IE2042
Database Management System for Security

2nd Year, 1st Semester
Assignment Submission

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Cyber Security

09/05/2024

Table of Contents

Assumptions

- The name attribute of the student entity is a composite attribute.
- The Phone_no attribute of the academic entity is a multivalued attribute.
- Students take courses and base on their performance they win awards.
- Students can win many awards and an award can have many students.
- Awards have names.
- The undergraduate students have student types, like Freshmen, Sophomore, Junior and Senior.

Assumptions for Trigger 1

- The course_audit_trigger trigger is executed automatically after each INSERT, UPDATE, or DELETE operation on the Course table.
- The trigger has access to several implicit variables provided by the database management system (INSERT, UPDATE, DELETE,).
- assumed that the user executing the DML operation has the necessary permissions to perform the operation and that the trigger executes.

Assumptions for Trigger 2

- The trigger assumes that it handles single row DELETE operations on the Program table.
- The trigger uses the RAISERROR statement to raise a custom error message when deletion is prevented due to associated courses.

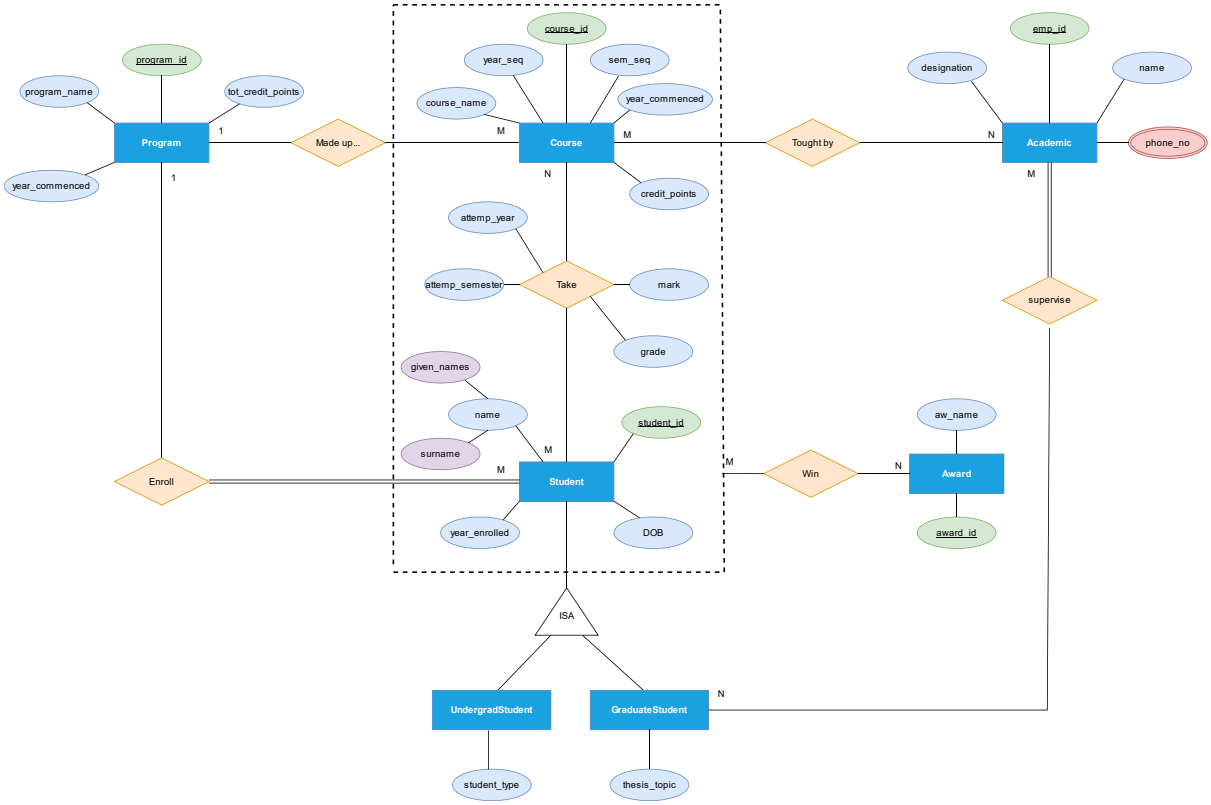
Assumptions for view 1

- Assumed that view 1 is used by academic staff to see academic performance of student.

Assumptions for view 2

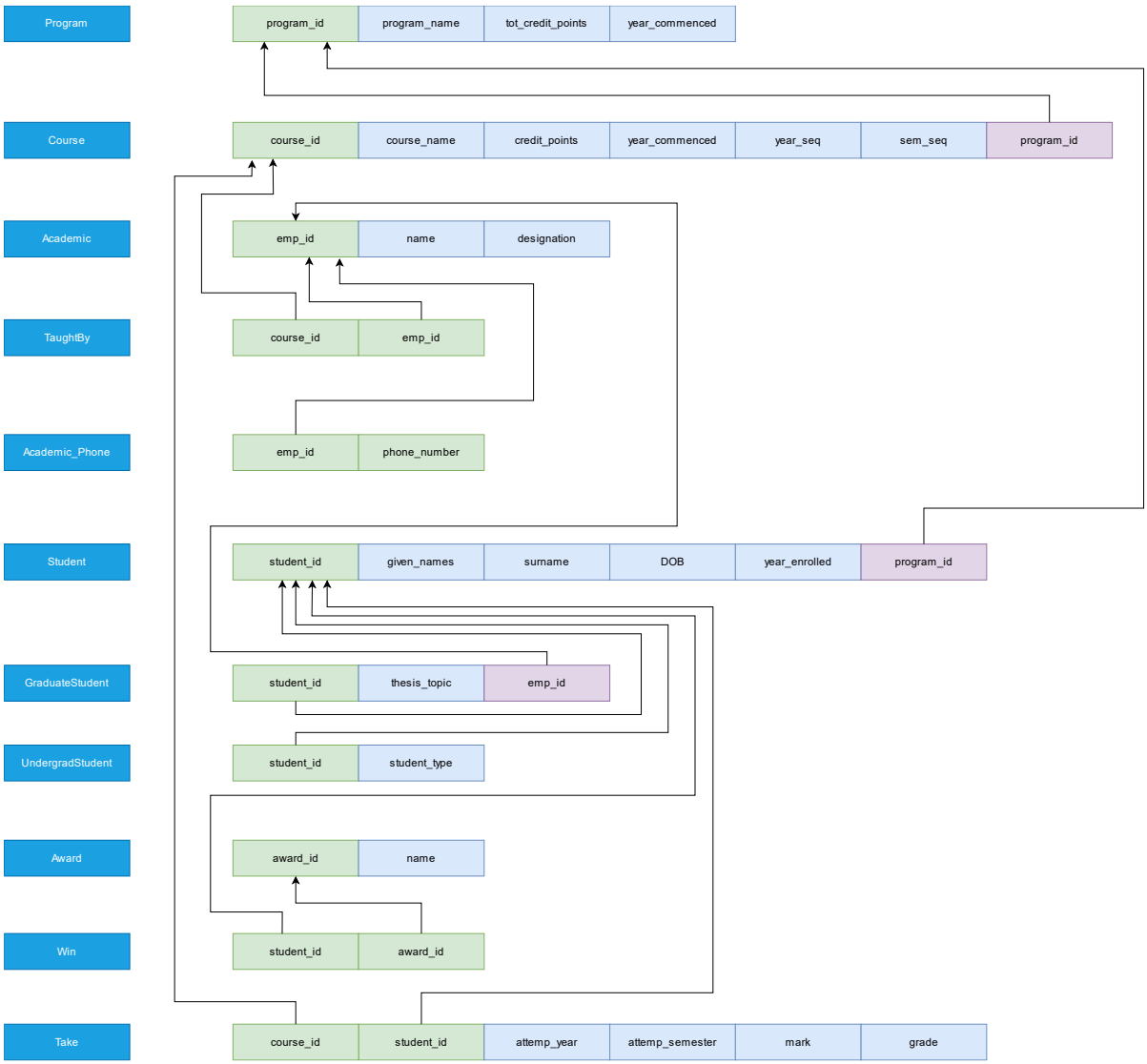
- Assumed that view 2 is used by students to see the courses that they have enrolled in.

