

SQL Tutorial – Views, Stored Procs & Triggers

CS340 | SQL TUTORIAL 8
ALI KHAWAJA

- *You might encounter errors in some of the following code. Fix the errors based on class discussion and your understanding of the issue.*
- *Pay attention to all T-SQL Code including the new built-in functions introduced in the code.*

1. Database and Schema Setup (SQL Server)

First, we create the database and the schema.

```
-- 1. Create the Database
IF DB_ID('LUMS') IS NOT NULL
    DROP DATABASE LUMS;
CREATE DATABASE LUMS;
GO
```

```
USE LUMS;
GO
```

```
-- 2. Create the Schema
IF SCHEMA_ID('cs340') IS NOT NULL
    DROP SCHEMA cs340;
CREATE SCHEMA cs340;
GO
```

2. Table Creation and Data Population

We create the three required tables using INT for IDs and enforce relationships with **Foreign Keys**.

A. cs340.students Table

```
CREATE TABLE cs340.students (  
    student_id INT PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    date_of_birth DATE,  
    major VARCHAR(50)  
);
```

```
-- Insert 15 rows of data  
INSERT INTO cs340.students (student_id, first_name, last_name, date_of_birth, major) VALUES  
(1001, 'Ali', 'Khan', '2003-05-10', 'CS'),  
(1002, 'Sara', 'Ahmed', '2004-11-20', 'EE'),  
(1003, 'Zain', 'Malik', '2003-01-15', 'CS'),  
(1004, 'Aisha', 'Raza', '2005-07-25', 'BBA'),  
(1005, 'Omar', 'Butt', '2002-12-01', 'CS'),  
(1006, 'Fatima', 'Siddiqui', '2004-03-30', 'HSS'),  
(1007, 'Kamran', 'Sheikh', '2003-09-05', 'EE'),  
(1008, 'Hina', 'Tariq', '2005-02-18', 'CS'),  
(1009, 'Junaid', 'Ali', '2004-06-12', 'BBA'),  
(1010, 'Saba', 'Iqbal', '2003-04-22', 'HSS'),  
(1011, 'Usman', 'Farooq', '2004-10-14', 'CS'),  
(1012, 'Nida', 'Javed', '2002-08-08', 'EE'),  
(1013, 'Rehan', 'Shah', '2005-01-28', 'CS'),  
(1014, 'Maha', 'Aslam', '2003-03-17', 'BBA'),  
(1015, 'Taimoor', 'Zahid', '2004-09-01', 'CS');
```

B. cs340.addresses Table

```
CREATE TABLE cs340.addresses (  
    address_id INT PRIMARY KEY,  
    student_id INT UNIQUE,  
    street VARCHAR(100),  
    city VARCHAR(50),  
    postal_code VARCHAR(10),  
    CONSTRAINT FK_StudentAddress FOREIGN KEY (student_id) REFERENCES cs340.students(student_id)  
);
```

```
-- Insert 15 rows of data  
INSERT INTO cs340.addresses (address_id, student_id, street, city, postal_code) VALUES  
(1, 1001, '12A Model Town', 'Lahore', '54000'),  
(2, 1002, 'B-4 Gulberg', 'Lahore', '54660'),  
(3, 1003, '34 Defense Rd', 'Karachi', '75500'),  
(4, 1004, '5 Block 10', 'Islamabad', '44000'),  
(5, 1005, '10-C DHA Phase 5', 'Lahore', '54792'),  
(6, 1006, '7 Railway Colony', 'Multan', '60000'),  
(7, 1007, '45 Clifton Block 9', 'Karachi', '75600'),  
(8, 1008, '9 Abbott Rd', 'Lahore', '54000'),  
(9, 1009, '20 Blue Area', 'Islamabad', '44000'),  
(10, 1010, '1 Industrial Area', 'Faisalabad', '38000'),  
(11, 1011, '11-B Tech Society', 'Lahore', '54000'),  
(12, 1012, '3 Sector F-7/3', 'Islamabad', '44000'),  
(13, 1013, '8 Sunset Blvd', 'Karachi', '75600'),  
(14, 1014, '15 Mall Road', 'Lahore', '54000'),
```

```
(15, 1015, '6 Garden Town', 'Rawalpindi', '46000');
```

