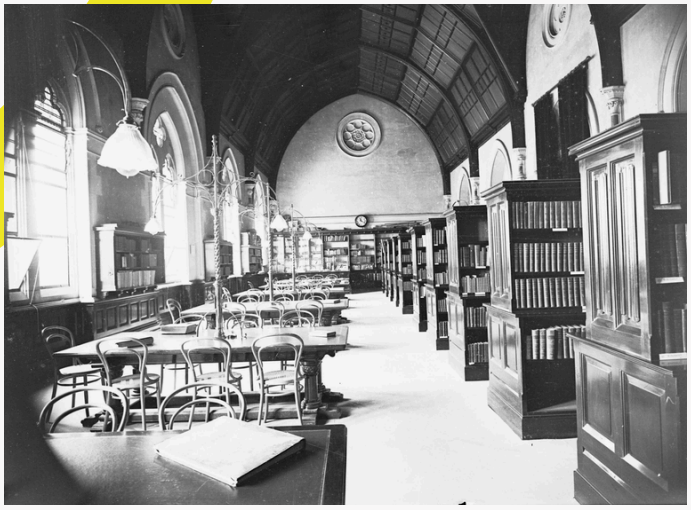
OBJECT RELATIONAL AND NO-SQL DB

report

University Enrollment System



INDIVIDUAL ASSIGNMENT 1
LUAN TRAN |
BS22DSY024

table content

01 sql development

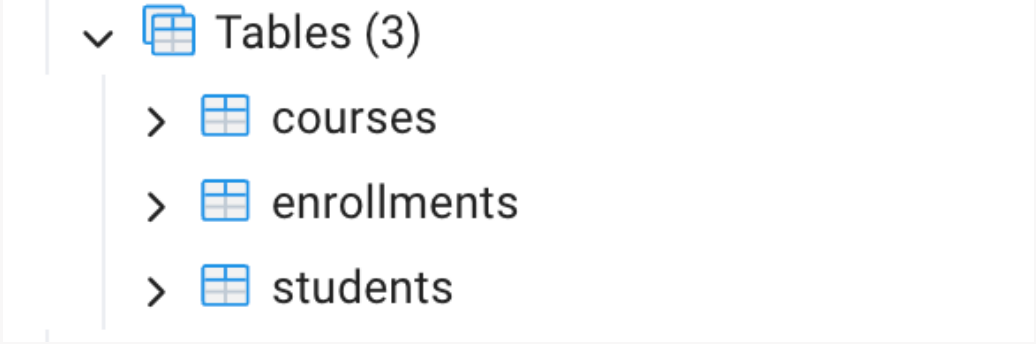
a. create table file

run **Create_Table.sql** file on pgAdmin

```sql
CREATE TABLE Students (
    student_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    major VARCHAR(50) );


CREATE TABLE Courses (
    course_id SERIAL PRIMARY KEY,
    course_name VARCHAR(50),
    department VARCHAR(50),
    max_capacity INT );

CREATE TABLE Enrollments (
    enrollment_id SERIAL PRIMARY KEY,
    student_id INT REFERENCES Students(student_id),
    course_id INT REFERENCES Courses(course_id),
    enrollment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
```



b. stored procedure

Create a stored procedure named EnrollStudent to perform student enrolment given student_id and course_id.

