
INSERT INTO SPONSORS (sponsor\_id, sponsorship\_type, sponsor\_name)

VALUES (p\_sponsor\_id, p\_sponsorship\_type, p\_sponsor\_name);

COMMIT;

END;

/



**5. This procedure updates the salary of a specific teacher.**

CREATE OR REPLACE PROCEDURE update\_teacher\_salary(

p\_teacher\_id INT,

p\_new\_salary DECIMAL

)

AS

BEGIN

UPDATE TEACHERS

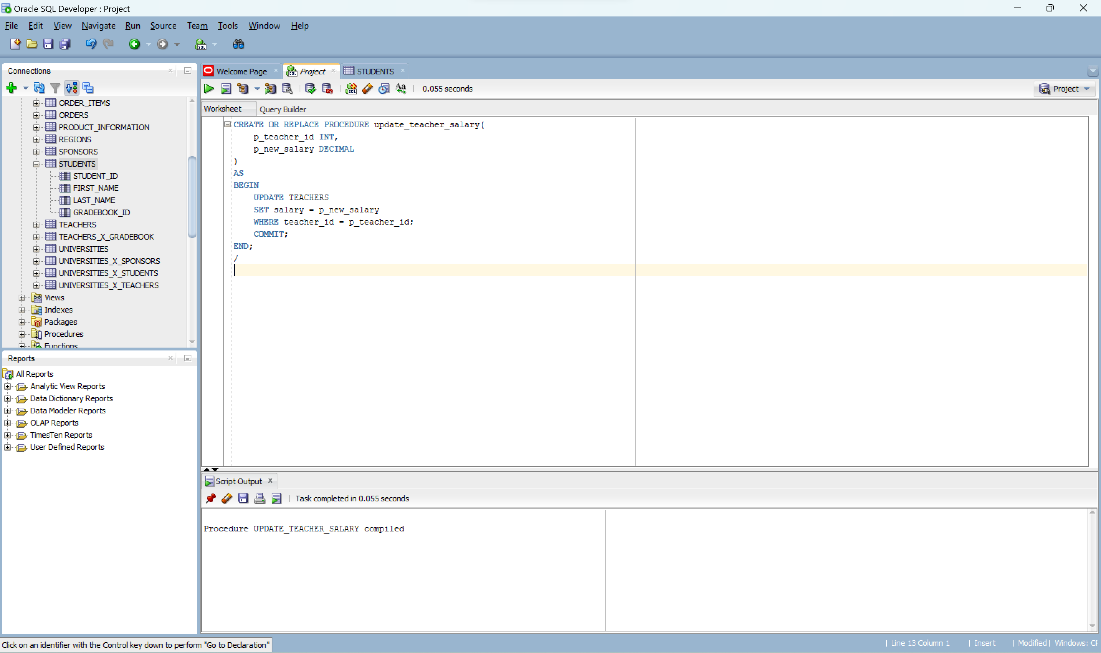
SET salary = p\_new\_salary

WHERE teacher\_id = p\_teacher\_id;

COMMIT;

END;

/



**6. This procedure deletes a student and the associated gradebook from the database.**

CREATE OR REPLACE PROCEDURE delete\_student(p\_student\_id INT)

AS

BEGIN

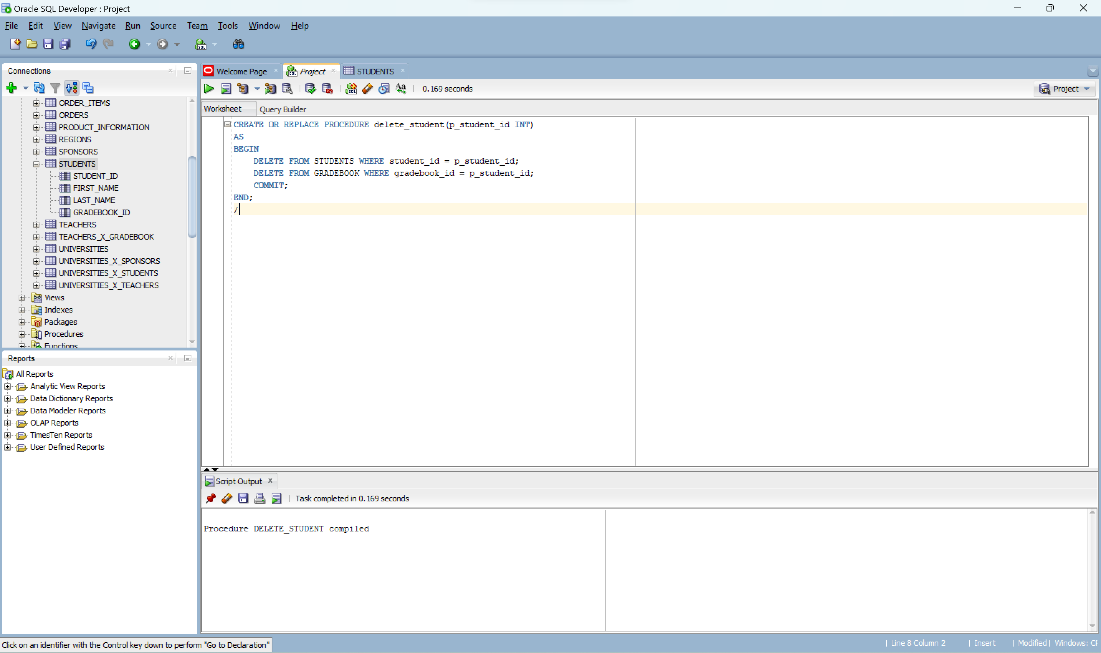
DELETE FROM STUDENTS WHERE student\_id = p\_student\_id;

