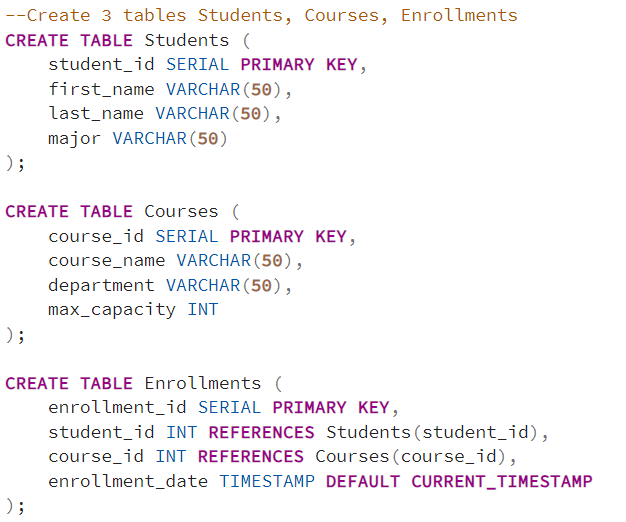
**Simplified Student Enrollment System**

**Part 1: SQL Development**

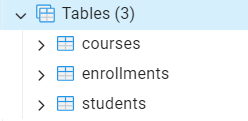
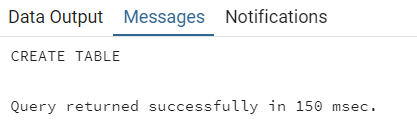
**Create Tables**

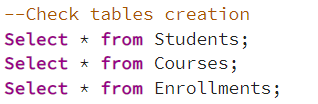
Create 3 tables: Students table, Courses table, and Enrollments table

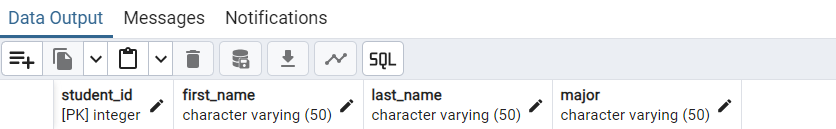
**Code:** Create 3 tables

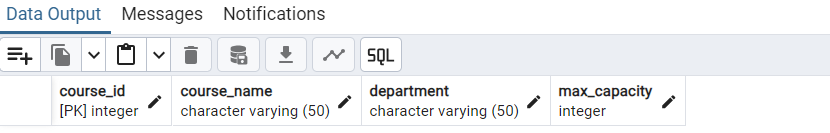


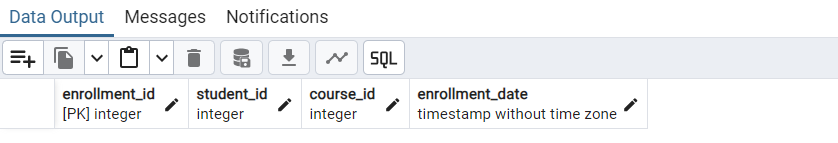
**Output**: Successful tables creation





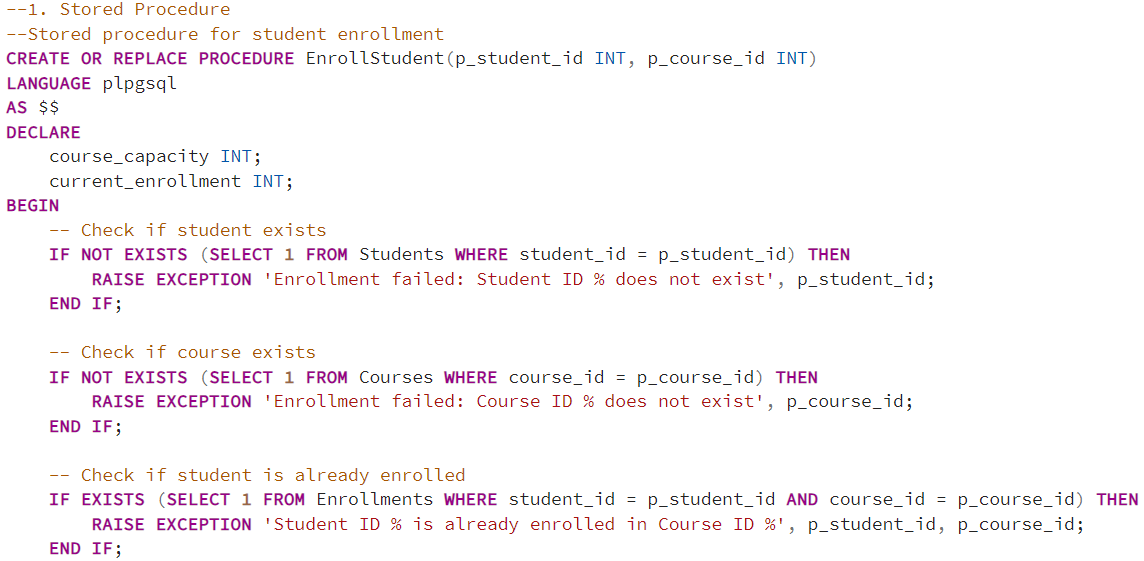


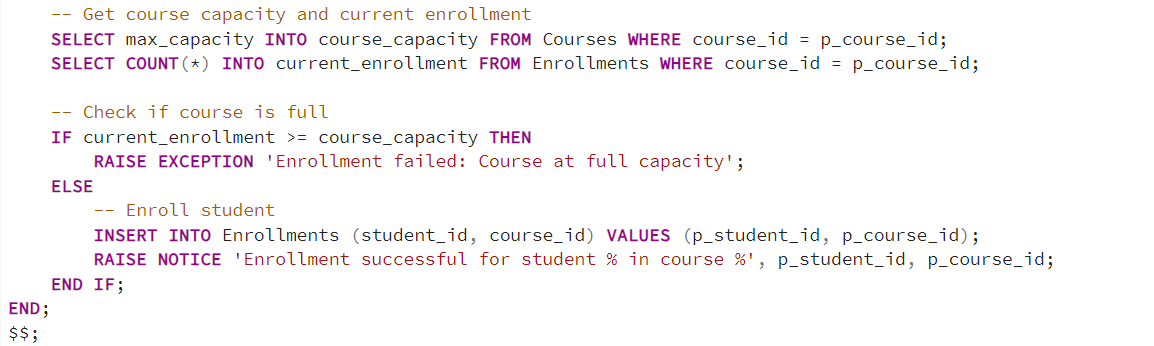




# **1.1. Create Stored Procedure for Student Enrollment**

**Code**: Create a stored procedure named EnrollStudent()

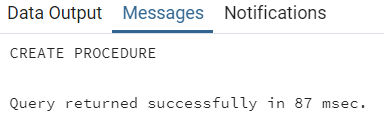




**Explanation**: In the stored procedure, checks for:

* Student existence: if not exist → raise exception that student does not exist
* Course existence
* Enrollment status: check if student is already enrolled or not
* Course capacity: check if the course is full or not
* If student does not exist and course is not full, then enroll student in course by INSERT INTO Enrollments table

**Output**: Successful creation of the stored procedure

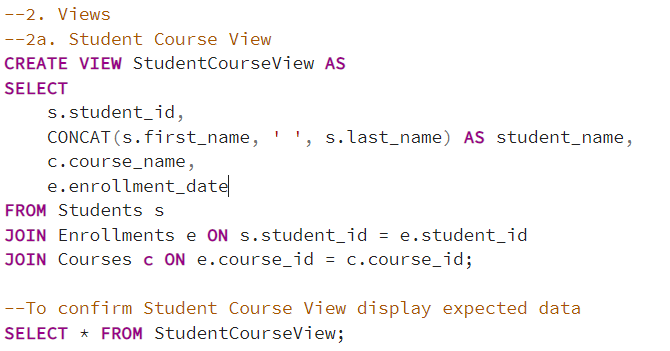


# **1.2. Create Views**

Create views for displaying enrollment and capacity information

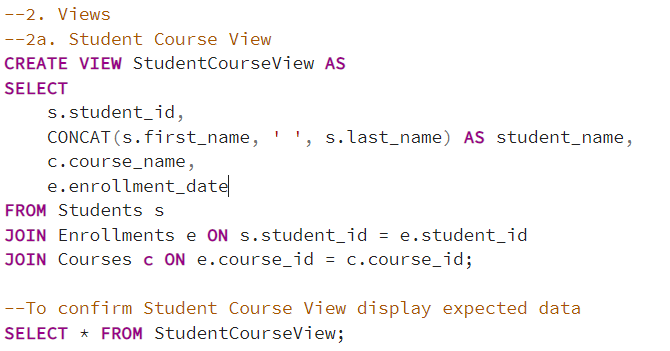
**Student Course View**

**Code**: Create StudentCourseView that combines data from Students table, Courses table, and Enrollments table to display details like student\_id, student\_name, course\_name, and enrollment\_date.



**Output**: StudentCourseView successful creation

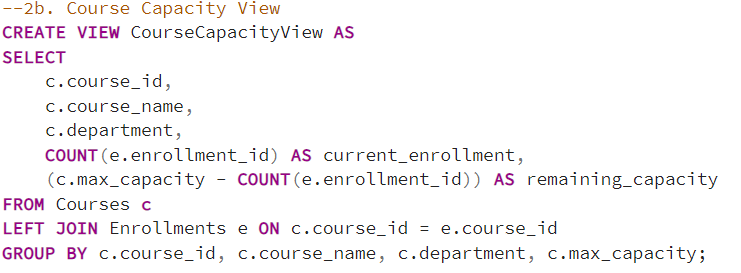




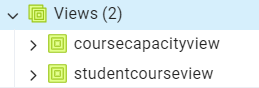
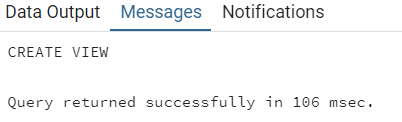


**Course capacity View**

**Code**: CourseCapacityView creation



**Output**: CourseCapacityView successful creation



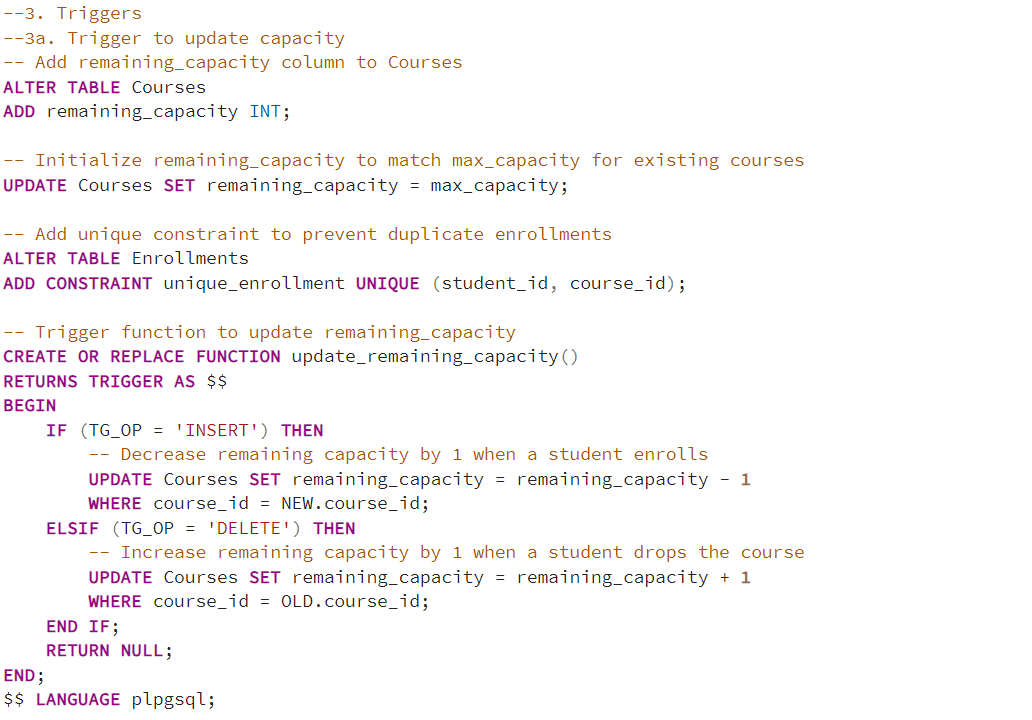


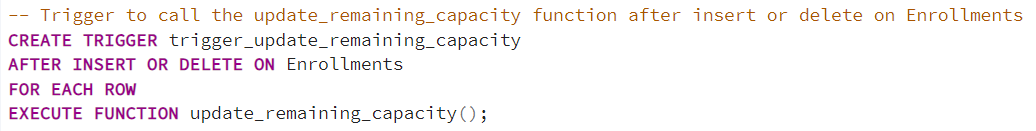


# **1.3. Create Triggers**

## **Trigger to Update Capacity**

**Code**: Create trigger to update capacity

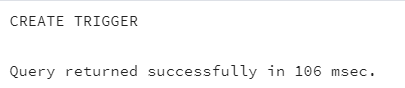




**Explanation**:

* Step 1: Add a new column remaining\_capacity to the Courses table, which will keep track of how many spots are still available in each course.
* Step 2: Set remaining\_capacity equal to max\_capacity for all existing courses, making sure remaining\_capacity starts with the correct value.
* Step 3: Add unique constraint to ensure that each student can enroll in each course only once, preventing duplicate enrollment records
* Step 4: Create a function update\_remaining\_capacity() to adjust the max\_capacity of a course based on enrollments.
  + If a new row is inserted into Enrollments table (a student enrolls), it decreases remaining\_capacity by 1 for the associated course\_id.
  + If a row is deleted from Enrollments table(a student withdraws), it increases remaining\_capacity by 1 for that course\_id.
* Step 5: Create trigger trigger\_update\_remaining\_capacity(): This trigger is activated after every INSERT or DELETE operation on the Enrollments table, ensuring that any changes to enrollments are immediately reflected in the course's max\_capacity .