ER Diagram


course_id
emp_id
program_id
year_seq
sem_seq
designation
name
program_name
tot_credit_points
course_name
year_commenced
Program
1
Made up...
M
Course
M
Tought by
N
Academic
phone_no
year_commenced
1
N
M
credit_points
attemp_year
attemp_semester
Take
mark
supervise
grade
given_names
name
student_id
aw_name
surname
M
M
M
N
Enroll
Student
Win
Award
year_enrolled
DOB
award_id
ISA
UndergradStudent
GraduateStudent
N
student_type
thesis_topic

Logical Model

Logical Model is already in 3NF.



SQL Queries

Table Creation

```sql
-- Program table
CREATE TABLE Program (
    program_id INT NOT NULL,
    program_name VARCHAR(255),
    total_credit_points INT,
    year_commenced INT,
    CONSTRAINT PK_Program PRIMARY KEY (program_id)
);


-- Course table
CREATE TABLE Course (
    course_id INT NOT NULL,
    program_id INT NOT NULL,
    course_name VARCHAR(255),
    credit_points INT,
    year_commenced INT,
    year_sequence INT,
    semester_sequence INT,
    CONSTRAINT PK_Course PRIMARY KEY (course_id),
    CONSTRAINT FK_Course_Program FOREIGN KEY (program_id)
REFERENCES Program(program_id)
);


-- Academic table
CREATE TABLE Academic (
    emp_id INT NOT NULL,
    name VARCHAR(255),
    designation VARCHAR(255),
    CONSTRAINT PK_Academic PRIMARY KEY (emp_id)
);
```

```sql
CREATE TABLE Academic_Phone (
    emp_id INT,
    phone_number VARCHAR(20),
    PRIMARY KEY (emp_id, phone_number),
    FOREIGN KEY (emp_id) REFERENCES Academic(emp_id)
);


-- Student table
CREATE TABLE Student (
    student_id INT NOT NULL,
    given_names VARCHAR(255),
    surname VARCHAR(255),
    date_of_birth DATE,
    year_enrolled INT,
    CONSTRAINT PK_Student PRIMARY KEY (student_id)
);


-- GraduateStudent and UndergradStudent tables (for ISA
relationship)
CREATE TABLE GraduateStudent (
    student_id INT NOT NULL,
    thesis_topic VARCHAR(255),
    emp_id    INT  NOT NULL,
    CONSTRAINT PK_GraduateStudent PRIMARY KEY
(student_id),
    CONSTRAINT FK_GraduateStudent_Student FOREIGN KEY
(student_id) REFERENCES Student(student_id),
    CONSTRAINT FK_GraduateStudent_Studen FOREIGN KEY
(emp_id) REFERENCES Academic(emp_id)
);


CREATE TABLE UndergradStudent (
    student_id INT NOT NULL,
    student_type VARCHAR(50),
    CONSTRAINT PK_UndergradStudent PRIMARY KEY
(student_id),
    CONSTRAINT FK_UndergradStudent_Student FOREIGN KEY
(student_id) REFERENCES Student(student_id)
);
```

```sql
-- Award table
CREATE TABLE Award (
    award_id INT NOT NULL,
    name VARCHAR(255),
    CONSTRAINT PK_Award PRIMARY KEY (award_id)
);



-- Take table
CREATE TABLE Take (
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    attempted_year INT,
    attempted_semester INT,
    mark DECIMAL(5,2),
    grade CHAR(1),
    CONSTRAINT PK_Take PRIMARY KEY (student_id,
course_id),
    CONSTRAINT FK_Take_Student FOREIGN KEY (student_id)
REFERENCES Student(student_id),
    CONSTRAINT FK_Take_Course FOREIGN KEY (course_id)
REFERENCES Course(course_id)
);



-- Win table
CREATE TABLE Win (
    student_id INT NOT NULL,
    award_id INT NOT NULL,
    CONSTRAINT PK_Win PRIMARY KEY (student_id, award_id),
    CONSTRAINT FK_Win_Student FOREIGN KEY (student_id)
REFERENCES Student(student_id),
    CONSTRAINT FK_Win_Award FOREIGN KEY (award_id)
REFERENCES Award(award_id)
);
```

```sql
-- TaughtBy table
CREATE TABLE TaughtBy (
    course_id INT NOT NULL,
    emp_id INT NOT NULL,
    CONSTRAINT PK_TaughtBy PRIMARY KEY (course_id,
emp_id),
    CONSTRAINT FK_TaughtBy_Course FOREIGN KEY (course_id)
REFERENCES Course(course_id),
    CONSTRAINT FK_TaughtBy_Academic FOREIGN KEY (emp_id)
REFERENCES Academic(emp_id)
);
```

Sample Data

```sql
-- Sample data for Program table
INSERT INTO Program (program_id, program_name,
total_credit_points, year_commenced)
VALUES
    (111, 'BSc in Computer Science          ', 120, 2018),
    (222, 'BA in English Literature         ', 90 , 2017),
    (333, 'Master of Business Administration', 60 , 2019),
    (444, 'MSc in Data Science              ', 60 , 2020),
    (555, 'BEng in Mechanical Engineering   ', 150, 2016),
    (666, 'MBBS                             ', 180, 2015);


-- Sample data for Course table
INSERT INTO Course (course_id,program_id, course_name,
credit_points, year_commenced, year_sequence,
semester_sequence)
VALUES
    (101,111,'Introduction to Programming', 4, 2018, 1, 1),
    (102,222,'Database Management Systems', 4, 2018, 2, 1),
    (103,222,'English Composition        ', 3, 2017, 1, 1),
    (104,333,'Marketing Management       ', 3, 2019, 1, 1),
    (105,111,'Data Mining                ', 4, 2020, 1, 2),
    (106,444,'Web Development            ', 4, 2018, 3, 1),
    (107,555,'Expert morter mechanic     ', 4, 2018, 2, 1);
```

```sql
-- Sample data for Student table
INSERT INTO Student (student_id, given_names, surname,
date_of_birth, year_enrolled)
VALUES
    (1001, 'Kamal'   ,'Silva'  , '1998-03-15', 2018),
    (1002, 'Nimal'   ,'Silva'  , '1997-05-20', 2019),
    (1003, 'Sunil'   ,'Perera' , '2001-08-10', 2020),
    (1004, 'Chaminda','Vaas'   , '1996-12-05', 2017),
    (1005, 'Dayas'   ,'Lakshan', '2000-07-25', 2018),
    (1006, 'Ruvan'   ,'Gomes'  , '1997-10-18', 2019);

-- Sample data for Academic table
INSERT INTO Academic (emp_id, name, designation)
VALUES
    (11, 'Dr. Chamal'   , 'Professor'          ),
    (22, 'Prof. Isira'  , 'Associate Professor'),
    (33, 'Dr. Kavindu'  , 'Assistant Professor'),
    (44, 'Dr. Piyal'    , 'Lecturer'           ),
    (55, 'Dr. Sahan'    , 'Lecturer'           ),
    (66, 'Dr. Banuka'   , 'Lecturer'           );


-- Sample data for GraduateStudent table
INSERT INTO GraduateStudent (student_id, thesis_topic,emp_id)
VALUES
    (1003, 'Analysis of Social Media Data',22),
    (1002, 'Analysis of database'         ,55),
    (1005, 'E-commerce Trends'            ,11);

-- Sample data for UndergradStudent table
INSERT INTO UndergradStudent (student_id, student_type)
VALUES
    (1001, 'Freshman' ),
    (1002, 'Sophomore'),
    (1004, 'Junior'   ),
    (1006, 'Senior'   );

-- Sample data for Award table
INSERT INTO Award (award_id, name)
VALUES
    (2001, 'Deans List'               ),
    (2002, 'Outstanding Student Award'),
    (2003, 'Research Excellence Award');
```

```sql
-- Sample data for Take table
INSERT INTO Take (student_id, course_id, attempted_year,
attempted_semester, mark, grade)
VALUES
    (1001, 101, 2018, 1, 85.5, 'A'),
    (1002, 102, 2019, 1, 78.0, 'B'),
    (1003, 105, 2020, 1, 92.5, 'A'),
    (1004, 103, 2017, 1, 88.0, 'A'),
    (1005, 104, 2018, 1, 75.5, 'B'),
    (1006, 106, 2019, 1, 81.0, 'B');

-- Sample data for Win table
INSERT INTO Win (student_id, award_id)
VALUES
    (1001, 2001),
    (1002, 2002),
    (1003, 2003),
    (1004, 2001),
    (1005, 2002),
    (1006, 2003);

-- Sample data for TaughtBy table
INSERT INTO TaughtBy (course_id, emp_id)
VALUES
    (101, 11),
    (102, 22),
    (103, 33),
    (104, 11),
    (105, 22),
    (106, 33);
```

