Dean H - Employee Performance Management System Design Document

Note - This project is not completed yet and I will still be working on this documentation as I produce the product. So there will be pseudocode to help structure the project and might change when I finish. But hopefully this will give you an idea of where I intend the project to go.

Introduction

The Employee Performance Management System is an ASP.NET Core MVC application designed to streamline the management of employee records and performance evaluations within an organization. By leveraging Microsoft SQL Server, Entity Framework Core, and SSRS, the system provides a comprehensive solution for tracking employee data, conducting performance reviews, and generating insightful reports. This application aims to enhance HR processes and decision-making by offering an intuitive interface for data management and analysis.

Key Features

- The Employee Records Management feature utilizes SQL Server to manage employee details and departmental assignments. This information is dynamically displayed on Razor pages, providing a user-friendly interface for viewing team details.

- The Performance Reviews feature utilizes SQL Server to record and manage employee performance evaluations. This information is dynamically displayed on Razor pages, providing a user-friendly interface for tracking and analyzing employee performance over time.

- The Department Management feature utilizes SQL Server to manage department details and performance metrics. This information is dynamically displayed on Razor pages, providing a user-friendly interface for monitoring and analyzing department performance.

- The Data Security and Role-Based Access Control feature ensures secure access to sensitive data by implementing role-based access control (RBAC). It utilizes SQL Server to enforce access permissions and maintain data integrity, adhering to proper database design principles. This ensures that only authorized users can view and modify sensitive data, promoting data integrity and consistency.

Tools and Technologies

ASP.NET Core MVC - For building the web application.
Entity Framework Core - For database operations.
Razor - For rendering dynamic HTML content.
SQL Server 2022 Developer Edition - For database management.
SQL Server Reporting Services (SSRS) - For creating and displaying reports.

System Architecture

The Employee Performance Management System follows a three-tier architecture, comprising the presentation layer, the application layer, and Data Layer. Each layer plays a distinct role in ensuring the functionality and performance of the system.

Presentation Layer

The presentation layer encompasses the user interface components responsible for displaying employee data, performance reviews, and department information. This layer includes web pages, views, and client-side scripts that facilitate user interaction with the application. Designed to be intuitive and user-friendly, the interface allows users to easily navigate through employee records and performance metrics.

Application Layer

Serving as the intermediary between the presentation layer and the data layer, the application layer handles business logic and data processing tasks. It comprises controllers, business logic components, and service layers responsible for processing user requests, fetching data from the data layer, and orchestrating the flow of information within the application. This layer ensures seamless integration of employee data and performance metrics retrieved from the data layer.