**Output**: Trigger successful creation



## **Trigger to Log Enrollment Events**

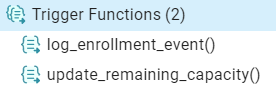
**Code**: Create trigger to log enrollment events



**Explanation**:

* Step 1: Create Enrollmentlog table to store log entries with a unique log\_id, action type (either 'ENROLL' or 'DROP'), student\_id, course\_id, and a timestamp (event\_time) of the action.
* Step 2: Create function log\_enrollment\_event() to detect actions on the Enrollments table:
  + If a student enrolls (INSERT), it logs the action as 'ENROLL' with student\_id and course\_id.
  + If a student withdraws (DELETE), it logs the action as 'DROP'.
* Step 3: Create trigger trigger\_log\_enrollment. This trigger runs after any INSERT or DELETE on Enrollments, automatically calling log\_enrollment\_event to record each action.

**Output**: Successful creation of 2 triggers



**Part 2: Python Integration**

# **Database Connection and Connection Test**

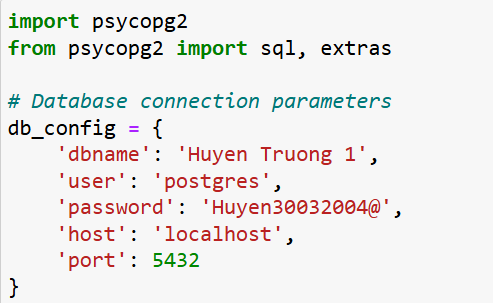
**Explanation**:

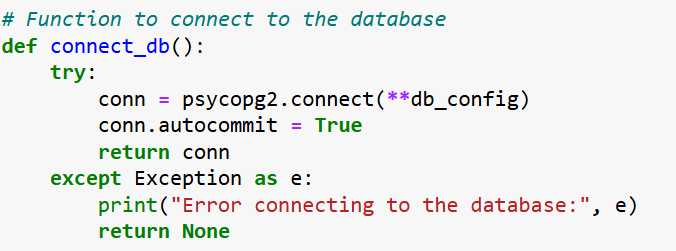
* The connect\_db() function establishes a connection to the PostgreSQL database using psycopg2 and returns a connection object.
* The test\_connection() function uses this connection to run a simple query, ensuring that the database is accessible.

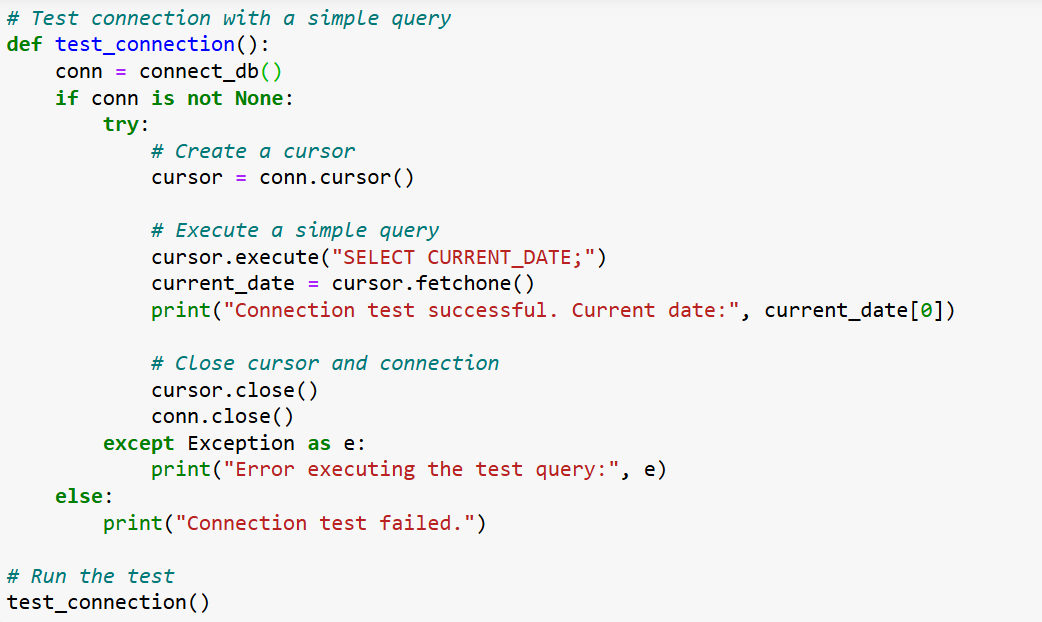
These functions verify database accessibility before further operations. test\_connection() confirms a successful connection with a basic date query.

**Code**:







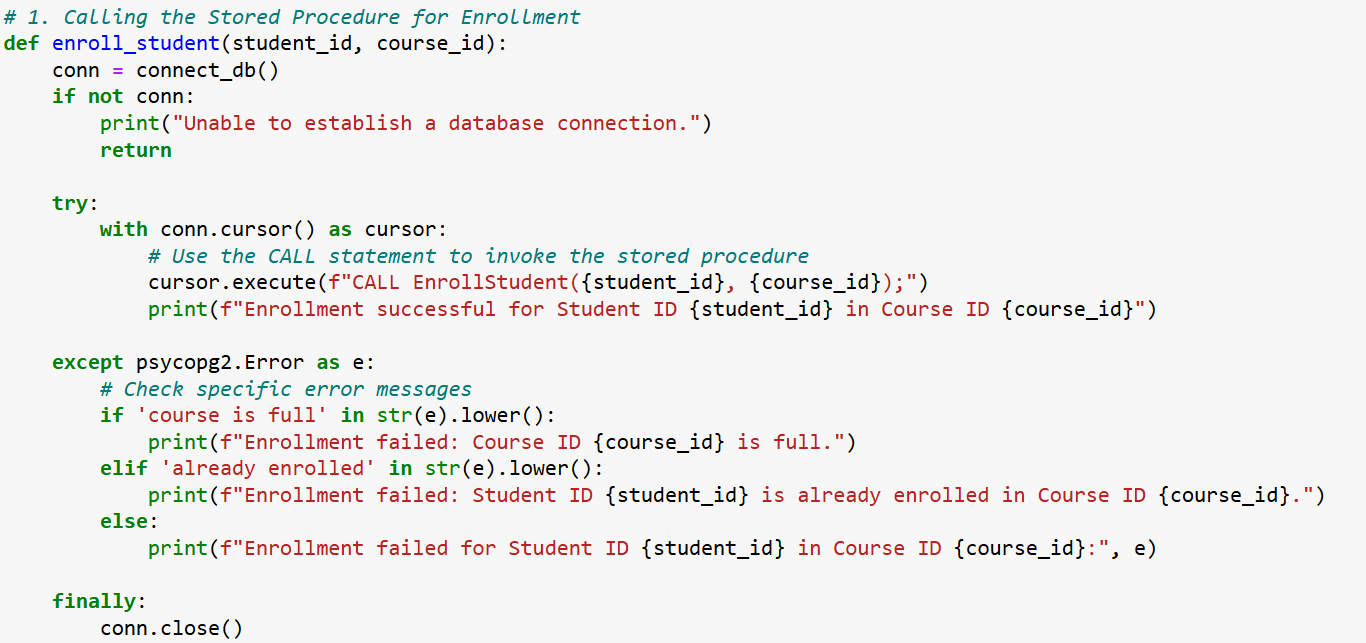


# **Calling the Stored Procedure**

**Explanation**:

* The function enroll\_student() takes student\_id and course\_id as input parameters, calling the stored procedure EnrollStudent in the database by using the CALL statement.
* The function handles exceptions for situations like a full course or a duplicate enrollment.

**Code**:



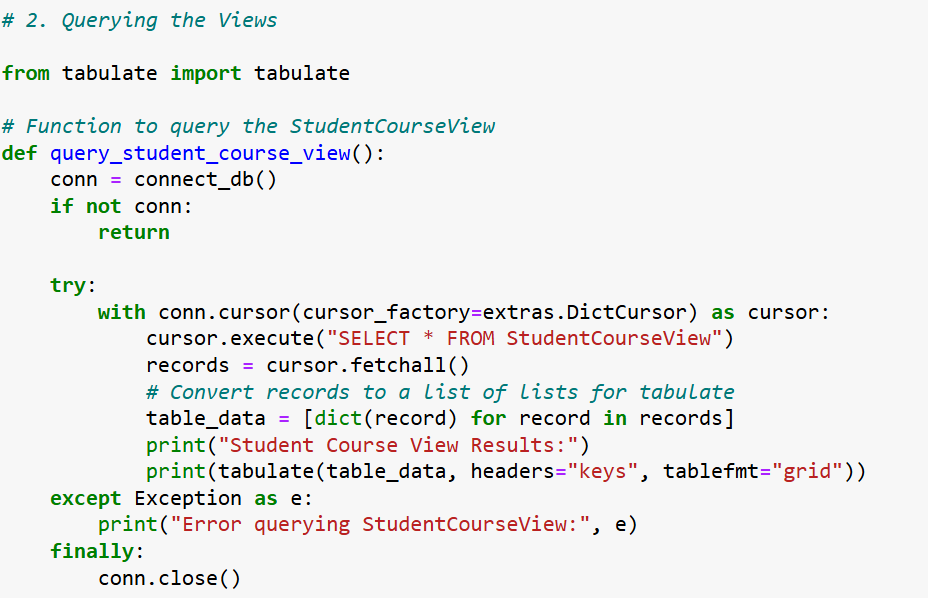
# **Querying the Views:**

**Explanation:**

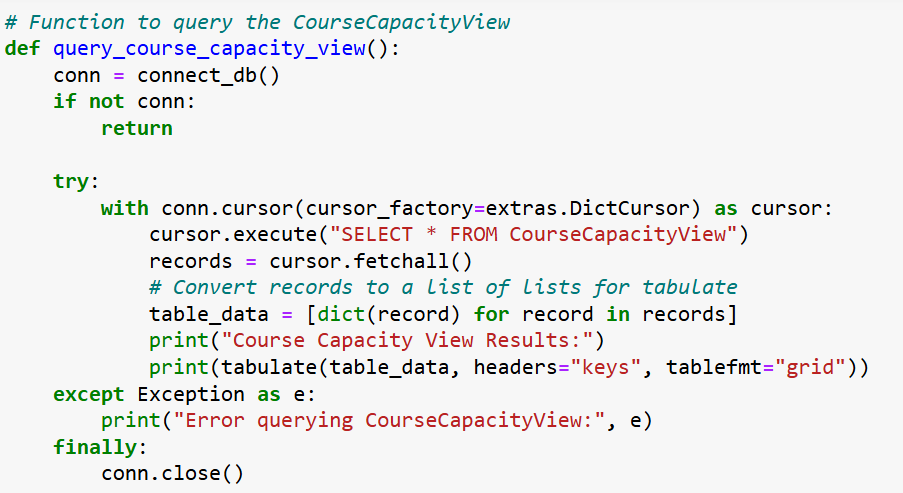
* query\_student\_course\_view()retrieves data from StudentCourseView to display each student’s enrollment details.
* query\_course\_capacity\_view()retrieves data from CourseCapacityView to show the course capacity and current enrollment.

Both functions use the tabulate library to improve readability of the output.