C. cs340.course_registrations Table

```
CREATE TABLE cs340.course_registrations (  
    registration_id INT PRIMARY KEY, IDENTITY (1, 1)  
    student_id INT,  
    course_code VARCHAR(10) NOT NULL,  
    semester VARCHAR(10) NOT NULL,  
    grade VARCHAR(2),  
    CONSTRAINT FK_StudentRegistration FOREIGN KEY (student_id) REFERENCES  
cs340.students(student_id)  
);
```

```
-- Insert 15 rows of data  
INSERT INTO cs340.course_registrations (registration_id, student_id, course_code, semester, grade)  
VALUES  
(1, 1001, 'CS340', 'Fall2023', 'A'),  
(2, 1003, 'CS340', 'Fall2023', 'B+'),  
(3, 1005, 'CS340', 'Fall2023', 'A-'),  
(4, 1008, 'CS340', 'Fall2023', 'B'),  
(5, 1011, 'CS340', 'Fall2023', 'A'),  
(6, 1013, 'CS340', 'Fall2023', 'C+'),  
(7, 1002, 'EE210', 'Fall2023', 'A'),  
(8, 1007, 'EE210', 'Fall2023', 'B'),  
(9, 1012, 'EE210', 'Fall2023', 'A-'),  
(10, 1004, 'MGMT101', 'Fall2023', 'B+'),  
(11, 1009, 'MGMT101', 'Fall2023', 'A'),
```

```
(12, 1014, 'MGMT101', 'Fall2023', 'B'),  
(13, 1001, 'SS101', 'Spring2023', 'A-'),  
(14, 1006, 'SS101', 'Spring2023', 'A'),  
(15, 1010, 'SS101', 'Spring2023', 'B+');
```

3. Views: Tutorial and Demonstration

A. Creating and Selecting Data from a View

Goal: Create a view showing the full details of all CS students.

```
-- Create the View  
CREATE VIEW cs340.CS_Students_Details AS  
SELECT  
    s.student_id,  
    s.first_name + ' ' + s.last_name AS full_name,  
    s.major,  
    a.city,  
    r.course_code,  
    r.grade  
FROM  
    cs340.students s  
JOIN  
    cs340.addresses a ON s.student_id = a.student_id  
LEFT JOIN  
    cs340.course_registrations r ON s.student_id = r.student_id  
WHERE  
    s.major = 'CS';
```

```
GO
```

```
-- Select Data through the View
```

```
SELECT * FROM cs340.CS_Students_Details WHERE city = 'Lahore';
```

Explanation: The view combines data from three base tables. When queried, the underlying SELECT statement runs, presenting the combined data as a single table.

B. Updating Data Using a View (Updatable View)

Goal: Change a student's address using a view based on a single table.

```
-- Create an Updatable View (Single Table)
```

```
CREATE VIEW cs340.Lahore_Addresses AS
```

```
SELECT
```

```
    student_id,
```

```
    street,
```

```
    city,
```

```
    postal_code
```

```
FROM
```

```
    cs340.addresses
```

```
WHERE
```

```
    city = 'Lahore';
```

```
GO
```

```
-- Update Data through the View
```

```
UPDATE cs340.Lahore_Addresses
```

```
SET street = '25-B New Model Town', postal_code = '54001'
```

```
WHERE student_id = 1001;
```

GO

```
-- Verify the change in the *underlying table*
```

```
SELECT student_id, street, postal_code FROM cs340.addresses WHERE student_id = 1001;
```

Explanation: Since the view is based on a single table (cs340.addresses) and doesn't contain aggregates, the **update is directly propagated** to the base table.