Stored_Procedure.sql file

```sql
CREATE OR REPLACE PROCEDURE EnrollStudent(p_student_id INT, p_course_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    student_exists BOOLEAN;
    course_exists BOOLEAN;
    current_enrollment INT;
    course_max_capacity INT;
BEGIN
    -- Verify if the student with p_student_id exists in the Students table.
    -- student_exists = TRUE if student exists, otherwise FALSE.
    SELECT EXISTS (SELECT 1 FROM Students WHERE student_id = p_student_id) INTO student_exists;

    -- If student_exists is FALSE, raise an exception "Student does not exist."
    IF NOT student_exists THEN
        RAISE EXCEPTION 'Student does not exist';
    END IF;

    -- Check if the course with p_course_id exists in the Courses table.
    -- course_exists = TRUE if course exists, otherwise FALSE.
    SELECT EXISTS (SELECT 1 FROM Courses WHERE course_id = p_course_id) INTO course_exists;

    -- If course_exists is FALSE, raise an exception "Course does not exist."
    IF NOT course_exists THEN
        RAISE EXCEPTION 'Course does not exist';
    END IF;

   /* Course Capacity Check:
   current_enrollment: the number of students currently enrolled in the specified course,
                       counted by matching p_course_id in the Enrollments table.
   course_max_capacity: the course's maximum capacity, retrieved from the Courses table.
   */
   SELECT COUNT(e.enrollment_id), c.max_capacity INTO current_enrollment, course_max_capacity
   FROM Enrollments e
   JOIN Courses c ON e.course_id = c.course_id
   WHERE e.course_id = p_course_id
   GROUP BY c.max_capacity;

   /* If current_enrollment exceeds or equals course_max_capacity, raise an exception
   stating "Course is full." */
   IF current_enrollment >= course_max_capacity THEN
       RAISE EXCEPTION 'Course is full';
   END IF;

   /* Enroll the student if not already enrolled:
   If the student is not already enrolled in the course, insert a new entry
   into the Enrollments table for p_student_id and p_course_id. */
   IF NOT EXISTS (SELECT 1 FROM Enrollments WHERE student_id = p_student_id AND course_id = p_course_id) THEN
       INSERT INTO Enrollments (student_id, course_id) VALUES (p_student_id, p_course_id);
   ELSE
       /* If the student is already enrolled, raise a notice stating "Student is already enrolled
       in this course," without inserting a new record. */
       RAISE EXCEPTION 'Student is already enrolled in this course';
   END IF;
END;
$$;
```

```sql
CREATE OR REPLACE PROCEDURE EnrollStudent(p_student_id INT, p_course_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    student_exists BOOLEAN;
    course_exists BOOLEAN;
    current_enrollment INT;
    course_max_capacity INT;
BEGIN
    -- Verify if the student with p_student_id exists in the Students table.
    -- student_exists = TRUE if student exists, otherwise FALSE.
    SELECT EXISTS (SELECT 1 FROM Students WHERE student_id = p_student_id) INTO student_exists;

    -- If student_exists is FALSE, raise an exception "Student does not exist."
    IF NOT student_exists THEN
        RAISE EXCEPTION 'Student does not exist';
    END IF;

    -- Check if the course with p_course_id exists in the Courses table.
    -- course_exists = TRUE if course exists, otherwise FALSE.
    SELECT EXISTS (SELECT 1 FROM Courses WHERE course_id = p_course_id) INTO course_exists

    -- If course_exists is FALSE, raise an exception "Course does not exist."
    IF NOT course_exists THEN
        RAISE EXCEPTION 'Course does not exist';
    END IF;

    /* Course Capacity Check:
    current_enrollment: the number of students currently enrolled in the specified course,
                        counted by matching p_course_id in the Enrollments table.
    course_max_capacity: the course's maximum capacity, retrieved from the Courses table.
    */
    SELECT COUNT(e.enrollment_id), c.max_capacity INTO current_enrollment, course_max_capacity
    FROM Enrollments e
    JOIN Courses c ON e.course_id = c.course_id
    WHERE e.course_id = p_course_id
    GROUP BY c.max_capacity;

    /* If current_enrollment exceeds or equals course_max_capacity, raise an exception
    stating "Course is full." */
    IF current_enrollment >= course_max_capacity THEN
        RAISE EXCEPTION 'Course is full';
    END IF;

    /* Enroll the student if not already enrolled:
    If the student is not already enrolled in the course, insert a new entry
    into the Enrollments table for p_student_id and p_course_id. */
    IF NOT EXISTS (SELECT 1 FROM Enrollments WHERE student_id = p_student_id AND course_id = p_course_id) THEN
        INSERT INTO Enrollments (student_id, course_id) VALUES (p_student_id, p_course_id);
    ELSE
        /* If the student is already enrolled, raise a notice stating "Student is already enrolled
        in this course," without inserting a new record. */
        RAISE EXCEPTION 'Student is already enrolled in this course';
    END IF;
END;
$$;
```

input variables

declare some variables

1

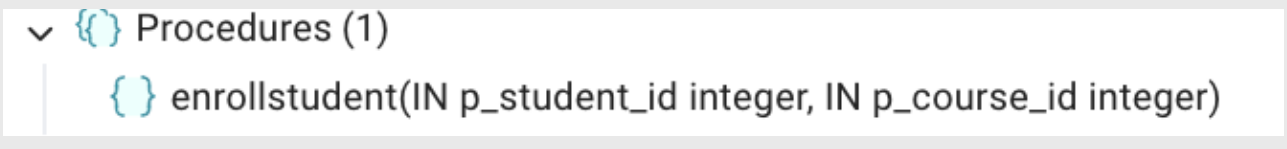
2

3

4

Edge Cases :

1. **Student does not exist**
2. **Course does not exist**
3. **Course is full**
4. **Student is already enrolled in this course**



c. views

Views.sql file

Create two Views :

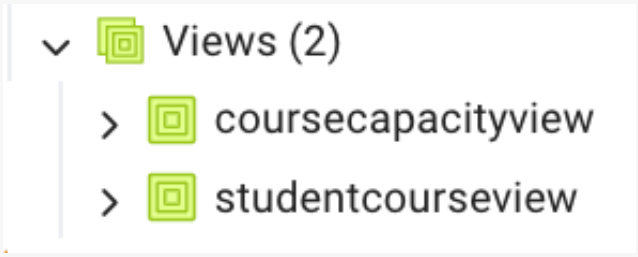
- ***StudentCourseView*** *combines data from the Students, Courses, and Enrolments tables.*
- ***CourseCapacityView*** *shows the number of students enrolled in each course and the remaining capacity.*

```sql
-- student course
-- Drop the view if it already exists
DROP VIEW IF EXISTS StudentCourseView;

CREATE VIEW StudentCourseView AS
SELECT
    s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name, -- combine first name and last name
    c.course_name,
    e.enrollment_date
FROM Students s
-- ensuring that only students who are enrolled in courses are selected.
JOIN Enrollments e ON s.student_id = e.student_id
-- linking each enrollment to the corresponding course.
JOIN Courses c ON e.course_id = c.course_id;

-- course capacity
-- Drop the view if it already exists
DROP VIEW IF EXISTS CourseCapacityView;

CREATE VIEW CourseCapacityView AS
SELECT
    c.course_id,
    c.course_name,
    c.department,
    COUNT(e.enrollment_id) AS current_enrollment,
    c.max_capacity - COUNT(e.enrollment_id) AS remaining_capacity
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id;
```



c. views

```sql
-- student course
-- Drop the view if it already exists
DROP VIEW IF EXISTS StudentCourseView;

CREATE VIEW StudentCourseView AS
SELECT
    s.student_id,
    CONCAT(s.first_name, ' ', s.last_name) AS student_name, -- combine first name and last name
    c.course_name,
    e.enrollment_date
FROM Students s
-- ensuring that only students who are enrolled in courses are selected.
JOIN Enrollments e ON s.student_id = e.student_id
-- linking each enrollment to the corresponding course.
JOIN Courses c ON e.course_id = c.course_id;

-- course capacity
-- Drop the view if it already exists
DROP VIEW IF EXISTS CourseCapacityView;

CREATE VIEW CourseCapacityView AS
SELECT
    c.course_id,
    c.course_name,
    c.department,
    COUNT(e.enrollment_id) AS current_enrollment,
    c.max_capacity - COUNT(e.enrollment_id) AS remaining_capacity
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id;
```

