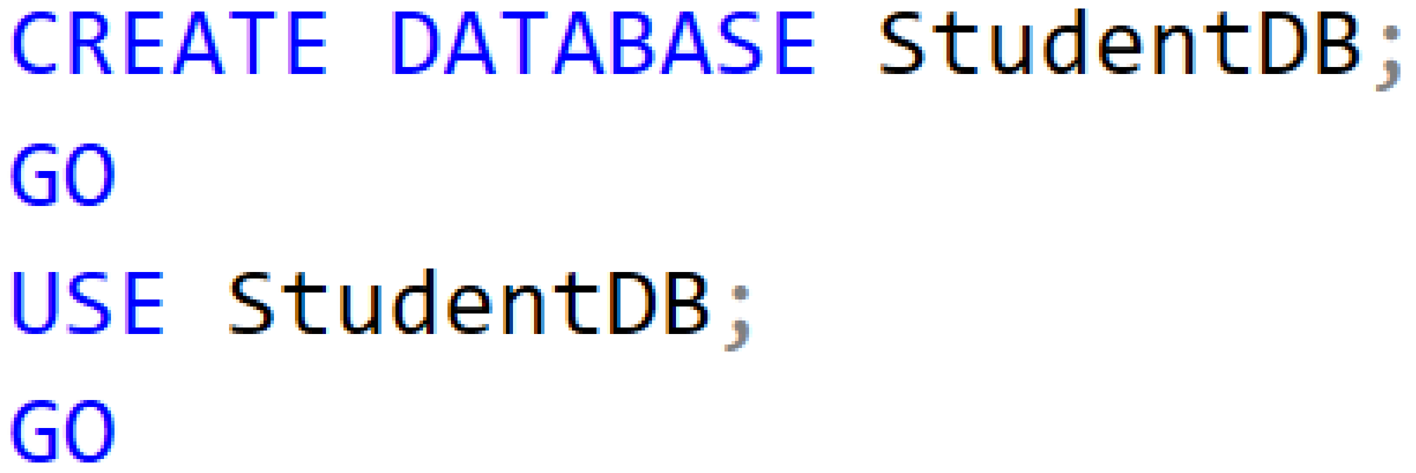
STUDENT MANAGEMENT SYSTEM SQL PROJECT

Overview

This SQL Server project creates a complete StudentDB with 6 core tables—Students, Courses, Faculty, Enrollments, Payments, and StudentAudit—for managing academic records efficiently. Nonclustered indexes optimize query performance on key columns like Email and StudentID, while 3 stored procedures (AddStudent, EnrollStudent, AddPayment) handle essential operations with built-in validation and error handling. An audit trigger automatically logs all INSERT/UPDATE/DELETE changes on student records for full compliance tracking. The system includes realistic sample data and 8 comprehensive unit tests that validate UDFs, procedures, and reporting queries end-to-end Project.

Table of Content

Phase 1 - Database creation

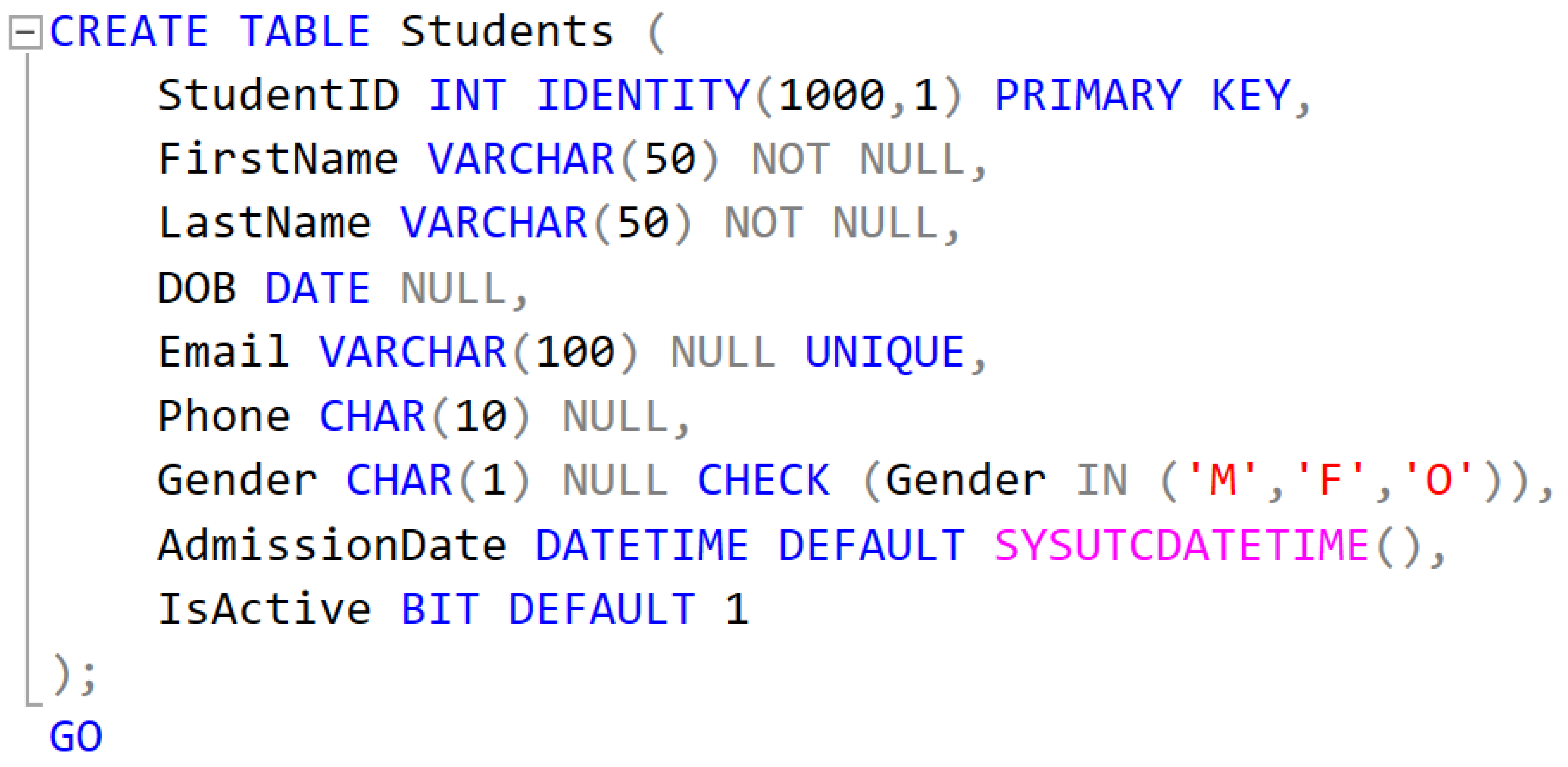


```
CREATE DATABASE StudentDB;
GO
USE StudentDB;
GO
```

Phase 2 – Table Creation

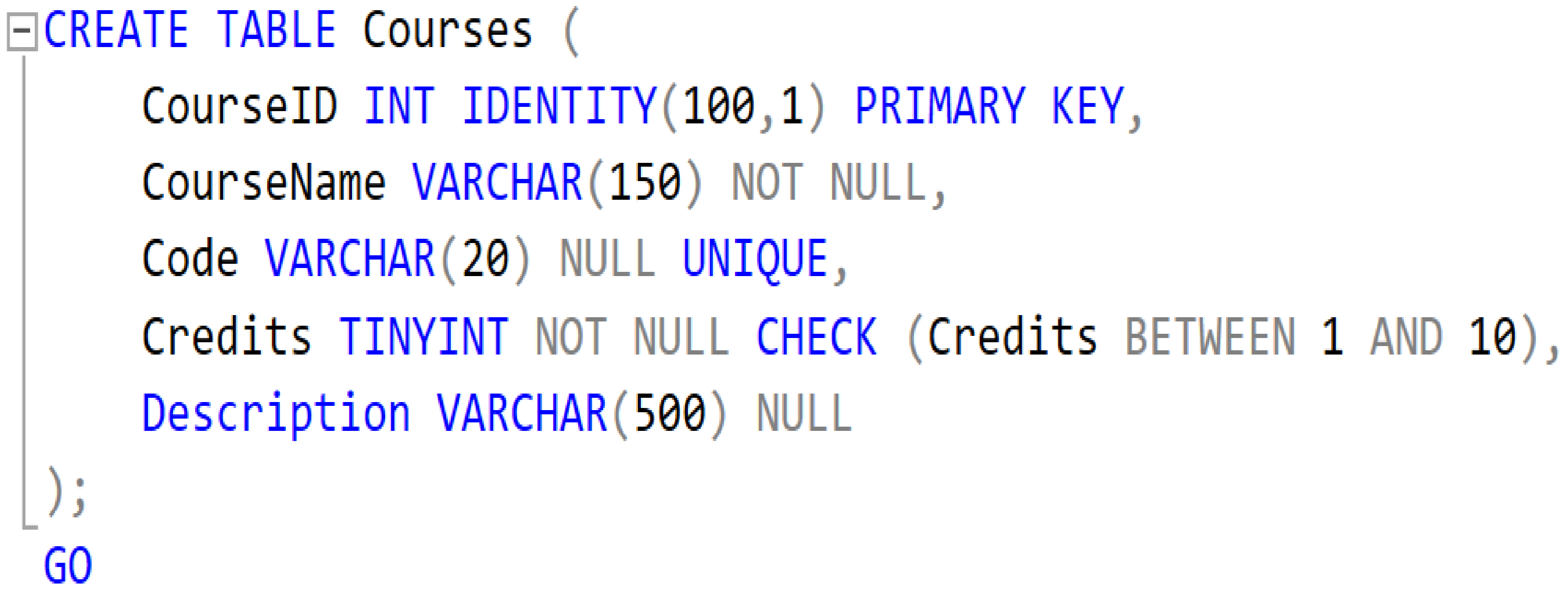
- Students
- Courses
- Faculty
- Enrolments
- Payments
- StudentAudit

•Students



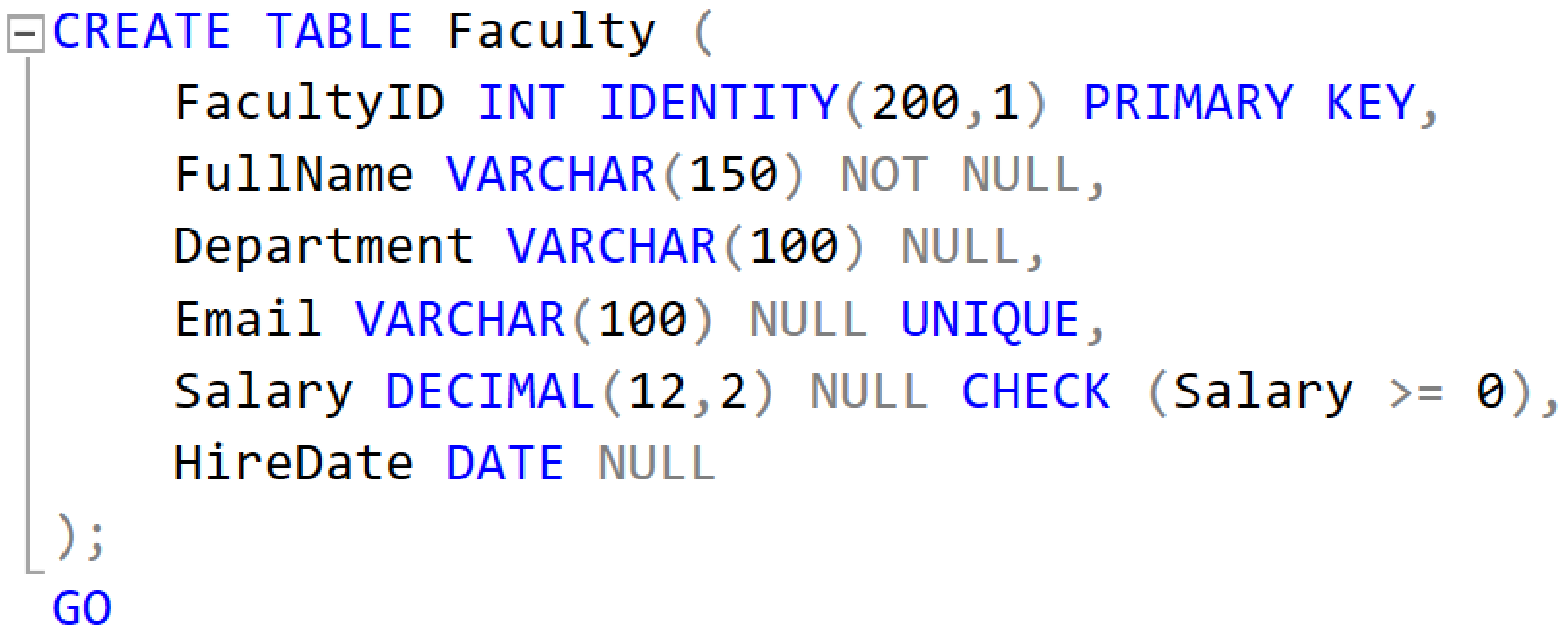
```sql
CREATE TABLE Students (
    StudentID INT IDENTITY(1000,1) PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DOB DATE NULL,
    Email VARCHAR(100) NULL UNIQUE,
    Phone CHAR(10) NULL,
    Gender CHAR(1) NULL CHECK (Gender IN ('M','F','O')),
    AdmissionDate DATETIME DEFAULT SYSUTCDATETIME(),
    IsActive BIT DEFAULT 1
);
GO
```