**Query the Student Course View**



**Query the Course Capacity View**

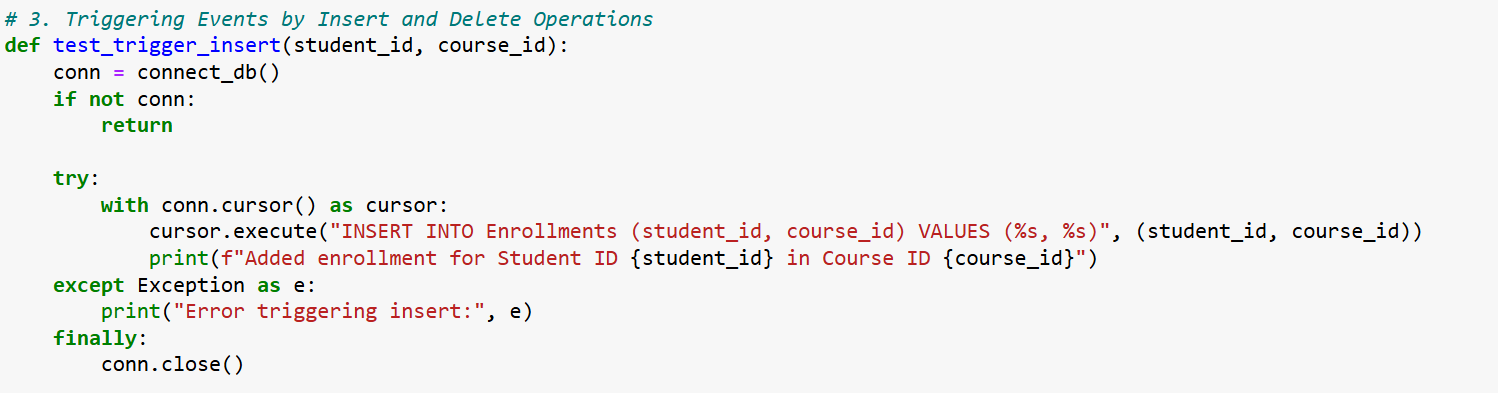


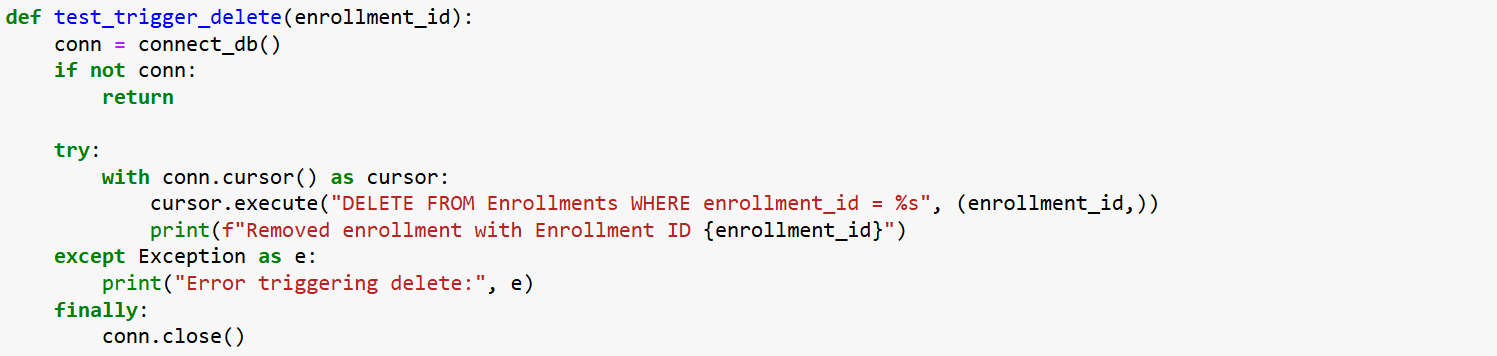
# **Trigger events**

**Explanation:**

* test\_trigger\_insert() inserts a new enrollment record, triggering any associated database actions like updating course capacity and logging.
* test\_trigger\_delete() removes an enrollment, triggering updates in course capacity and logging if such triggers exist.

**Code:**





The test results and outcomes for the triggers are detailed in the test cases section below.

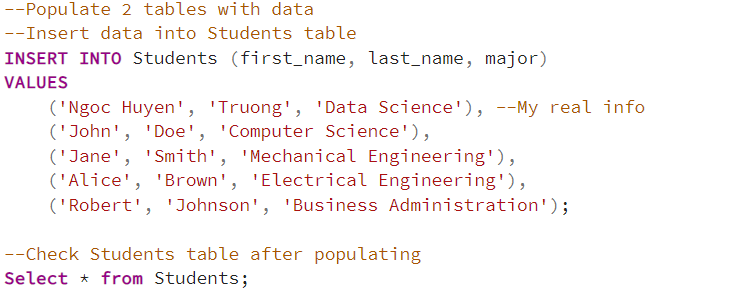
**Test Plans**

# **Populate data**

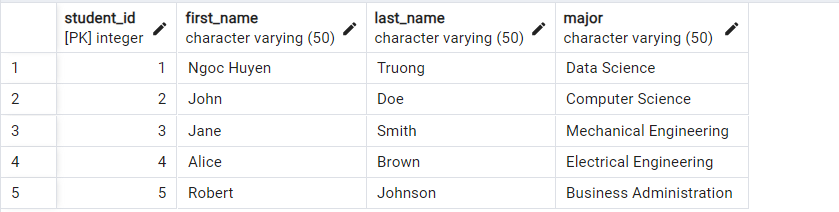
Populate 2 tables Students table and Courses table with data in SQL code:

Since the Students table defines student\_id as SERIAL and the Courses table defines course\_id as SERIAL, unique integer IDs are automatically generated for each student and course upon data insertion. Therefore, only the relevant column information was provided when populating the tables.

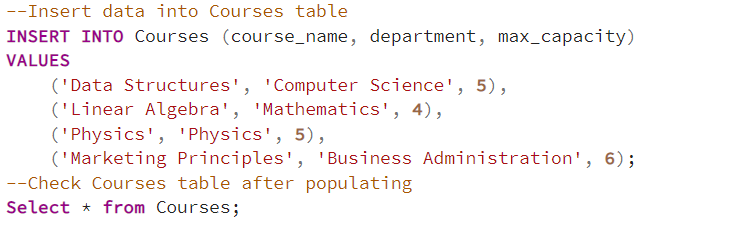
**Code**: Insert data into Students table:



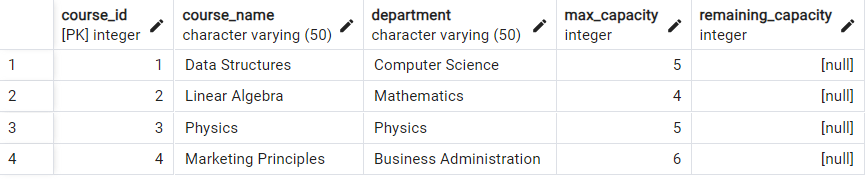
**Output**: Students table after inserting data



**Code**: Insert data into Courses table:



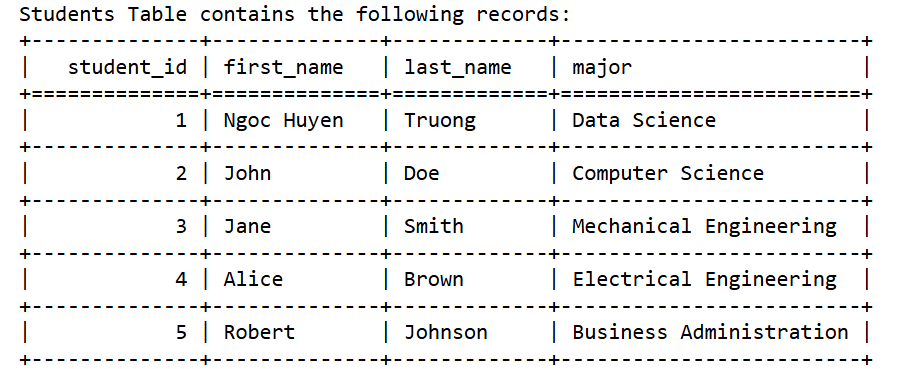
**Output**: Courses table after inserting data



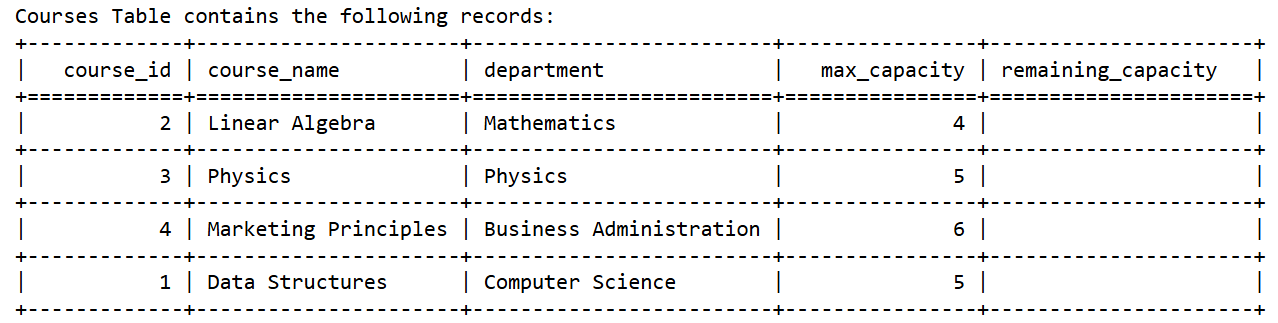
For now, leave the Enrollments table empty, as it will be populated during the stored procedure tests.

**Review 2 tables Students and Courses in Python**

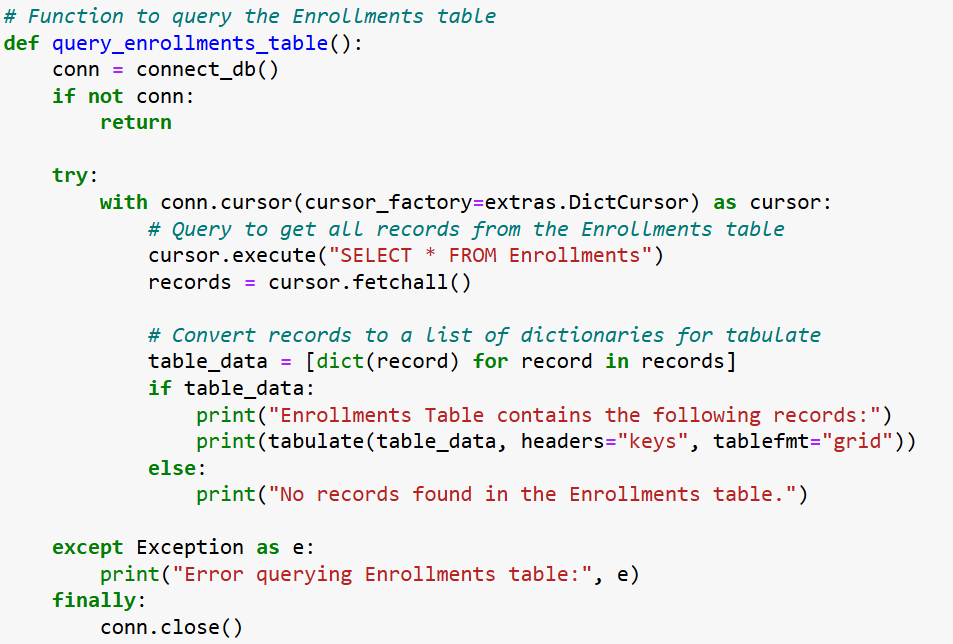
Students table:

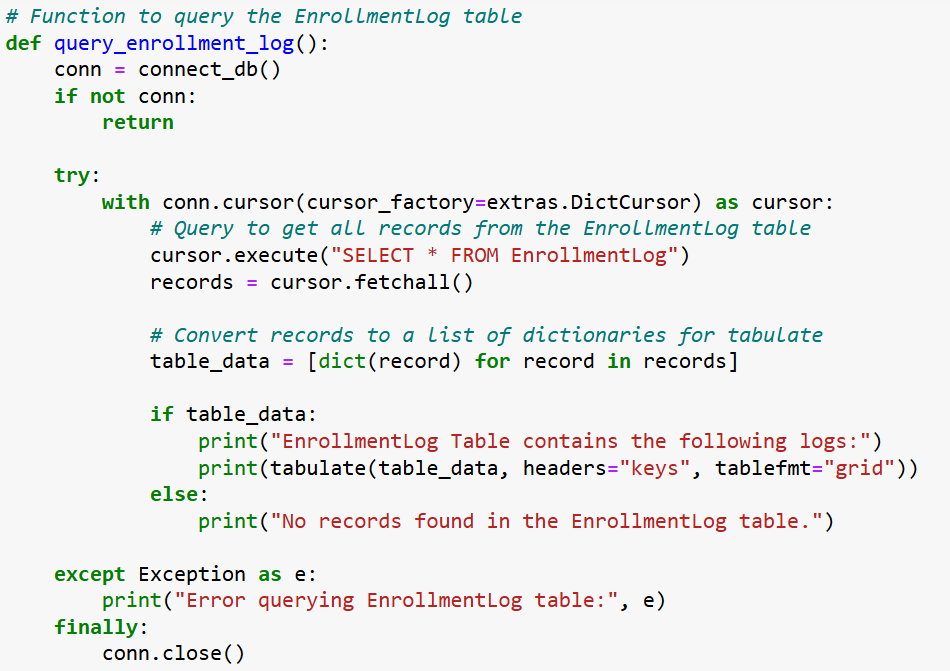


Courses table:

Before proceeding with the test cases, I created two functions:

* **Function to query the Enrollments table**: This function checks the Enrollments table to verify if a student has been successfully enrolled in a course.
* **Function to query the EnrollmentLog table**: This function queries the EnrollmentLog table to ensure that an enrollment record was added when the student enrolls or drops a course.