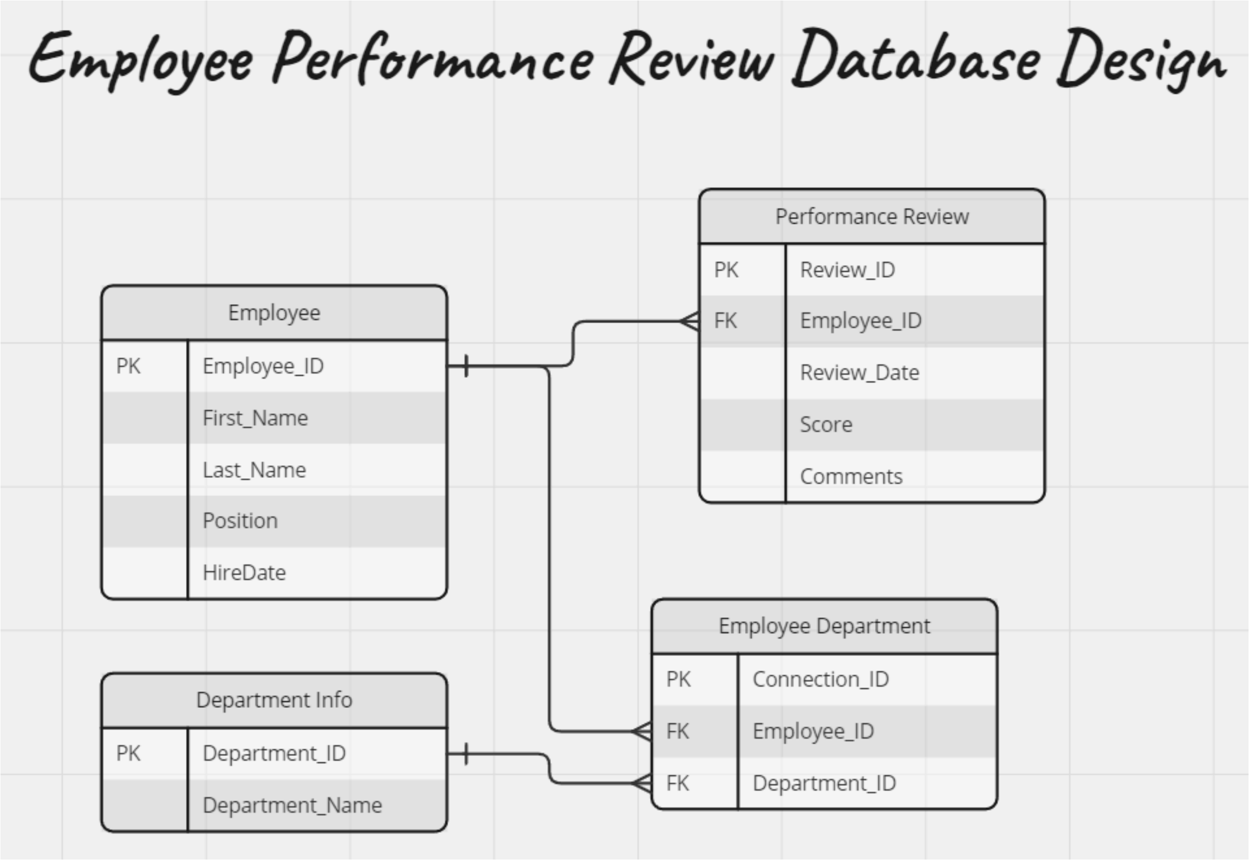
Data Layer

The data layer is responsible for storing and managing persistent data used by the application. It typically consists of a relational database management system (RDBMS), such as SQL Server, which stores employee records, performance reviews, department information, and other relevant data. The data layer ensures data integrity, consistency, and security through proper database design, including the definition of tables, relationships, and constraints. It also provides mechanisms for data retrieval, modification, and deletion, enabling efficient access to and manipulation of data by the application layer.

Scalability and Performance Considerations

The system architecture of the application is designed to be scalable and performant, capable of handling increasing loads and user interactions. Horizontal scalability is achieved through load balancing and auto-scaling mechanisms, allowing the application to dynamically allocate resources based on demand. Performance optimizations, such as efficient database queries and caching strategies, are implemented to minimize latency and improve responsiveness. Additionally, asynchronous processing and parallelization techniques are employed to enhance throughput and resource utilization, ensuring optimal performance under varying workloads.

Database Design



Employee Performance Review Database Design

Employee	
PK	Employee_ID
	First_Name
	Last_Name
	Position
	HireDate

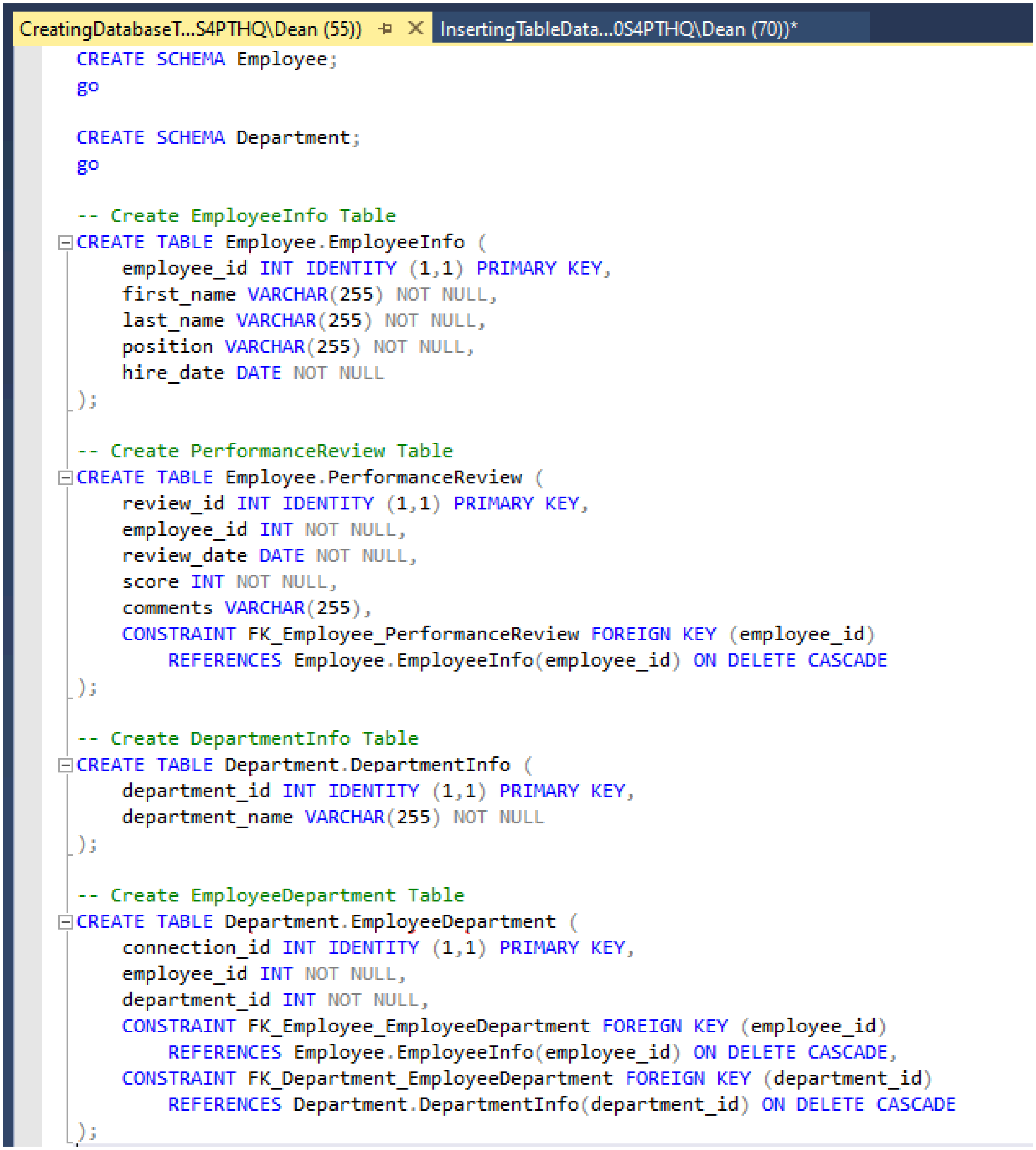
Performance Review	
PK	Review_ID
FK	Employee_ID
	Review_Date
	Score
	Comments