- **current_enrollment** : counts how many students are currently enrolled in the course
- **remaining_capacity** : the remaining capacity of the course by subtracting the current number of enrollments from the maximum capacity of the course.

This groups the result by course_id, ensuring the aggregation (COUNT) is calculated per course.

The LEFT JOIN ensures that even courses with zero enrollments are included in the results.

d. triggers

Triggers.sql file

Initiate step :

- **add new column ‘course_capacity’ to Course table**
- **create new table EnrollmentLog**

```
-- Initiate
ALTER TABLE Courses DROP COLUMN IF EXISTS course_capacity;
DO $$
BEGIN
    -- Check if the `course_capacity` column exists in the `Courses` table
    IF NOT EXISTS (
        SELECT 1
        FROM information_schema.columns
        WHERE table_name = 'courses' AND column_name = 'course_capacity'
    ) THEN
        -- Add `course_capacity` column and set default to 0
        ALTER TABLE Courses ADD COLUMN course_capacity INT DEFAULT 0;

        -- Add a check constraint to ensure `course_capacity` is between 0 and `max_capacity`
        ALTER TABLE Courses
        ADD CONSTRAINT check_capacity_range
        CHECK (course_capacity >= 0 AND course_capacity <= max_capacity);
    END IF;
END $$;

CREATE TABLE IF NOT EXISTS EnrollmentLog (
    log_id SERIAL PRIMARY KEY,          -- Primary key for the log entries
    student_id INT NOT NULL,            -- The ID of the student
    course_id INT NOT NULL,             -- The ID of the course
    action VARCHAR(10) NOT NULL,        -- 'enroll' or 'drop'
    action_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP  -- Time of action
);
```

Trigger 1 : Update course_capacity column in Course table when a student enrolls or drop a course

- **TG_OP = 'INSERT' (enroll) : increase course_capacity by 1**
- **TG_OP = 'DELETE' (drop) : decrease course_capacity by 1**

Create a trigger named update_capacity_trigger, which is triggered AFTER INSERT OR DELETE ACTION on ENROLLMENTS table

```
-- Step 1: Update course capacity whenever a student enrolls or drops a course

-- Function to update the course_capacity when a student enrolls or drops a course
CREATE OR REPLACE FUNCTION update_course_capacity()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
        -- Decrement course_capacity when a student enrolls
        UPDATE Courses
        SET course_capacity = course_capacity + 1
        WHERE course_id = NEW.course_id;
    ELSIF (TG_OP = 'DELETE') THEN
        -- Increment course_capacity when a student drops the course
        UPDATE Courses
        SET course_capacity = course_capacity - 1
        WHERE course_id = OLD.course_id;
    END IF;

    RETURN NULL; -- Return NULL as we don't need to modify the row itself
END;
$$ LANGUAGE plpgsql;

-- Drop existing trigger if it exists
DROP TRIGGER IF EXISTS update_capacity_trigger ON Enrollments;

-- Create new trigger to update course_capacity when students enroll or drop a course
CREATE TRIGGER update_capacity_trigger
AFTER INSERT OR DELETE ON Enrollments
FOR EACH ROW
EXECUTE FUNCTION update_course_capacity();
```

Trigger 2 : log enrollment events (enroll and drop action) in EnrollmentLog

Create a trigger named enrollment_log_trigger, which is triggered AFTER INSERT OR DELETE ACTION on ENROLLMENTS table

- **TG_OP = 'INSERT' (enroll) : log the NEW pair (student_id, course_id) with action 'enroll'**
- **TG_OP = 'DELETE' (drop) : log the OLD (student_id, course_id) with action 'drop'**

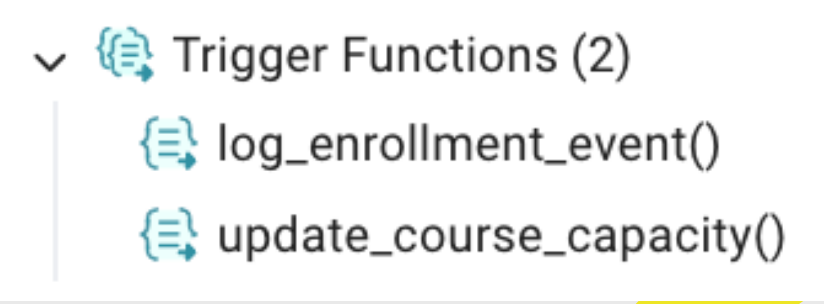
```
-- Step 2: Log enrollment events (enroll and drop actions) in EnrollmentLog

-- Function to log enrollment events (enroll and drop actions)
CREATE OR REPLACE FUNCTION log_enrollment_event()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
        -- Log enrollment event
        INSERT INTO EnrollmentLog (student_id, course_id, action)
        VALUES (NEW.student_id, NEW.course_id, 'enroll');
    ELSIF (TG_OP = 'DELETE') THEN
        -- Log drop event
        INSERT INTO EnrollmentLog (student_id, course_id, action)
        VALUES (OLD.student_id, OLD.course_id, 'drop');
    END IF;

    RETURN NULL; -- No need to modify the row itself
END;
$$ LANGUAGE plpgsql;

-- Drop existing trigger if it exists
DROP TRIGGER IF EXISTS enrollment_log_trigger ON Enrollments;

-- Create trigger for logging enrollment events (enroll and drop actions)
CREATE TRIGGER enrollment_log_trigger
AFTER INSERT OR DELETE ON Enrollments
FOR EACH ROW
EXECUTE FUNCTION log_enrollment_event();
```



python integration

02

Pre-install

Install any necessary library / package if not install

```
!pip install psycopg2
!pip install pandas
```