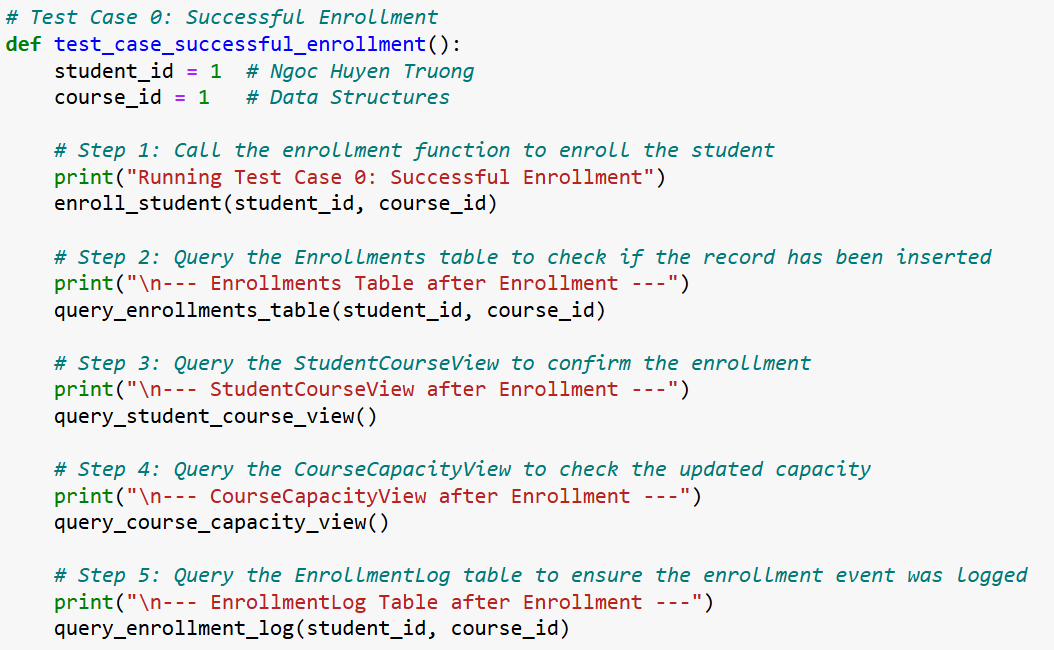
**1. Function to query the Enrollments table** 

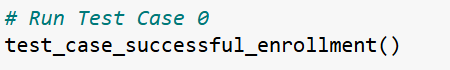
**2. Function to query the EnrollmentLog table** 

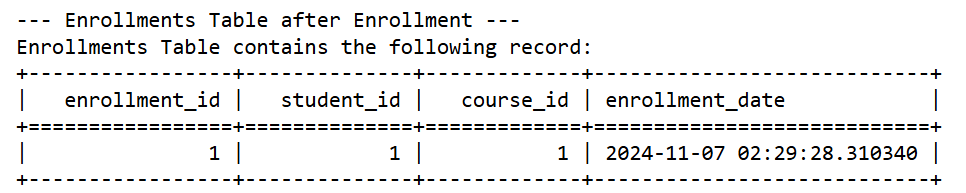
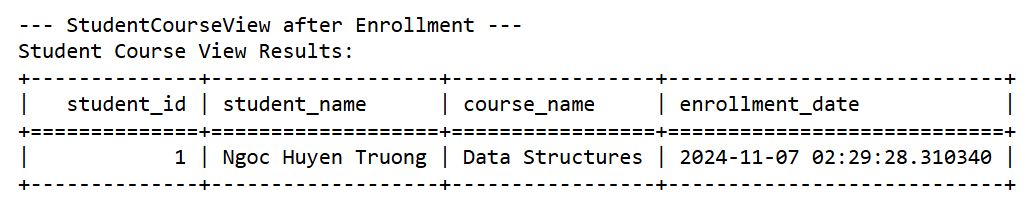
# **Test Plan: University Enrollment System**

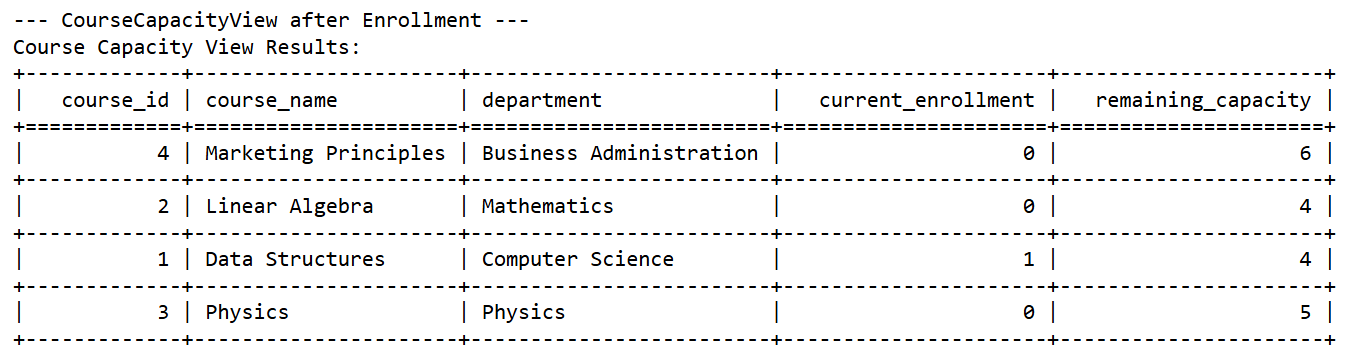
|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 0: Successful Enrollment (Sample test)** **Objective**: Ensure that a student can be successfully enrolled in a course when the course has available capacity and the student is not already enrolled.  · **Preconditions**:  1. Students table has a student with student\_id = 1 (Ngoc Huyen Truong).  2. Courses table has a course with course\_id = 1 (Data Structures) and max\_capacity = 5.  3. There are currently fewer than 5 students enrolled in course 1.  · **Steps**:  1. Execute the EnrollStudent(1, 1) stored procedure from the Python script.  2. Query the Enrollments table to check if the record has been inserted.  3. Query the StudentCourseView to confirm the enrollment.  4. Query the CourseCapacityView to ensure the remaining capacity is reduced by 1.  · **Expected Results**:  1. Enrollments table has a new record for student\_id = 1 and course\_id = 1.  2. StudentCourseView displays the student and course with the correct enrollment\_date.  3. CourseCapacityView shows the remaining\_capacity decreased by 1.  4. The EnrollmentLog table logs the enrollment event with the action ENROLL. |

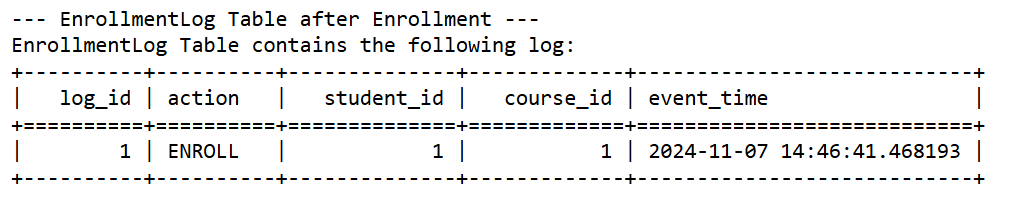
**Code**:

**Output**:



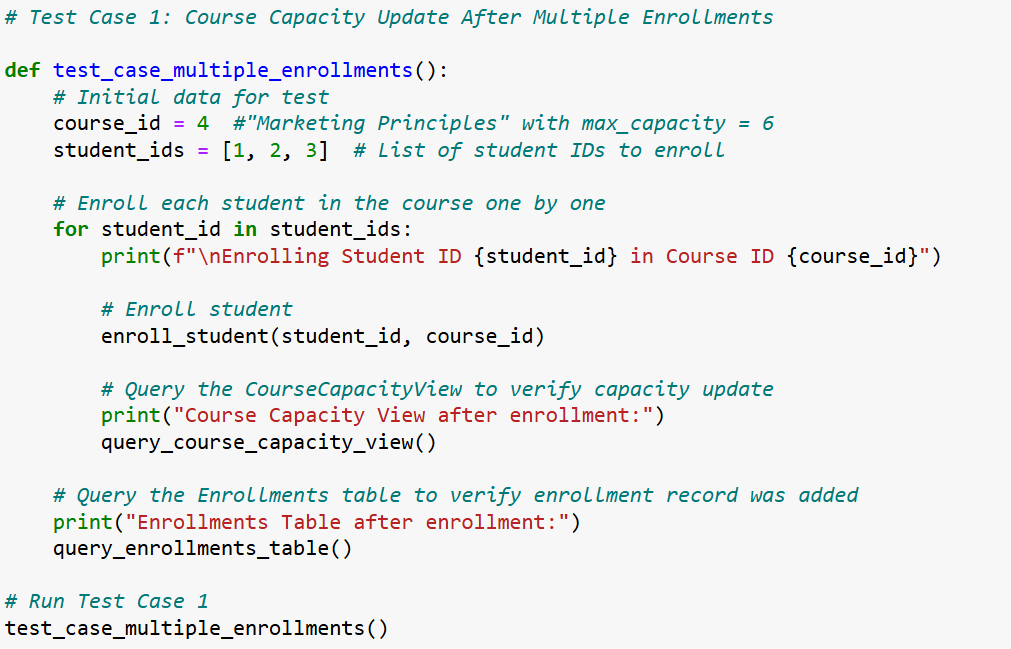
1. Enrollments table has a new record for student\_id = 1 and course\_id = 1.
2. StudentCourseView displays the student and course with the correct enrollment\_date.
3. CourseCapacityView shows the remaining\_capacity decreased by 1. After enrolling the student with student\_id = 1 in the Data Structures course, which has a maximum capacity of 5, the CourseCapacityView was checked to verify the enrollment impact. The view now reflects an updated current\_enrollment = 1 and remaining\_capacity = 4, confirming that the enrollment was processed correctly and the capacity has been decreased by 1 as expected.