•Courses



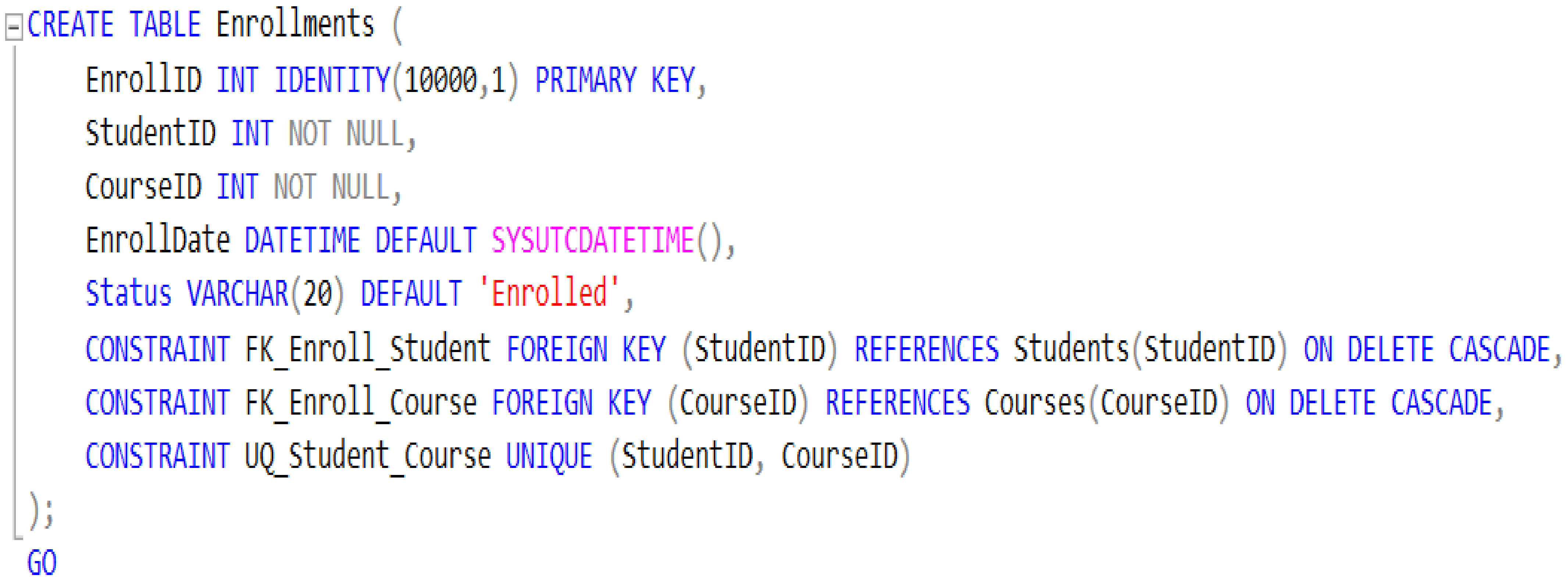
```sql
CREATE TABLE Courses (
    CourseID INT IDENTITY(100,1) PRIMARY KEY,
    CourseName VARCHAR(150) NOT NULL,
    Code VARCHAR(20) NULL UNIQUE,
    Credits TINYINT NOT NULL CHECK (Credits BETWEEN 1 AND 10),
    Description VARCHAR(500) NULL
);
GO
```

•Faculty



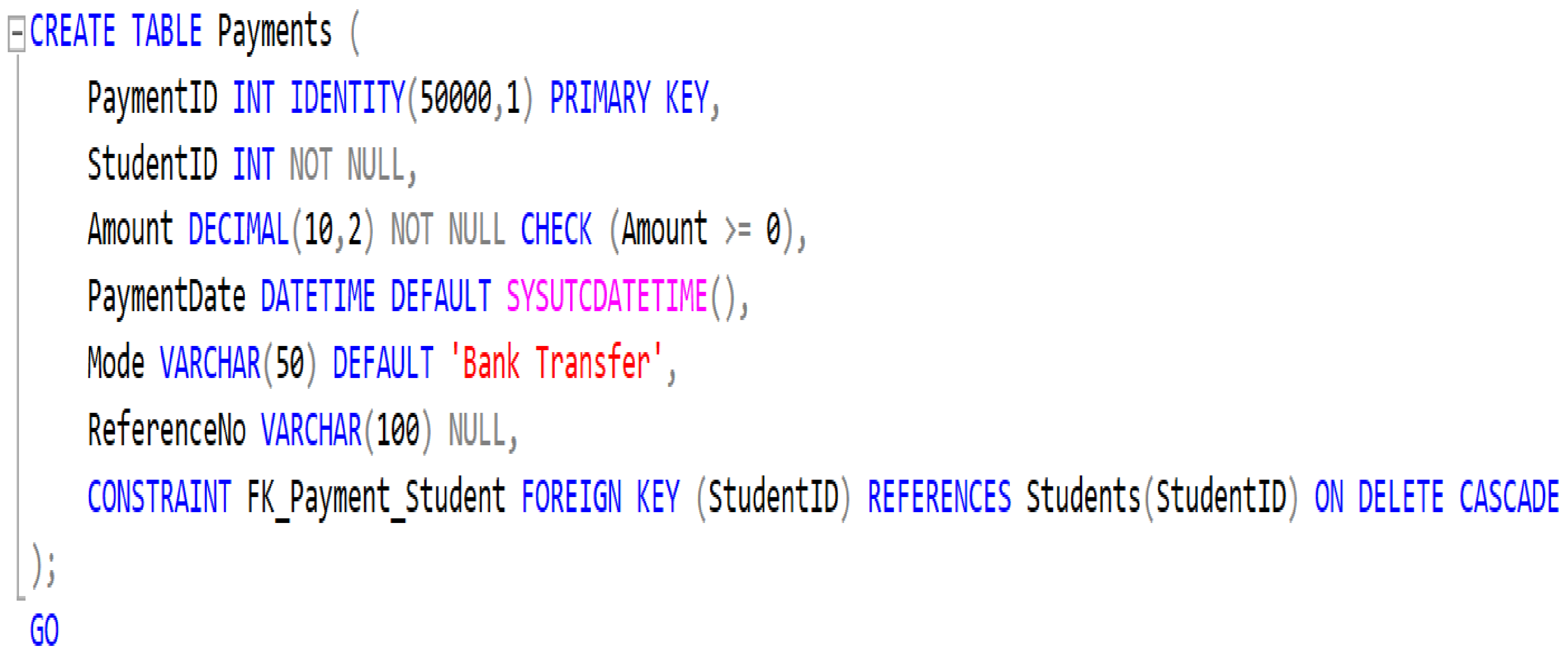
```sql
CREATE TABLE Faculty (
    FacultyID INT IDENTITY(200,1) PRIMARY KEY,
    FullName VARCHAR(150) NOT NULL,
    Department VARCHAR(100) NULL,
    Email VARCHAR(100) NULL UNIQUE,
    Salary DECIMAL(12,2) NULL CHECK (Salary >= 0),
    HireDate DATE NULL
);
GO
```

•Enrolments



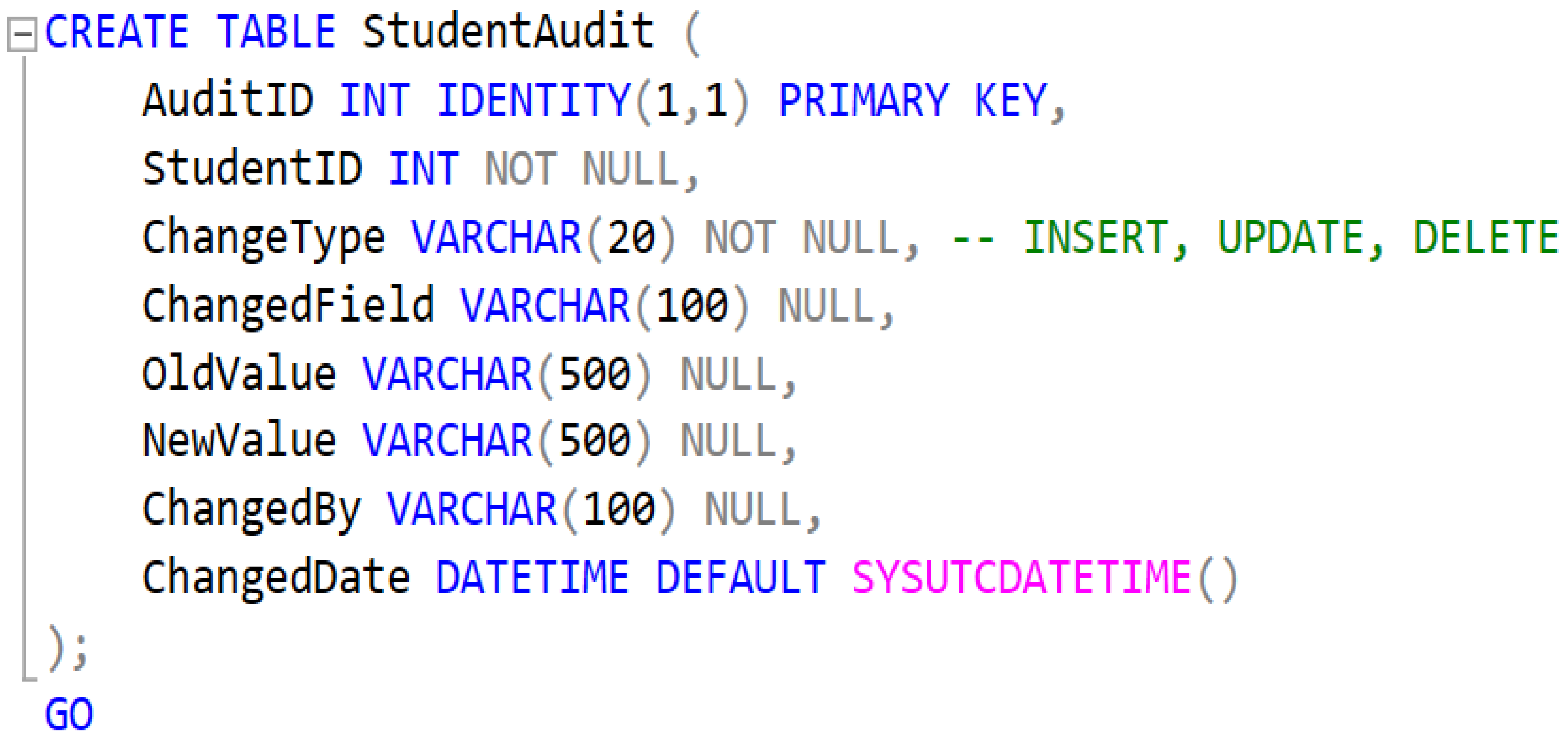
```sql
CREATE TABLE Enrollments (
    EnrollID INT IDENTITY(10000,1) PRIMARY KEY,
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    EnrollDate DATETIME DEFAULT SYSUTCDATETIME(),
    Status VARCHAR(20) DEFAULT 'Enrolled',
    CONSTRAINT FK_Enroll_Student FOREIGN KEY (StudentID) REFERENCES Students(StudentID) ON DELETE CASCADE,
    CONSTRAINT FK_Enroll_Course FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE,
    CONSTRAINT UQ_Student_Course UNIQUE (StudentID, CourseID)
);
GO
```

