Module Title: Advanced Databases and Modelling

Module Code: IS3S662

Module Leader/Tutor: Joseph Griffiths

Assessment Title: Set exercise / tasks - not-time constrained

Student Name: Dilukshika Sivanathan

Student No: 30073395



Table of Contents

[Introduction 3](#_Toc184740204)

[Database Schema Design 3](#_Toc184740205)

[Tables Created 3](#_Toc184740206)

[Question 01) 3](#_Toc184740207)

[1. Create the tables of courses and modules 3](#_Toc184740208)

[Output Screenshot 4](#_Toc184740209)

[2. Write the PL/SQL procedure to insert records into both tables with validation. 5](#_Toc184740210)

[Output Screenshots: 7](#_Toc184740211)

[3. Populate the Tables using data from the appendix 9](#_Toc184740212)

[Question 02) 17](#_Toc184740213)

[Output Screenshots 18](#_Toc184740214)

[Question 03) 20](#_Toc184740215)

[Output Screenshots 22](#_Toc184740216)

[Question 04) 25](#_Toc184740217)

[Output Screenshots 27](#_Toc184740218)

[Question 5) 30](#_Toc184740219)

[Output Screenshots 31](#_Toc184740220)

[Question 6) 35](#_Toc184740221)

[Output Screenshots 36](#_Toc184740222)

[Reference 39](#_Toc184740223)

# Introduction

This report summarizes a PL/SQL database management system designed to handle course and module tracking. The assignment demonstrates advanced database programming techniques, including table creation, data validation, procedural logic, and database manipulation.

# Database Schema Design

## Tables Created

1. Course Table

* course\_id (VARCHAR2, Primary Key)
* title (VARCHAR2)
* description (VARCHAR2)
* course\_leader (VARCHAR2)
* date\_modified (DATE)
* number\_of\_credits (NUMBER)

1. Module Table

* module\_id (VARCHAR2, Primary Key)
* course\_id (VARCHAR2, Foreign Key referencing course table)

# Question 01)

Steps for the populate the table:

Create the tables of courses and modules.

Write the PL/SQL procedure to insert records into both tables with validation.

Populate the tables using data from the appendix.

## Create the tables of courses and modules

-- Create the course and modules tables

CREATE TABLE course (

course\_id VARCHAR2(13) NOT NULL PRIMARY KEY,

title VARCHAR2(200),

description VARCHAR2(2000),

course\_leader VARCHAR2(200),

date\_modified DATE,

number\_of\_credits NUMBER

);

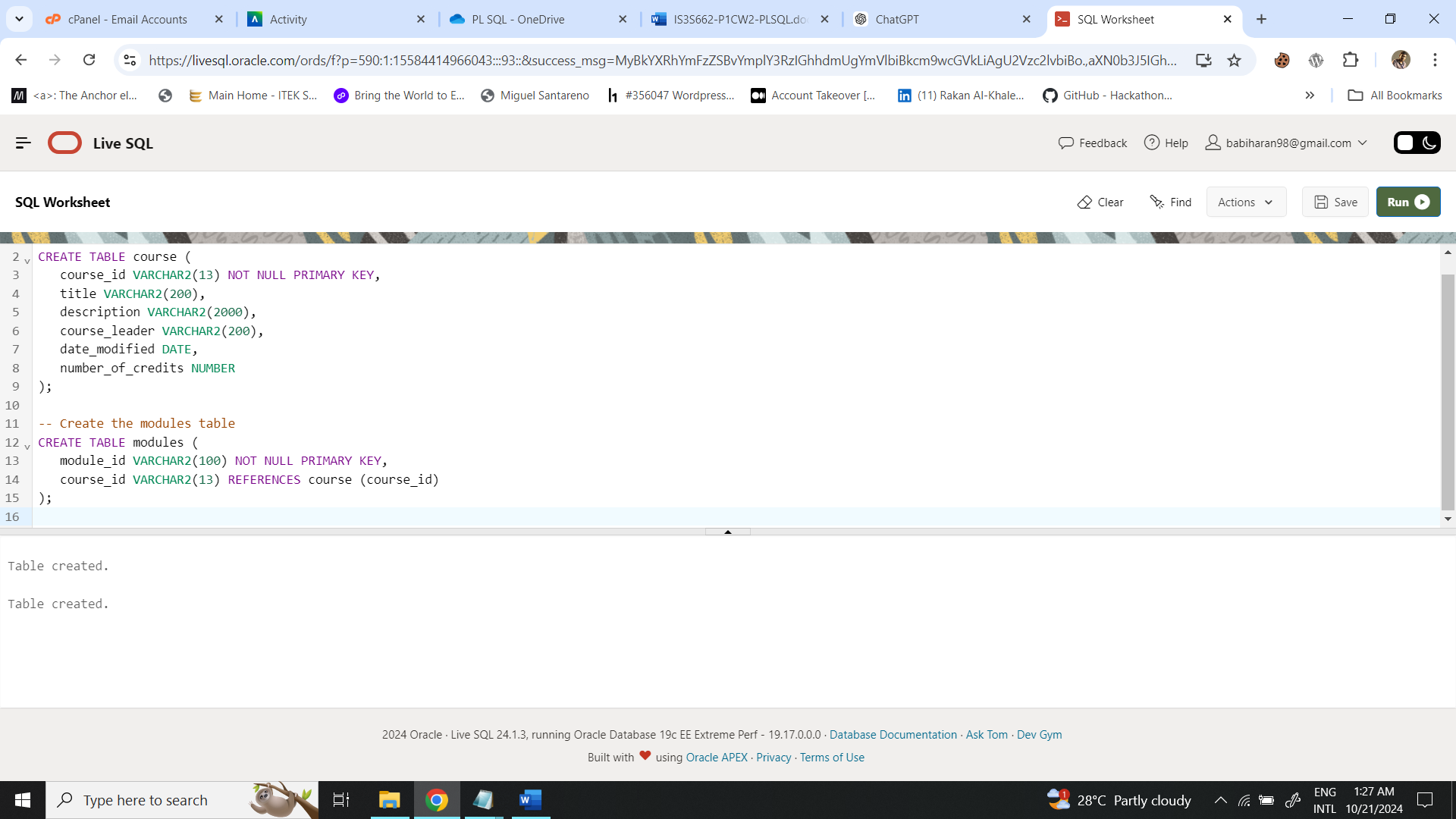
CREATE TABLE modules (

module\_id VARCHAR2(100) NOT NULL PRIMARY KEY,

course\_id VARCHAR2(13) REFERENCES course (course\_id)

);

## Output Screenshot



## ***Write the PL/SQL procedure to insert records into both tables with validation.***

CREATE OR REPLACE PROCEDURE add\_course\_and\_modules (

p\_course\_id IN VARCHAR2,

p\_title IN VARCHAR2,

p\_description IN VARCHAR2,

p\_course\_leader IN VARCHAR2,

p\_date\_modified IN DATE,

p\_number\_of\_credits IN NUMBER,

p\_module\_ids IN SYS.ODCIVARCHAR2LIST -- Using Oracle's collection type

) IS

-- Exception to handle invalid input

invalid\_input EXCEPTION;