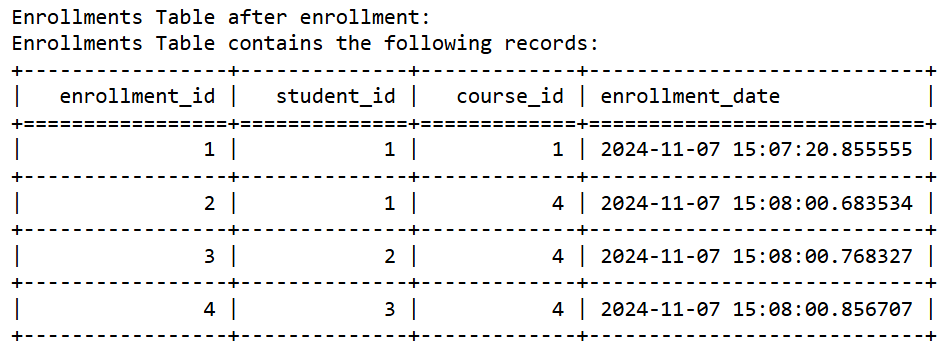
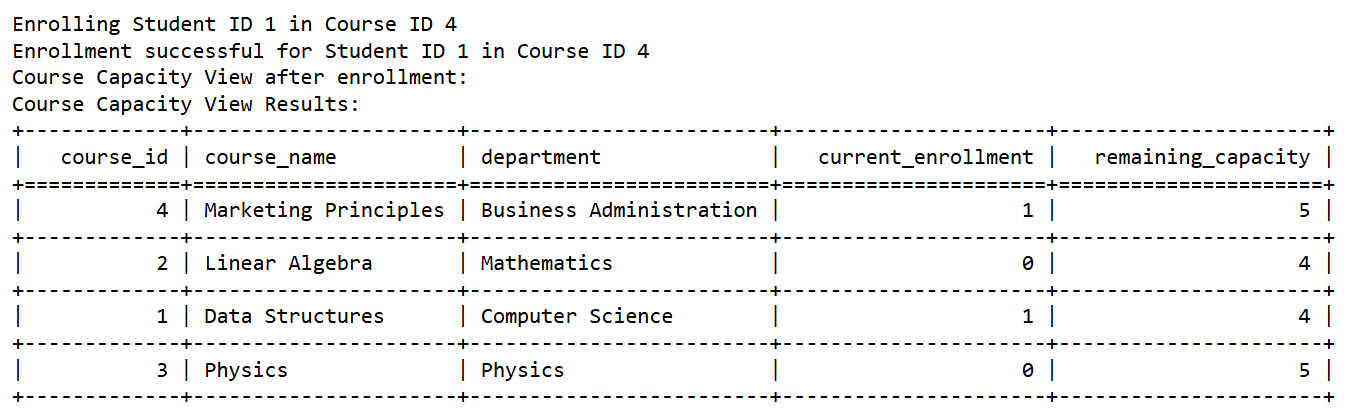
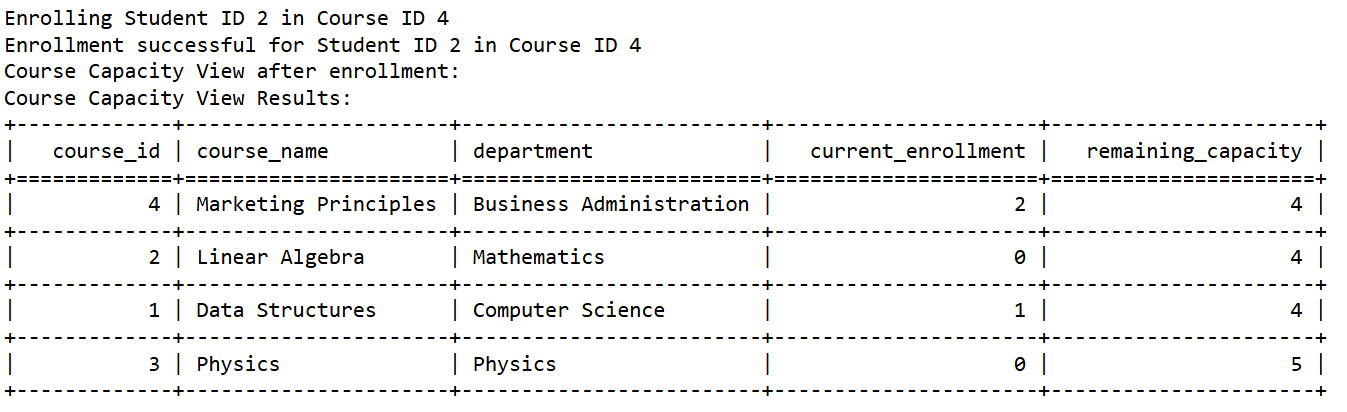
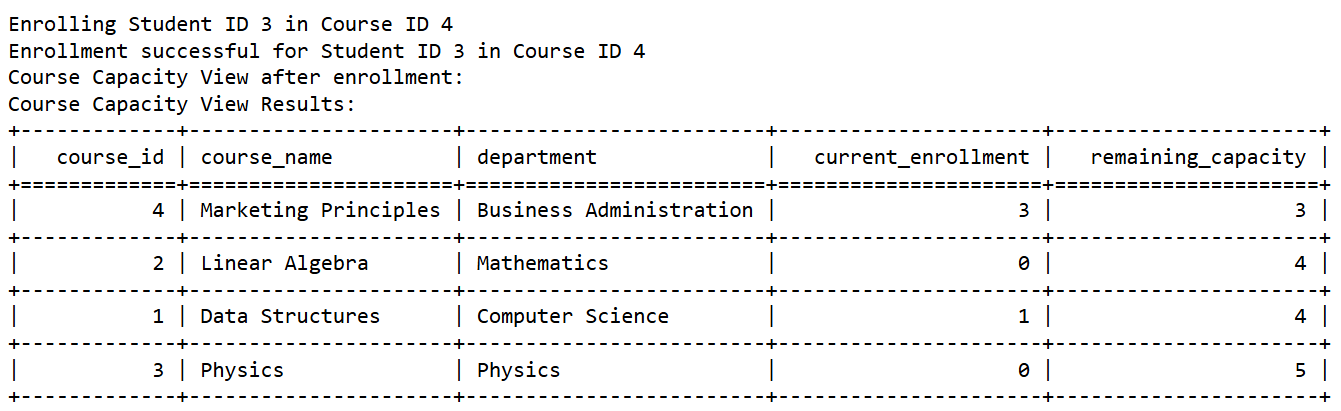
4. The EnrollmentLog table logs the enrollment event with the action ENROLL 

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 1: Course Capacity Update After Multiple Enrollments** **Objective**: Confirm that the system accurately updates the remaining course capacity as students enroll.  · **Preconditions**:  1. Course course\_id = 4 (e.g., Marketing Principles) has a max capacity of 6.  2. Multiple students are not yet enrolled in course\_id = 4.  · **Steps**:  1. Enroll three students (e.g., student\_id = 1, student\_id = 2, and student\_id = 3) in course\_id = 4 one by one by executing the EnrollStudent() stored procedure from the Python script.  2. Query the CourseCapacityView after each enrollment to confirm remaining capacity decreases by 1 each time.  · **Expected Results**:  1. Enrollments table has a new record for 3 students student\_id = 1,student\_id = 2, and student\_id = 3 in course\_id = 4.  2. After each enrollment, CourseCapacityView reflects the adjusted remaining capacity. |

**Code**:

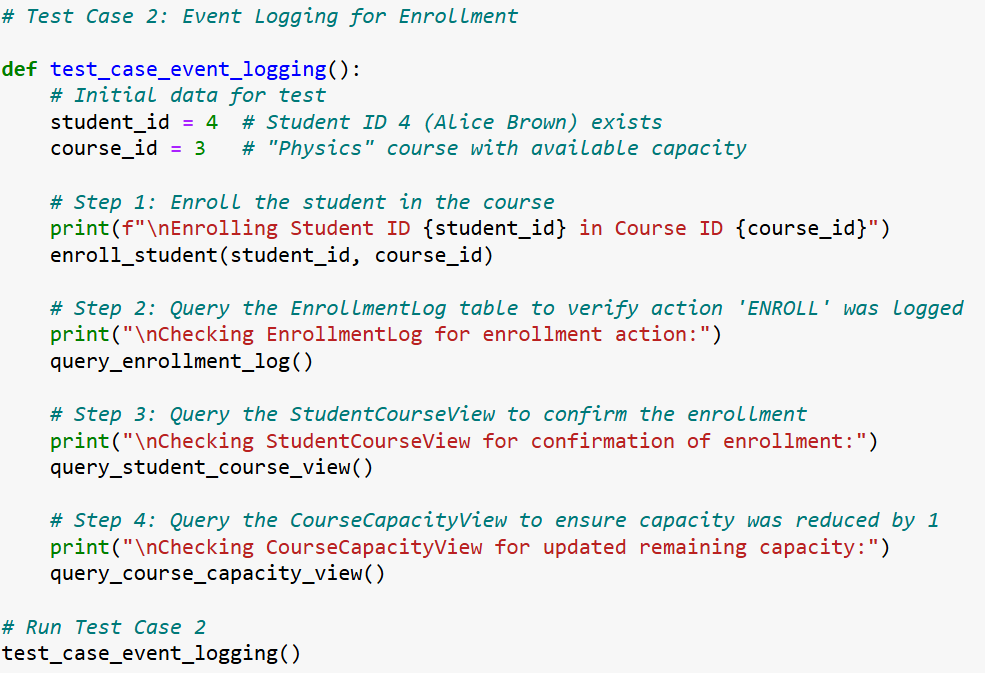


**Output:**

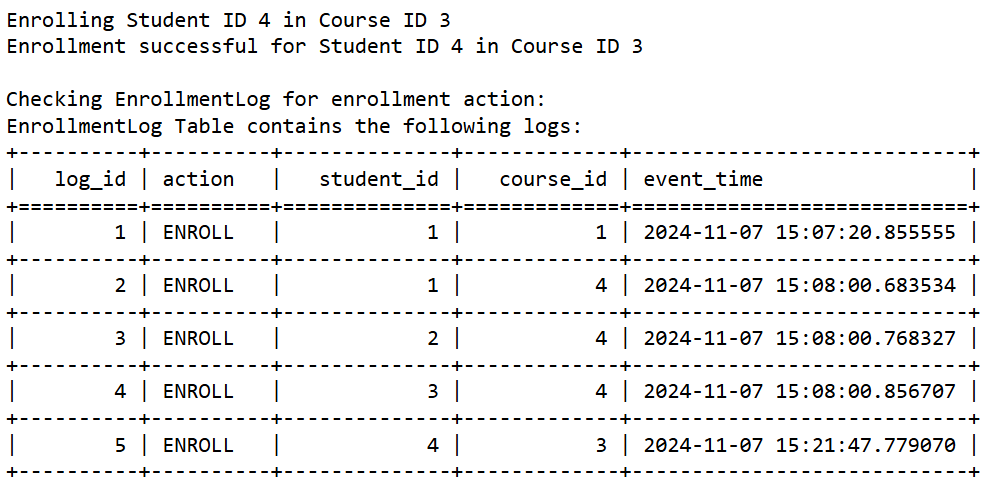
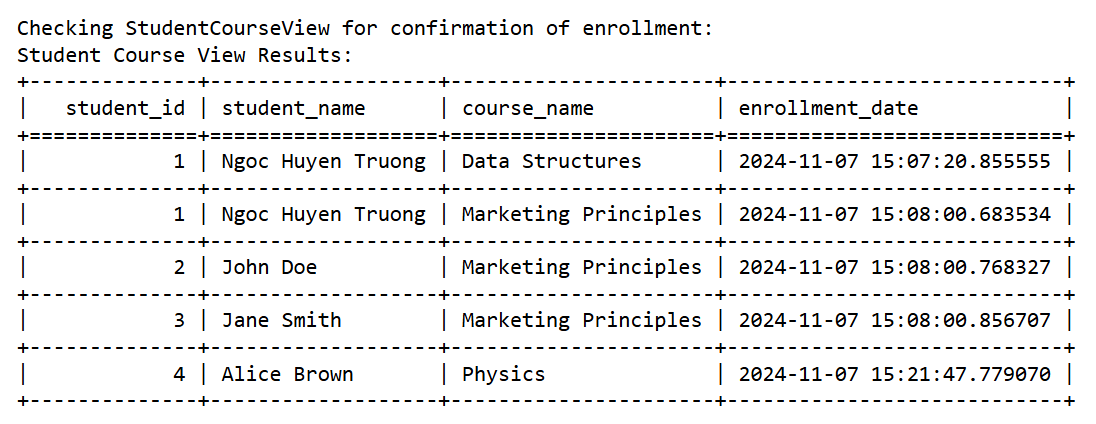
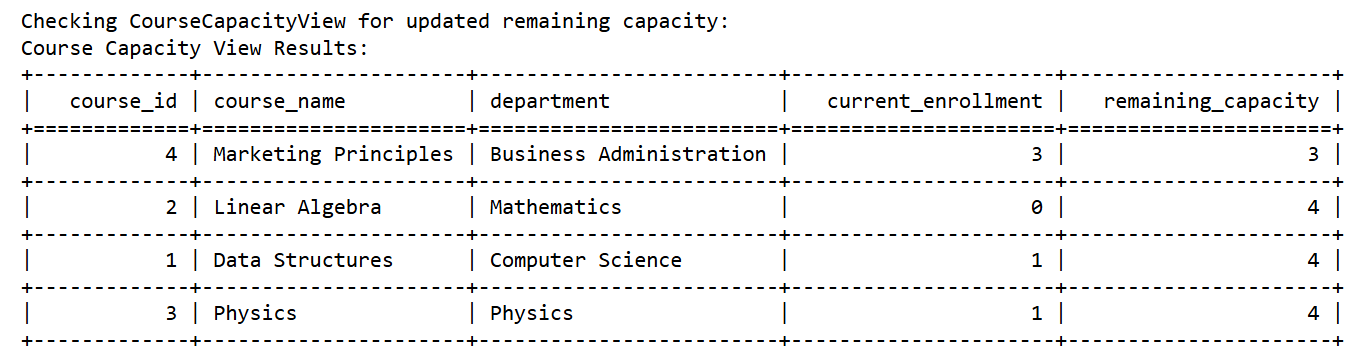
1. Enrollments table has a new record for 3 students student\_id = 1,student\_id = 2, and student\_id = 3 in course\_id = 4. 
2. After each enrollment, CourseCapacityView reflects the adjusted remaining capacity. For student\_id = 1 in course\_id = 4 (max capacity = 6), after successful enrollment, current\_emrollment = 1 and remaining\_capacity = 5.****For student\_id = 2 in course\_id = 4 (max capacity = 6), after successful enrollment, current\_emrollment = 2 and remaining\_capacity = 4.****For student\_id = 3 in course\_id = 4 (max capacity = 6), after successful enrollment, current\_emrollment = 3 and remaining\_capacity = 3.****