Pre-define some data

```
import psycopg2
import pandas as pd

conn_params = {
    "dbname": "postgres",
    "user": "postgres",
    "password": "12345",
    "host": "localhost",
    "port": "5432",
}
```

Define connection parameters for PostgreSQL include :

- **dbname**: name of the database to connect to.
- **user**: username used to authenticate.
- **password**: password associated with the username.
- **host**: the host of the database, typically "localhost"
- **port**: the port number the database is running on (default PostgreSQL port is 5432)

```
students_data = [
    ("John", "Doe", "Computer Science"),
    ("Jane", "Smith", "Mathematics"),
    ("Emily", "Johnson", "Physics"),
    ("Michael", "Brown", "Engineering"),
    ("Luan", "Tran", "Data Science"),
]

courses_data = [
    ("Introduction to Python", "Computer Science", 2),
    ("Calculus I", "Mathematics", 1),
    ("Physics I", "Physics", 5),
    ("Engineering Design", "Engineering", 3),
]
```

✓ 0.5s

Define some necessary functions

a) connect to database

```python
def connect():
    """Establishes a connection to the PostgreSQL database and returns the connection and cursor."""
    try:
        conn = psycopg2.connect(**conn_params)
        cur = conn.cursor()
        print("Connection to PostgreSQL established successfully.")
        return conn, cur
    except OperationalError as e:
        print("The error occurred:", e)
        return None, None


def check_connect(conn, cur):
    assert conn is not None, "Database connection failed!"
    print("Database connection is active.")
    assert cur is not None, "Cursor creation failed!"
    print("Cursor created successfully.")
    return True
```

✓ 0.0s

def connect()

```
conn, cur = connect()
```

✓ 0.0s

```
Connection to PostgreSQL established successfully.
```

def check_connect()

```
check_connect(conn, cur)
```

✓ 0.0s

```
Database connection is active.
Cursor created successfully.
```

Define some necessary functions

b) re-style dataframe for visualisation purpose

def display_df()

```
def display_df(df):
    styled_df = (
        df.style.set_properties(
            **{
                "text-align": "center",
                "border": "1px solid black",
                "color": "black",
            }  # Set border and text color
        )
        .set_table_styles(
            [  # Set bold header, background color for header, and black text
                {
                    "selector": "th",
                    "props": [
                        ("font-weight", "bold"),
                        ("background-color", "#f2f2f2"),
                        ("color", "black"),  # Ensure header text is black
                    ],
                }
            ]
        )
        .applymap(
            lambda x: "background-color: #f9f9f9", subset=pd.IndexSlice[::2, :]
        )  # Row background color for even rows
        .applymap(
            lambda x: "background-color: #e0e0e0", subset=pd.IndexSlice[1::2, :]
        )  # Row background color for odd rows
    )
    return styled_df
```

Inital Database

Feeding some initial data into Students and Courses table

```
def precondition():
    try:
        conn, cur = connect()
        check_connect(conn, cur)
        # If the connection is active, proceed with data insertion
        if conn is not None and cur is not None:
            # Insert data into the Students table
            for student in students_data:
                cur.execute(
                    """
                INSERT INTO Students (first_name, last_name, major)
                VALUES (%s, %s, %s)
                """,
                    student,
                )

            # Insert data into the Courses table
            for course in courses_data:
                cur.execute(
                    """
                INSERT INTO Courses (course_name, department, max_capacity)
                VALUES (%s, %s, %s)
                """,
                    course,
                )

            # Commit the transactions to save the data
            conn.commit()
            print("\nData inserted successfully into Students and Courses tables.")

            # Query and display the inserted data to confirm
            cur.execute("SELECT * FROM Students")
            students_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("\nStudents Table Data:")
            df = pd.DataFrame(students_result, columns=column_names)
            styled_df = display_df(df)
            display(styled_df)

            cur.execute("SELECT * FROM Courses")
            courses_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("\nCourses Table Data:")
            df = pd.DataFrame(
                courses_result,
                columns=column_names,
            )
            styled_df = display_df(df)
            display(styled_df)

    except AssertionError as e:
        print(e)
    except Exception as e:
        print("\nThe error occurred:", e)
    finally:
        # Ensure the cursor and connection are closed if they were created
        if "cur" in locals() and cur is not None:
            cur.close()
        if "conn" in locals() and conn is not None:
            conn.close()
        print("Connection closed.")
```

Inital Database

Insert values from pre-defined lists students_data, courses_data into Students, Courses table, respectively

```
students_data = [
    ("John", "Doe", "Computer Science"),
    ("Jane", "Smith", "Mathematics"),
    ("Emily", "Johnson", "Physics"),
    ("Michael", "Brown", "Engineering"),
    ("Luan", "Tran", "Data Science"),
]

courses_data = [
    ("Introduction to Python", "Computer Science", 2),
    ("Calculus I", "Mathematics", 1),
    ("Physics I", "Physics", 5),
    ("Engineering Design", "Engineering", 3),
]
```

✓ 0.5s

```
precondition()
```

✓ 0.3s

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

Data inserted successfully into Students and Courses tables.