BEGIN

-- Validate input for course\_id, title, course\_leader, and number\_of\_credits

IF p\_course\_id IS NULL OR p\_title IS NULL OR p\_course\_leader IS NULL OR p\_number\_of\_credits IS NULL THEN

RAISE invalid\_input;

END IF;

IF p\_number\_of\_credits <= 0 THEN

RAISE invalid\_input;

END IF;

-- Insert data into the course table

INSERT INTO course (

course\_id, title, description, course\_leader, date\_modified, number\_of\_credits

) VALUES (

p\_course\_id, p\_title, p\_description, p\_course\_leader, p\_date\_modified, p\_number\_of\_credits

);

-- Insert corresponding modules into the modules table

FOR i IN 1..p\_module\_ids.COUNT LOOP

INSERT INTO modules (module\_id, course\_id)

VALUES (p\_module\_ids(i), p\_course\_id);

END LOOP;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN invalid\_input THEN

-- Rollback in case of invalid input

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Invalid input values.');

WHEN OTHERS THEN

-- Handle other exceptions

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

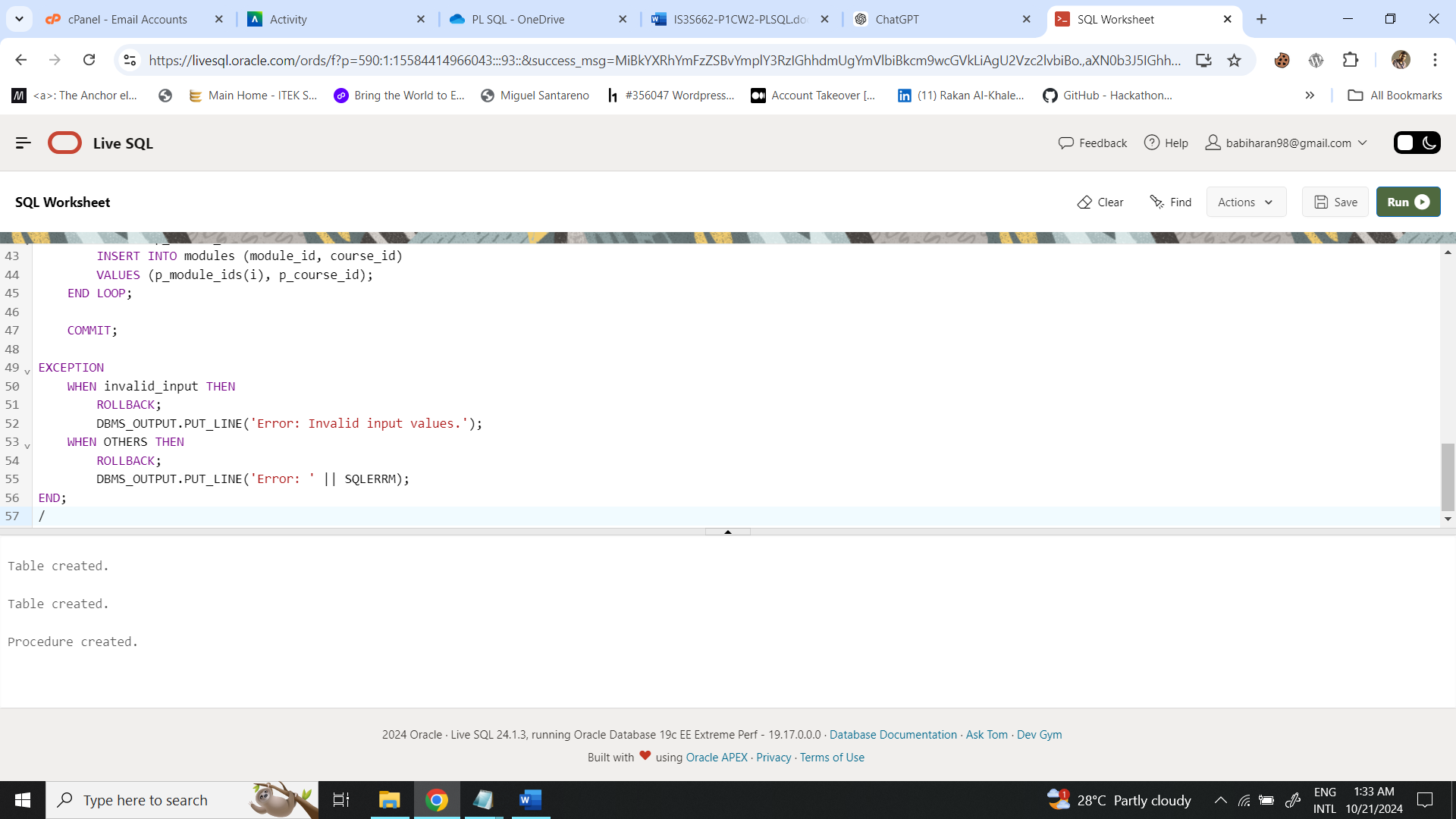
END;