Department Info	
PK	Department_ID
	Department_Name

Employee Department	
PK	Connection_ID
FK	Employee_ID
FK	Department_ID

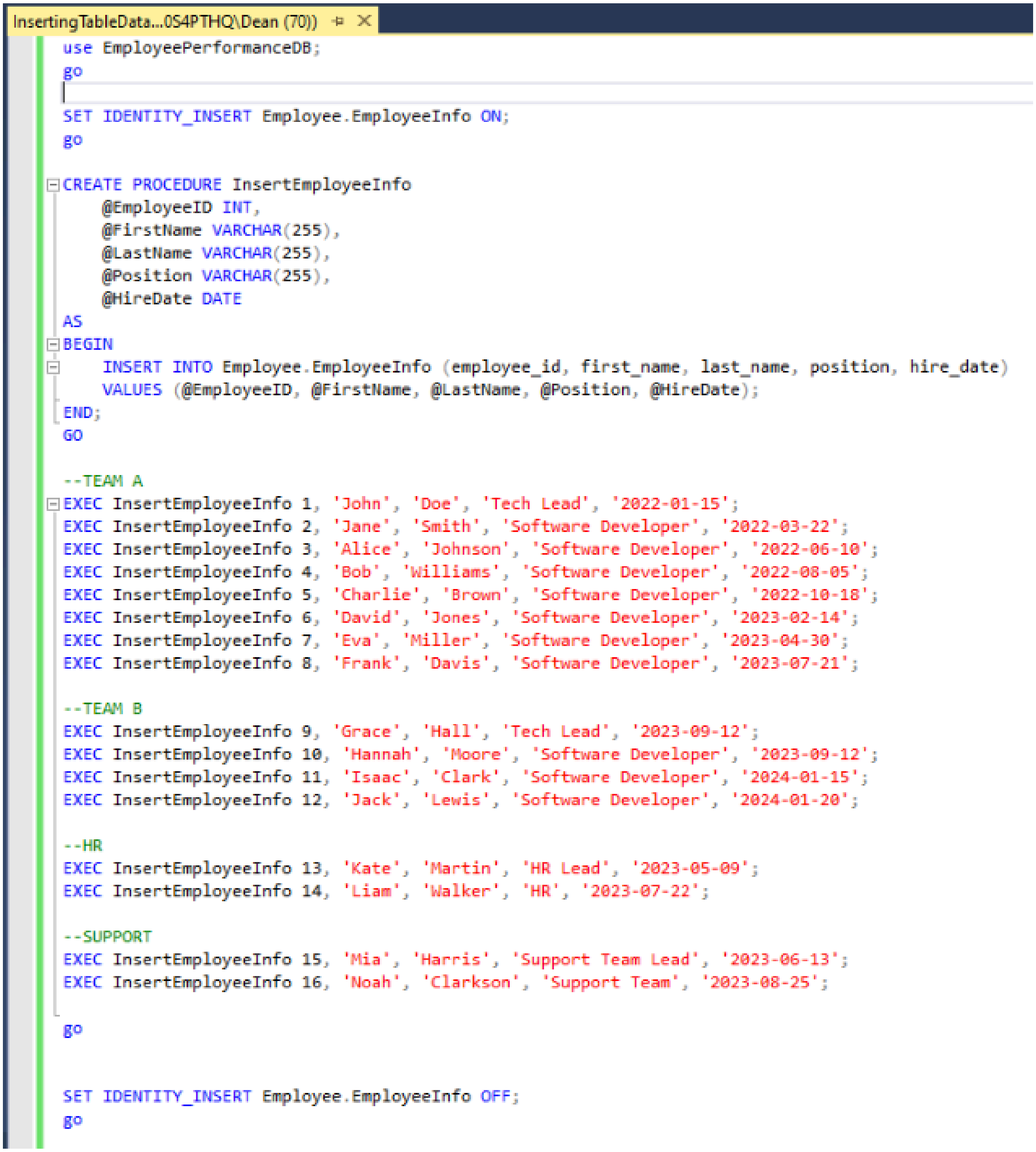
SQL Database Creation

Table Creation

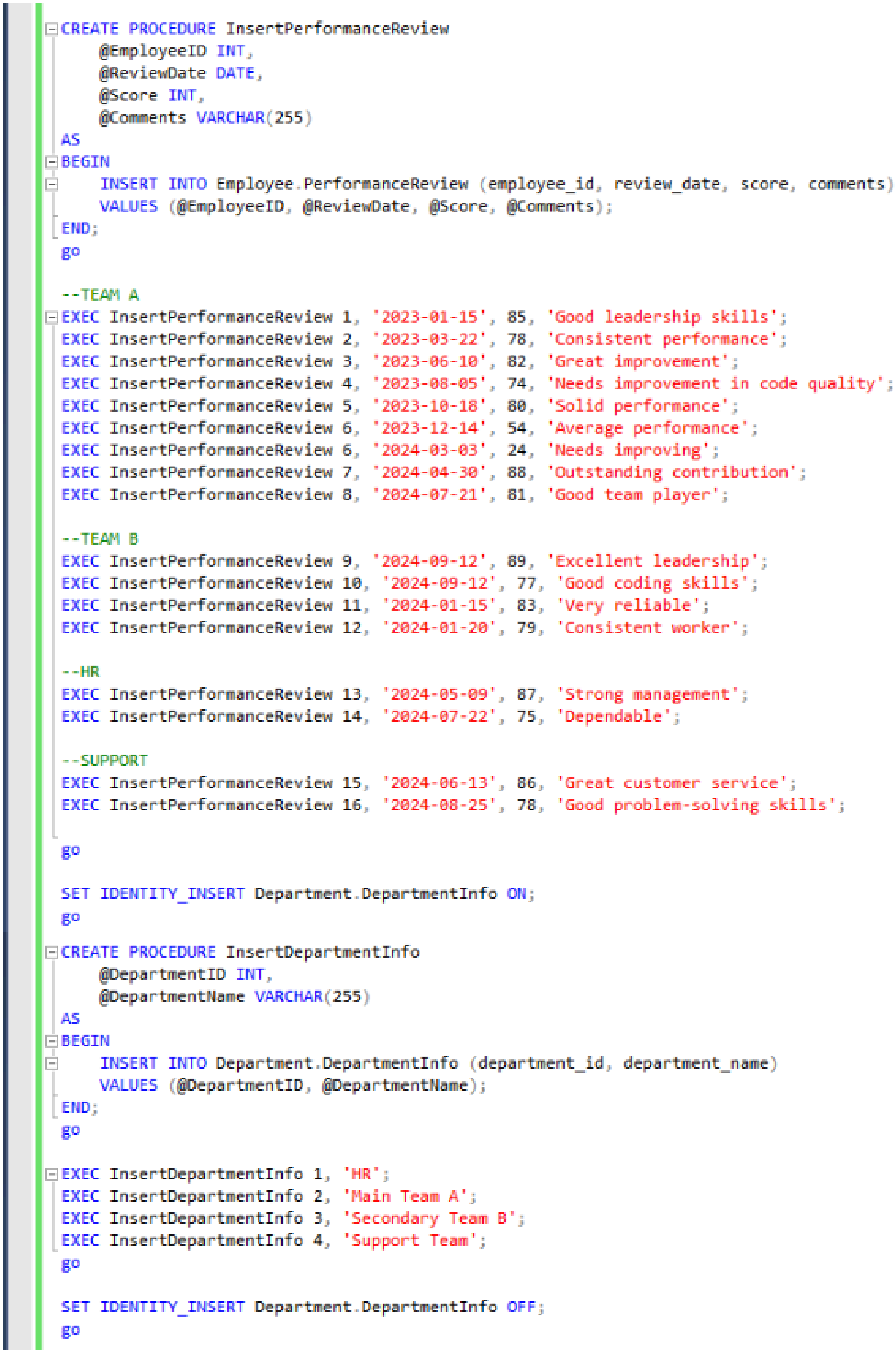

CreatingDatabaseT...S4PTHQ\Dean (55))
InsertingTableData...0S4PTHQ\Dean (70))*

```
CREATE SCHEMA Employee;
go

CREATE SCHEMA Department;
go

-- Create EmployeeInfo Table
CREATE TABLE Employee.EmployeeInfo (
    employee_id INT IDENTITY (1,1) PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    position VARCHAR(255) NOT NULL,
    hire_date DATE NOT NULL
);

-- Create PerformanceReview Table
CREATE TABLE Employee.PerformanceReview (
    review_id INT IDENTITY (1,1) PRIMARY KEY,
    employee_id INT NOT NULL,
    review_date DATE NOT NULL,
    score INT NOT NULL,
    comments VARCHAR(255),
    CONSTRAINT FK_Employee_PerformanceReview FOREIGN KEY (employee_id)
        REFERENCES Employee.EmployeeInfo(employee_id) ON DELETE CASCADE
);

-- Create DepartmentInfo Table
CREATE TABLE Department.DepartmentInfo (
    department_id INT IDENTITY (1,1) PRIMARY KEY,
    department_name VARCHAR(255) NOT NULL
);

-- Create EmployeeDepartment Table
CREATE TABLE Department.EmployeeDepartment (
    connection_id INT IDENTITY (1,1) PRIMARY KEY,
    employee_id INT NOT NULL,
    department_id INT NOT NULL,
    CONSTRAINT FK_Employee_EmployeeDepartment FOREIGN KEY (employee_id)
        REFERENCES Employee.EmployeeInfo(employee_id) ON DELETE CASCADE,
    CONSTRAINT FK_Department_EmployeeDepartment FOREIGN KEY (department_id)
        REFERENCES Department.DepartmentInfo(department_id) ON DELETE CASCADE
);
```