C. Non-Updateable View

Goal: Demonstrate a view that cannot be updated due to aggregation.

```
-- Create a Non-Updateable View (using aggregation/GROUP BY)
```

```
CREATE VIEW cs340.Major_Performance_Summary AS
```

```
SELECT
```

```
    major,
```

```
    COUNT(s.student_id) AS total_students,
```

```
    AVG(CASE WHEN r.grade = 'A' THEN 4.0
```

```
           WHEN r.grade = 'A-' THEN 3.7
```

```
           WHEN r.grade = 'B+' THEN 3.3
```

```
           WHEN r.grade = 'B' THEN 3.0
```

```
           WHEN r.grade = 'B-' THEN 2.7
```

```
           WHEN r.grade = 'C+' THEN 2.3
```

```
           WHEN r.grade = 'C' THEN 2.0
```

```
           WHEN r.grade = 'C-' THEN 1.7
```

```
           WHEN r.grade = 'D+' THEN 1.3
```

```
           WHEN r.grade = 'D' THEN 1.0
```

```
           WHEN r.grade = 'F' THEN 0.0
```

```
           ELSE 2.0 END) AS average_gpa
```

```
FROM
```

```
    cs340.students s
JOIN
    cs340.course_registrations r ON s.student_id = r.student_id
GROUP BY
    major;
```

```
-- Attempt to update the view (This will fail with an error: 'View or function
'cs340.Major_Performance_Summary' is not updateable...')
UPDATE cs340.Major_Performance_Summary
SET total_students = 10
WHERE major = 'CS';
```

Explanation: The view is non-updateable because columns like **total_students** and **average_gpa** are **calculated aggregates**. SQL Server cannot determine which individual student rows to modify in the base tables to satisfy an update on these calculated fields.

4. Stored Procedures: Tutorial and Demonstration

A **Stored Procedure** is a precompiled set of T-SQL statements, used for security, performance, and abstraction of complex logic.

A. Creating and Calling a Stored Procedure (Updating Data)

Goal: Create a procedure to update a student's major.

```
-- Create the Stored Procedure
CREATE PROCEDURE cs340.sp_Update_Student_Major
    @p_student_id INT,
    @p_new_major VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON; -- Prevents returning the count of rows affected
```

```
-- Check if the student exists
IF NOT EXISTS (SELECT 1 FROM cs340.students WHERE student_id = @p_student_id)
BEGIN
    -- Use RAISERROR for custom error handling
    RAISERROR('Student ID %d not found.', 16, 1, @p_student_id);
    RETURN 1; -- Return value for error
END

-- Update the major
UPDATE cs340.students
SET major = @p_new_major
WHERE student_id = @p_student_id;

RETURN 0; -- Return value for success
END;
GO
```

```
-- Call the Stored Procedure (Updating Data)
-- Change student 1003's major from 'CS' to 'DS'
EXEC cs340.sp_Update_Student_Major @p_student_id = 1003, @p_new_major = 'DS';
GO
```

```
-- Verify the update
SELECT student_id, major FROM cs340.students WHERE student_id = 1003;
```

B. Using Return Values

In T-SQL, a procedure returns an integer **status code** using RETURN, typically 0 for success and non-zero for errors.

Goal: Execute the procedure and capture the integer return value.

```
-- 1. Declare a variable to capture the return status
DECLARE @Status INT;

-- 2. Execute the procedure, assigning the return value to the variable
EXEC @Status = cs340.sp_Update_Student_Major @p_student_id = 1005, @p_new_major = 'DS';

-- 3. Check and display the status
IF @Status = 0
    PRINT 'Procedure executed successfully (Status: ' + CAST(@Status AS VARCHAR) + ')';
ELSE
    PRINT 'Procedure encountered an error (Status: ' + CAST(@Status AS VARCHAR) + ')';
GO
```

```
-- Demonstrate error return (using a non-existent ID)
DECLARE @ErrorStatus INT;
EXEC @ErrorStatus = cs340.sp_Update_Student_Major @p_student_id = 9999, @p_new_major = 'Invalid';

-- The RAISERROR will print a message, and the variable will capture the non-zero return value (1).
PRINT 'Error Status Captured: ' + CAST(@ErrorStatus AS VARCHAR);
```

5. Triggers: AFTER and INSTEAD OF

Triggers are special stored procedures that execute automatically in response to data modification events.