•Payments



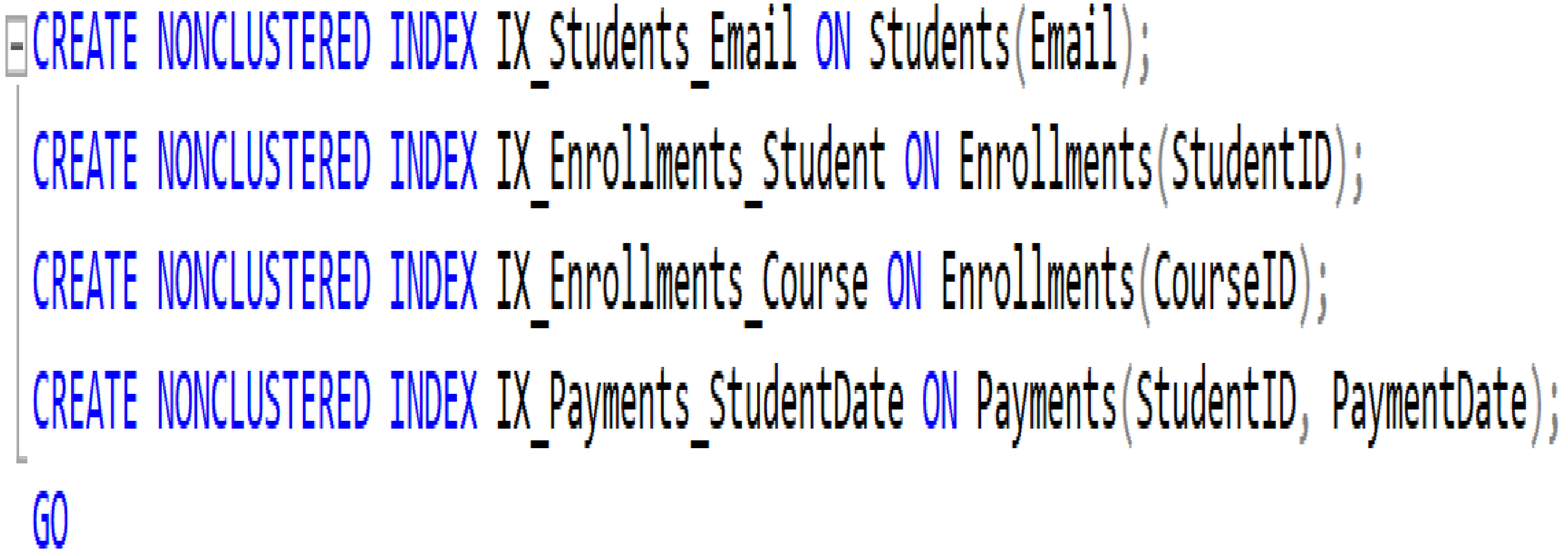
```
CREATE TABLE Payments (
    PaymentID INT IDENTITY(50000,1) PRIMARY KEY,
    StudentID INT NOT NULL,
    Amount DECIMAL(10,2) NOT NULL CHECK (Amount >= 0),
    PaymentDate DATETIME DEFAULT SYSUTCDATETIME(),
    Mode VARCHAR(50) DEFAULT 'Bank Transfer',
    ReferenceNo VARCHAR(100) NULL,
    CONSTRAINT FK_Payment_Student FOREIGN KEY (StudentID) REFERENCES Students(StudentID) ON DELETE CASCADE
);
GO
```

•StudentAudit



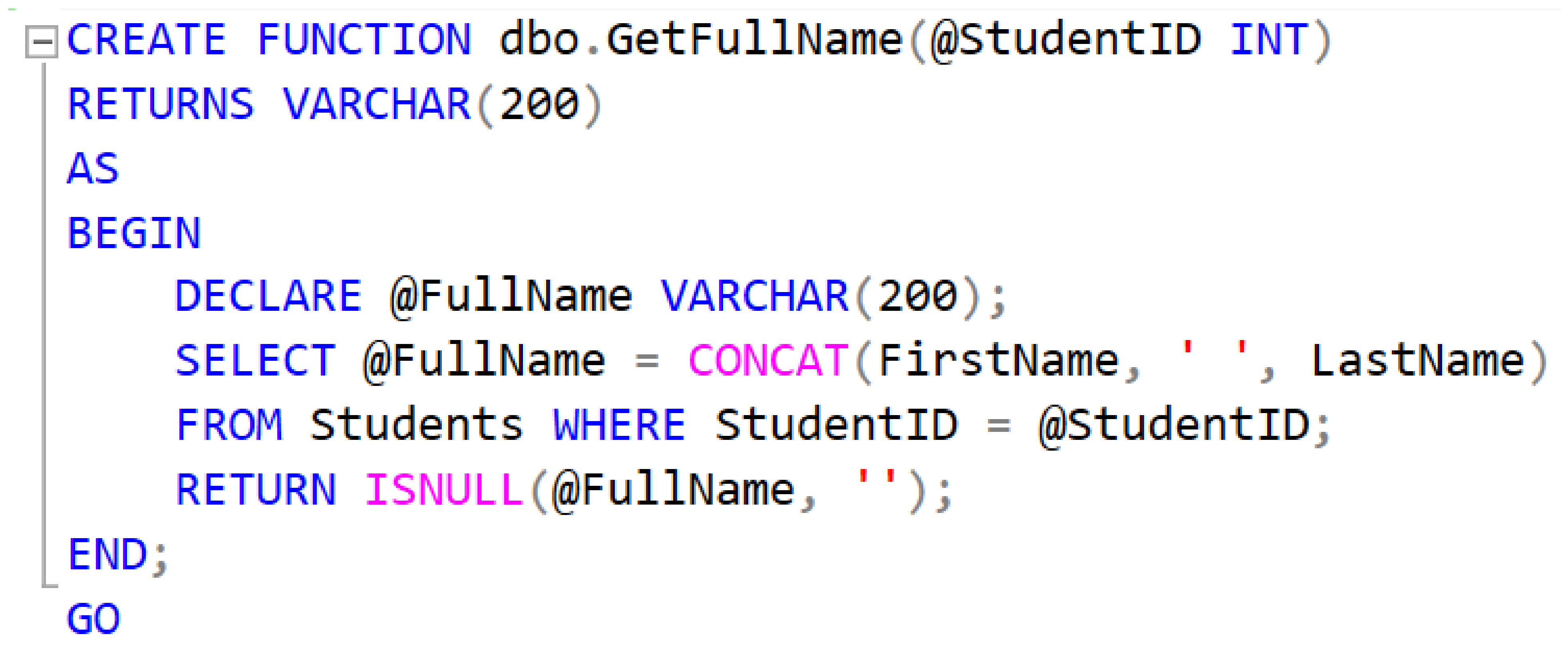
```
CREATE TABLE StudentAudit (
    AuditID INT IDENTITY(1,1) PRIMARY KEY,
    StudentID INT NOT NULL,
    ChangeType VARCHAR(20) NOT NULL, -- INSERT, UPDATE, DELETE
    ChangedField VARCHAR(100) NULL,
    OldValue VARCHAR(500) NULL,
    NewValue VARCHAR(500) NULL,
    ChangedBy VARCHAR(100) NULL,
    ChangedDate DATETIME DEFAULT SYSUTCDATETIME()
);
GO
```

Phase 3 - Indexes



```sql
CREATE NONCLUSTERED INDEX IX_Students_Email ON Students(Email);
CREATE NONCLUSTERED INDEX IX_Enrollments_Student ON Enrollments(StudentID);
CREATE NONCLUSTERED INDEX IX_Enrollments_Course ON Enrollments(CourseID);
CREATE NONCLUSTERED INDEX IX_Payments_StudentDate ON Payments(StudentID, PaymentDate);
GO
```

Phase 4 - UDF Implementation: GetFullName

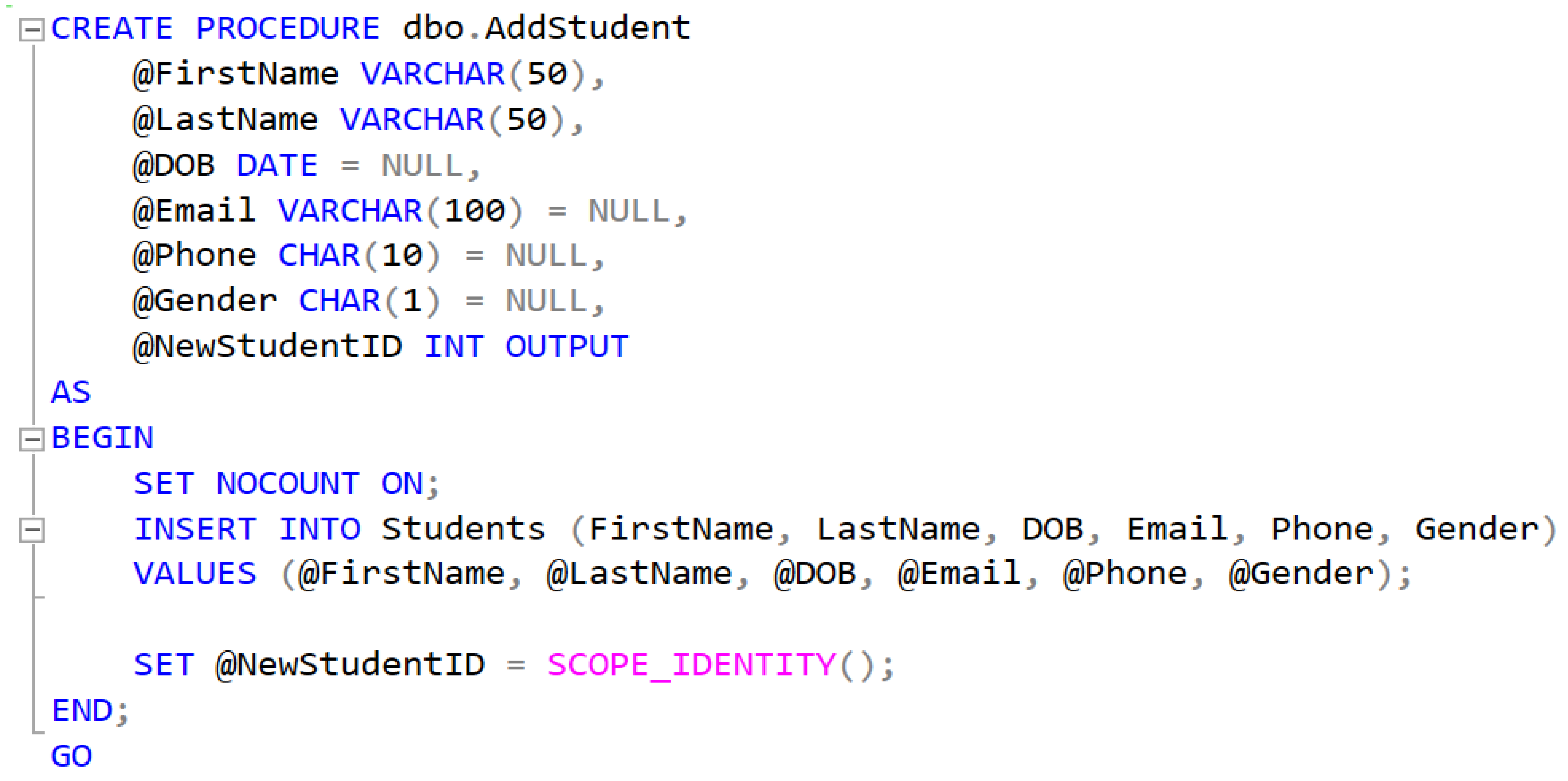


```sql
CREATE FUNCTION dbo.GetFullName(@StudentID INT)
RETURNS VARCHAR(200)
AS
BEGIN
    DECLARE @FullName VARCHAR(200);
    SELECT @FullName = CONCAT(FirstName, ' ', LastName)
    FROM Students WHERE StudentID = @StudentID;
    RETURN ISNULL(@FullName, '');
END;
GO
```