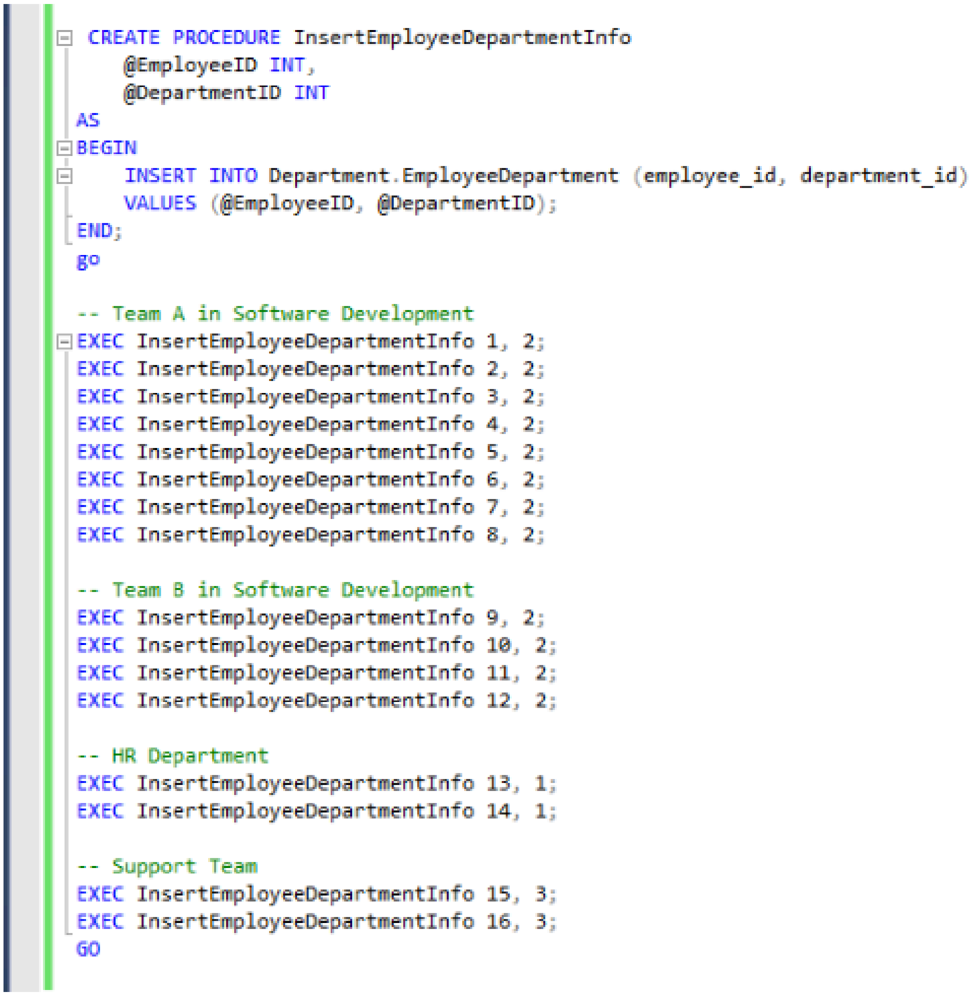
Inserting Data



```
use EmployeePerformanceDB;
go

SET IDENTITY_INSERT Employee.EmployeeInfo ON;
go

CREATE PROCEDURE InsertEmployeeInfo
    @EmployeeID INT,
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Position VARCHAR(255),
    @HireDate DATE
AS
BEGIN
    INSERT INTO Employee.EmployeeInfo (employee_id, first_name, last_name, position, hire_date)
    VALUES (@EmployeeID, @FirstName, @LastName, @Position, @HireDate);
END;
GO

--TEAM A
EXEC InsertEmployeeInfo 1, 'John', 'Doe', 'Tech Lead', '2022-01-15';
EXEC InsertEmployeeInfo 2, 'Jane', 'Smith', 'Software Developer', '2022-03-22';
EXEC InsertEmployeeInfo 3, 'Alice', 'Johnson', 'Software Developer', '2022-06-10';
EXEC InsertEmployeeInfo 4, 'Bob', 'Williams', 'Software Developer', '2022-08-05';
EXEC InsertEmployeeInfo 5, 'Charlie', 'Brown', 'Software Developer', '2022-10-18';
EXEC InsertEmployeeInfo 6, 'David', 'Jones', 'Software Developer', '2023-02-14';
EXEC InsertEmployeeInfo 7, 'Eva', 'Miller', 'Software Developer', '2023-04-30';
EXEC InsertEmployeeInfo 8, 'Frank', 'Davis', 'Software Developer', '2023-07-21';

--TEAM B
EXEC InsertEmployeeInfo 9, 'Grace', 'Hall', 'Tech Lead', '2023-09-12';
EXEC InsertEmployeeInfo 10, 'Hannah', 'Moore', 'Software Developer', '2023-09-12';
EXEC InsertEmployeeInfo 11, 'Isaac', 'Clark', 'Software Developer', '2024-01-15';
EXEC InsertEmployeeInfo 12, 'Jack', 'Lewis', 'Software Developer', '2024-01-20';

--HR
EXEC InsertEmployeeInfo 13, 'Kate', 'Martin', 'HR Lead', '2023-05-09';
EXEC InsertEmployeeInfo 14, 'Liam', 'Walker', 'HR', '2023-07-22';

--SUPPORT
EXEC InsertEmployeeInfo 15, 'Mia', 'Harris', 'Support Team Lead', '2023-06-13';
EXEC InsertEmployeeInfo 16, 'Noah', 'Clarkson', 'Support Team', '2023-08-25';

go


SET IDENTITY_INSERT Employee.EmployeeInfo OFF;
go
```



```
CREATE PROCEDURE InsertPerformanceReview
    @EmployeeID INT,
    @ReviewDate DATE,
    @Score INT,
    @Comments VARCHAR(255)
AS
BEGIN
    INSERT INTO Employee.PerformanceReview (employee_id, review_date, score, comments)
    VALUES (@EmployeeID, @ReviewDate, @Score, @Comments);
END;
go

--TEAM A
EXEC InsertPerformanceReview 1, '2023-01-15', 85, 'Good leadership skills';
EXEC InsertPerformanceReview 2, '2023-03-22', 78, 'Consistent performance';
EXEC InsertPerformanceReview 3, '2023-06-10', 82, 'Great improvement';
EXEC InsertPerformanceReview 4, '2023-08-05', 74, 'Needs improvement in code quality';
EXEC InsertPerformanceReview 5, '2023-10-18', 80, 'Solid performance';
EXEC InsertPerformanceReview 6, '2023-12-14', 54, 'Average performance';
EXEC InsertPerformanceReview 6, '2024-03-03', 24, 'Needs improving';
EXEC InsertPerformanceReview 7, '2024-04-30', 88, 'Outstanding contribution';
EXEC InsertPerformanceReview 8, '2024-07-21', 81, 'Good team player';

--TEAM B
EXEC InsertPerformanceReview 9, '2024-09-12', 89, 'Excellent leadership';
EXEC InsertPerformanceReview 10, '2024-09-12', 77, 'Good coding skills';
EXEC InsertPerformanceReview 11, '2024-01-15', 83, 'Very reliable';
EXEC InsertPerformanceReview 12, '2024-01-20', 79, 'Consistent worker';

--HR
EXEC InsertPerformanceReview 13, '2024-05-09', 87, 'Strong management';
EXEC InsertPerformanceReview 14, '2024-07-22', 75, 'Dependable';

--SUPPORT
EXEC InsertPerformanceReview 15, '2024-06-13', 86, 'Great customer service';
EXEC InsertPerformanceReview 16, '2024-08-25', 78, 'Good problem-solving skills';

go

SET IDENTITY_INSERT Department.DepartmentInfo ON;
go

CREATE PROCEDURE InsertDepartmentInfo
    @DepartmentID INT,
    @DepartmentName VARCHAR(255)
AS
BEGIN
    INSERT INTO Department.DepartmentInfo (department_id, department_name)
    VALUES (@DepartmentID, @DepartmentName);
END;
go

EXEC InsertDepartmentInfo 1, 'HR';
EXEC InsertDepartmentInfo 2, 'Main Team A';
EXEC InsertDepartmentInfo 3, 'Secondary Team B';
EXEC InsertDepartmentInfo 4, 'Support Team';
go

SET IDENTITY_INSERT Department.DepartmentInfo OFF;
go
```