A. AFTER Trigger (ON Tables)

Goal: Log every change to a student's address into an auditing table.

Create Audit Table:

```
CREATE TABLE cs340.Address_Audit (  
    audit_id INT PRIMARY KEY IDENTITY(1,1),  
    student_id INT,  
    old_city VARCHAR(50),  
    new_city VARCHAR(50),  
    change_date DATETIME DEFAULT GETDATE(),  
    operation_type VARCHAR(10)  
);  
GO
```

Create AFTER UPDATE Trigger:

```
CREATE TRIGGER cs340.trg_After_Address_Update  
ON cs340.addresses  
AFTER UPDATE  
AS  
BEGIN  
    SET NOCOUNT ON;  
  
    -- Check if the 'city' column was modified  
    IF UPDATE(city)  
    BEGIN  
        -- Insert the old and new values into the audit table  
        INSERT INTO cs340.Address_Audit (student_id, old_city, new_city, operation_type)  
        SELECT
```

```

        d.student_id,
        d.city AS old_city,
        i.city AS new_city,
        'UPDATE'
FROM
    DELETED d -- 'DELETED' holds the state *before* the update
INNER JOIN
    INSERTED i ON d.student_id = i.student_id -- 'INSERTED' holds the state *after* the update
WHERE
    d.city <> i.city; -- Only log if the city actually changed
END
END;
GO

```

Demonstration:

```

-- Update an address
UPDATE cs340.addresses
SET city = 'Lahore Cantt'
WHERE student_id = 1003; -- Zain Malik's city changes from Karachi
GO

```

```

-- Check the Audit Table
SELECT * FROM cs340.Address_Audit WHERE student_id = 1003;

```

B. INSTEAD OF Trigger (ON Views)

Goal: Make our non-updatable, multi-table view (cs340.CS_Students_Details) insertable by redirecting the insertion to the two underlying tables (students and course_registrations).

```

CREATE TRIGGER cs340.trg_InsteadOf_Insert_CS_Details

```

```

ON cs340.CS_Students_Details
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    -- Temp variable to store split name parts
    DECLARE @FirstName VARCHAR(50), @LastName VARCHAR(50);
    -- 1. Insert into the students table
    -- Since full_name is one column, we have to parse it.
    INSERT INTO cs340.students (student_id, first_name, last_name, major)
    SELECT
        i.student_id,
        LEFT(i.full_name, CHARINDEX(' ', i.full_name) - 1), -- Extract first name
        SUBSTRING(i.full_name, CHARINDEX(' ', i.full_name) + 1, 50), -- Extract last name
        i.major
    FROM
        INSERTED i
    WHERE
        NOT EXISTS (SELECT 1 FROM cs340.students s WHERE s.student_id = i.student_id); -- Only
insert if student is new

    -- 2. Insert into the course_registrations table
    INSERT INTO cs340.course_registrations (student_id, course_code, semester, grade)
    SELECT
        student_id,
        course_code,
        'Fall2024', -- Default semester, since the view doesn't contain it in our INSERT statement
        grade

```

```
FROM
    INSERTED
WHERE course_code IS NOT NULL; -- Ensure a course is provided
END;
GO
```

Demonstration:

```
-- Insert a new student AND their course registration via the View
SET IDENTITY_INSERT ON;
GO
INSERT INTO cs340.CS_Students_Details (student_id, full_name, major, course_code, grade)
VALUES (1016, 'Hamza Jamil', 'CS', 'CS340', 'A+');
GO
SET IDENTITY_INSERT OFF;
GO
```

```
-- Verify the insertion in the base tables
SELECT * FROM cs340.students WHERE student_id = 1016;
SELECT * FROM cs340.course_registrations WHERE student_id = 1016;
```

Explanation: The INSTEAD OF INSERT trigger **intercepts** the insertion attempt on the view and executes the logic to correctly insert the data into the two underlying base tables (students and course_registrations). This makes a typically non-insertable view functional.