Students Table Data:
```

	student_id	first_name	last_name	major
0	1	John	Doe	Computer Science
1	2	Jane	Smith	Mathematics
2	3	Emily	Johnson	Physics
3	4	Michael	Brown	Engineering
4	5	Luan	Tran	Data Science

```
Courses Table Data:
```

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	2	Calculus I	Mathematics	1	0
2	3	Physics I	Physics	5	0
3	4	Engineering Design	Engineering	3	0

```
Connection closed.
```

Stored Procedure

def query_stored_procedure_enroll_student() to test stored procedure EnrollStudent in our database making enrollment for given *student_id* and *course_id*

```
def query_stored_procedure_enroll_student(student_id=None, course_id=None):
    # Prompt for student ID and course ID
    if student_id is None:
        student_id = input("Enter the student ID: ")
    if course_id is None:
        course_id = input("Enter the course ID: ")
    try:
        conn, cur = connect()
        check_connect(conn, cur)
        # If the connection is active, proceed with data insertion
        if conn is not None and cur is not None:
            # Enroll the student in the course
            cur.execute(
                """
            CALL enrollstudent(%s, %s)
            """,
                (student_id, course_id),
            )
            conn.commit()
            print("Student enrolled successfully in the course.")

            # Query and display the updated data to confirm
            cur.execute("SELECT * FROM Enrollments")
            enrollments_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("\nEnrollments Table Data:")
            df = pd.DataFrame(
                enrollments_result,
                columns=column_names,
            )

            styled_df = display_df(df)
            display(styled_df)
    except AssertionError as e:
        print(e)
    except Exception as e:
        print("\nThe error occurred:", e)
    finally:
        # Ensure the cursor and connection are closed if they were created
        if "cur" in locals() and cur is not None:
            cur.close()
        if "conn" in locals() and conn is not None:
            conn.close()
        print("Connection closed.\n")


query_stored_procedure_enroll_student()
```

Stored Procedure

def query_stored_procedure_enroll_student ()

can be run in two ways, either pass 2 parameters (student_id, course_id) or using prompt to type input value

- **using prompt to type input value to make student_id 5 enroll course_id 3**

(5, 3)

```
In [*]: query_stored_procedure_enroll_student()
        Enter the student ID: 5
        Enter the course ID: 3
```

```
In [11]: query_stored_procedure_enroll_student()
         Enter the student ID: 5
         Enter the course ID: 3
         Connection to PostgreSQL established successfully.
         Database connection is active.
         Cursor created successfully.
         Student enrolled successfully in the course.

         Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:01:36.351944

```
         Connection closed.
```

- **pass 2 parameters (student_id, course_id) to make student_id 1 enroll course_id 3**

(1, 3)

```
In [13]: query_stored_procedure_enroll_student(1, 3)
         Connection to PostgreSQL established successfully.
         Database connection is active.
         Cursor created successfully.
         Student enrolled successfully in the course.

         Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:01:36.351944
1	2	1	3	2024-11-08 15:03:39.162075

```
         Connection closed.
```

Views

def query_views() to test if our 2 views StudentCourseView and CourseCapacityView working properly.

```
def query_views():
    try:
        conn, cur = connect()
        check_connect(conn, cur)
        # If the connection is active, proceed with data insertion
        if conn is not None and cur is not None:
            # Query and display the view data
            cur.execute("SELECT * FROM StudentCourseView")
            student_courses_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("\nStudentCourses View Data:")
            df = pd.DataFrame(
                student_courses_result,
                columns=column_names,
            )
            styled_df = display_df(df)
            display(styled_df)

            cur.execute("SELECT * FROM CourseCapacityView")
            course_capacity_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("\nCourseCapacity View Data:")
            df = pd.DataFrame(
                course_capacity_result,
                columns=column_names,
            )
            styled_df = display_df(df)
            display(styled_df)

    except AssertionError as e:
        print(e)
    except Exception as e:
        print("\nThe error occurred:", e)
    finally:
        # Ensure the cursor and connection are closed if they were created
        if "cur" in locals() and cur is not None:
            cur.close()
        if "conn" in locals() and conn is not None:
            conn.close()
        print("Connection closed.\n")
```

Views

```
query_views()
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.
```

Two enrollments (5, 3) and (1, 3) are showned here

StudentCourses View Data:

	student_id	student_name	course_name	enrollment_date
0	5	Luan Tran	Physics I	2024-11-08 12:02:03.826997
1	1	John Doe	Physics I	2024-11-08 12:03:15.293865

CourseCapacity View Data:

	course_id	course_name	department	current_enrollment	remaining_capacity
0	4	Engineering Design	Engineering	0	3
1	2	Calculus I	Mathematics	0	1
2	1	Introduction to Python	Computer Science	0	2
3	3	Physics I	Physics	2	3

```
Connection closed.
```

Capacity of Physics course (course_id 3) gets updated

Triggers