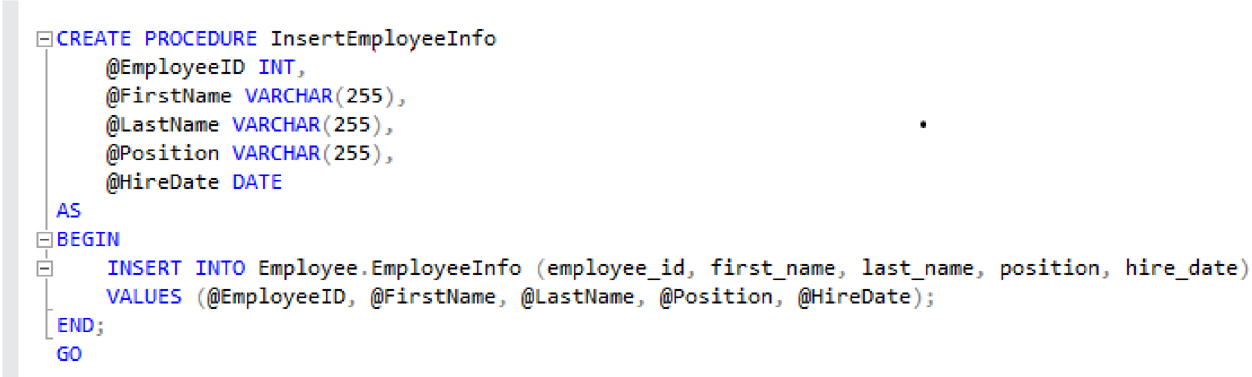


```sql
 CREATE PROCEDURE InsertEmployeeDepartmentInfo
    @EmployeeID INT,
    @DepartmentID INT
AS
BEGIN
    INSERT INTO Department.EmployeeDepartment (employee_id, department_id)
    VALUES (@EmployeeID, @DepartmentID);
END;
go

-- Team A in Software Development
EXEC InsertEmployeeDepartmentInfo 1, 2;
EXEC InsertEmployeeDepartmentInfo 2, 2;
EXEC InsertEmployeeDepartmentInfo 3, 2;
EXEC InsertEmployeeDepartmentInfo 4, 2;
EXEC InsertEmployeeDepartmentInfo 5, 2;
EXEC InsertEmployeeDepartmentInfo 6, 2;
EXEC InsertEmployeeDepartmentInfo 7, 2;
EXEC InsertEmployeeDepartmentInfo 8, 2;

-- Team B in Software Development
EXEC InsertEmployeeDepartmentInfo 9, 2;
EXEC InsertEmployeeDepartmentInfo 10, 2;
EXEC InsertEmployeeDepartmentInfo 11, 2;
EXEC InsertEmployeeDepartmentInfo 12, 2;

-- HR Department
EXEC InsertEmployeeDepartmentInfo 13, 1;
EXEC InsertEmployeeDepartmentInfo 14, 1;

-- Support Team
EXEC InsertEmployeeDepartmentInfo 15, 3;
EXEC InsertEmployeeDepartmentInfo 16, 3;
GO
```