Phase 5 – Stored Procedures

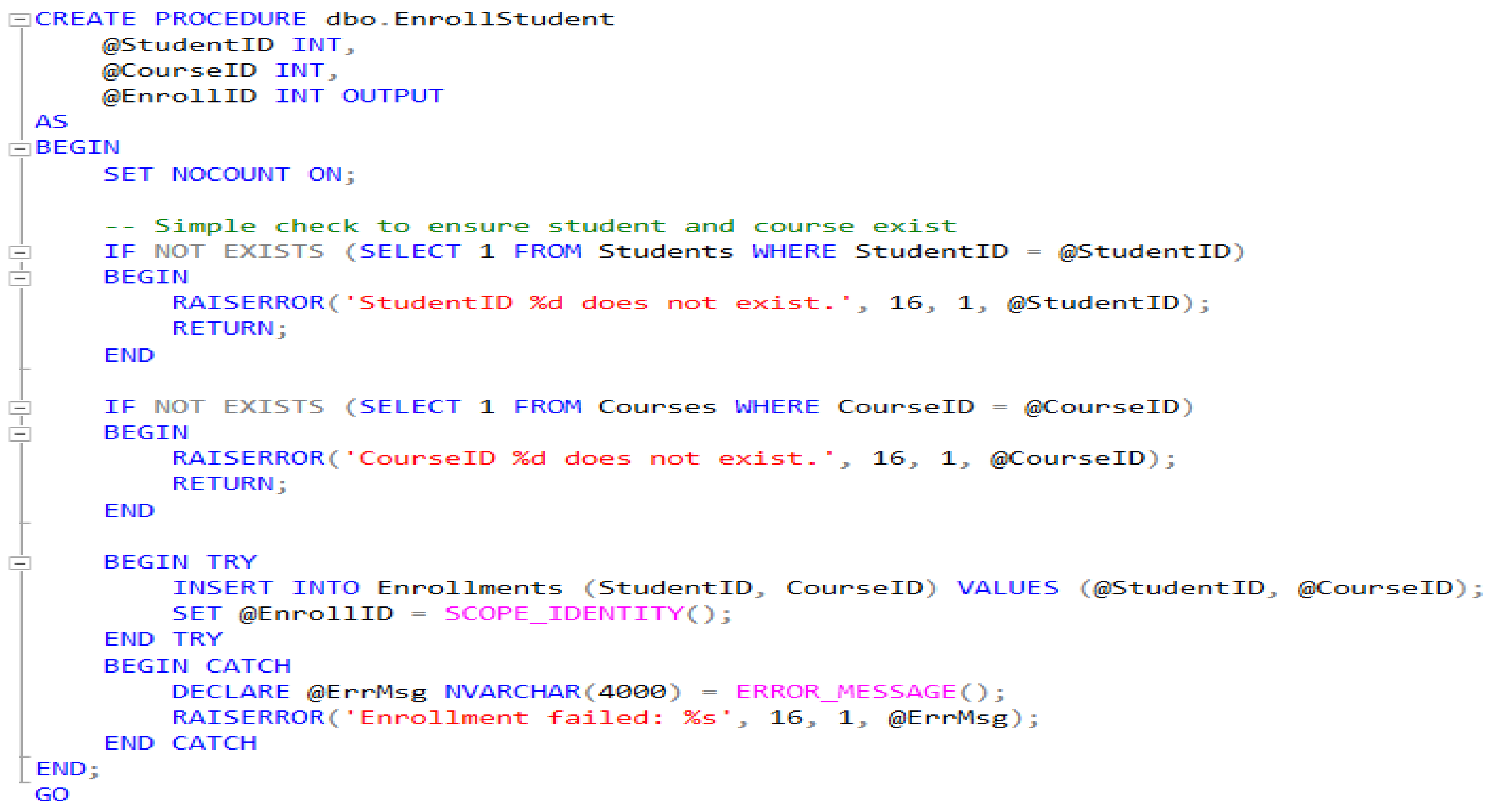
- AddStudent
- EnrollStudent
- AddPayment

•AddStudent



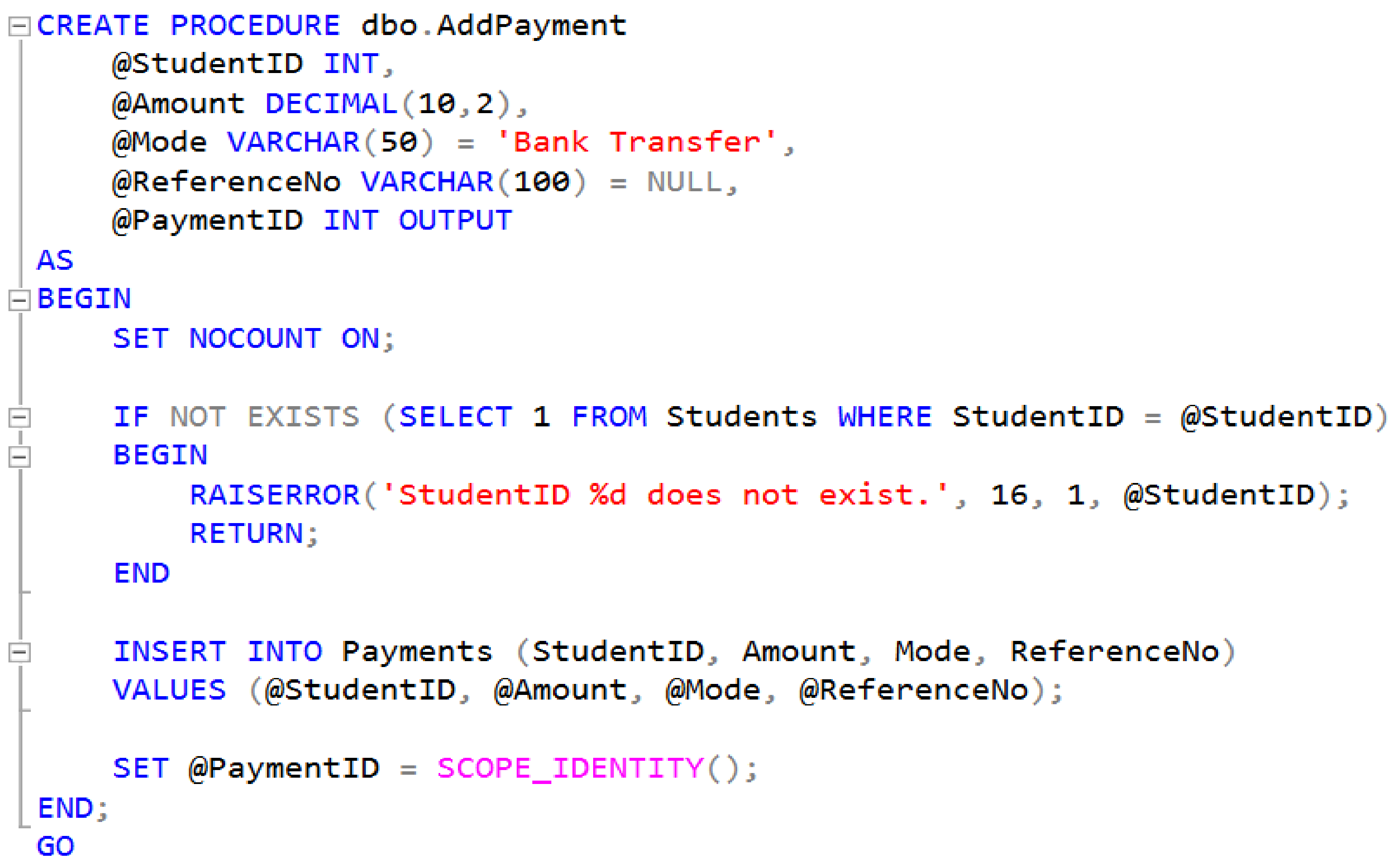
```
CREATE PROCEDURE dbo.AddStudent
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @DOB DATE = NULL,
    @Email VARCHAR(100) = NULL,
    @Phone CHAR(10) = NULL,
    @Gender CHAR(1) = NULL,
    @NewStudentID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Students (FirstName, LastName, DOB, Email, Phone, Gender)
    VALUES (@FirstName, @LastName, @DOB, @Email, @Phone, @Gender);

    SET @NewStudentID = SCOPE_IDENTITY();
END;
GO
```

•EnrollStudent



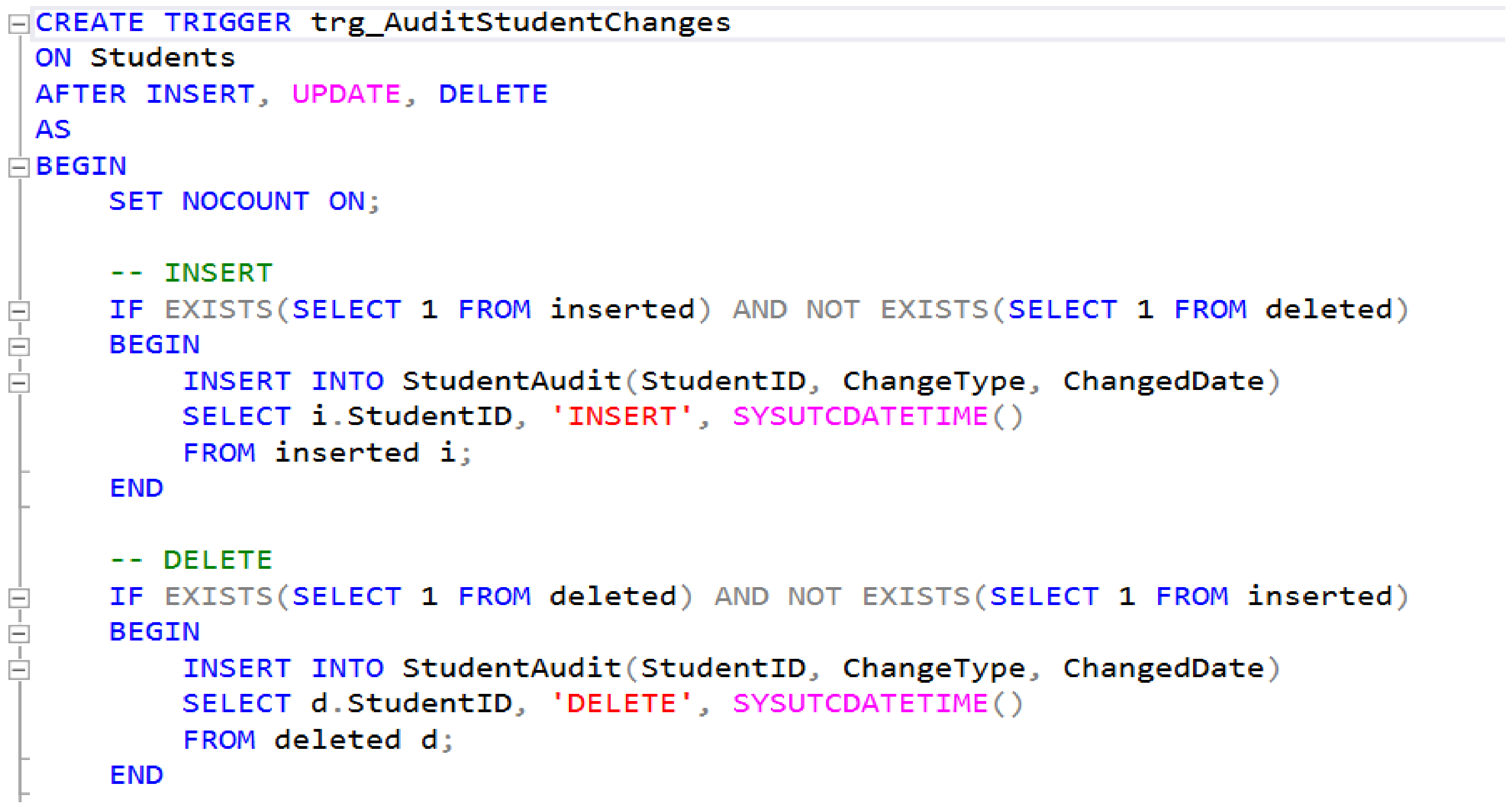
```sql
CREATE PROCEDURE dbo.EnrollStudent
    @StudentID INT,
    @CourseID INT,
    @EnrollID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    -- Simple check to ensure student and course exist
    IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('StudentID %d does not exist.', 16, 1, @StudentID);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        RAISERROR('CourseID %d does not exist.', 16, 1, @CourseID);
        RETURN;
    END

    BEGIN TRY
        INSERT INTO Enrollments (StudentID, CourseID) VALUES (@StudentID, @CourseID);
        SET @EnrollID = SCOPE_IDENTITY();
    END TRY
    BEGIN CATCH
        DECLARE @ErrMsg NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR('Enrollment failed: %s', 16, 1, @ErrMsg);
    END CATCH
END;
GO
```