```
def trigger_event(student_id, course_id, action="enroll"):
    try:
        conn, cur = connect()
        check_connect(conn, cur)
        # If the connection is active, proceed with data insertion
        if conn is not None and cur is not None:
            # Enroll or drop the student from the course based on the action
            if action == "enroll":
                cur.execute("""CALL enrollstudent(%s, %s)""", (student_id, course_id))
                print("Student enrolled successfully in the course.")
            elif action == "drop":
                # Check if the record exists before attempting to delete
                cur.execute(
                    """
                    SELECT 1 FROM Enrollments
                    WHERE student_id = %s AND course_id = %s
                    """,
                    (student_id, course_id),
                )
                if cur.fetchone():
                    cur.execute(
                        """
                        DELETE FROM Enrollments
                        WHERE student_id = %s AND course_id = %s
                        """,
                        (student_id, course_id),
                    )
                    print("Student dropped successfully from the course.")
                else:
                    raise Exception("The student is not enrolled in the course.")

            conn.commit()

            # Query and display the updated data to confirm
            cur.execute("SELECT * FROM EnrollmentLog")
            enrollment_log_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("Check the updated enrollment log after the event:")
            print("\nEnrollmentLog Table Data:")
            df = pd.DataFrame(
                enrollment_log_result,
                columns=column_names,
            )
            styled_df = display_df(df)
            display(styled_df)

            cur.execute("SELECT * FROM Courses")
            courses_result = cur.fetchall()
            column_names = [desc[0] for desc in cur.description]
            print("Check the updated course capacity after the event:")
            print("\nCourses Table Data:")
            df = pd.DataFrame(
                courses_result,
                columns=column_names,
            )
            styled_df = display_df(df)
            display(styled_df)

    except AssertionError as e:
        print(e)
    except Exception as e:
        print("\nThe error occurred:", e)
    finally:
        # Ensure the cursor and connection are closed if they were created
        if "cur" in locals() and cur is not None:
            cur.close()
        if "conn" in locals() and conn is not None:
            conn.close()
        print("Connection closed.\n")
```

def trigger_event() to test if our 2 triggers log_enrollment_event() and update_course_capacity() working properly.

Triggers

Input **student_id, course_id, action (“enroll” / “drop”)**
Implement INSERT (enroll) or DELETE (drop) enrollments onto our database

```python
def trigger_event(student_id, course_id, action="enroll"):
    try:
        conn, cur = connect()
        check_connect(conn, cur)
        # If the connection is active, proceed with data insertion
        if conn is not None and cur is not None:
            # Enroll or drop the student from the course based on the action
            if action == "enroll":
                cur.execute("""CALL enrollstudent(%s, %s)""", (student_id, course_id))
                print("Student enrolled successfully in the course.")
            elif action == "drop":
                # Check if the record exists before attempting to delete
                cur.execute(
                    """
                    SELECT 1 FROM Enrollments
                    WHERE student_id = %s AND course_id = %s
                    """,
                    (student_id, course_id),
                )
                if cur.fetchone():
                    cur.execute(
                        """
                        DELETE FROM Enrollments
                        WHERE student_id = %s AND course_id = %s
                        """,
                        (student_id, course_id),
                    )
                    print("Student dropped successfully from the course.")
                else:
                    raise Exception("The student is not enrolled in the course.")

            conn.commit()
```

Edge case : Delete enrollment which does not exist --> Raise Exception : “The student is not enrolled in the course.”

Triggers

After implementing INSERT (enroll) or DELETE (drop), print out EnrollmentLog and Courses table to confirm triggers

```
        # Query and display the updated data to confirm
        cur.execute("SELECT * FROM EnrollmentLog")
        enrollment_log_result = cur.fetchall()
        column_names = [desc[0] for desc in cur.description]
        print("Check the updated enrollment log after the event:")
        print("\nEnrollmentLog Table Data:")
        df = pd.DataFrame(
            enrollment_log_result,
            columns=column_names,
        )
        styled_df = display_df(df)
        display(styled_df)

        cur.execute("SELECT * FROM Courses")
        courses_result = cur.fetchall()
        column_names = [desc[0] for desc in cur.description]
        print("Check the updated course capacity after the event:")
        print("\nCourses Table Data:")
        df = pd.DataFrame(
            courses_result,
            columns=column_names,
        )
        styled_df = display_df(df)
        display(styled_df)

except AssertionError as e:
    print(e)
except Exception as e:
    print("\nThe error occurred:", e)
finally:
    # Ensure the cursor and connection are closed if they were created
    if "cur" in locals() and cur is not None:
        cur.close()
    if "conn" in locals() and conn is not None:
        conn.close()
    print("Connection closed.\n")
```

Triggers

student_id 3 enroll course_id 3

```
trigger_event(3, 3, "enroll")
✓ 0.0s

Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.
Student enrolled successfully in the course.
Check the updated enrollment log after the event:

EnrollmentLog Table Data:
```

the event get updated in EnrollmentLog

	log_id	student_id	course_id	action	action_timestamp
0	1	5	3	enroll	2024-11-08 15:39:34.412706
1	2	1	3	enroll	2024-11-08 15:39:38.236148
2	3	3	3	enroll	2024-11-08 15:39:49.074563

```
Check the updated course capacity after the event:

Courses Table Data:
```

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	2	Calculus I	Mathematics	1	0
2	4	Engineering Design	Engineering	3	0
3	3	Physics I	Physics	5	3

course_capacity increase by 1

```
Connection closed.
```

Triggers

student_id 3 drop course_id 3

```
trigger_event(3, 3, "drop")
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.
Student dropped successfully from the course.
Check the updated enrollment log after the event:

EnrollmentLog Table Data:
```

the event get updated in EnrollmentLog

	log_id	student_id	course_id	action	action_timestamp
0	1	5	3	enroll	2024-11-08 12:02:03.826997
1	2	1	3	enroll	2024-11-08 12:03:15.293865
2	3	3	3	enroll	2024-11-08 12:05:12.990964
3	4	3	3	drop	2024-11-08 12:05:24.700885