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 2: Event Logging for Enrollment** **Objective**: Verify that each enrollment action is logged in the EnrollmentLog table.  · **Preconditions**:  1. Students table has a student with student\_id = 4 (Alice Brown).  2. Courses table has a course with course\_id = 3 (Physics).  3. Student with student\_id = 4 is not yet enrolled in course\_id = 3.  · **Steps**:  1. Execute the EnrollStudent(4, 3) stored procedure from the Python script.  2. Query the EnrollmentLog table to check if the action was logged with action = ENROLL.  3. Query the StudentCourseView to confirm the enrollment.  4. Query the CourseCapacityView to ensure the remaining capacity is reduced by 1.  · **Expected Results**:  1. Enrollments table has a new record for student\_id = 4 and course\_id = 3.  2. StudentCourseView displays the student and course with the correct enrollment\_date.  3. CourseCapacityView shows the remaining\_capacity decreased by 1.  4. The EnrollmentLog table logs the enrollment event with the action ENROLL. |

**Code:**

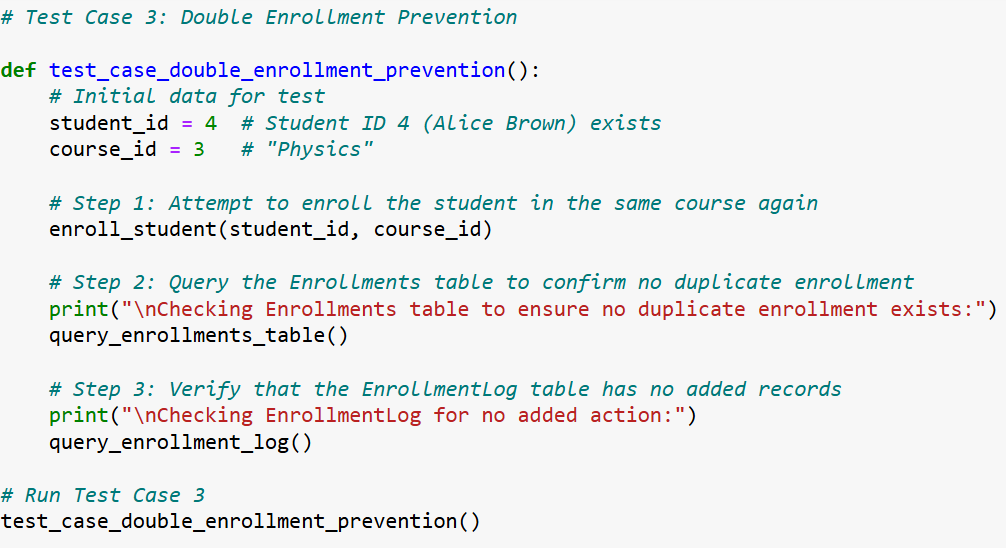
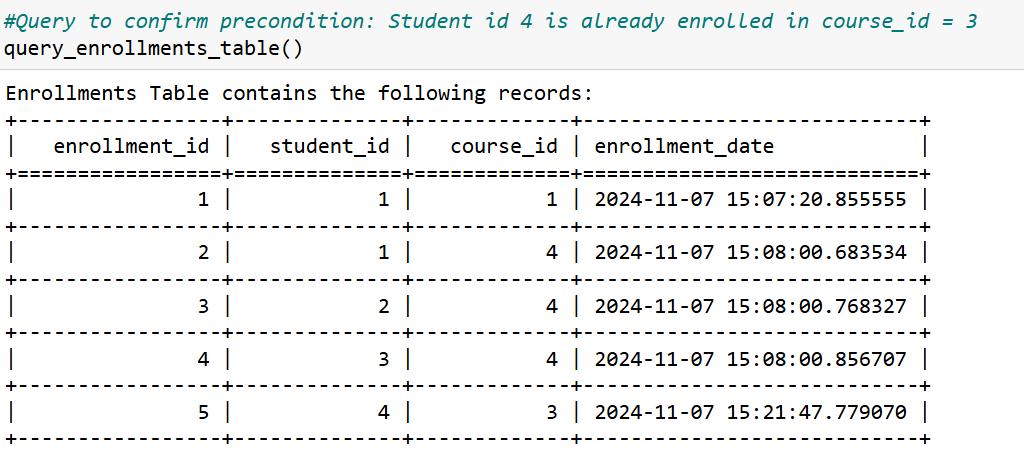


**Output**:

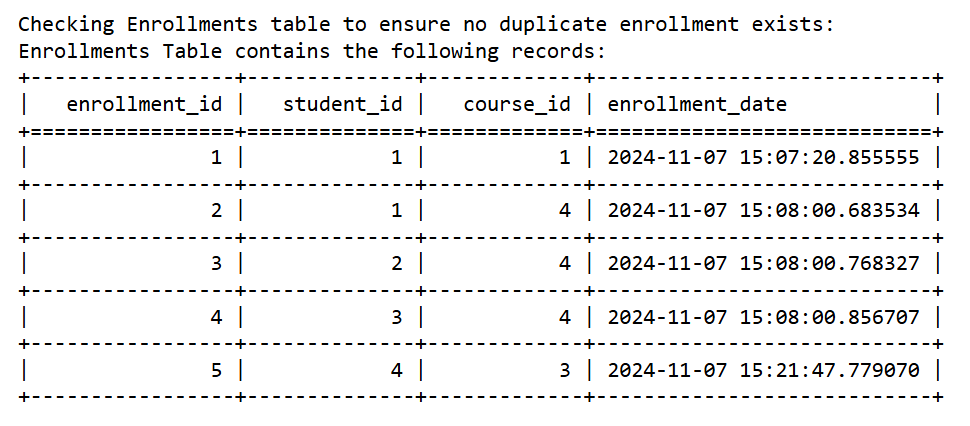
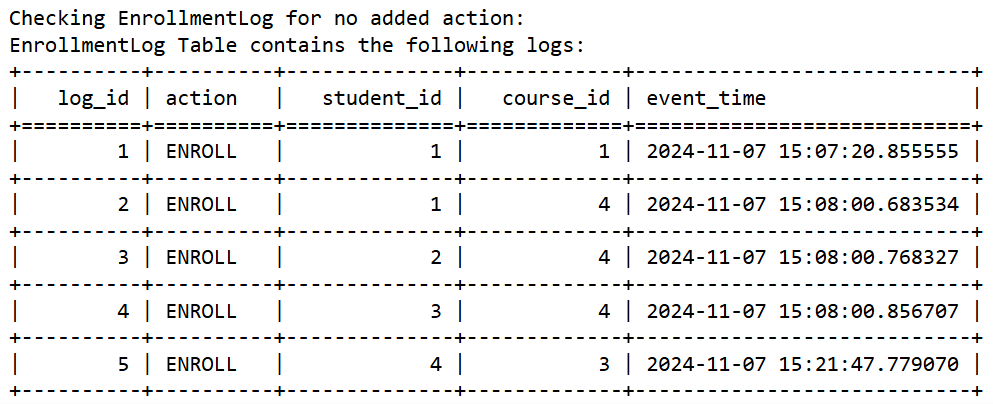
1. The EnrollmentLog table logs the enrollment event with the action ENROLL.
2. StudentCourseView displays the student and course with the correct enrollment\_date. The view shows that student\_id = 4 (Alice Brown) is enrolled course\_id = 3 (Physics)
3. CourseCapacityView shows the remaining\_capacity of the Physics course decreased by 1.

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 3: Double Enrollment Prevention** **Objective**: Ensure that a student cannot enroll in the same course twice.  · **Preconditions**:  1. Students table has a student with student\_id = 4 (Alice Brown).  2. Courses table has a course with course\_id = 3 (Physics).  3. Student with student\_id = 4 is already enrolled in course\_id = 3.  · **Steps**:  1. Execute the EnrollStudent(4, 3) stored procedure from the Python script to try enrolling this student again.  2. Query the Enrollments table to verify no duplicate enrollment.  3. Verify that the EnrollmentLog table logs the "already enrolled" status.  · **Expected Results**:  1. Enrollment fails, and a message "Enrollment failed: Student ID 4 is already enrolled in Course ID 3" is displayed.  2. No duplicate record is added to the Enrollments table.  3. The EnrollmentLog table has no added action. |

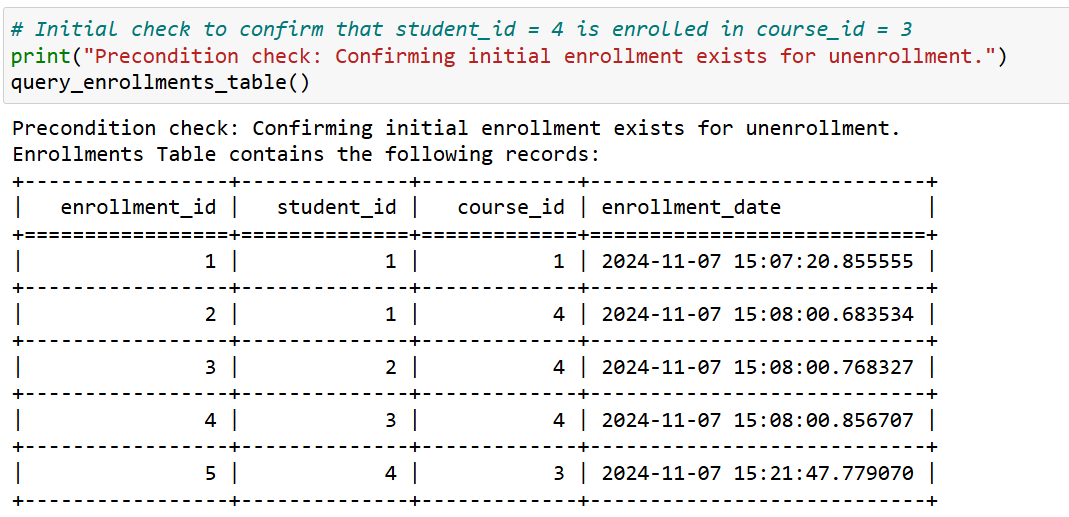
**Code**: Precondition: student\_id = 4 is already enrolled in course\_id = 3

The last row in the Enrollments table shows that a student with student\_id = 4 is already enrolled in course\_id = 3.

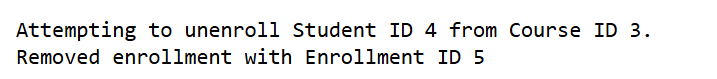
**Output**:

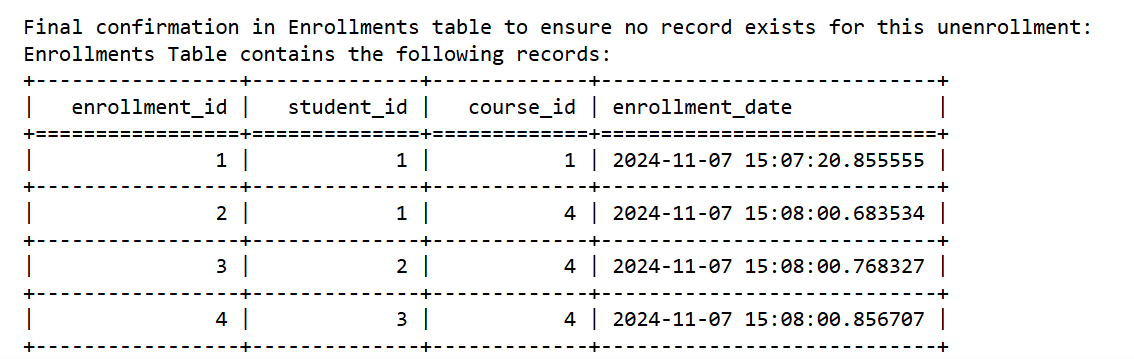
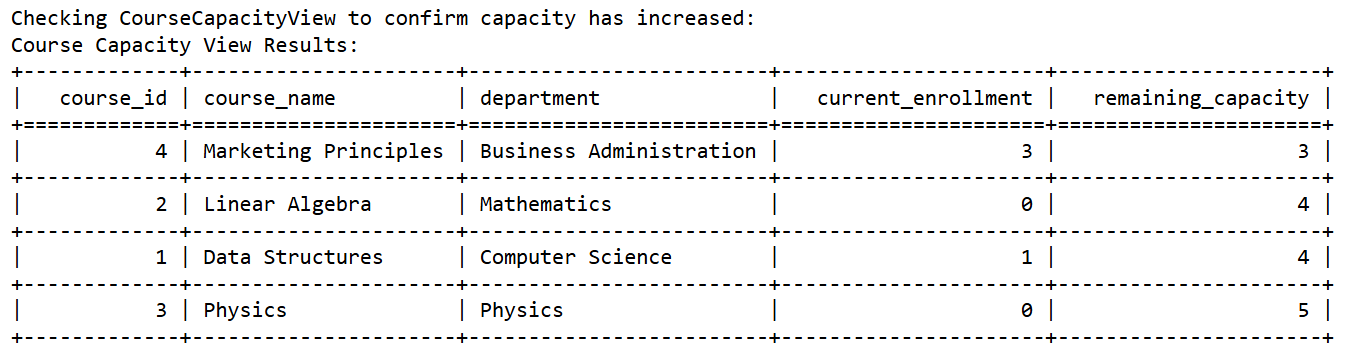
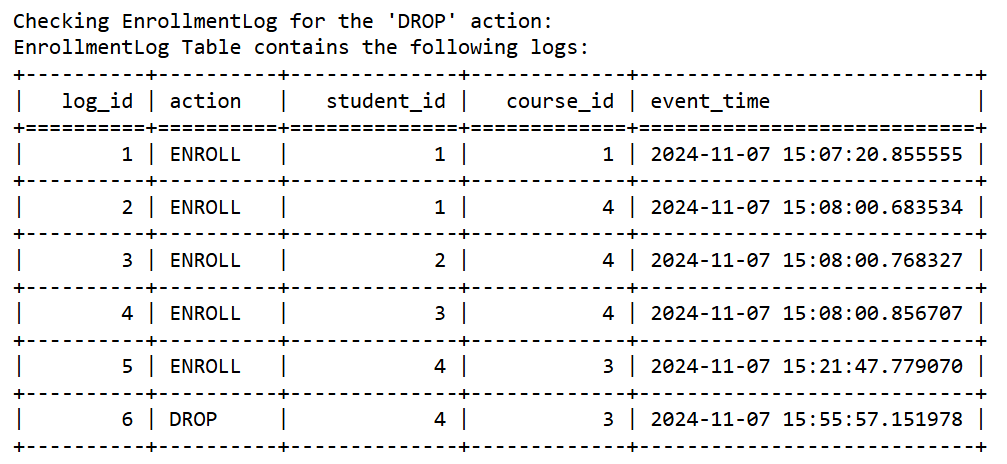
1. Enrollment fails, and a message "Enrollment failed: Student ID 4 is already enrolled in Course ID 3" is displayed. 
2. No duplicate record is added to the Enrollments table. There is only 1 enrollment for student\_id = 4 in course\_id = 3.
3. The EnrollmentLog table has no added action compared to previous test case 2.

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 4: Successful Unenrollment** **Objective**: Verify that a student can successfully unenroll from a course and the system updates capacity accordingly.  · **Preconditions**:  1. Enrollments table exists of a student with student\_id = 4 in course\_id = 3.  2. CourseCapacityView shows current capacity data.  · **Steps**:  1. Remove the enrollment for student\_id = 4 in course\_id = 3 using test\_trigger\_delete().  2. Query the CourseCapacityView to ensure the remaining capacity increased by 1.  3. Verify that EnrollmentLog records the DROP action.  · **Expected Results**:  1. Enrollments table now does not have the record for student\_id = 4 and course\_id = 3.  2. CourseCapacityView shows the remaining\_capacity increased by 1.  3. The EnrollmentLog table logs the unenrollment event with the action DROP. |

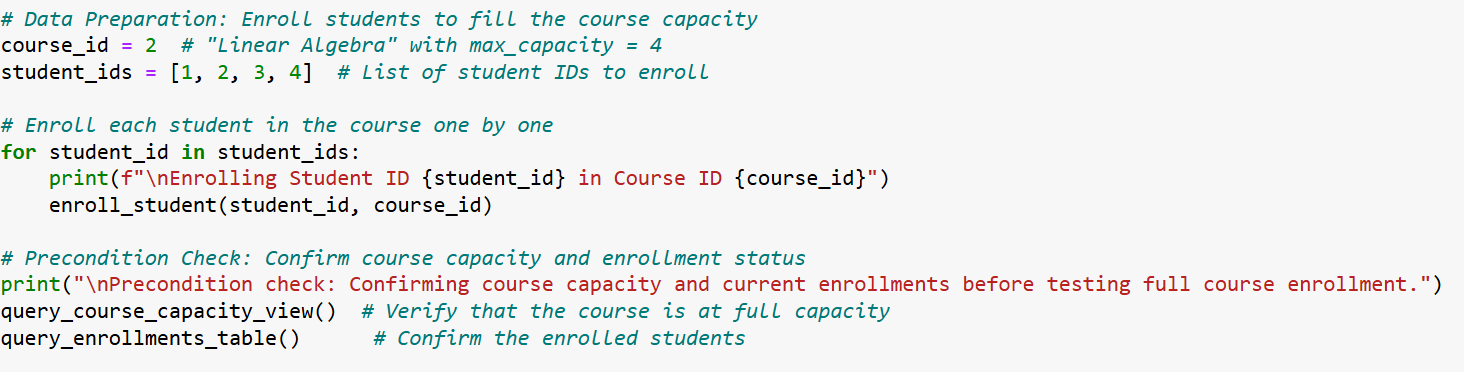
**Code**: Precondition: Enrollments table exists of a student with student\_id = 4 in course\_id = 3.

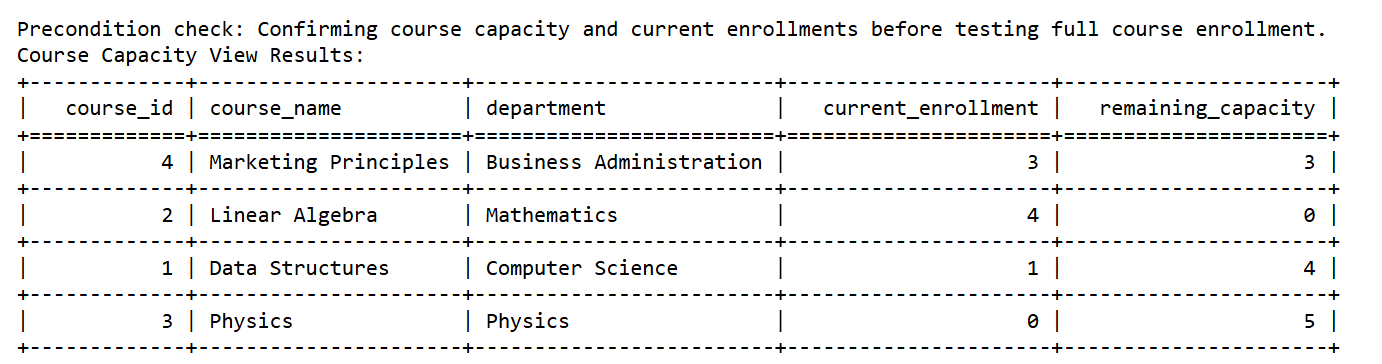
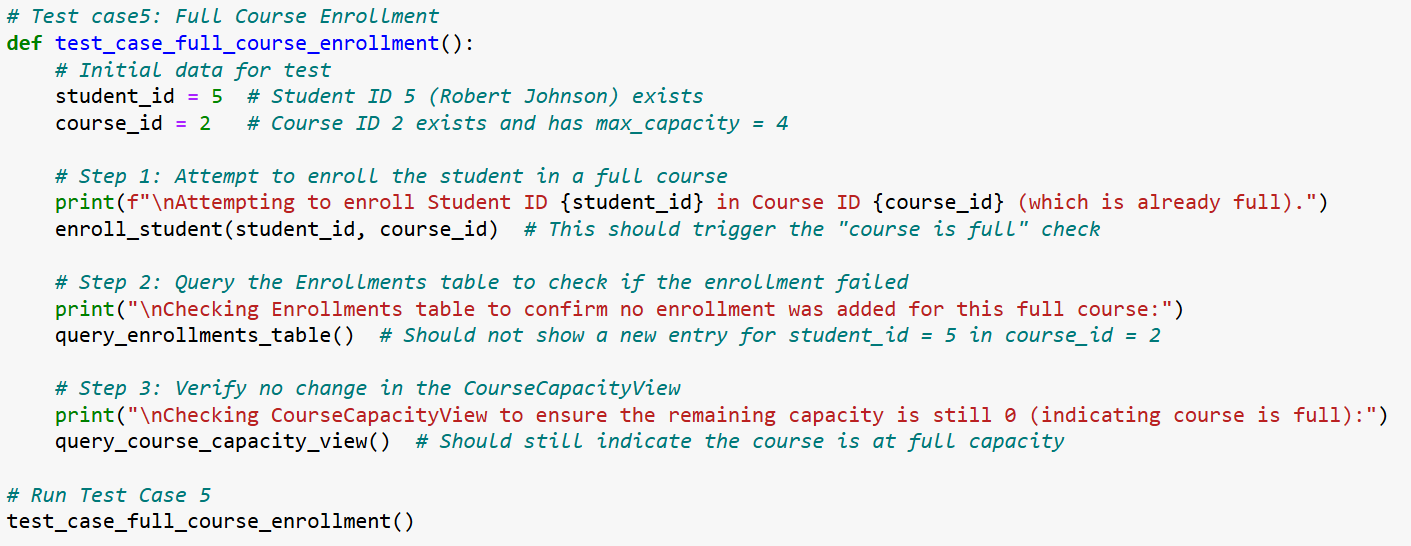
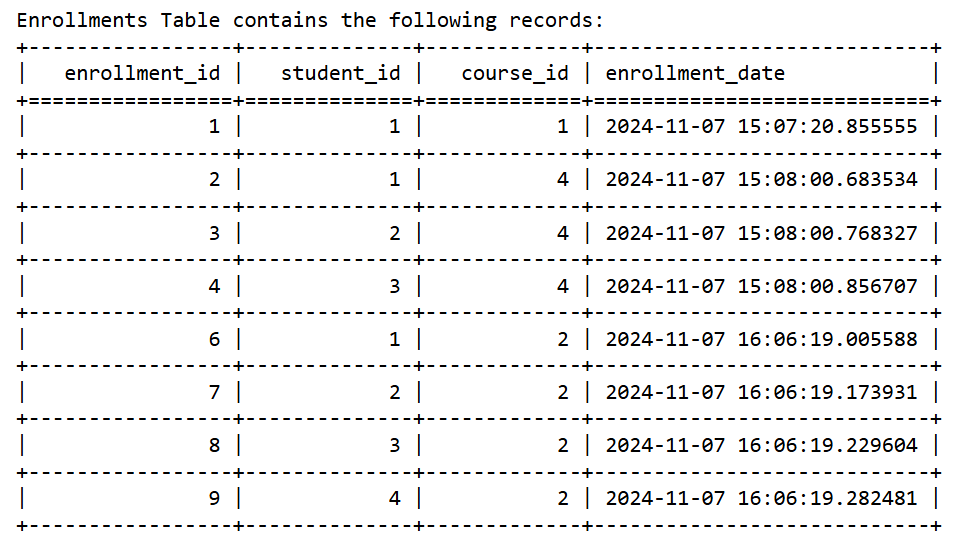
**Output:**