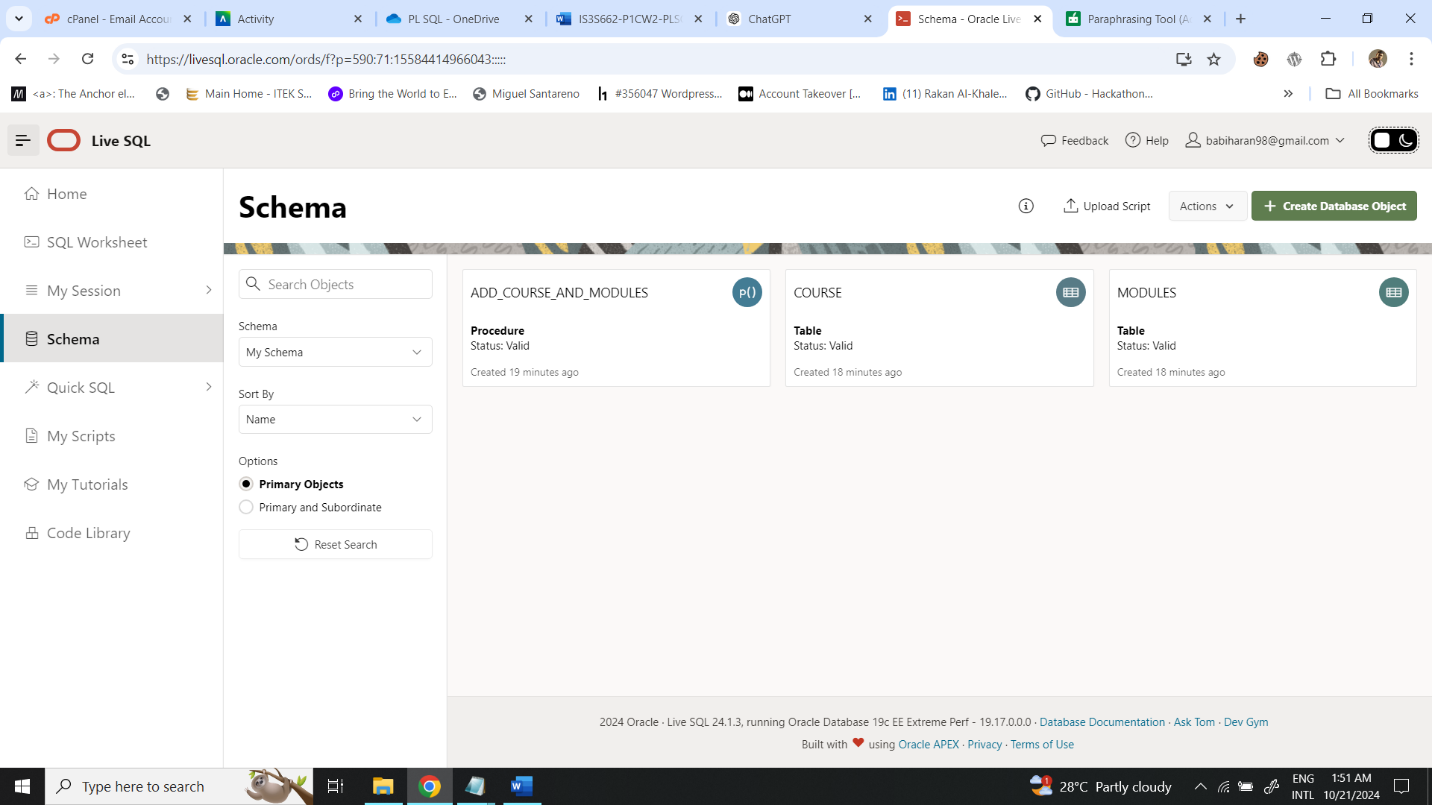
/

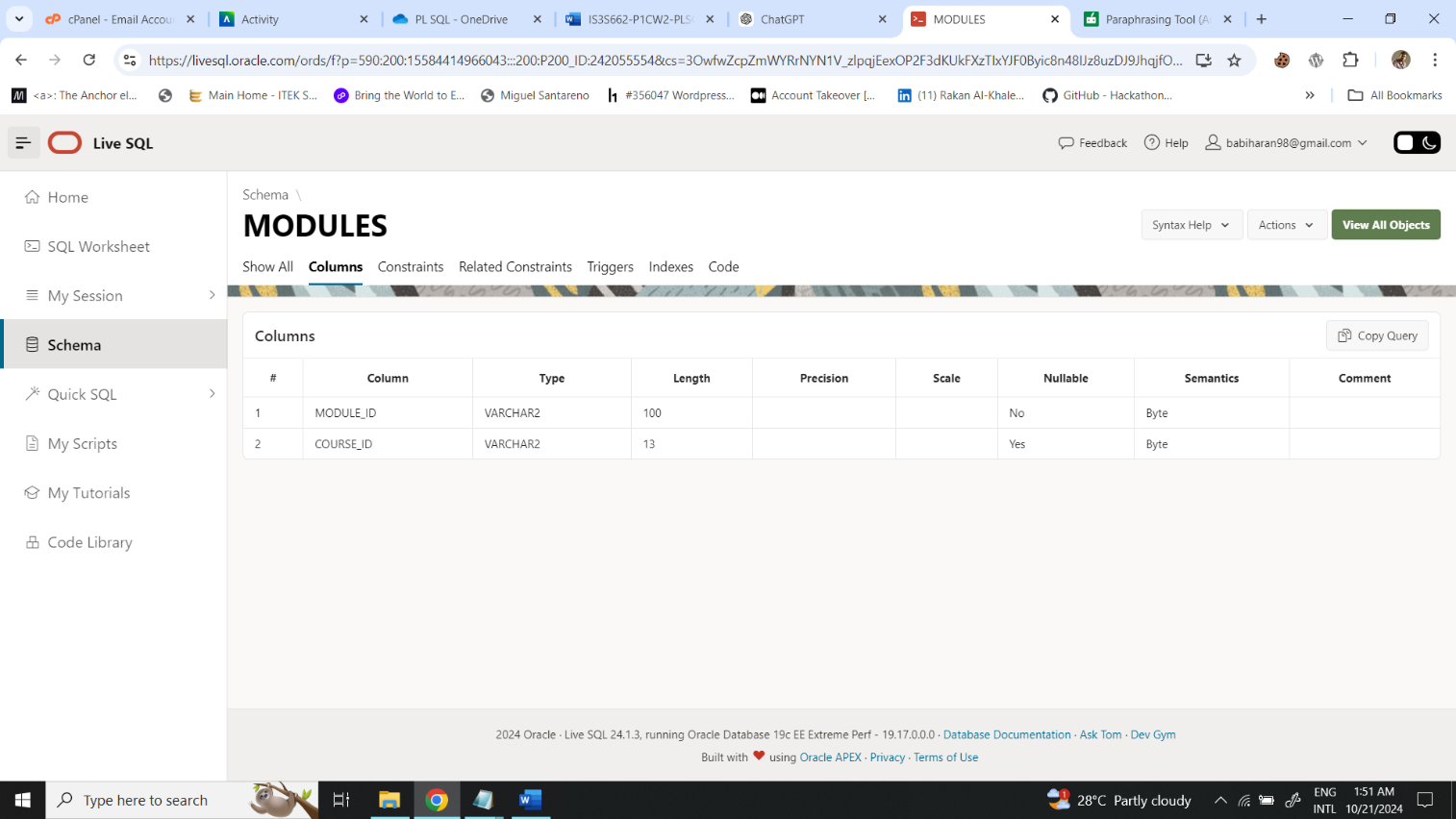
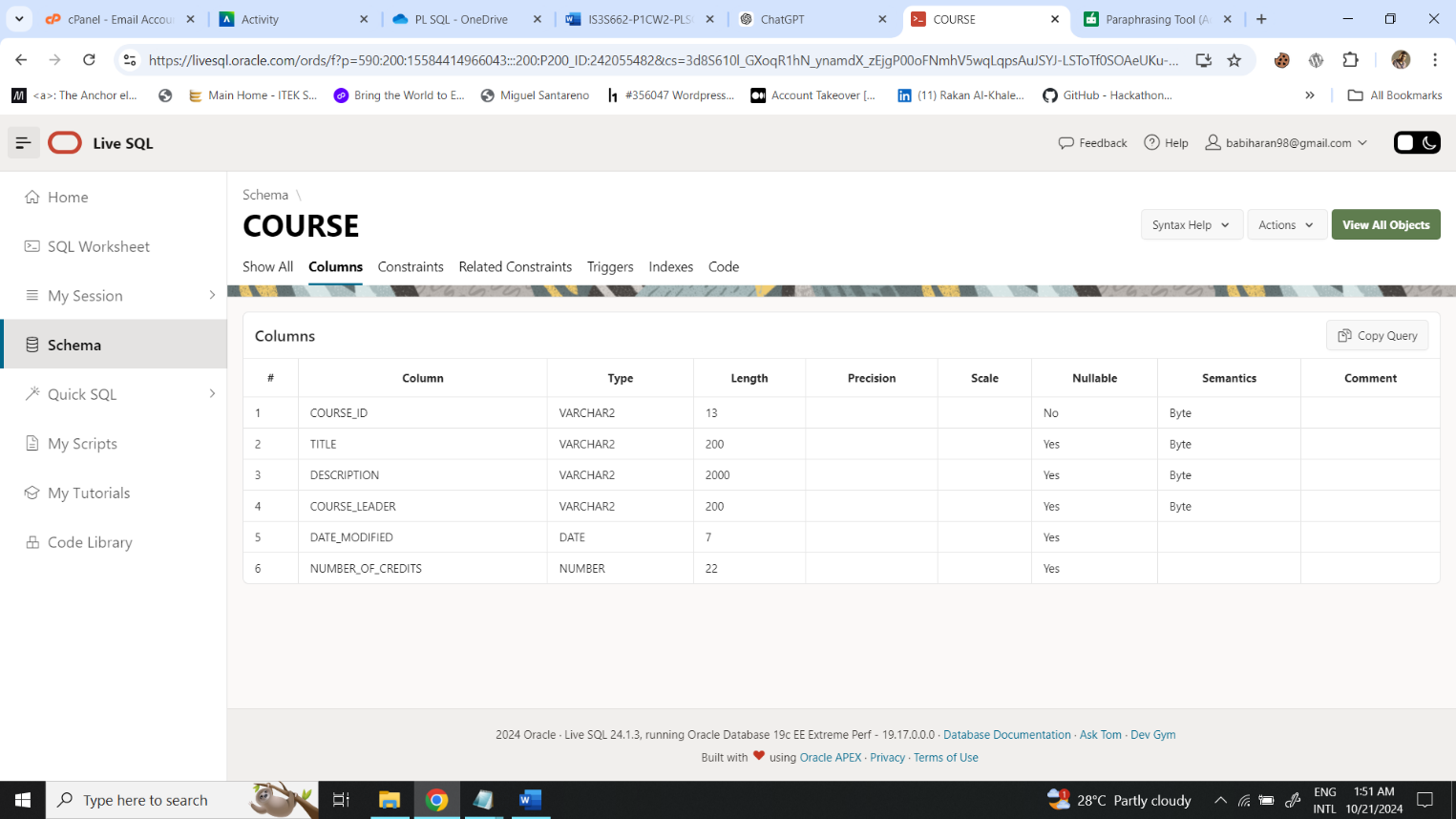
**Explanation:**