•AddPayment



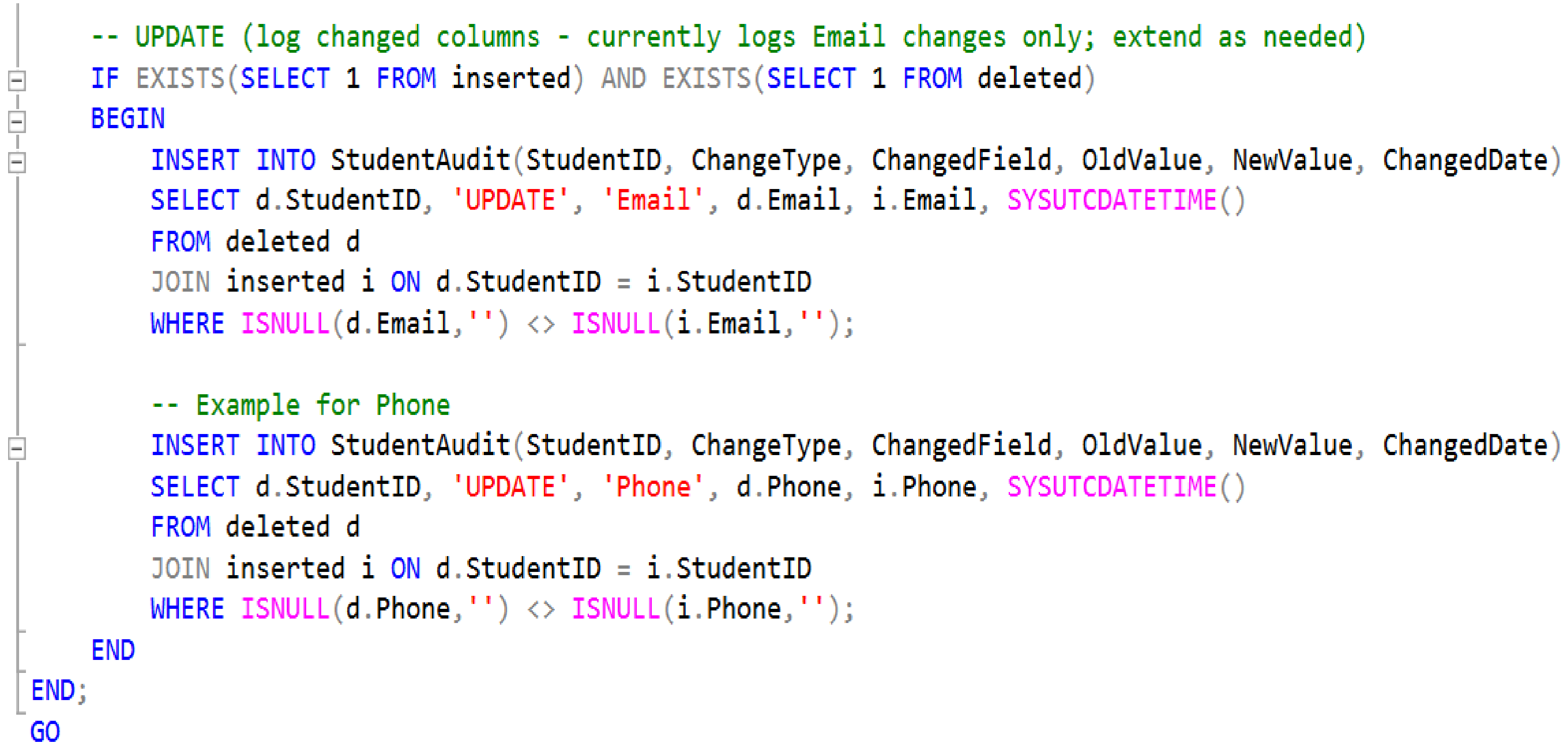
```sql
CREATE PROCEDURE dbo.AddPayment
    @StudentID INT,
    @Amount DECIMAL(10,2),
    @Mode VARCHAR(50) = 'Bank Transfer',
    @ReferenceNo VARCHAR(100) = NULL,
    @PaymentID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('StudentID %d does not exist.', 16, 1, @StudentID);
        RETURN;
    END

    INSERT INTO Payments (StudentID, Amount, Mode, ReferenceNo)
    VALUES (@StudentID, @Amount, @Mode, @ReferenceNo);

    SET @PaymentID = SCOPE_IDENTITY();
END;
GO
```

Phase 6 - Triggers (audit)



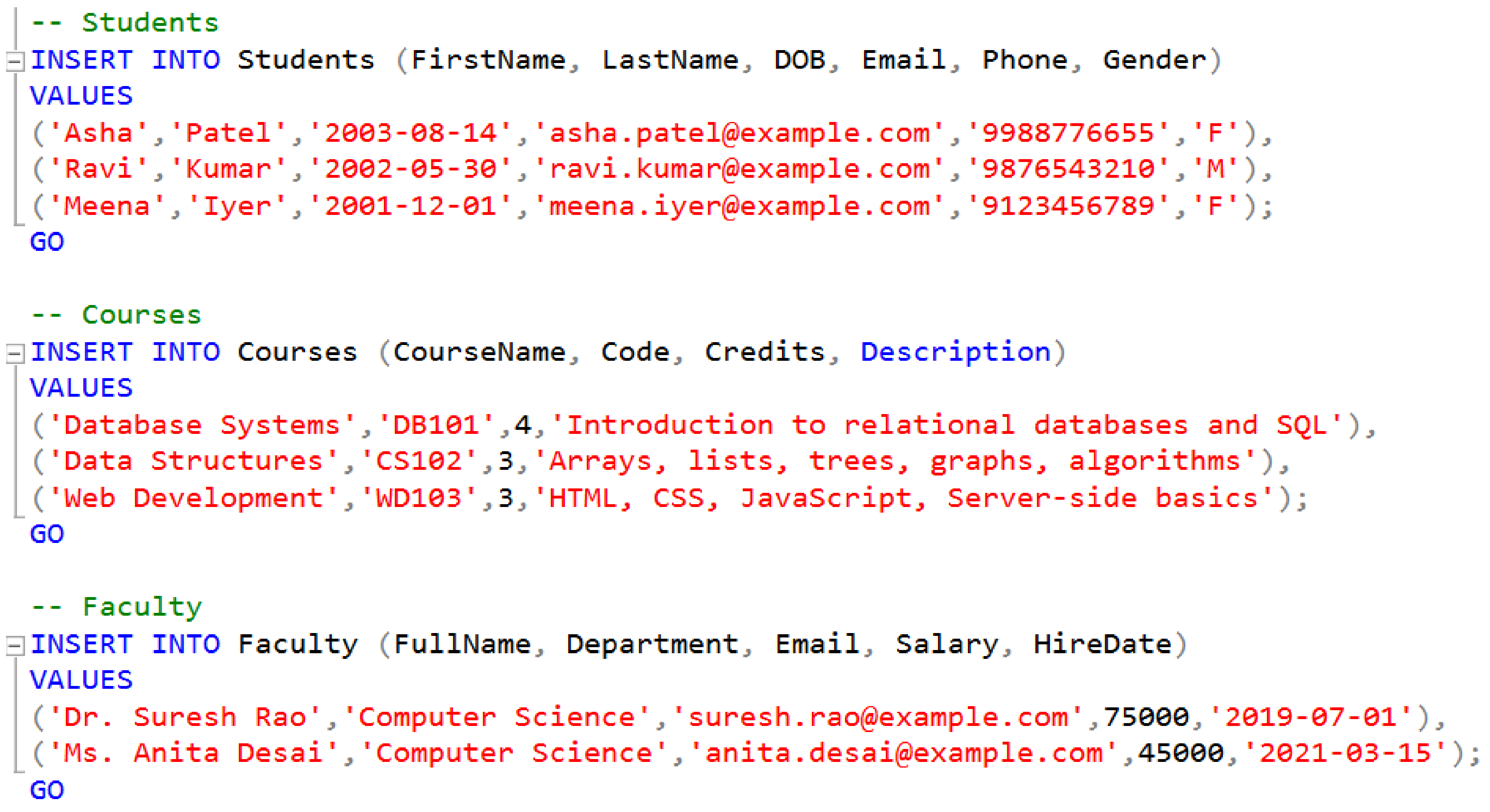
```sql
CREATE TRIGGER trg_AuditStudentChanges
ON Students
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- INSERT
    IF EXISTS(SELECT 1 FROM inserted) AND NOT EXISTS(SELECT 1 FROM deleted)
    BEGIN
        INSERT INTO StudentAudit(StudentID, ChangeType, ChangedDate)
        SELECT i.StudentID, 'INSERT', SYSUTCDATETIME()
        FROM inserted i;
    END

    -- DELETE
    IF EXISTS(SELECT 1 FROM deleted) AND NOT EXISTS(SELECT 1 FROM inserted)
    BEGIN
        INSERT INTO StudentAudit(StudentID, ChangeType, ChangedDate)
        SELECT d.StudentID, 'DELETE', SYSUTCDATETIME()
        FROM deleted d;
    END
```

Phase 6 - Triggers (audit)



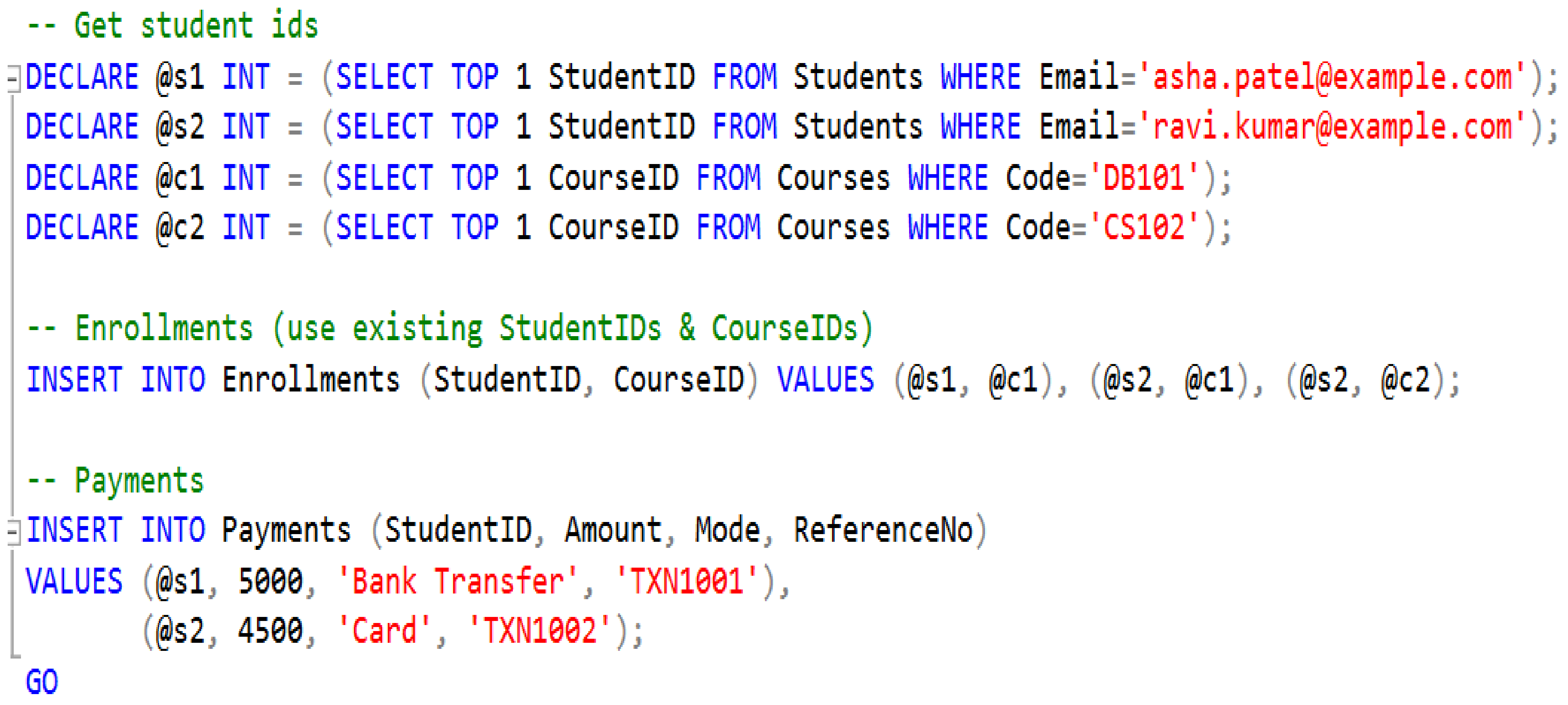
```sql
    -- UPDATE (log changed columns - currently logs Email changes only; extend as needed)
    IF EXISTS(SELECT 1 FROM inserted) AND EXISTS(SELECT 1 FROM deleted)
    BEGIN
        INSERT INTO StudentAudit(StudentID, ChangeType, ChangedField, OldValue, NewValue, ChangedDate)
        SELECT d.StudentID, 'UPDATE', 'Email', d.Email, i.Email, SYSUTCDATETIME()
        FROM deleted d
        JOIN inserted i ON d.StudentID = i.StudentID
        WHERE ISNULL(d.Email,'') <> ISNULL(i.Email,'');

        -- Example for Phone
        INSERT INTO StudentAudit(StudentID, ChangeType, ChangedField, OldValue, NewValue, ChangedDate)
        SELECT d.StudentID, 'UPDATE', 'Phone', d.Phone, i.Phone, SYSUTCDATETIME()
        FROM deleted d
        JOIN inserted i ON d.StudentID = i.StudentID
        WHERE ISNULL(d.Phone,'') <> ISNULL(i.Phone,'');
    END
END;
GO
```