Triggers

```sql
/*
1. Audit Trigger:

This trigger can be used to track changes made to Course
tables in the database. This trigger captures information
about who made the change, what action was performed
(INSERT, UPDATE, DELETE), and when the change occurred.
*/
```

```sql
--Course_Audit table to add the data that genarated by the
Course Audit Trigger.

CREATE TABLE Course_Audit (
    audit_id INT PRIMARY KEY,
    course_id INT,
    action VARCHAR(10),
    modified_by VARCHAR(255),
    modified_at DATETIME
);


--Trigger for INSERT, UPDATE, DELETE operations on Course
table

CREATE TRIGGER Course_Audit_Trigger
ON Course
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Action VARCHAR(10);

    IF EXISTS (SELECT * FROM inserted)
    BEGIN
        IF EXISTS (SELECT * FROM deleted)
        BEGIN
            -- Update operation
            SET @Action = 'UPDATE';
        END
        ELSE
        BEGIN
            -- Insert operation
            SET @Action = 'INSERT';
        END
    END
    ELSE
    BEGIN
            -- Delete operation
        SET @Action = 'DELETE';
    END
```

```
    -- Insert into Course_Audit table
    INSERT INTO Course_Audit (audit_id, course_id, action,
modified_by, modified_at)
    SELECT
        ISNULL((SELECT MAX(audit_id) FROM Course_Audit),
0) + ROW_NUMBER() OVER (ORDER BY (SELECT NULL)),
        COALESCE(i.course_id, d.course_id),
        @Action,
        SUSER_SNAME(),
        GETDATE()
    FROM inserted i
    FULL OUTER JOIN deleted d ON i.course_id =
d.course_id;
END;


/*
2. Prevent Delete Trigger:

This trigger prevent the deletion of records from the
Program table if there are courses associated with that
program. Here's how you can create such a trigger:
*/

--Trigger to prevent deletion of Program records if
associated courses exist.

CREATE TRIGGER Prevent_Delete_Program
ON Program
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if there are any associated courses
    IF EXISTS (SELECT 1 FROM Course WHERE program_id IN
(SELECT program_id FROM deleted))
    BEGIN
        -- If there are associated courses, raise an error
and rollback the transaction
```

```
        RAISERROR ('Cannot delete program with associated
courses.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- If there are no associated courses, proceed with
the deletion
    DELETE FROM Program WHERE program_id IN (SELECT
program_id FROM deleted);
END;
```

Views

```
/*
user 1. Academic Staff:

View 1: Student Performance View:**
This view displays the performance of students in courses
taught by academic staff, including the students name,
course name, mark,
and grade.
*/
```

```
CREATE VIEW Student_Performance_View AS
SELECT
    s.given_names + ' ' + s.surname AS student_name,
    c.course_name,
    t.mark,
    t.grade
FROM
    Student s
JOIN
```

```
    Take t ON s.student_id = t.student_id
JOIN
    Course c ON t.course_id = c.course_id
JOIN
    TaughtBy tb ON c.course_id = tb.course_id
JOIN
    Academic a ON tb.emp_id = a.emp_id;


--Call the view

SELECT * FROM Student_Performance_View;


/*
user 1. Student:

View 1: Enrolled Courses View:**
create a view provides students with information about the
courses they are currently enrolled in, including the
course name, credit points,
and the semester.
*/

CREATE VIEW Enrolled_Courses_View AS
SELECT
    s.student_id,
    s.given_names + ' ' + s.surname AS student_name,
    c.course_name,
    c.credit_points,
    c.semester_sequence
FROM
    Student s
JOIN
    Take t ON s.student_id = t.student_id
JOIN
    Course c ON t.course_id = c.course_id;

--call the view
```

```sql
SELECT * FROM Enrolled_Courses_View;
```

Indexes

```sql
-- Index to optimize query 1: Retrieve the names of all
academics who have the designation "Lecturer"
CREATE INDEX IX_Academic_Designation ON Academic
(designation);

-- Index to optimize query 2: Retrieve the list of names
of students studying a given course "X"
CREATE INDEX IX_Take_CourseID ON Take (course_id);
```

Functions

```sql
--Funtions

--SQL function to retrieve the names of all academics who
have the designation "Lecturer"

CREATE FUNCTION GetLecturers1()
RETURNS TABLE
AS
RETURN
(
    SELECT name
    FROM Academic
    WHERE designation = 'Lecturer'
);

--call the Function

SELECT * FROM GeteLcturers1();

--SQL function to retrieve the list of names of students
studying a given course

CREATE FUNCTION GetStudentsByCourse1
(
```

```
    @CourseName VARCHAR(255)
)
RETURNS TABLE
AS
RETURN
(
    SELECT s.given_names + ' ' + s.surname AS student_name
    FROM Student s
    INNER JOIN Take t ON s.student_id = t.student_id
    INNER JOIN Course c ON t.course_id = c.course_id
    WHERE c.course_name = @CourseName
);


--call the Function_____

SELECT * FROM GetStudentsByCourse1('Introduction to Programming');
```

Procedures

```
--call the Procedure

EXEC GetLecturers;

--Retrieve the list of names of students studying a given
course:

CREATE PROCEDURE GetStudentsByCourse
    @CourseName VARCHAR(255)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT s.given_names + ' ' + s.surname AS student_name
    FROM Student s
    INNER JOIN Take t ON s.student_id = t.student_id
    INNER JOIN Course c ON t.course_id = c.course_id
    WHERE c.course_name = @CourseName;
END;


--call the Procedure
```

```
EXEC GetStudentsByCourse @CourseName = 'Marketing Management';
```