* **Input Validation:** The procedure verifies that if course\_id, title, course\_leader, and number\_of\_credits are provided and if number\_of\_credits is a positive number.
* **Inserting Course:** This procedure inserts a new course record into the course table.
* **Inserting Modules:** A collection (Oracle's SYS.ODCIVARCHAR2LIST) is used to pass multiple module\_id values, and each module is inserted into the modules table.
* **Exception Handling:** The transaction is rolled back if any errors are found, including invalid inputs.

## Output Screenshots:







## Populate the Tables using data from the appendix

1. **Inserting Course C001 (Computing)**

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('IS1S464', 'IS3S662');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C001',

p\_title => 'Computing',

p\_description => 'Focuses on developing skills that employers demand for roles in the IT industry.',

p\_course\_leader => 'Richard Jones',

p\_date\_modified => TO\_DATE('20-APR-2024', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

);

END;

/

1. **Inserting Course C002 (Business)**

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('BS1S737', 'BS3S374');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C002',

p\_title => 'Business',

p\_description => 'Focuses on management and looks at the different functions of business and how they inter-relate.',

p\_course\_leader => 'Thomas Page',

p\_date\_modified => TO\_DATE('03-MAR-2021', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

);

END;

/

1. **Inserting Course C003 (History)**

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('HS3S773', 'HS2S484');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C003',

p\_title => 'History',

p\_description => 'Focus on modern history, from about 1450 to the present day, including regional, national, European, American and global perspectives.',

p\_course\_leader => 'Adam Richards',

p\_date\_modified => TO\_DATE('12-MAY-2023', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

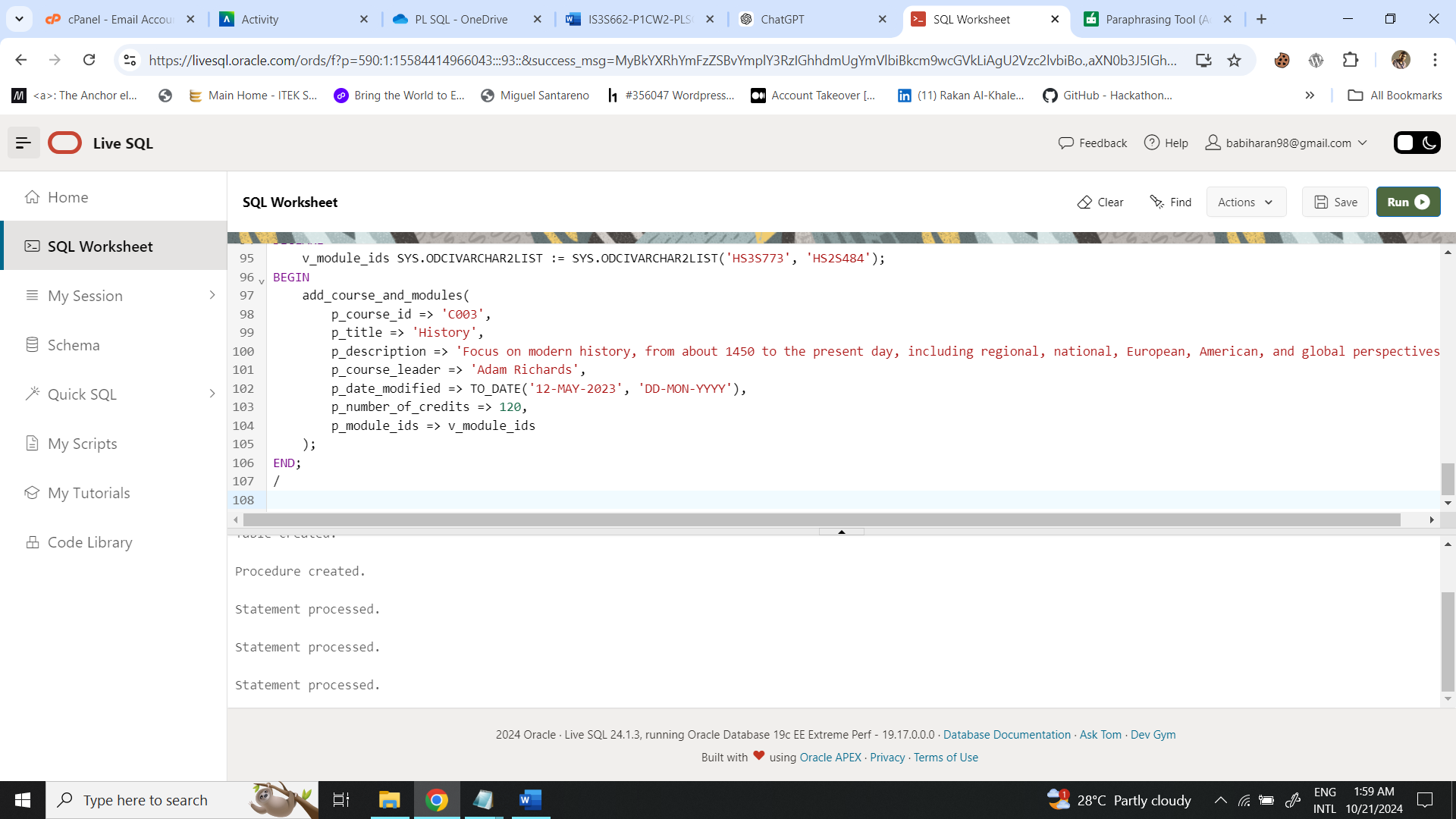
);