Phase 7 - Sample data (DML)



```sql
-- Students
INSERT INTO Students (FirstName, LastName, DOB, Email, Phone, Gender)
VALUES
('Asha','Patel','2003-08-14','asha.patel@example.com','9988776655','F'),
('Ravi','Kumar','2002-05-30','ravi.kumar@example.com','9876543210','M'),
('Meena','Iyer','2001-12-01','meena.iyer@example.com','9123456789','F');
GO

-- Courses
INSERT INTO Courses (CourseName, Code, Credits, Description)
VALUES
('Database Systems','DB101',4,'Introduction to relational databases and SQL'),
('Data Structures','CS102',3,'Arrays, lists, trees, graphs, algorithms'),
('Web Development','WD103',3,'HTML, CSS, JavaScript, Server-side basics');
GO

-- Faculty
INSERT INTO Faculty (FullName, Department, Email, Salary, HireDate)
VALUES
('Dr. Suresh Rao','Computer Science','suresh.rao@example.com',75000,'2019-07-01'),
('Ms. Anita Desai','Computer Science','anita.desai@example.com',45000,'2021-03-15');
GO
```

Phase 7 - Sample data (DML)

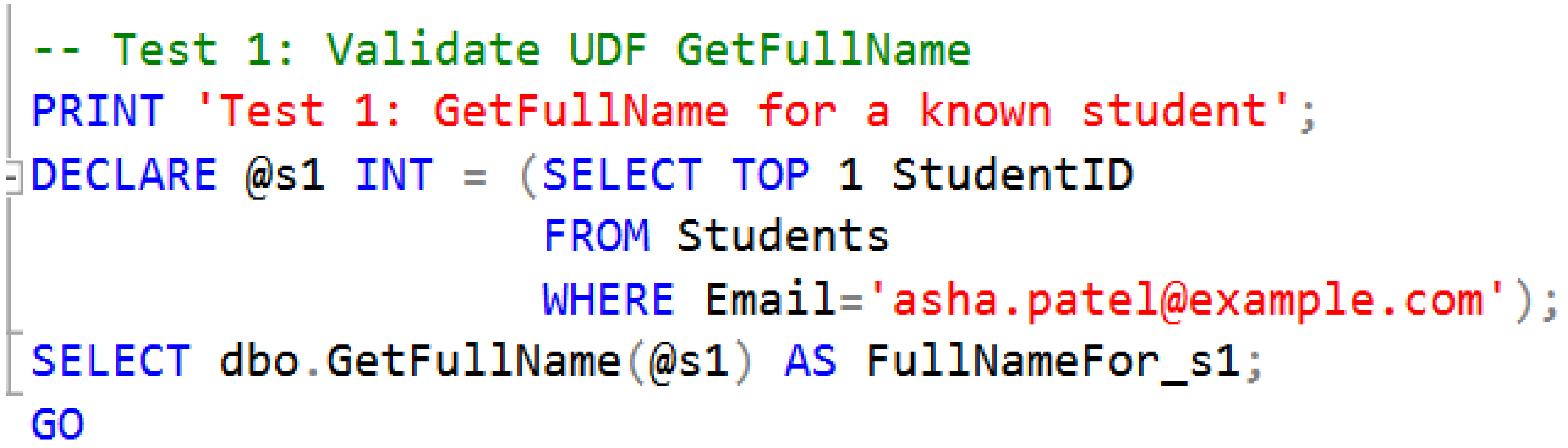


```sql
-- Get student ids
DECLARE @s1 INT = (SELECT TOP 1 StudentID FROM Students WHERE Email='asha.patel@example.com');
DECLARE @s2 INT = (SELECT TOP 1 StudentID FROM Students WHERE Email='ravi.kumar@example.com');
DECLARE @c1 INT = (SELECT TOP 1 CourseID FROM Courses WHERE Code='DB101');
DECLARE @c2 INT = (SELECT TOP 1 CourseID FROM Courses WHERE Code='CS102');

-- Enrollments (use existing StudentIDs & CourseIDs)
INSERT INTO Enrollments (StudentID, CourseID) VALUES (@s1, @c1), (@s2, @c1), (@s2, @c2);

-- Payments
INSERT INTO Payments (StudentID, Amount, Mode, ReferenceNo)
VALUES (@s1, 5000, 'Bank Transfer', 'TXN1001'),
       (@s2, 4500, 'Card', 'TXN1002');
GO
```

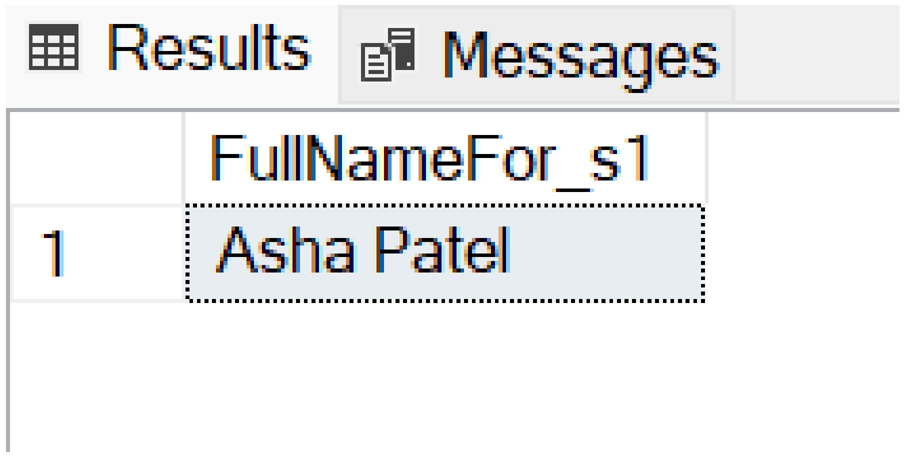
Phase 8 - Simple unit tests (validation queries & procedure calls)

- Validate UDF GetFullName
- AddStudent procedure
- EnrollStudent procedure - positive case
- EnrollStudent procedure - duplicate enrollment should fail due to unique constraint
- AddPayment procedure
- Trigger audit - update student email and phone, then check audit table
- Reporting query - students with their courses
- Payments summary

• Validate UDF GetFullName



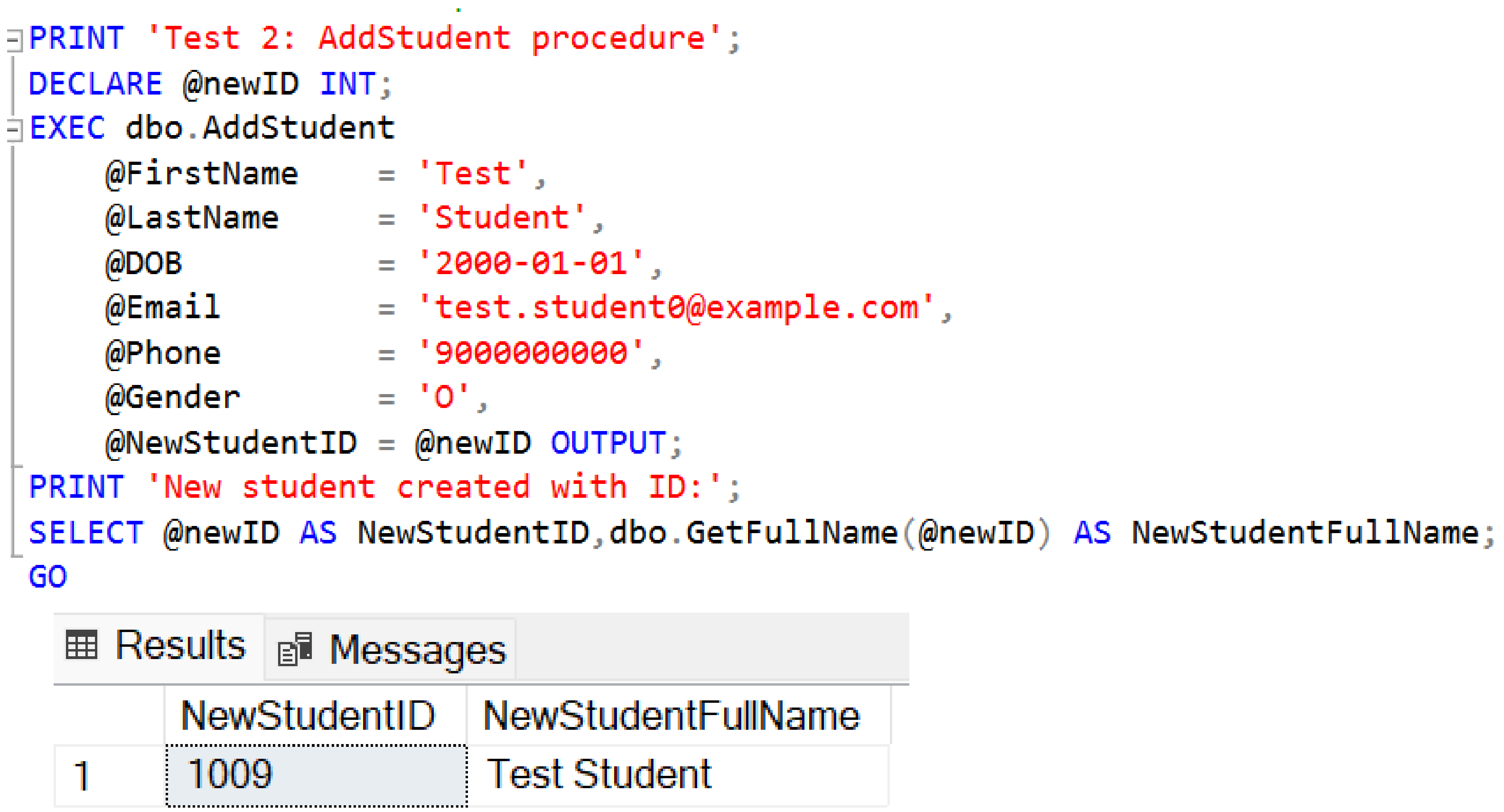
```sql
-- Test 1: Validate UDF GetFullName
PRINT 'Test 1: GetFullName for a known student';
DECLARE @s1 INT = (SELECT TOP 1 StudentID
                   FROM Students
                   WHERE Email='asha.patel@example.com');
SELECT dbo.GetFullName(@s1) AS FullNameFor_s1;
GO
```