Stored Procedures SQL

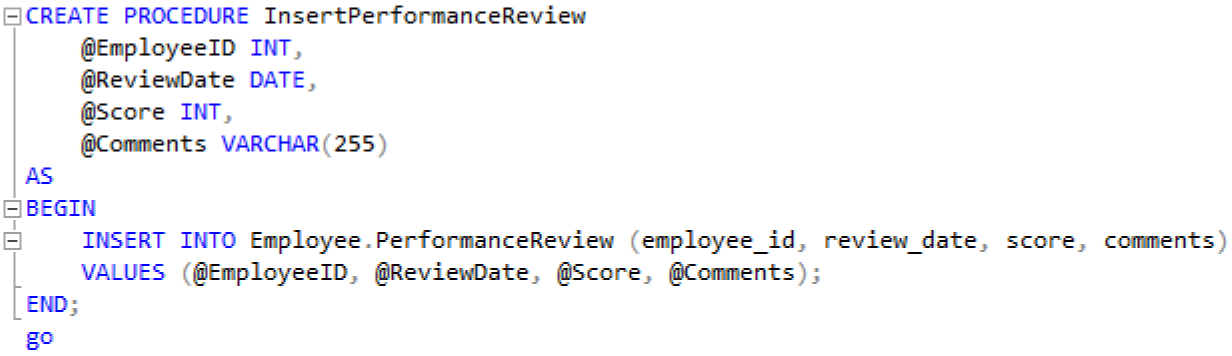
Insert Employee Info

First time setting up the database I needed access to the primary key.



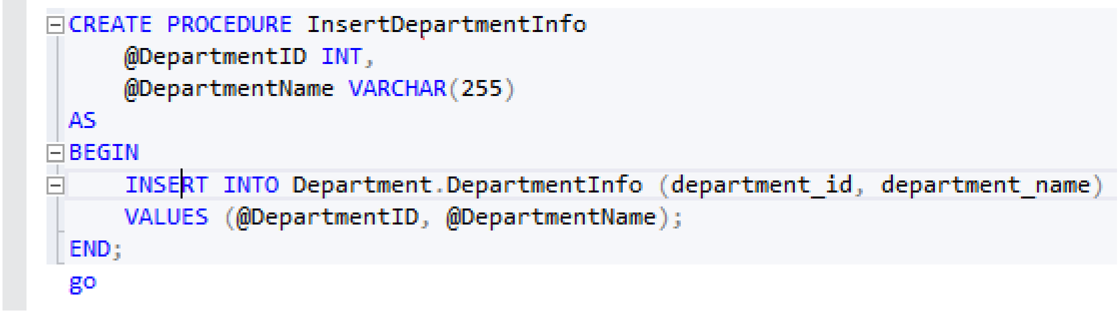
```
CREATE PROCEDURE InsertEmployeeInfo
    @EmployeeID INT,
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Position VARCHAR(255),
    @HireDate DATE
AS
BEGIN
    INSERT INTO Employee.EmployeeInfo (employee_id, first_name, last_name, position, hire_date)
    VALUES (@EmployeeID, @FirstName, @LastName, @Position, @HireDate);
END;
GO
```

Insert Performance Review



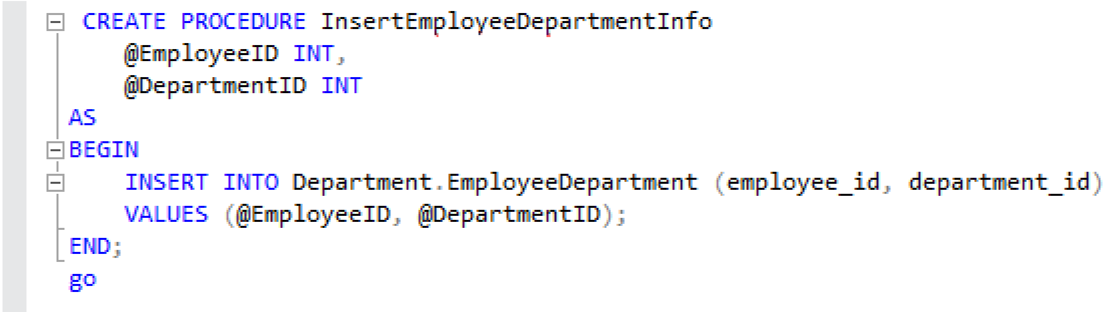
```
CREATE PROCEDURE InsertPerformanceReview
    @EmployeeID INT,
    @ReviewDate DATE,
    @Score INT,
    @Comments VARCHAR(255)
AS
BEGIN
    INSERT INTO Employee.PerformanceReview (employee_id, review_date, score, comments)
    VALUES (@EmployeeID, @ReviewDate, @Score, @Comments);
END;
go
```