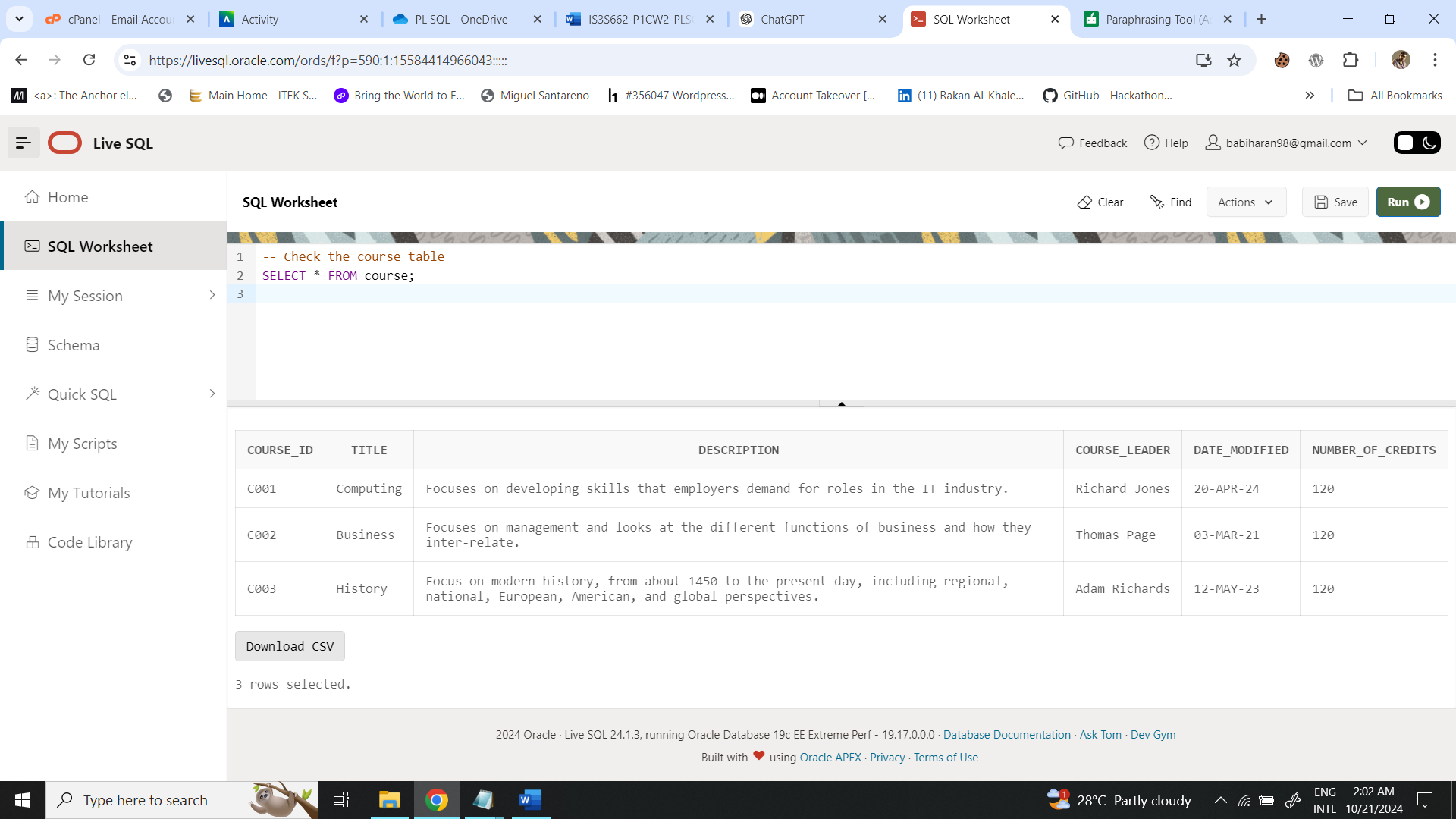
END;