Results | Messages

```
   FullNameFor_s1
1  Asha Patel
```

• AddStudent procedure

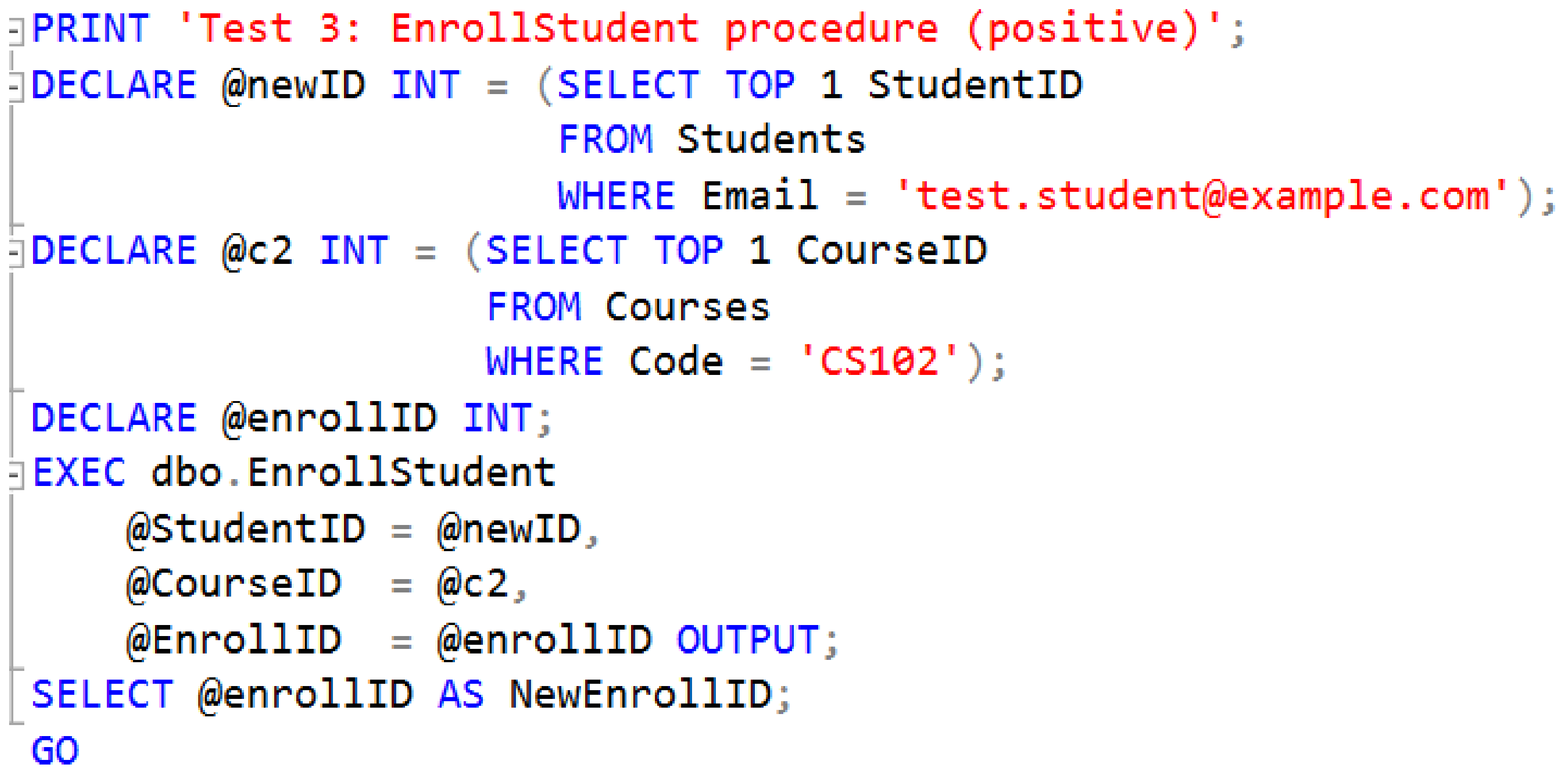


```
PRINT 'Test 2: AddStudent procedure';
DECLARE @newID INT;
EXEC dbo.AddStudent
    @FirstName    = 'Test',
    @LastName     = 'Student',
    @DOB          = '2000-01-01',
    @Email        = 'test.student0@example.com',
    @Phone        = '9000000000',
    @Gender       = 'O',
    @NewStudentID = @newID OUTPUT;
PRINT 'New student created with ID:';
SELECT @newID AS NewStudentID,dbo.GetFullName(@newID) AS NewStudentFullName;
GO
```

Results Messages

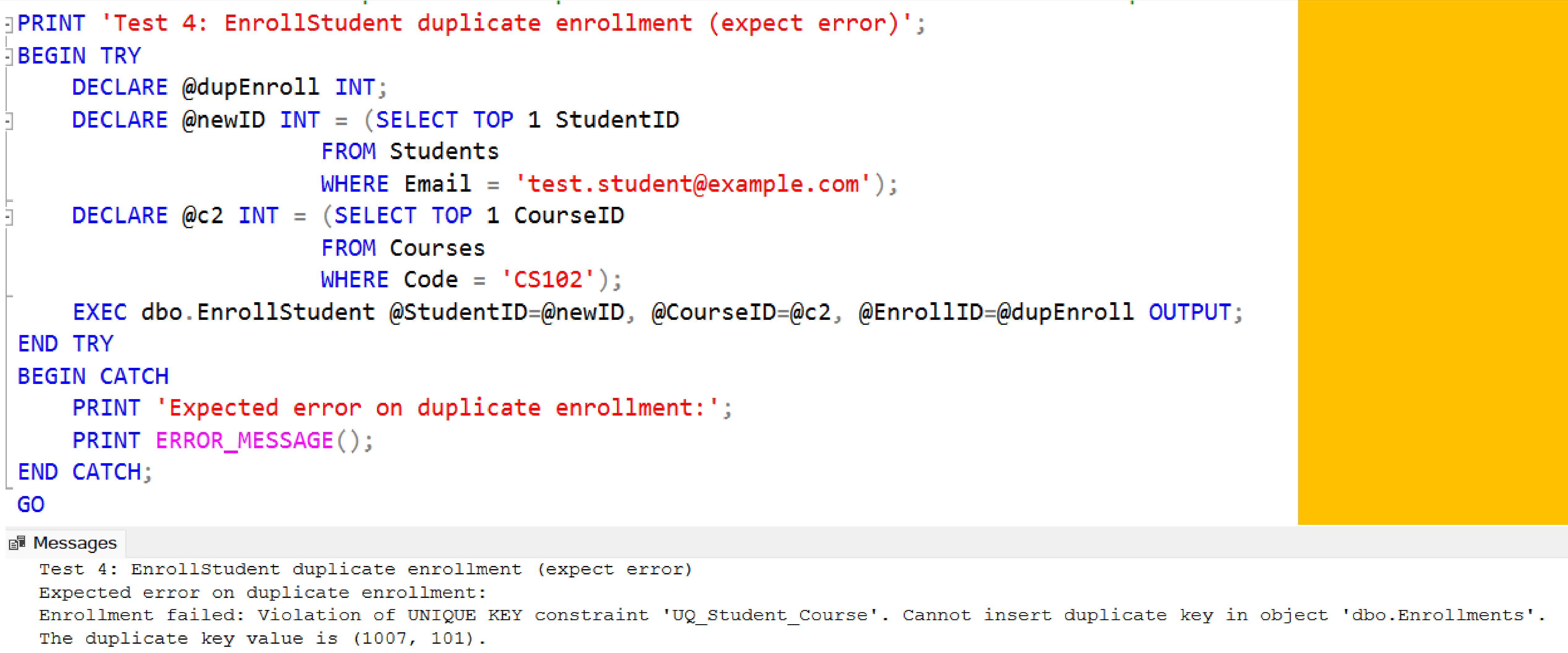
	NewStudentID	NewStudentFullName
1	1009	Test Student

- **EnrollStudent procedure - positive case**



```
PRINT 'Test 3: EnrollStudent procedure (positive)';
DECLARE @newID INT = (SELECT TOP 1 StudentID
                      FROM Students
                      WHERE Email = 'test.student@example.com');
DECLARE @c2 INT = (SELECT TOP 1 CourseID
                   FROM Courses
                   WHERE Code = 'CS102');
DECLARE @enrollID INT;
EXEC dbo.EnrollStudent
    @StudentID = @newID,
    @CourseID  = @c2,
    @EnrollID  = @enrollID OUTPUT;
SELECT @enrollID AS NewEnrollID;
GO
```

- **EnrollStudent procedure - duplicate enrollment should fail due to unique constraint**

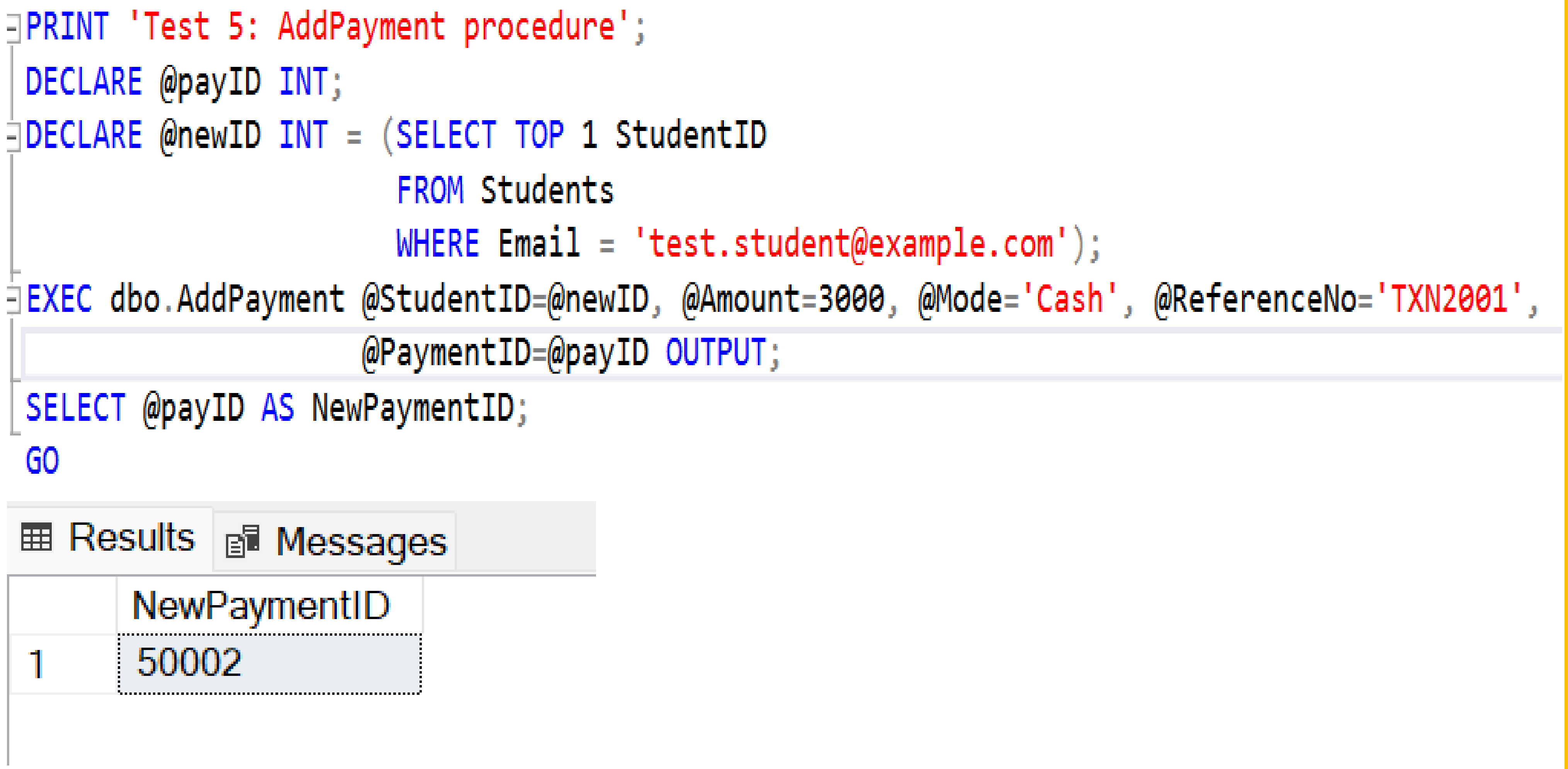


```
PRINT 'Test 4: EnrollStudent duplicate enrollment (expect error)';
BEGIN TRY
    DECLARE @dupEnroll INT;
    DECLARE @newID INT = (SELECT TOP 1 StudentID
                          FROM Students
                          WHERE Email = 'test.student@example.com');
    DECLARE @c2 INT = (SELECT TOP 1 CourseID
                          FROM Courses
                          WHERE Code = 'CS102');
    EXEC dbo.EnrollStudent @StudentID=@newID, @CourseID=@c2, @EnrollID=@dupEnroll OUTPUT;
END TRY
BEGIN CATCH
    PRINT 'Expected error on duplicate enrollment:';
    PRINT ERROR_MESSAGE();
END CATCH;
GO
```

Messages

```
Test 4: EnrollStudent duplicate enrollment (expect error)
Expected error on duplicate enrollment:
Enrollment failed: Violation of UNIQUE KEY constraint 'UQ_Student_Course'. Cannot insert duplicate key in object 'dbo.Enrollments'.
The duplicate key value is (1007, 101).
```