Screenshots of the SQL Queries

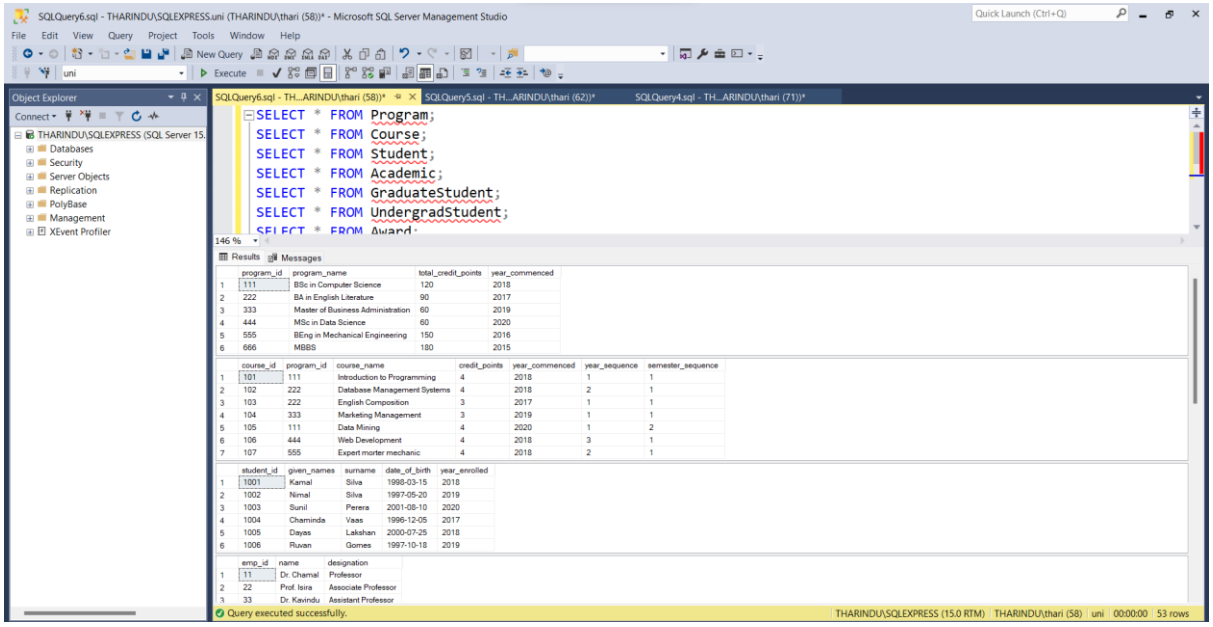


Figure 1: Table Creation 01

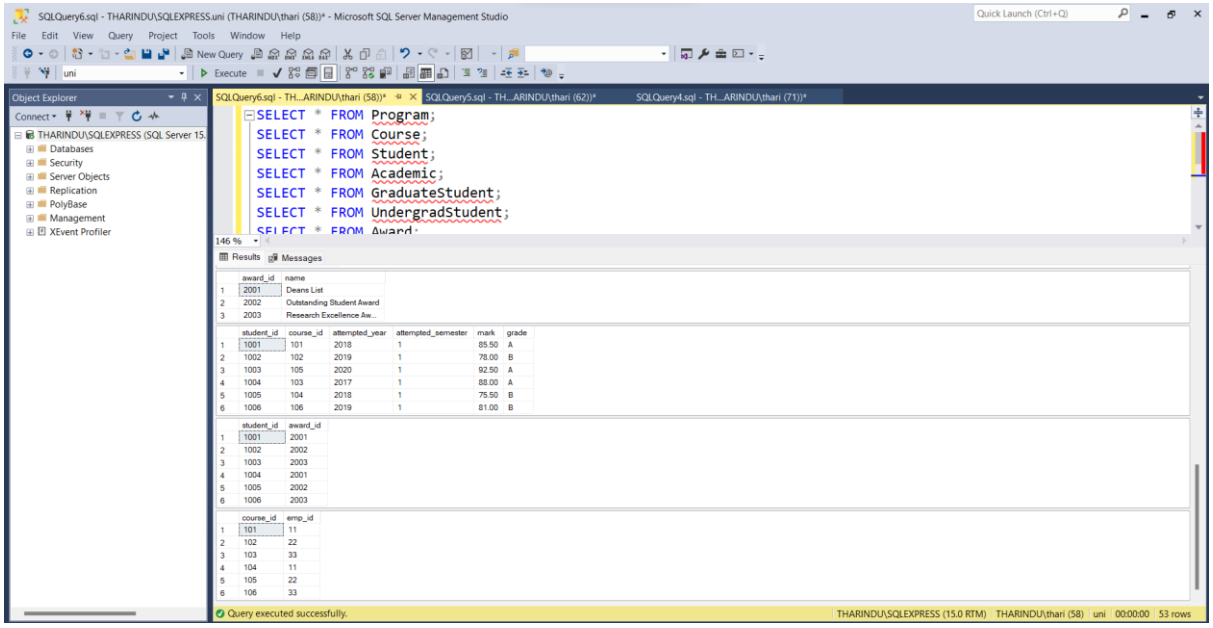


Figure 2: Table Creation 02

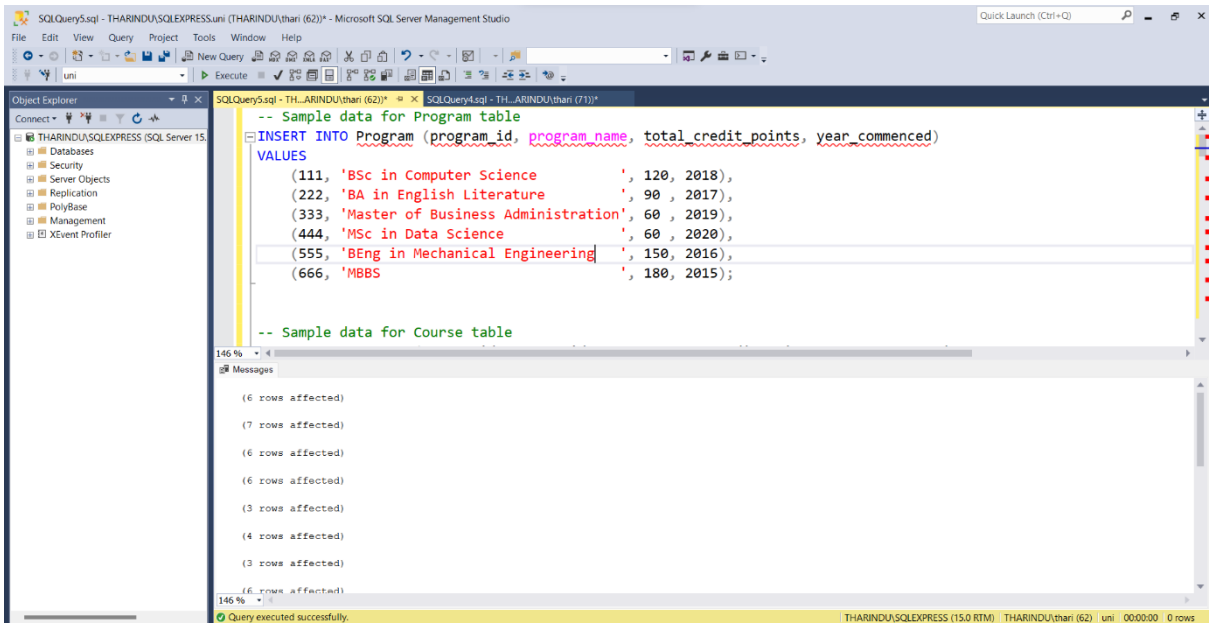


Figure 3: Sample Data Insertion

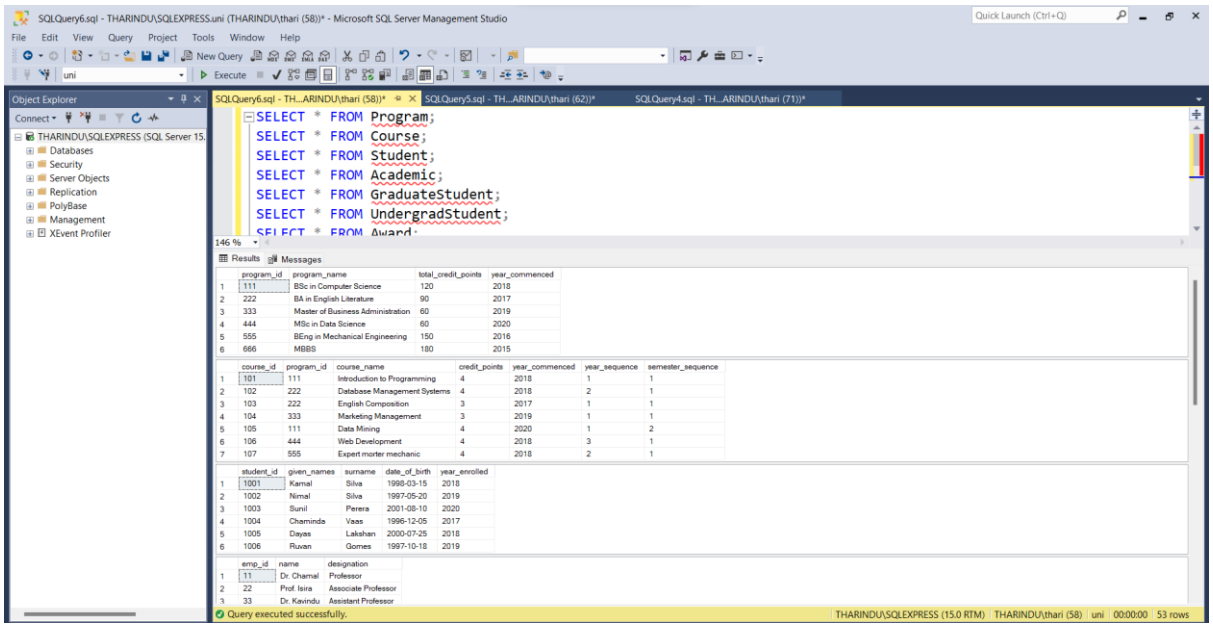


Figure 4:Table Structure with Sample Data 01

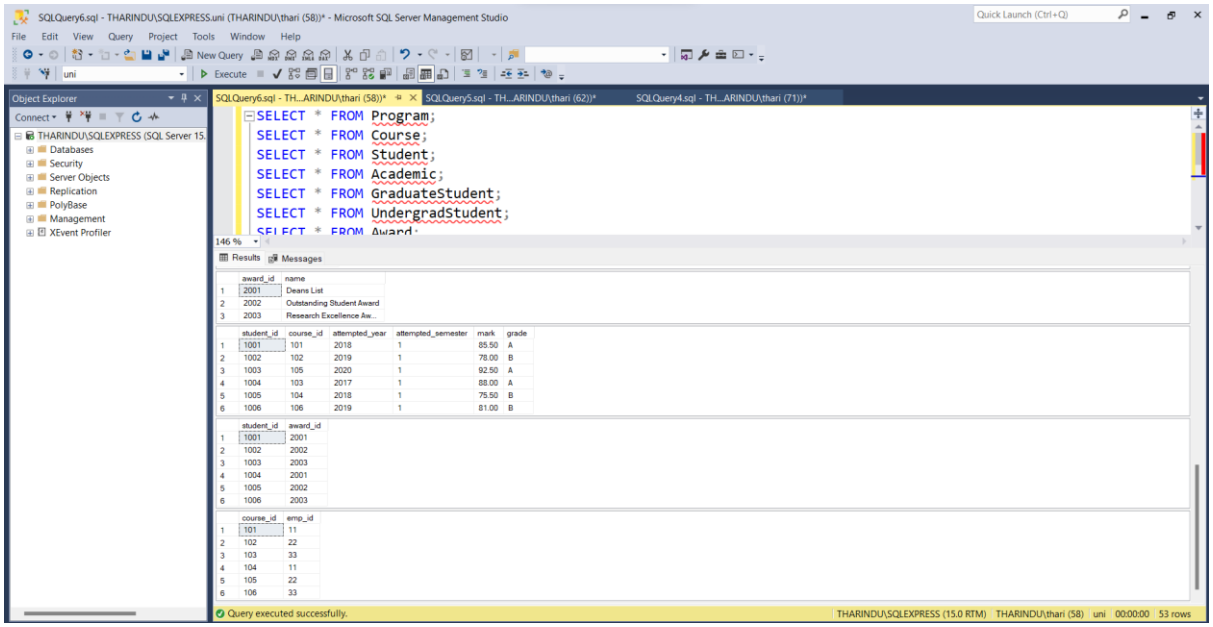


Figure 5:Tabe Structure with Sample Data 02

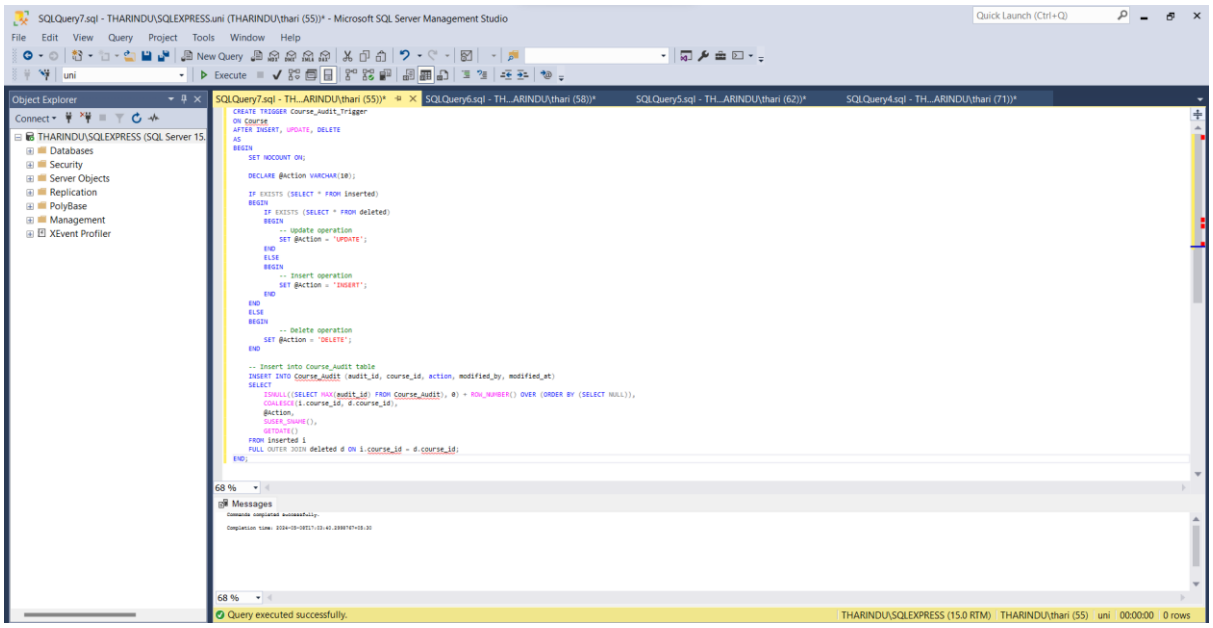


Figure 6: Trigger 01

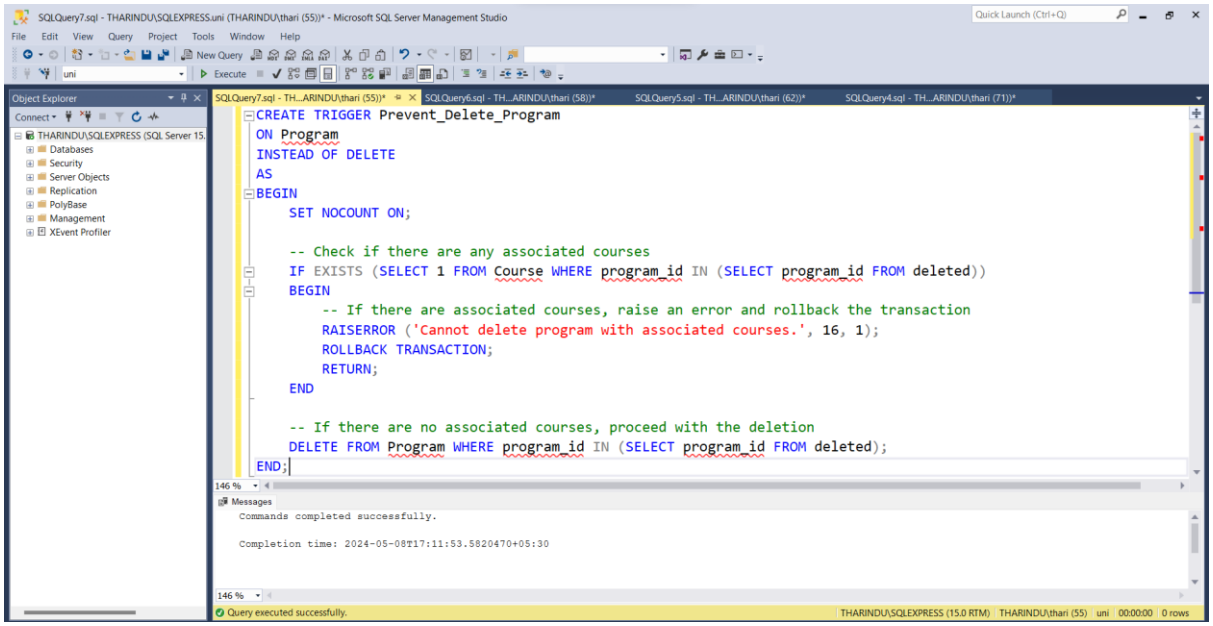


Figure 7: Trigger 02

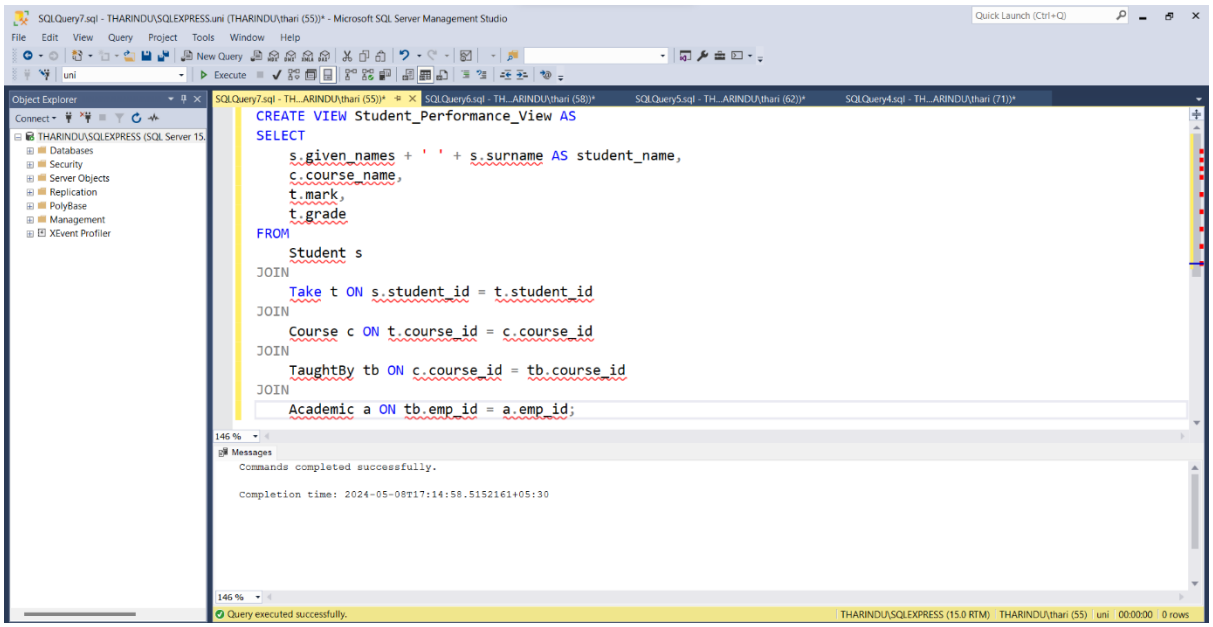


Figure 8: Views 01

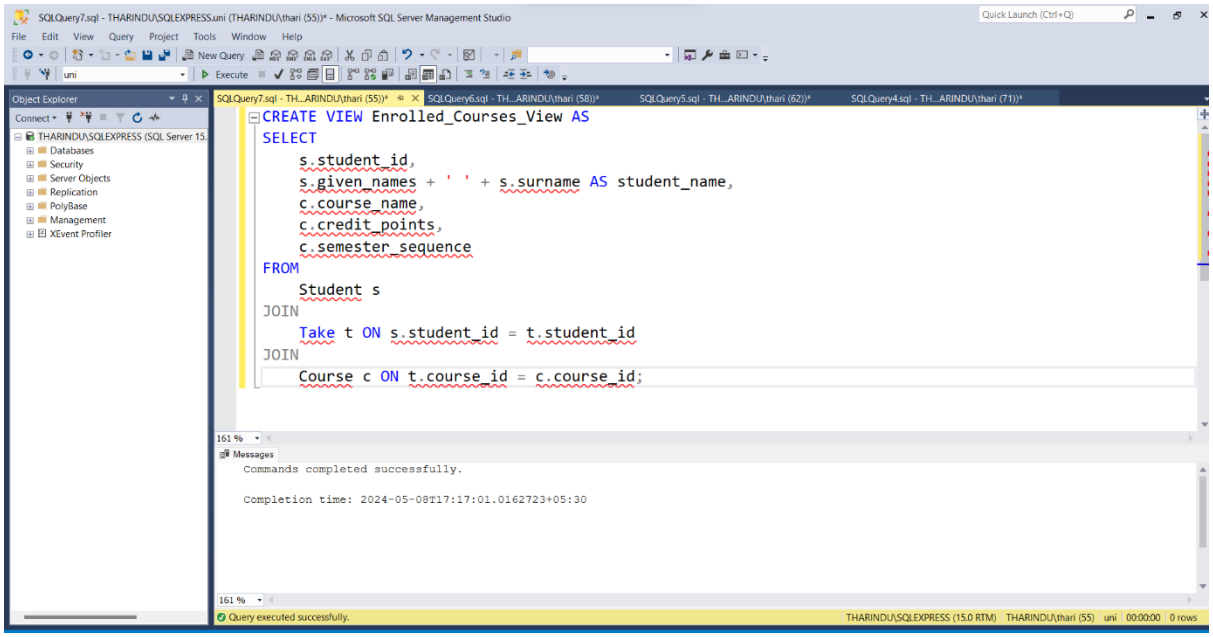


Figure 9: Views 02

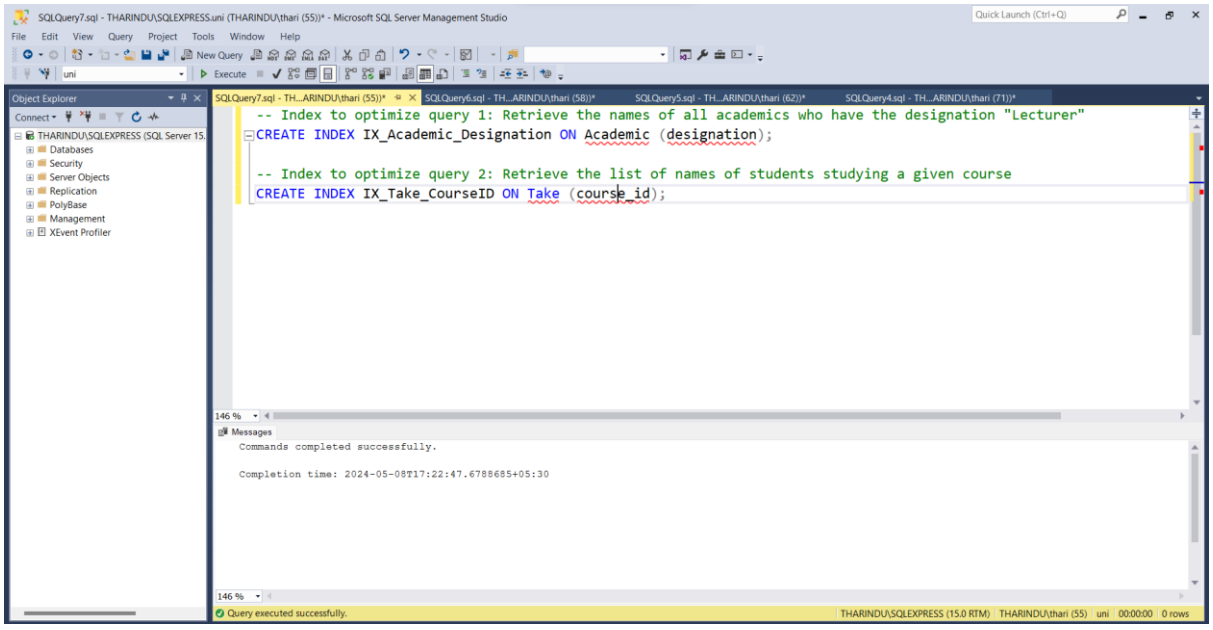


Figure 10: Indexes

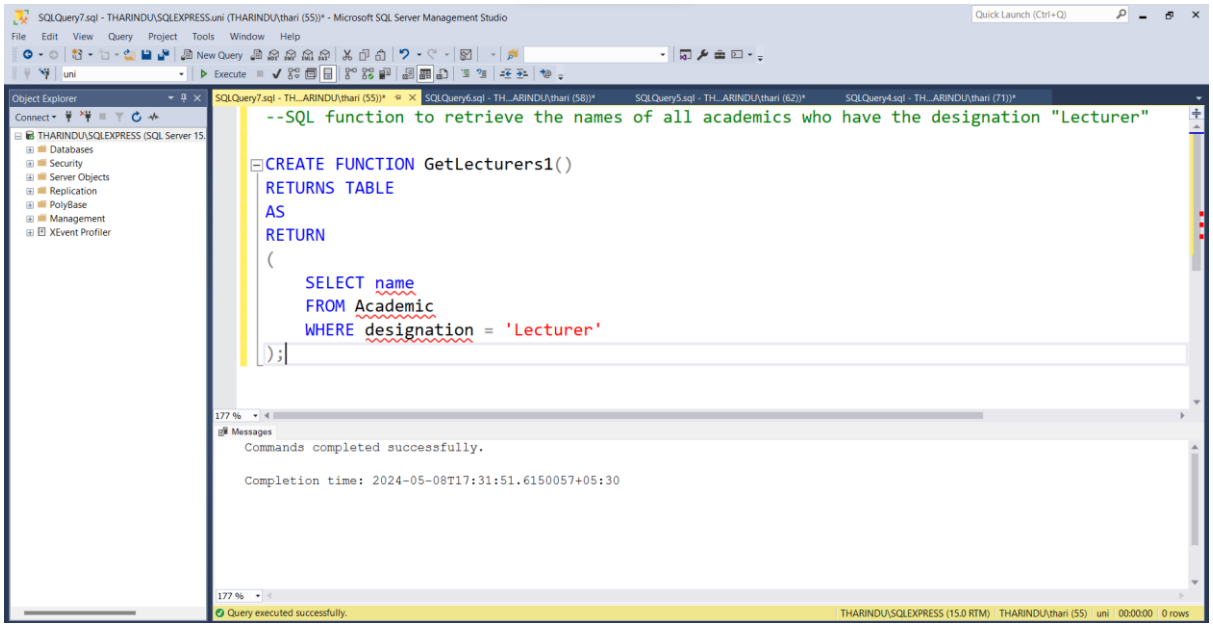


Figure 11: Functions 01

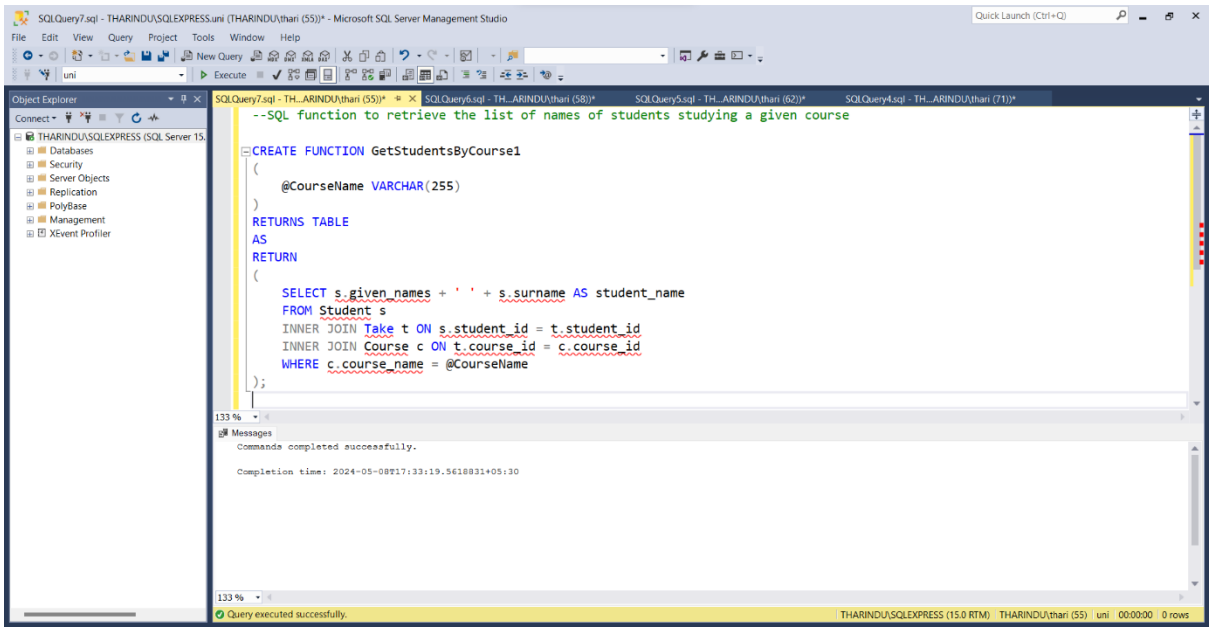


Figure 12: Functions 02

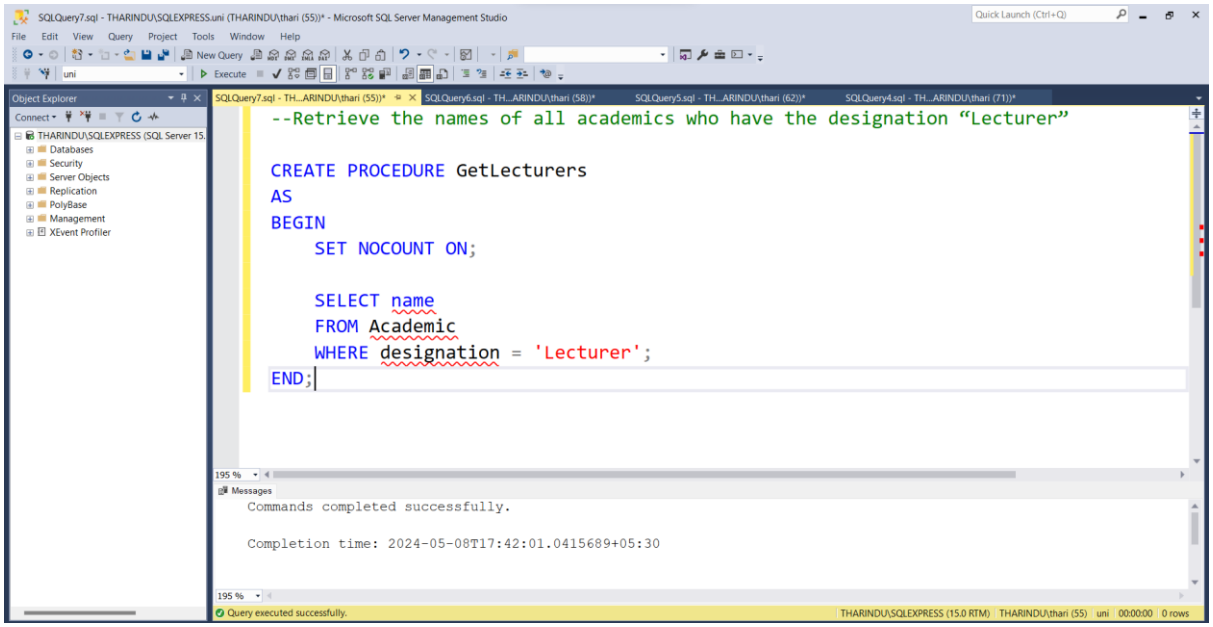


Figure 13: Procedures 01

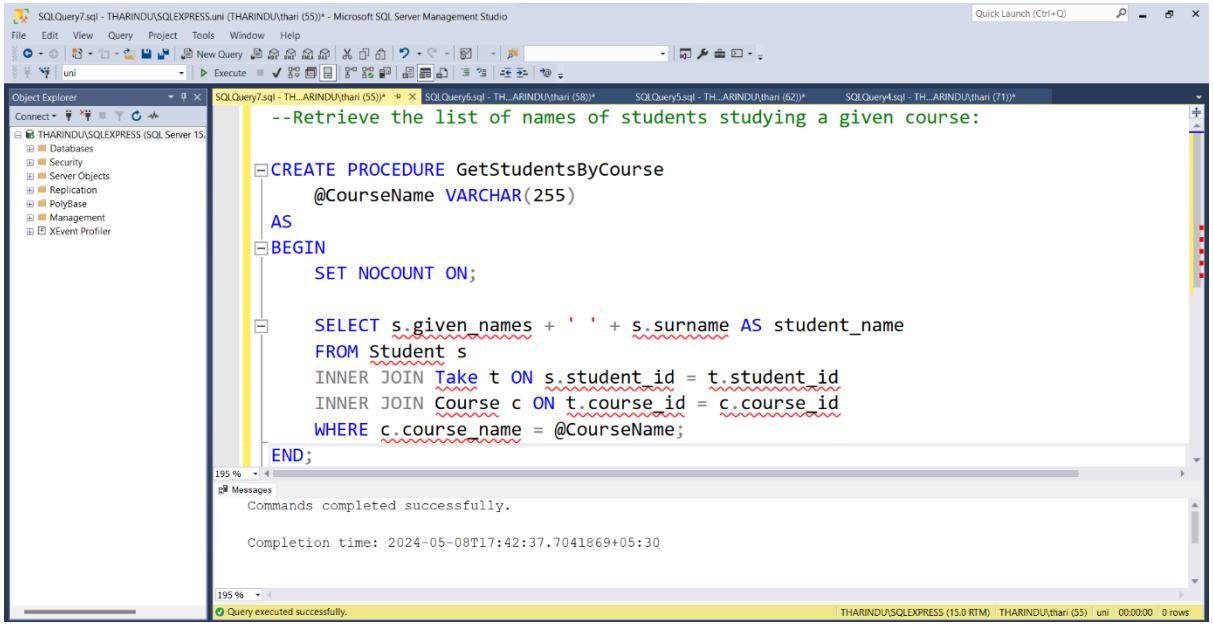


Figure 14: Procedures 02