/

**To verify that the records were inserted correctly, run the following SQL queries:**

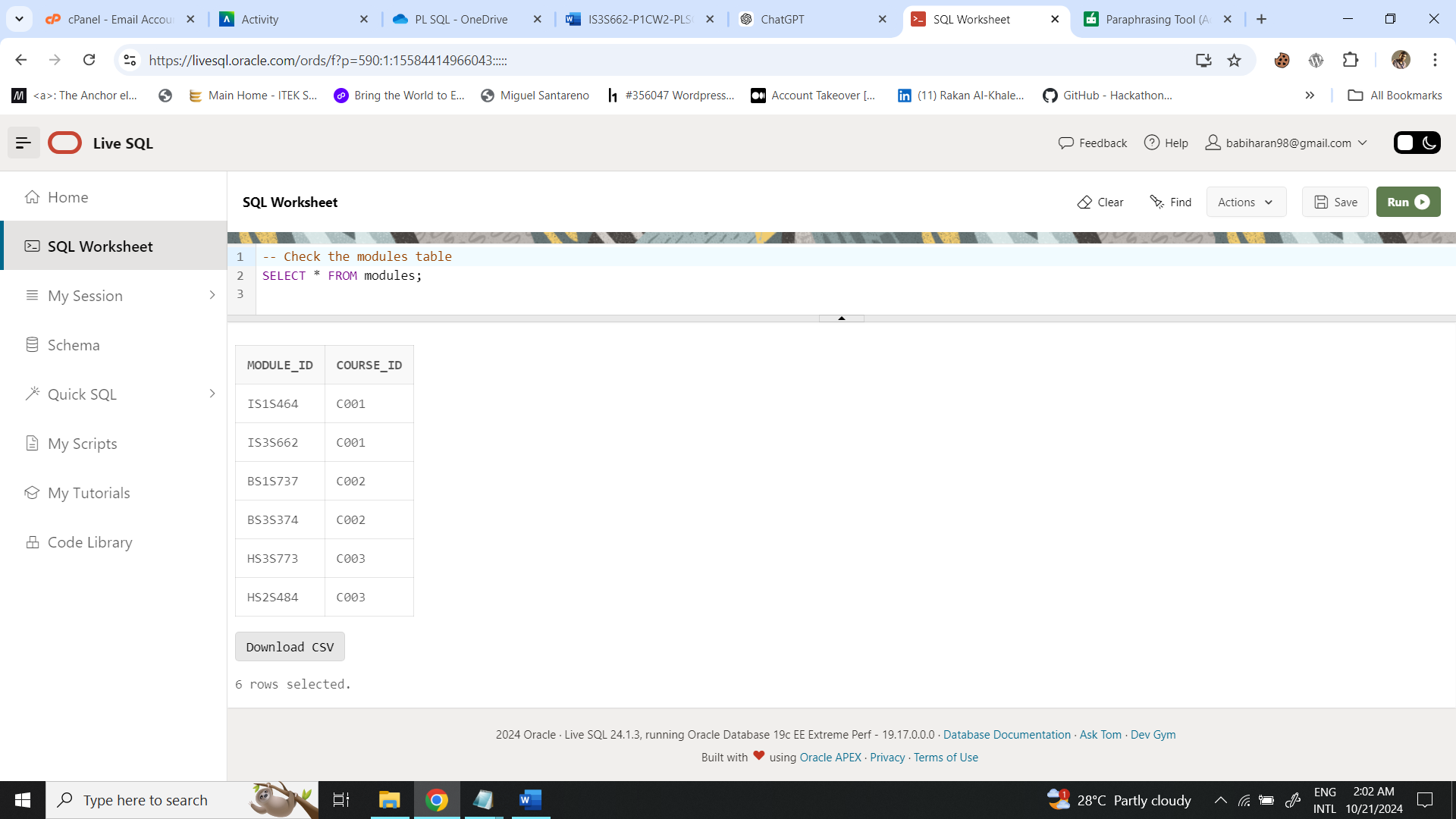
-- Check the course table

SELECT \* FROM course;



-- Check the modules table

SELECT \* FROM modules;



**Whole Script**

-- 1. Create the course and modules tables

CREATE TABLE course (

course\_id VARCHAR2(13) NOT NULL PRIMARY KEY,

title VARCHAR2(200),

description VARCHAR2(2000),

course\_leader VARCHAR2(200),

date\_modified DATE,

number\_of\_credits NUMBER

);

CREATE TABLE modules (

module\_id VARCHAR2(100) NOT NULL PRIMARY KEY,

course\_id VARCHAR2(13) REFERENCES course (course\_id)

);

-- 2. Create the procedure to insert a new course and modules

CREATE OR REPLACE PROCEDURE add\_course\_and\_modules (

p\_course\_id IN VARCHAR2,

p\_title IN VARCHAR2,

p\_description IN VARCHAR2,

p\_course\_leader IN VARCHAR2,

p\_date\_modified IN DATE,

p\_number\_of\_credits IN NUMBER,

p\_module\_ids IN SYS.ODCIVARCHAR2LIST

) IS

invalid\_input EXCEPTION;

BEGIN

IF p\_course\_id IS NULL OR p\_title IS NULL OR p\_course\_leader IS NULL OR p\_number\_of\_credits IS NULL THEN

RAISE invalid\_input;

END IF;

IF p\_number\_of\_credits <= 0 THEN

RAISE invalid\_input;

END IF;

INSERT INTO course (

course\_id, title, description, course\_leader, date\_modified, number\_of\_credits

) VALUES (

p\_course\_id, p\_title, p\_description, p\_course\_leader, p\_date\_modified, p\_number\_of\_credits

);

FOR i IN 1..p\_module\_ids.COUNT LOOP

INSERT INTO modules (module\_id, course\_id)

VALUES (p\_module\_ids(i), p\_course\_id);

END LOOP;

COMMIT;

EXCEPTION

WHEN invalid\_input THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Invalid input values.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

-- 3. Insert records into the course and modules tables using the procedure

-- Example for course C001

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('IS1S464', 'IS3S662');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C001',

p\_title => 'Computing',

p\_description => 'Focuses on developing skills that employers demand for roles in the IT industry.',

p\_course\_leader => 'Richard Jones',

p\_date\_modified => TO\_DATE('20-APR-2024', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

);

END;

/

-- Example for course C002

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('BS1S737', 'BS3S374');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C002',

p\_title => 'Business',

p\_description => 'Focuses on management and looks at the different functions of business and how they inter-relate.',

p\_course\_leader => 'Thomas Page',

p\_date\_modified => TO\_DATE('03-MAR-2021', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

);

END;

/

-- Example for course C003

DECLARE

v\_module\_ids SYS.ODCIVARCHAR2LIST := SYS.ODCIVARCHAR2LIST('HS3S773', 'HS2S484');

BEGIN

add\_course\_and\_modules(

p\_course\_id => 'C003',

p\_title => 'History',

p\_description => 'Focus on modern history, from about 1450 to the present day, including regional, national, European, American, and global perspectives.',

p\_course\_leader => 'Adam Richards',

p\_date\_modified => TO\_DATE('12-MAY-2023', 'DD-MON-YYYY'),

p\_number\_of\_credits => 120,

p\_module\_ids => v\_module\_ids

);

END;

/

# Question 02)

Write a procedure that retrieves a count on the number of courses and modules.

-- QUESTION 2

-- Procedure that retrieves a count on the number of courses and modules

CREATE OR REPLACE PROCEDURE

get\_course\_and\_module\_count IS

v\_course\_count NUMBER;

v\_module\_count NUMBER;

BEGIN

--Get the total number of courses

SELECT COUNT(\*) into v\_course\_count FROM course;

--Get the total number of modules

SELECT COUNT(\*) into v\_module\_count FROM modules;

--OUTPUT of counts

DBMS\_OUTPUT.PUT\_LINE('Total Number of Courses: ' || v\_course\_count );

DBMS\_OUTPUT.PUT\_LINE('Total Number of Modules: ' || v\_module\_count );

END;

/

-- To execute the counts for courses and modules

BEGIN

get\_course\_and\_module\_count;

END;

/

## A screenshot of a computer Description automatically generatedOutput Screenshots

**Explanation:**

1. Procedure Header

SQL - CREATE OR REPLACE PROCEDURE get\_course\_and\_module\_count IS

* The **CREATE OR REPLACE** clause ensures the procedure is created or replaces an existing one with the same name.
* The procedure name is get\_course\_and\_module\_count.
* Next comes the declaration section with IS, in which local variables are declared.

1. Variable Declarations

SQL - v\_course\_count NUMBER;

v\_module\_count NUMBER;

* Two variables are declared:
* **v\_course\_count:** This will store the count of rows in the course table.
* **v\_module\_count:** This will store the count of rows in the modules table.

1. Logic for Retrieving Counts

SQL - SELECT COUNT(\*) INTO v\_course\_count FROM course;

* The **COUNT(\*) SELECT** statement counts total number of rows in a course table.
* The result is stored in the variable **v\_course\_count** using the **INTO** clause.

SQL - SELECT COUNT(\*) INTO v\_module\_count FROM modules;

* This statement counts the same for the modules table and puts the result in **v\_module\_count.**

1. Output the Results

SQL - DBMS\_OUTPUT.PUT\_LINE('Total Number of Courses: ' || v\_course\_count);

DBMS\_OUTPUT.PUT\_LINE('Total Number of Modules: ' || v\_module\_count);

* **DBMS\_OUTPUT.PUT\_LINE** is a utility that outputs messages to the console in Oracle SQL Developer or any compatible environment.
* These lines print the counts of courses and modules concatenated with descriptive text.

1. **Execution**

The procedure is executed with the following block:

SQL - BEGIN

get\_course\_and\_module\_count;

END;

* This calls the stored procedure **get\_course\_and\_module\_count** which fetches and displays the counts.