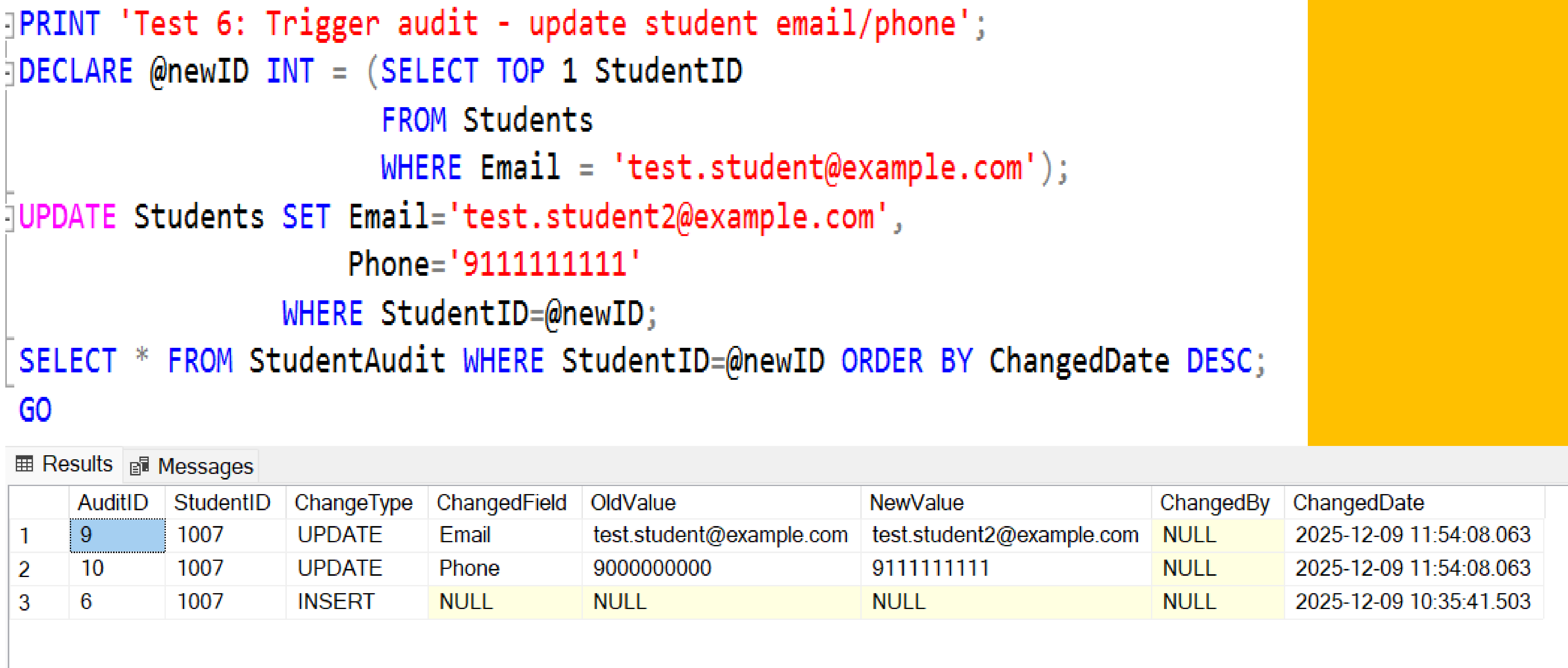
• AddPayment procedure



```
PRINT 'Test 5: AddPayment procedure';
DECLARE @payID INT;
DECLARE @newID INT = (SELECT TOP 1 StudentID
                      FROM Students
                      WHERE Email = 'test.student@example.com');
EXEC dbo.AddPayment @StudentID=@newID, @Amount=3000, @Mode='Cash', @ReferenceNo='TXN2001',
                    @PaymentID=@payID OUTPUT;
SELECT @payID AS NewPaymentID;
GO
```

```
   NewPaymentID
1  50002
```

• Trigger audit - update student email and phone, then check audit table

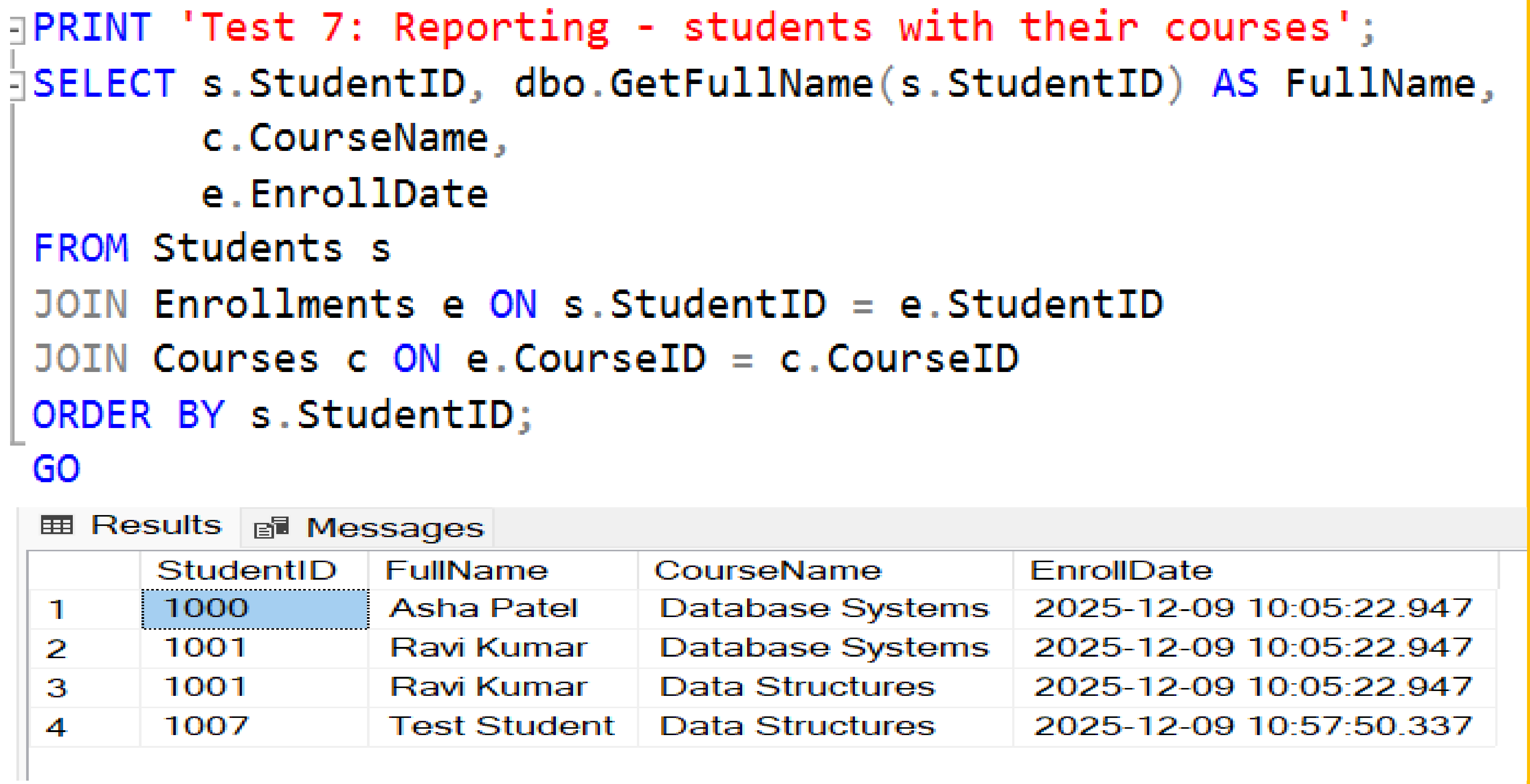


```sql
PRINT 'Test 6: Trigger audit - update student email/phone';
DECLARE @newID INT = (SELECT TOP 1 StudentID
                      FROM Students
                      WHERE Email = 'test.student@example.com');
UPDATE Students SET Email='test.student2@example.com',
                    Phone='9111111111'
                WHERE StudentID=@newID;
SELECT * FROM StudentAudit WHERE StudentID=@newID ORDER BY ChangedDate DESC;
GO
```

Results | Messages

	AuditID	StudentID	ChangeType	ChangedField	OldValue	NewValue	ChangedBy	ChangedDate
1	9	1007	UPDATE	Email	test.student@example.com	test.student2@example.com	NULL	2025-12-09 11:54:08.063
2	10	1007	UPDATE	Phone	9000000000	9111111111	NULL	2025-12-09 11:54:08.063
3	6	1007	INSERT	NULL	NULL	NULL	NULL	2025-12-09 10:35:41.503

- **Reporting query - students with their courses**

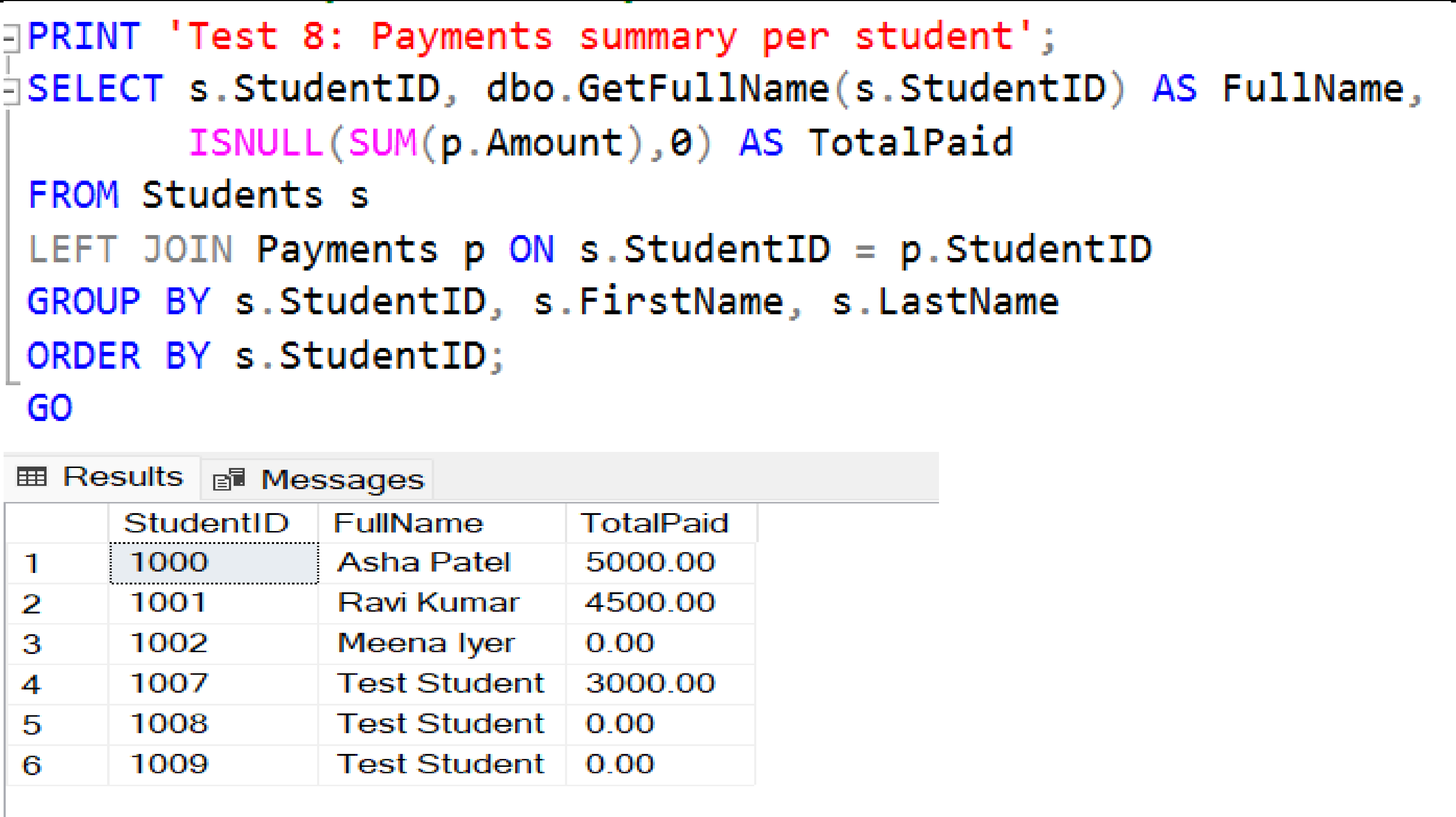


```
PRINT 'Test 7: Reporting - students with their courses';
SELECT s.StudentID, dbo.GetFullName(s.StudentID) AS FullName,
       c.CourseName,
       e.EnrollDate
FROM Students s
JOIN Enrollments e ON s.StudentID = e.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
ORDER BY s.StudentID;
GO
```

Results | Messages

	StudentID	FullName	CourseName	EnrollDate
1	1000	Asha Patel	Database Systems	2025-12-09 10:05:22.947
2	1001	Ravi Kumar	Database Systems	2025-12-09 10:05:22.947
3	1001	Ravi Kumar	Data Structures	2025-12-09 10:05:22.947
4	1007	Test Student	Data Structures	2025-12-09 10:57:50.337

• Payments summary



```
PRINT 'Test 8: Payments summary per student';
SELECT s.StudentID, dbo.GetFullName(s.StudentID) AS FullName,
       ISNULL(SUM(p.Amount),0) AS TotalPaid
FROM Students s
LEFT JOIN Payments p ON s.StudentID = p.StudentID
GROUP BY s.StudentID, s.FirstName, s.LastName
ORDER BY s.StudentID;
GO
```

Results | Messages

	StudentID	FullName	TotalPaid
1	1000	Asha Patel	5000.00
2	1001	Ravi Kumar	4500.00
3	1002	Meena Iyer	0.00
4	1007	Test Student	3000.00
5	1008	Test Student	0.00
6	1009	Test Student	0.00

THANK YOU