Database Vulnerabilities

SQL Injection:

SQL injection is a method where input fields are altered and the unauthorized access of database or operational steps together with malicious SQL statement are carried out.This vulnerability occurs when user input is not properly validated or sanitized before being used in SQL queries.

Techniques and Impact:

1.Utilizing SQL injection flaws, it is possible to get a hold of the systems and their components, modify the information, or gain access to the database.

2.Attackers can apply a range of the techniques like Union-based SQL injections, Error-based SQL injections, Blind SQL injections and Piggy-backed queries to achieve a successful exploitation.

3.The impact can range from data breaches, data tampering, and data loss to complete system compromise, depending on the privileges granted to the database user.

Mitigation and Countermeasures:

1.Input Validation: Enact a rigorous interface validation and sanitization rules for all the user inputs meant before it is used in SQL queries.

2.Parameterized Queries: Use parameterized queries or prepared statements to delimit the input with SQL code features and stop injection attacks.

3.Least Privilege Principle: Grant the minimum necessary privileges to database users and applications to limit the potential impact of successful attacks.

4.Regular Updates and Patches: APPLYING THE OPERATIONAL PROCEDURE: keep database software along with its components upgraded with the latest security patches and updates.

5.Web Application Firewalls (WAFs): Block SQL injections through the WAF that will monitoring and filtering of these malicious requests.

6.Error Handling: Implement proper error handling and logging mechanisms to detect and respond to potential attacks.

7.Regular Security Audits: Carry out regular security audits and penetration testing, which reveal SQL Injection flaws and expedite their repairs.

Excessive Privileges:

Excessive privileges refer to granting more permissions or access rights than necessary to users, applications, or services interacting with the database. This weakness consists of misconfigurations if the authorization mechanisms for access are not properly implemented or the principle of least privilege is not complied with.

Techniques and Impact:

The attackers can use more privileges than necessary to get into confidential data or to exercise access which is unauthorized inside the database. Insiders with excessive privileges can intentionally or unintentionally misuse their access rights, leading to data breaches or system compromises. Bad software or code scripts, being run under too much privilege, might carry knowledge of destroying, or getting to confidential details.

Mitigation and Countermeasures:

1. Principle of Least Privilege: Follow the principle of least privilege by restricting users and programs to obtain only the permissions they are necessary for to complete the intended operations.

2. Role-based Access Control (RBAC): Implement RBAC to manage and restrict access based on predefined roles and responsibilities.

3. Periodic Access Reviews: Frequently go through user and application permissions evaluation in order to discover and remove clearance overstepping behavior.

4. Separation of Duties: Don't give all the tasks and responsibilities to one person or role only. This will ensure backup and keep people from misusing.

5. Auditing and Monitoring: Implement auditing and monitoring mechanisms to track and log database activities, enabling detection of suspicious or unauthorized actions.

6. Access Control Policies: Set up and apply strong access management policies that outline the policies for giving, removing and accepting the access grants.

7. Security Awareness Training: Offer a routine information technology security orientation sessions for the users to highlight the necessity of protecting the data and following the secure guidelines and best practices.

Student Registration Number	Student Name
IT22249852	Karunarathna. P.M.T.L.
IT22083678	Sahan H.P.T
IT22230010	Umayanga H.L.A
IT22110220	Apeksha M.k.S.R