# Question 03)

Write a procedure getCourseDetails which accepts a Course ID and returns the Course Title, Course Leader, Description, Date Modified and number of credits. The main block should call the procedure with a course id and output the course’s details.

-- QUESTION 3

CREATE OR REPLACE PROCEDURE getCourseDetails (

p\_course\_id IN VARCHAR2

) IS

v\_title VARCHAR2(200);

v\_leader VARCHAR2(200);

v\_description VARCHAR2(2000);

v\_date\_modified DATE;

v\_credits NUMBER;

BEGIN

-- Retrieve course details

SELECT

title,

course\_leader,

description,

date\_modified,

number\_of\_credits

INTO

v\_title,

v\_leader,

v\_description,

v\_date\_modified,

v\_credits

FROM course

WHERE course\_id = p\_course\_id;

-- Output the details

DBMS\_OUTPUT.PUT\_LINE('Course Title: ' || v\_title);

DBMS\_OUTPUT.PUT\_LINE('Course Leader: ' || v\_leader);

DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_description);

DBMS\_OUTPUT.PUT\_LINE('Date Modified: ' || TO\_CHAR(v\_date\_modified, 'DD-MON-YYYY'));

DBMS\_OUTPUT.PUT\_LINE('Number of Credits: ' || v\_credits);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No course found with ID: ' || p\_course\_id);

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error retrieving course details: ' || SQLERRM);

END;

/

-- Main block to call the procedure

BEGIN

getCourseDetails('C001');

END;

/

## Output Screenshots

A screenshot of a computer

Description automatically generated

**Explanation:**

1. Procedure Header

SQL - CREATE OR REPLACE PROCEDURE getCourseDetails (

p\_course\_id IN VARCHAR2

)

* This would define a stored procedure called **getCourseDetails**. This procedure takes one parameter, **p\_course\_id**, of **VARCHAR2** data type (in other words, string) representing unique identifier of a course.

1. SQL Query to Fetch Course Details

SQL –

**SELECT**

**title,**

**course\_leader,**

**description,**

**date\_modified,**

**number\_of\_credits**

**INTO**

**v\_title,**

**v\_leader,**

**v\_description,**

**v\_date\_modified,**

**v\_credits**

**FROM course**

**WHERE course\_id = p\_course\_id;**

* This **SELECT** statement queries the course table to retrieve the details of the course with the provided **course\_id**. The **INTO** clause is used to assign the selected values into the local variables:
  + **v\_title** will hold the value of title.
  + **v\_leader** will hold the value of **course\_leader**.
  + **v\_description** will hold the value of description.
  + **v\_date\_modified** will hold the value of **date\_modified.**
  + **v\_credits** will hold the value of **number\_of\_credits.**
* The **WHERE** clause filters the records based on the **course\_id** passed as an argument to the procedure.

1. Output the Course Details

SQL –

**DBMS\_OUTPUT.PUT\_LINE('Course Title: ' || v\_title);**

**DBMS\_OUTPUT.PUT\_LINE('Course Leader: ' || v\_leader);**

**DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_description);**

**DBMS\_OUTPUT.PUT\_LINE('Date Modified: ' || TO\_CHAR(v\_date\_modified, 'DD-MON- YYYY'));**

**DBMS\_OUTPUT.PUT\_LINE('Number of Credits: ' || v\_credits);**

* Once the course details have been retrieved and stored in these variables, we use **DBMS\_OUTPUT.PUT\_LINE** to print these to the output. Here**, TO\_CHAR** is used to format the **v\_date\_modified** value into a readable date format; hence, it will appear as 09-DEC-2024.

1. Exception Handling

SQL –

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('No course found with ID: ' || p\_course\_id);**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error retrieving course details: ' || SQLERRM);**

* **EXCEPTION** HANDLES are for handling the potential error arising during execution of the procedure:
  + **NO\_DATA\_FOUND:** If there is no record returned by the **SELECT** query-which implies that there does not exist any course with this provided **course\_id-**the system writes a message that it could not find any course.
  + WHEN OTHERS: This handles any other exceptions (general errors) that may occur, and it outputs the error message using **SQLERRM** (a built-in function that returns the error message).

# Question 04)

Write a PL/SQL block which utilises the getCourseDetails procedure and prints the data for each record.

--QUESTION 4

DECLARE

-- Cursor definition to fetch course details

-- This cursor retrieves all the course details from the 'course' table.

CURSOR course\_cursor IS

SELECT course\_id,

title,

description,

course\_leader,

date\_modified,

number\_of\_credits

FROM course; -- Selecting from the 'course' table

-- Variables to store each course's details as we fetch them

v\_course\_id VARCHAR2(13); -- Variable to hold the course ID

v\_title VARCHAR2(200); -- Variable to hold the course title

v\_description VARCHAR2(2000); -- Variable to hold the course description

v\_course\_leader VARCHAR2(200); -- Variable to hold the name of the course leader

v\_date\_modified DATE; -- Variable to hold the last modified date of the course

v\_number\_of\_credits NUMBER; -- Variable to hold the number of credits for the course

BEGIN

-- Opening the cursor to start fetching data

OPEN course\_cursor;

-- Loop to process each course fetched from the cursor

LOOP

-- Fetch the next record into the variables

FETCH course\_cursor INTO v\_course\_id, v\_title, v\_description, v\_course\_leader, v\_date\_modified, v\_number\_of\_credits;

-- Exit the loop when there are no more records in the cursor

EXIT WHEN course\_cursor%NOTFOUND;

-- Output the details of the course

-- Printing the course information using DBMS\_OUTPUT

DBMS\_OUTPUT.PUT\_LINE('Course ID: ' || v\_course\_id);

DBMS\_OUTPUT.PUT\_LINE('Title: ' || v\_title);

DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_description);

DBMS\_OUTPUT.PUT\_LINE('Course Leader: ' || v\_course\_leader);

DBMS\_OUTPUT.PUT\_LINE('Date Modified: ' || TO\_CHAR(v\_date\_modified, 'DD-MON-YYYY')); -- Formatting the date

DBMS\_OUTPUT.PUT\_LINE('Number of Credits: ' || v\_number\_of\_credits);

DBMS\_OUTPUT.PUT\_LINE('-----------------------------'); -- A separator for readability

END LOOP;

-- Close the cursor to release the resources

CLOSE course\_cursor;

EXCEPTION

-- Exception handling to catch any errors

WHEN OTHERS THEN

-- If there's an error, print the error message

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

-- Ensure the cursor is closed if an error occurs

IF course\_cursor%ISOPEN THEN

CLOSE course\_cursor;

END IF;

END;

/

## A screenshot of a computer Description automatically generatedOutput Screenshots

**Explanation:**

1. Cursor Definition:

* The cursor **course\_cursor** is declared to pick up certain columns from the table **course:**
* **course\_id:** Unique identifier for the course, e.g., 'CS101', 'MATH202'.
* **title:** The name of the course, e.g., 'Introduction to Computer Science'. description: A small
* **description:** about the course, e.g., 'This course will introduce the basics of programming'.
* **course\_leader:** Leader/instructor name for the course, e.g., 'Dr. Smith'.
* **date\_modified:** When the course details were last modified, e.g., '2023-05-01'.
* **number\_of\_credits:** The number of credits assigned to the course (e.g., '3').