```
Check the updated course capacity after the event:

Courses Table Data:
```

course_capacity decrease by 1

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	2	Calculus I	Mathematics	1	0
2	4	Engineering Design	Engineering	3	0
3	3	Physics I	Physics	5	2

```
Connection closed.
```

03 deliverables

Our current database

```
current_database()
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

Students Table Data:
```

	student_id	first_name	last_name	major
0	1	John	Doe	Computer Science
1	2	Jane	Smith	Mathematics
2	3	Emily	Johnson	Physics
3	4	Michael	Brown	Engineering
4	5	Luan	Tran	Data Science

```
Courses Table Data:
```

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	2	Calculus I	Mathematics	1	0
2	4	Engineering Design	Engineering	3	0
3	3	Physics I	Physics	5	2

```
Connection closed.
```

03 deliverables

Our current tracking tables

```
track_database()
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 12:02:03.826997
1	2	1	3	2024-11-08 12:03:15.293865

```
Courses Table Data:
```

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	2	Calculus I	Mathematics	1	0
2	4	Engineering Design	Engineering	3	0
3	3	Physics I	Physics	5	2

```
EnrollmentLog Table Data:
```

	log_id	student_id	course_id	action	action_timestamp
0	1	5	3	enroll	2024-11-08 12:02:03.826997
1	2	1	3	enroll	2024-11-08 12:03:15.293865
2	3	3	3	enroll	2024-11-08 12:05:12.990964
3	4	3	3	drop	2024-11-08 12:05:24.700885

```
Connection closed.
```

03 deliverables

Test Plan: University Enrollment System

Test Case 1: Course is Full

Objective: Ensure the system raises an exception when attempting to enroll a student in a course that has reached its maximum capacity.

Preconditions:

1. Students table contains students with student_id = 5 and 4.
2. Courses table contains a course with course_id=2 and max_capacity=1.
3. There are exactly 0 students already enrolled in course 2.

Steps:

1. Execute EnrollStudent(5, 2) stored procedure.
2. Execute EnrollStudent(4, 2) stored procedure.
3. Catch the exception that is raised.
4. Confirm that the exception message is "Course is full."

Expected Results:

- An exception is raised with the message "Course is full."
- No changes are made to the Enrollments table after EnrollStudent(4, 2).
- The exception should stop further execution.

EnrollStudent(5, 2) makes course 2 full

Test case 1 : Course is full

```
query_stored_procedure_enroll_student(5, 2)
✓ 0.0s
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.
Student enrolled successfully in the course.

Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

```
Connection closed.
```

Course is full, can not implement EnrollStudent(4, 2)

```
query_stored_procedure_enroll_student(4, 2)
✓ 0.0s
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

The error occurred: Course is full
CONTEXT:  PL/pgSQL function enrollstudent(integer,integer) line 40 at RAISE

Connection closed.
```

No changes in Enrollments table

```
Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

Test Case 2: Course Does Not Exist

Objective: Verify that the system raises an exception when attempting to enroll in a non-existent course.

Preconditions:

1. Students table contains a student with student_id = 1.
2. Courses table does not contain a course with course_id = 10.

Steps:

1. Execute EnrollStudent(1, 10) stored procedure.
2. Catch the exception that is raised.
3. Confirm that the exception message is "Course does not exist."

Expected Results:

- An exception is raised with the message "Course does not exist."
- No changes are made to the Enrollments table.
- The exception should stop further execution.

Test case 2 : Course does not exist

```
query_stored_procedure_enroll_student(1, 10)
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

The error occurred: Course does not exist
CONTEXT:  PL/pgSQL function enrollstudent(integer,integer) line 23 at RAISE

Connection closed.
```

No changes in Enrollments table

Enrollments Table Data:

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

Test Case 3: Student Does Not Exist

Objective: Verify that the system raises an exception when attempting to enroll a student who does not exist.

Preconditions:

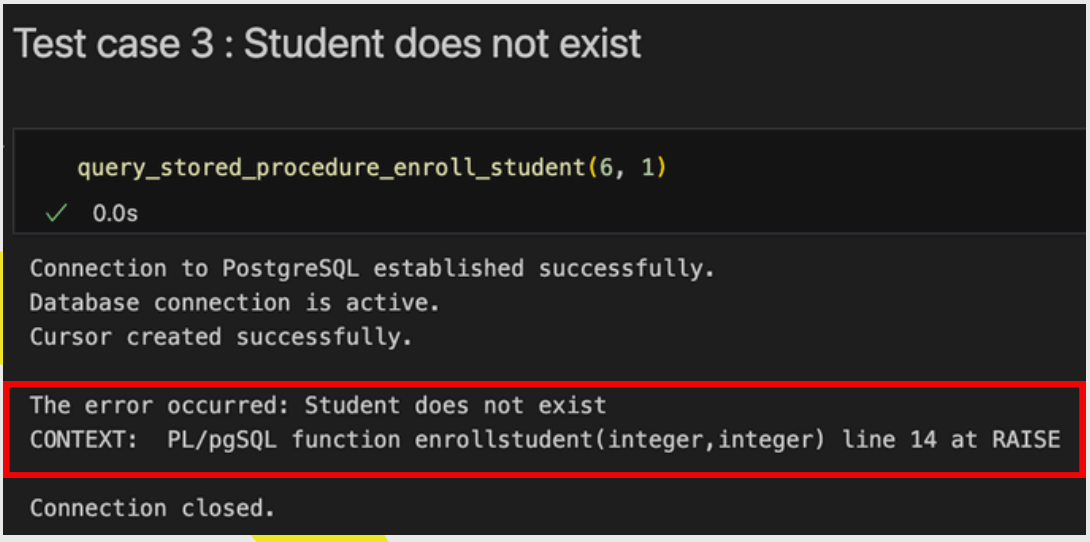
1. Students table does not contain a student with student_id = 6.
2. Courses table has a course with course_id = 1 and max_capacity = 2.