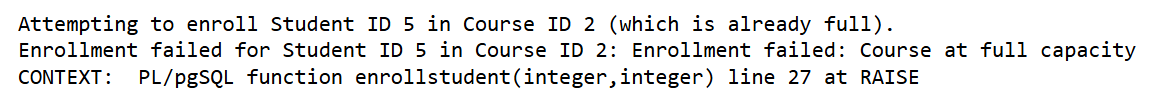
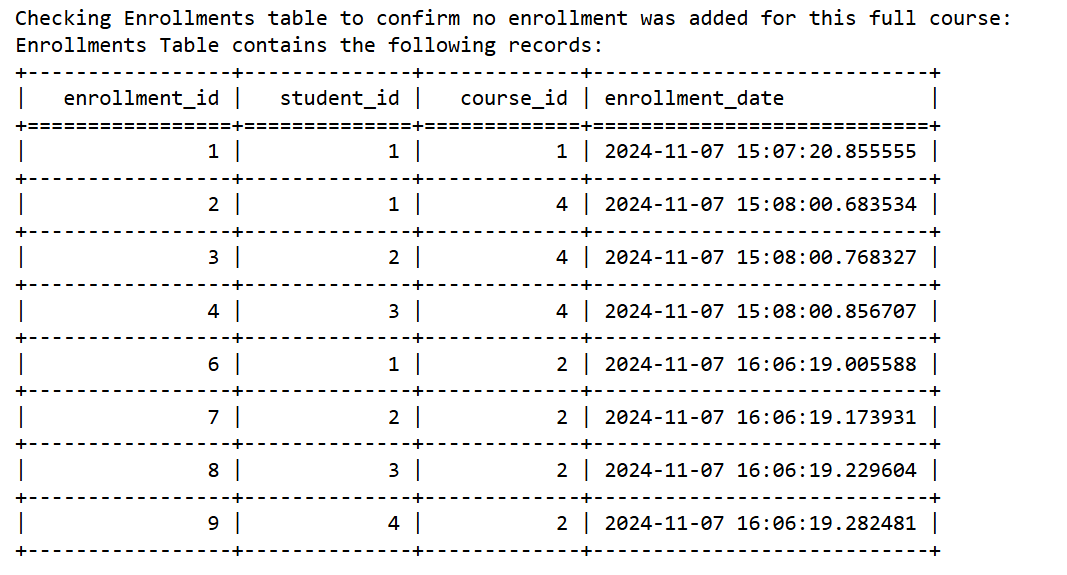
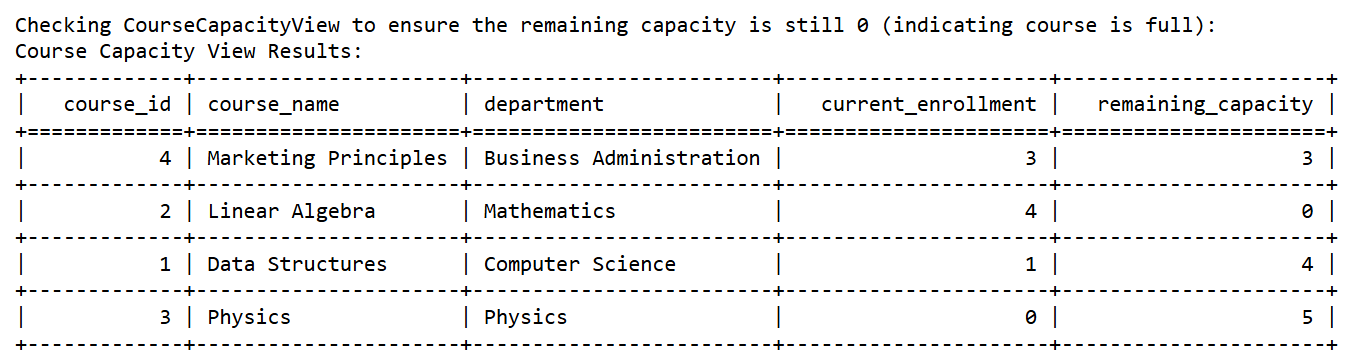


1. Enrollments table now does not have the record for student\_id = 4 and course\_id = 3 (enrollment\_id = 5).
2. CourseCapacityView shows the remaining\_capacity increased by 1. The current\_enrollment of Physics course is back to 0 and remaining\_capacity = 5
3. The EnrollmentLog table logs the unenrollment event with the action DROP.

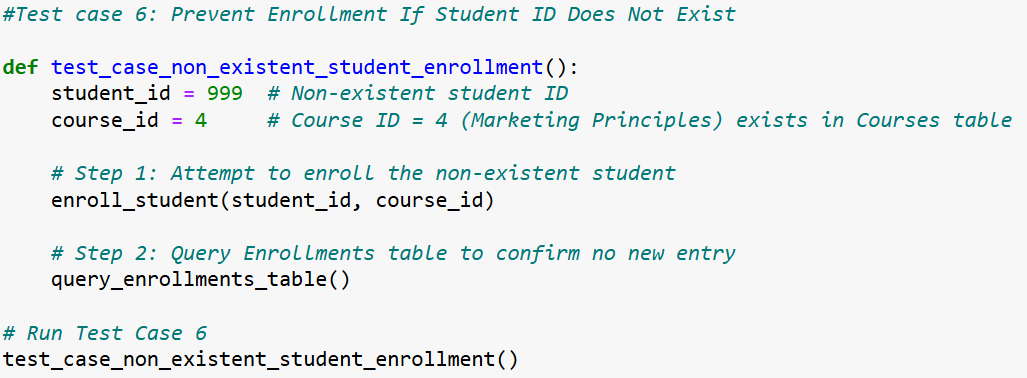
|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 5: Full Course Enrollment** **Objective**: Verify that the system prevents a student from enrolling in a course that has reached its maximum capacity.  · **Preconditions**:  1. Students table has a student with student\_id = 5 (Robert Johnson).  2. Courses table has a course with course\_id = 2 and max\_capacity = 4.  3. Currently, 4 students are enrolled in course 2.  · **Steps**:  1. Execute the EnrollStudent(5, 2) stored procedure from the Python script.  2. Query the Enrollments table to check if the enrollment failed.  3. Verify no change in the CourseCapacityView.  · **Expected Results**:  1. Enrollment fails, and a message "Enrollment failed: Course ID 2 is full." is displayed.  2. Enrollments table remains unchanged for this student and course.  3. CourseCapacityView shows no change in the remaining capacity. |

**Code**: Data preparation: Enroll students with ID 1, 2, 3, 4 into course\_id = 4 so that the course\_id = 4 (Linear Algebra) reaches the max\_capacity = 4.**Precondition**: Currently, 4 students are enrolled in course 2 (Linear Algebra), the course is at full capacity.

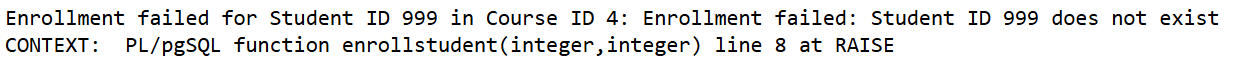
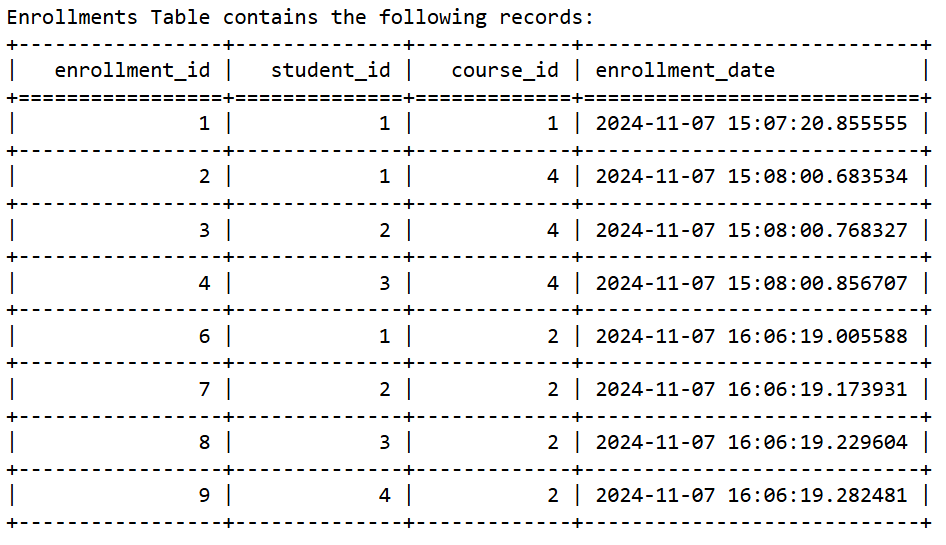
The Enrollments tables now has 9 enrollments:**Output**:

1. Enrollment fails, and a message "Enrollment failed for Student ID 5 in Course ID 2: Course at full capacity." is displayed. 
2. Enrollments table remains unchanged for this student and course (still having 9 enrollments)
3. CourseCapacityView shows no change in the remaining capacity, still 0 indicating the course is full. ****

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 6: Prevent Enrollment If Student ID Does Not Exist** **Objective**: Verify that an enrollment attempt fails if the student does not exist in the **Students** table.  · **Preconditions**:  1. course\_id = 4 exists in the Courses table (Marketing Principles).  2. There is no record in the Students table for student\_id = 999.  · **Steps**:  1. Execute the EnrollStudent(999, 4) to attempt enrolling a non-existent student in course\_id = 4.  2. Catch any exception and check the error message.  3. Query the Enrollments table to confirm no new entry in the **Enrollments** table for student\_id = 999 and course\_id = 4.  · **Expected Results**:  1. Enrollment fails, and a message "Enrollment failed: Student ID 999 does not exist." is displayed.  2. Enrollments table remains unchanged, with no entries for student\_id = 999 and course\_id = 4. |

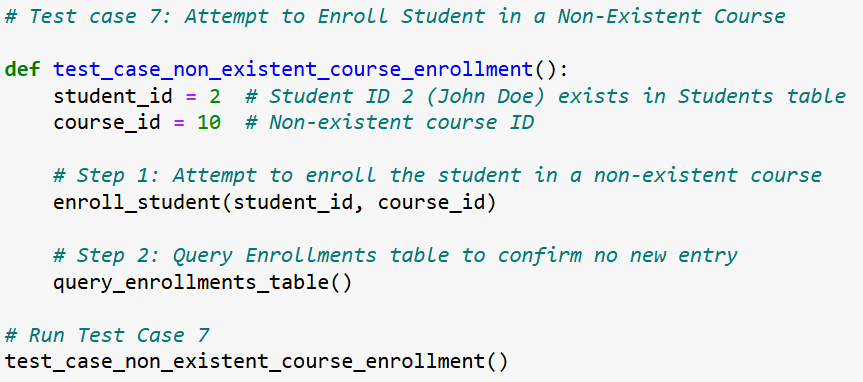
**Code:**

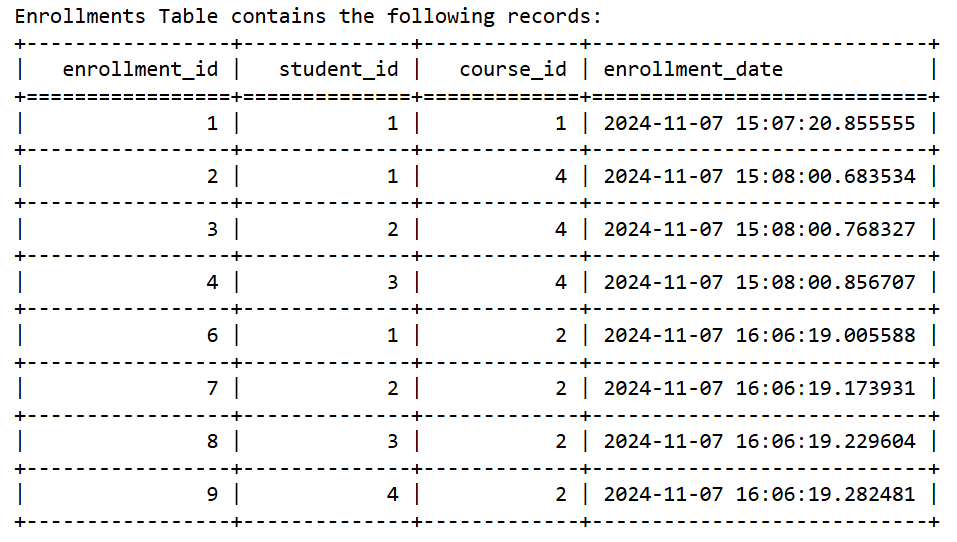
**Output**:

1. Enrollment fails, and a message "Enrollment failed: Student ID 999 does not exist." is displayed. 
2. Enrollments table remains unchanged, with no entries for student\_id = 999 and course\_id = 4.

|  |
| --- |
| **Test Plan: University Enrollment System** **Test Case 7: Attempt to Enroll Student in a Non-Existent Course** **Objective**: Ensure that enrollment fails when a student tries to enroll in a course that does not exist.  · **Preconditions**:  1. The Students table contains a record with student\_id = 2 (John Doe).  2. The Courses table does not contain a course with course\_id = 10.  · **Steps**:  1. Execute the EnrollStudent(2,10) to attempt enrolling student\_id = 2 in course\_id = 10.  2. Catch any exception and check the error message.  3. Query the Enrollments table to confirm no new records in the **Enrollments** table for student\_id = 2 and course\_id = 10.  · **Expected Results**:  1. Enrollment fails, and a message "Enrollment failed: Course ID 10 does not exist." is displayed.  2. Enrollments table remains unchanged, with no entries for student\_id = 2 and course\_id = 10. |

**Code**:

**Output**:

1. Enrollment fails, and a message "Enrollment failed: Course ID 10 does not exist." is displayed. 
2. Enrollments table remains unchanged, with no entries for student\_id = 2 and course\_id = 10.****