SQL -

CURSOR course\_cursor IS

SELECT course\_id,

title,

description,

course\_leader,

date\_modified,

number\_of\_credits

FROM course;

1. Opening the Cursor:

SQL – OPEN course\_cursor;

* The **OPEN** statement opens the cursor. This initiates the process by which records will be fetched from the course table.

1. Loop to Process Each Record:

SQL - FETCH course\_cursor INTO v\_course\_id, v\_title, v\_description, v\_course\_leader, v\_date\_modified, v\_number\_of\_credits;

* **LOOP** iterates through each fetched record from the cursor. The **FETCH** statement retrieves a row of data from the cursor into the declared variables - **v\_course\_id, v\_title, v\_description, v\_course\_leader, v\_date\_modified, v\_number\_of\_credits.**

SQL - EXIT WHEN course\_cursor%NOTFOUND;

* This is an infinite loop until the cursor reaches a state where it has no more records, which in this case, is when the condition says **EXIT WHEN** **course\_cursor%NOTFOUND**;. That means, after reaching the end of the result set.

1. Closing the Cursor:

SQL - CLOSE course\_cursor;

* The **CLOSE** statement closes the cursor after all records have been processed. This closes the cursor and frees the resources.

# Question 5)

Write a procedure that deletes a course and all corresponding modules from the database.

-- QUESTION 5

CREATE OR REPLACE PROCEDURE delete\_course\_and\_modules (

p\_course\_id IN VARCHAR2

) IS

BEGIN

-- First, delete the modules associated with the course

DELETE FROM modules

WHERE course\_id = p\_course\_id;

-- Then, delete the course itself

DELETE FROM course

WHERE course\_id = p\_course\_id;

-- Commit the changes

COMMIT;

-- Output success message

DBMS\_OUTPUT.PUT\_LINE('Course and corresponding modules deleted successfully.');

EXCEPTION

-- Exception handling to handle any errors during deletion

WHEN OTHERS THEN

-- Rollback any changes if an error occurs

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

## Output Screenshots

A screenshot of a computer

Description automatically generated

-- Call the procedure

DECLARE

v\_course\_id VARCHAR2(13) := 'C001'; -- Specify the course ID to delete

BEGIN

-- Call the delete\_course\_and\_modules procedure

delete\_course\_and\_modules(p\_course\_id => v\_course\_id);

END;

/

A screenshot of a computer

Description automatically generated

SELECT \* FROM course;

A screenshot of a computer

Description automatically generated

SELECT \* FROM modules;

A screenshot of a computer

Description automatically generated

**Explanation:**

1. Procedure Declaration

SQL - CREATE OR REPLACE PROCEDURE delete\_course\_and\_modules (

p\_course\_id IN VARCHAR2

) IS

* **CREATE OR REPLACE PROCEDURE:** A statement that defines a new stored procedure or replaces an existing procedure in the database with the same name.
* **delete\_course\_and\_modules:** the name of the procedure.
* **p\_course\_id IN VARCHAR2:** This is the procedure parameter. The parameter **p\_course\_id** is an input (IN) of type **VARCHAR2,** which will be used to pass the **course ID** to the procedure. The procedure will delete the course with this ID together with all its associated modules.

1. Deleting Corresponding Modules

SQL - DELETE FROM modules

WHERE course\_id = p\_course\_id;

* This **DELETE** statement removes all records from the modules table where the **course\_id** matches the **p\_course\_id** parameter.
* **Reason:** We first delete the associated modules before deleting the course itself because the course may have foreign key constraints with the modules. In this way, the dependencies between the course and the modules will be handled first.

1. Deleting the Course

SQL - DELETE FROM course

WHERE course\_id = p\_course\_id;

* This **DELETE** statement, after having deleted the modules, removes the course record from the course table where **course\_id** is equal to p\_course\_id.
* This ensures that the course and all its related modules are deleted from the database.

1. Exception Handling

SQL - EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

* **EXCEPT** that this block will catch any errors upon the execution of the procedure.
* **WHEN OTHERS:** This catches all exceptions, i.e., any errors that are not explicitly handled elsewhere in the procedure.
* **ROLLBACK:** In case of any failure in the process of deletion, this command rolls back all changes that have happened in the procedure; for example, if the modules were deleted but the course deletion failed, the rollback will make both actions revert.
* **DBMS\_OUTPUT.PUT\_LINE:** In case of an error, this prints the error message - SQLERRM provides the error details.

# Question 6)

Write a trigger that reports how many modules are present after any insert/update/delete operation.

--QUESTION 6

CREATE OR REPLACE TRIGGER module\_count\_report

AFTER INSERT OR UPDATE OR DELETE ON modules

FOR EACH ROW

DECLARE

v\_module\_count NUMBER;

BEGIN

-- Get the count of modules after the DML operation

SELECT COUNT(\*) INTO v\_module\_count FROM modules;

-- Output the module count

DBMS\_OUTPUT.PUT\_LINE('Total Number of Modules: ' || v\_module\_count);

END;

/

A screenshot of a computer

Description automatically generated

-- insert a course

INSERT INTO course (course\_id, title, description, course\_leader, date\_modified, number\_of\_credits)

VALUES ('C001', 'Computing', 'Focuses on developing skills...', 'Richard Jones', TO\_DATE('20-APR-2024', 'DD-MON-YYYY'), 120);

-- insert the module

INSERT INTO modules (module\_id, course\_id)

VALUES ('IS4S700', 'C001');

SELECT \* FROM modules;

SELECT \* FROM course;

## Output Screenshots

**A screenshot of a computer

Description automatically generated**

-- Update an existing module's ID (changing module\_id)

UPDATE modules

SET module\_id = 'IS4S701'

WHERE module\_id = 'IS4S700';

SELECT \* FROM modules;

SELECT \* FROM course;

**A screenshot of a computer

Description automatically generated**

-- Delete a module from the modules table

DELETE FROM modules

WHERE module\_id = 'IS4S701';

SELECT \* FROM modules;

SELECT \* FROM course;

**A screenshot of a computer

Description automatically generated**

**Explanation:**

* **AFTER INSERT OR UPDATE OR DELETE:** The trigger will be fired after any insert, update, or delete operation on the modules table.
* **FOR EACH ROW:** This ensures that the trigger will be executed once for each row affected by the DML operation though we are counting the total number of rows in the modules table.
* **DBMS\_OUTPUT.PUT\_LINE:** Outputs total number of modules in the modules table after the operation.
* The insert statement puts into the modules table a new module **- IS4S700**, to course **C001:.**
* The update query changes the **module\_id** of the module with **module\_id** = **IS4S700 to IS4S701** for testing.
* The delete query removes the module with **module\_id** = **IS4S701** from the modules table.

# Reference

GeeksforGeeks. (2024). *PL/SQL Triggers*. Available at: <https://www.geeksforgeeks.org/plsql-triggers/>. [Accessed – 08/12/2024]

‌