Steps:

1. Execute EnrollStudent(6, 1) stored procedure.
2. Catch the exception that is raised.
3. Confirm that the exception message is "Student does not exist."

Expected Results:

- An exception is raised with the message "Student does not exist."
- No changes are made to the Enrollments table.
- The exception should stop further execution.



No changes in Enrollments table

Enrollments Table Data:

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

Test Case 4: Student Already Enrolled

Objective: Ensure that the system raises a notice when attempting to enroll a student who is already enrolled in the course.

Preconditions:

1. Students table contains a student with student_id = 5.
2. Courses table contains a course with course_id=3 and max_capacity=5.
3. The student with student_id = 5 is already enrolled in course 3.

Steps:

1. Execute EnrollStudent(5, 3) stored procedure.
2. Confirm that a notice is raised with the message "Student is already enrolled in this course."
3. Query the Enrollments table to ensure no new enrollment record was added.
4. Confirm that the Enrollments table still contains the original record.

Expected Results:

- An exception is raised with the message "Student is already enrolled in this course."
- No changes are made to the Enrollments table.
- The original enrollment record remains intact.

current Enrollments table

Enrollments Table Data:

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

EnrollStudents(5, 3)

```
query_stored_procedure_enroll_student(5, 3)
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

The error occurred: Student is already enrolled in this course
CONTEXT:  PL/pgSQL function enrollstudent(integer,integer) line 53 at RAISE

Connection closed.
```

No changes in Enrollments table after execution

Enrollments Table Data:

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

Test Case 5: Trigger Error when Deleting a Non-Existing Enrollment Record and Verify EnrollmentLog

Objective: Ensure that the system correctly raises an error when attempting to delete a non-existing enrollment record and EnrollmentLog does not record wrong operation.

Preconditions:

1. Students table contains a student with student_id = 1.
2. Courses table contains a course with course_id = 1 and max_capacity = 2.
3. There are 0 students currently enrolled in course 1.

Steps:

1. Drop student 1 from course 1 by deleting their record in the Enrollments table.
2. Execute EnrollStudent(1, 1) again to enroll the student after a slot becomes available.
3. Confirm that the student is successfully enrolled.

Expected Results:

- An exception is raised with the message "The student is not enrolled in the course." after the first event
- The student is successfully enrolled in course 1 after the second event
- EnrollmentLog only records new Enroll action, not the Drop action
- The remaining capacity in the CourseCapacityView is reduced by 1.
- The Enrollments table reflects the new enrollment record.

Current Tracking tables

Enrollments Table Data:

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779

Courses Table Data:

	course_id	course_name	department	max_capacity	course_capacity
0	1	Introduction to Python	Computer Science	2	0
1	4	Engineering Design	Engineering	3	0
2	3	Physics I	Physics	5	2
3	2	Calculus I	Mathematics	1	1

EnrollmentLog Table Data:

	log_id	student_id	course_id	action	action_timestamp
0	1	5	3	enroll	2024-11-08 15:39:34.412706
1	2	1	3	enroll	2024-11-08 15:39:38.236148
2	3	3	3	enroll	2024-11-08 15:39:49.074563
3	4	3	3	drop	2024-11-08 15:48:34.553665
4	5	5	2	enroll	2024-11-08 15:54:24.715779

Try dropping enrollment (1, 1)

```
trigger_event(1, 1, "drop")
```

```
Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.

The error occurred: The student is not enrolled in the course.
Connection closed.
```

EnrollStudent(1, 1) again

```
trigger_event(1, 1, "enroll")
✓ 0.1s

Connection to PostgreSQL established successfully.
Database connection is active.
Cursor created successfully.
Student enrolled successfully in the course.
Check the updated enrollment log after the event:

EnrollmentLog Table Data:
```

	log_id	student_id	course_id	action	action_timestamp
0	1	5	3	enroll	2024-11-08 15:39:34.412706
1	2	1	3	enroll	2024-11-08 15:39:38.236148
2	3	3	3	enroll	2024-11-08 15:39:49.074563
3	4	3	3	drop	2024-11-08 15:48:34.553665
4	5	5	2	enroll	2024-11-08 15:54:24.715779
5	6	1	1	enroll	2024-11-08 16:42:57.602475

drop event is not recorded in the system, only the enroll event

```
Check the updated course capacity after the event:

Courses Table Data:
```

	course_id	course_name	department	max_capacity	course_capacity
0	4	Engineering Design	Engineering	3	0
1	3	Physics I	Physics	5	2
2	2	Calculus I	Mathematics	1	1
3	1	Introduction to Python	Computer Science	2	1

```
Connection closed.
```

Enrollments Table get updated

```
Enrollments Table Data:
```

	enrollment_id	student_id	course_id	enrollment_date
0	1	5	3	2024-11-08 15:39:34.412706
1	2	1	3	2024-11-08 15:39:38.236148
2	4	5	2	2024-11-08 15:54:24.715779
3	5	1	1	2024-11-08 16:42:57.602475

thank you

LUAN TRAN

BS22DSY024