DELETE FROM GRADEBOOK WHERE gradebook\_id = p\_student\_id;

COMMIT;

END;

/



**7. This package includes functions and procedures related to universities.**

CREATE PACKAGE University1\_Package AS

FUNCTION get\_total\_universities RETURN INT;

PROCEDURE add\_new\_university(

p\_university\_id INT,

p\_university\_name VARCHAR

);

PROCEDURE update\_university\_name(

p\_university\_id INT,

p\_new\_name VARCHAR

);

END University1\_Package;

***6. statement-level and row-level triggers***

1. *Statement-Level Trigger:*

CREATE OR REPLACE TRIGGER trg\_statement\_level

AFTER INSERT OR UPDATE OR DELETE ON SPONSORS

DECLARE

total\_sponsors INT;

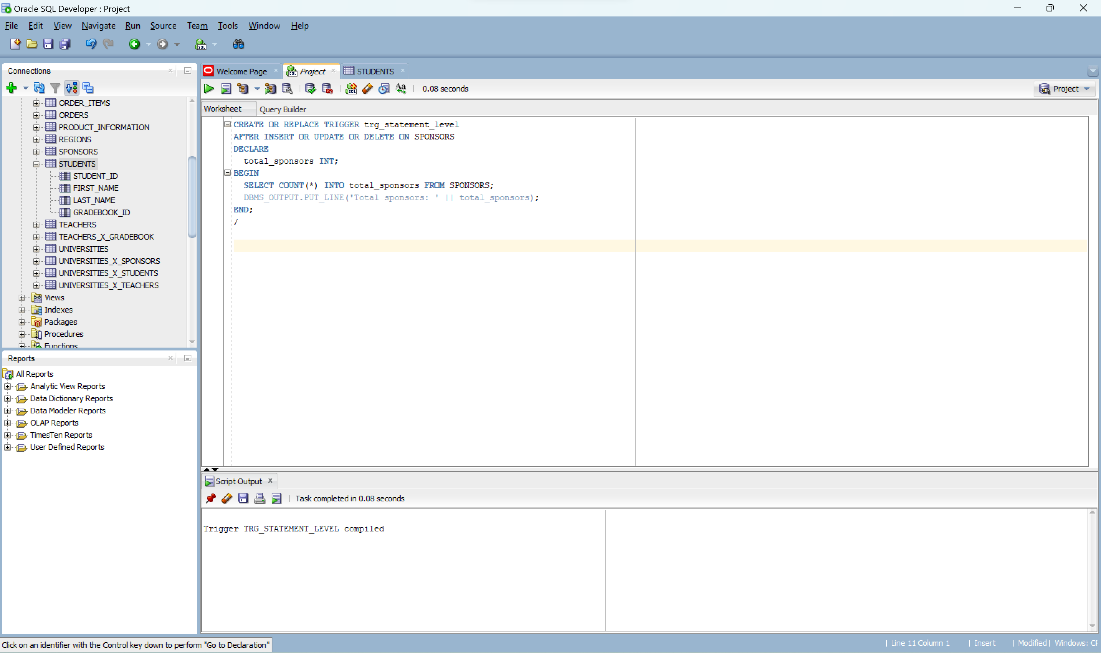
BEGIN

SELECT COUNT(\*) INTO total\_sponsors FROM SPONSORS;

DBMS\_OUTPUT.PUT\_LINE('Total sponsors: ' || total\_sponsors);

END;

/



1. *Row-Level Trigger:*

CREATE OR REPLACE TRIGGER trg\_row\_level

AFTER INSERT OR UPDATE OF no\_of\_presences ON GRADEBOOK

FOR EACH ROW

BEGIN

IF :NEW.no\_of\_presences < 10 THEN

DBMS\_OUTPUT.PUT\_LINE('Low attendance! Student: ' || :NEW.gradebook\_id);

END IF;

END;

/