Insert Department Info



```
CREATE PROCEDURE InsertDepartmentInfo
    @DepartmentID INT,
    @DepartmentName VARCHAR(255)
AS
BEGIN
    INSERT INTO Department.DepartmentInfo (department_id, department_name)
    VALUES (@DepartmentID, @DepartmentName);
END;
go
```

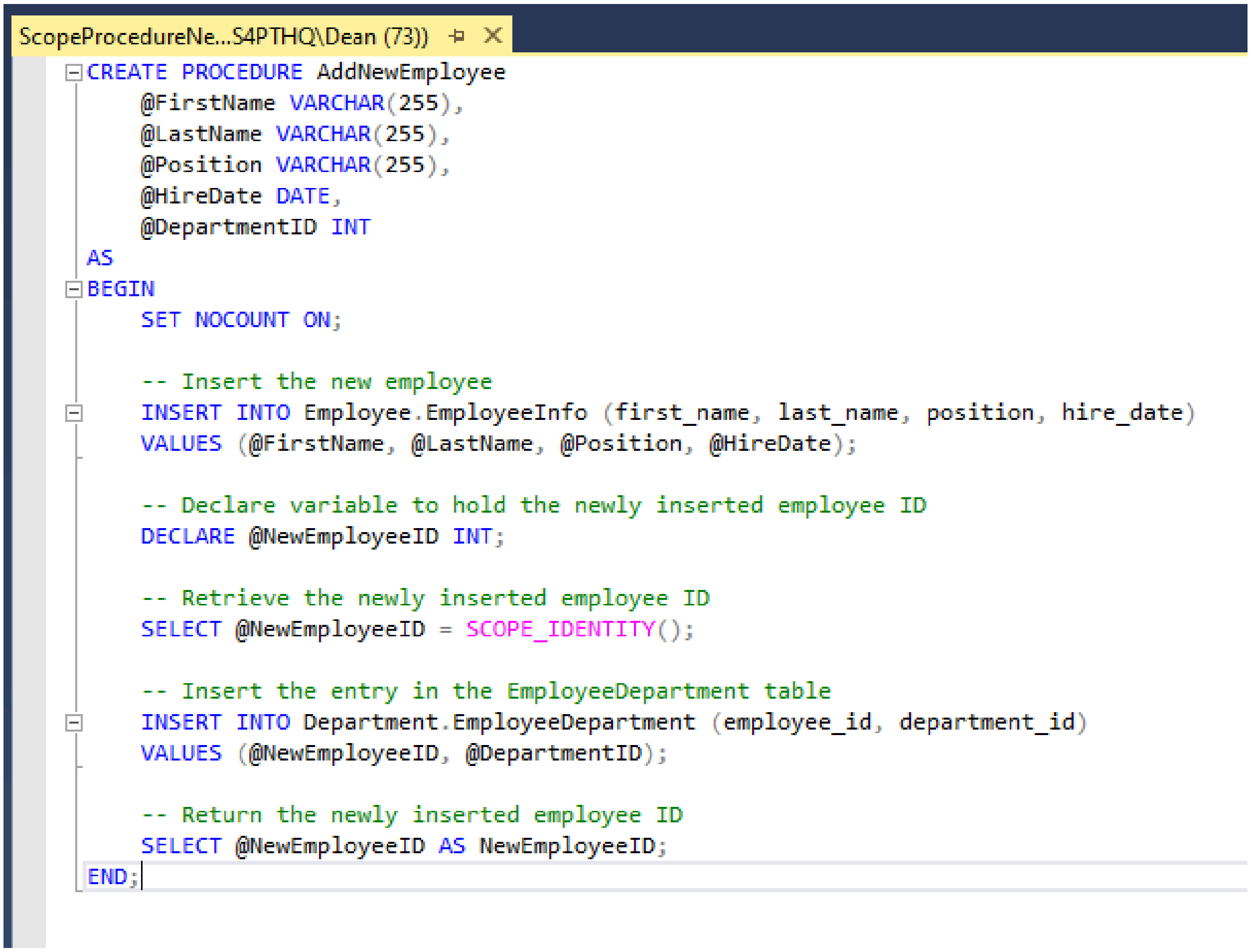
Insert Employee Department Info



```
CREATE PROCEDURE InsertEmployeeDepartmentInfo
    @EmployeeID INT,
    @DepartmentID INT
AS
BEGIN
    INSERT INTO Department.EmployeeDepartment (employee_id, department_id)
    VALUES (@EmployeeID, @DepartmentID);
END;
go
```

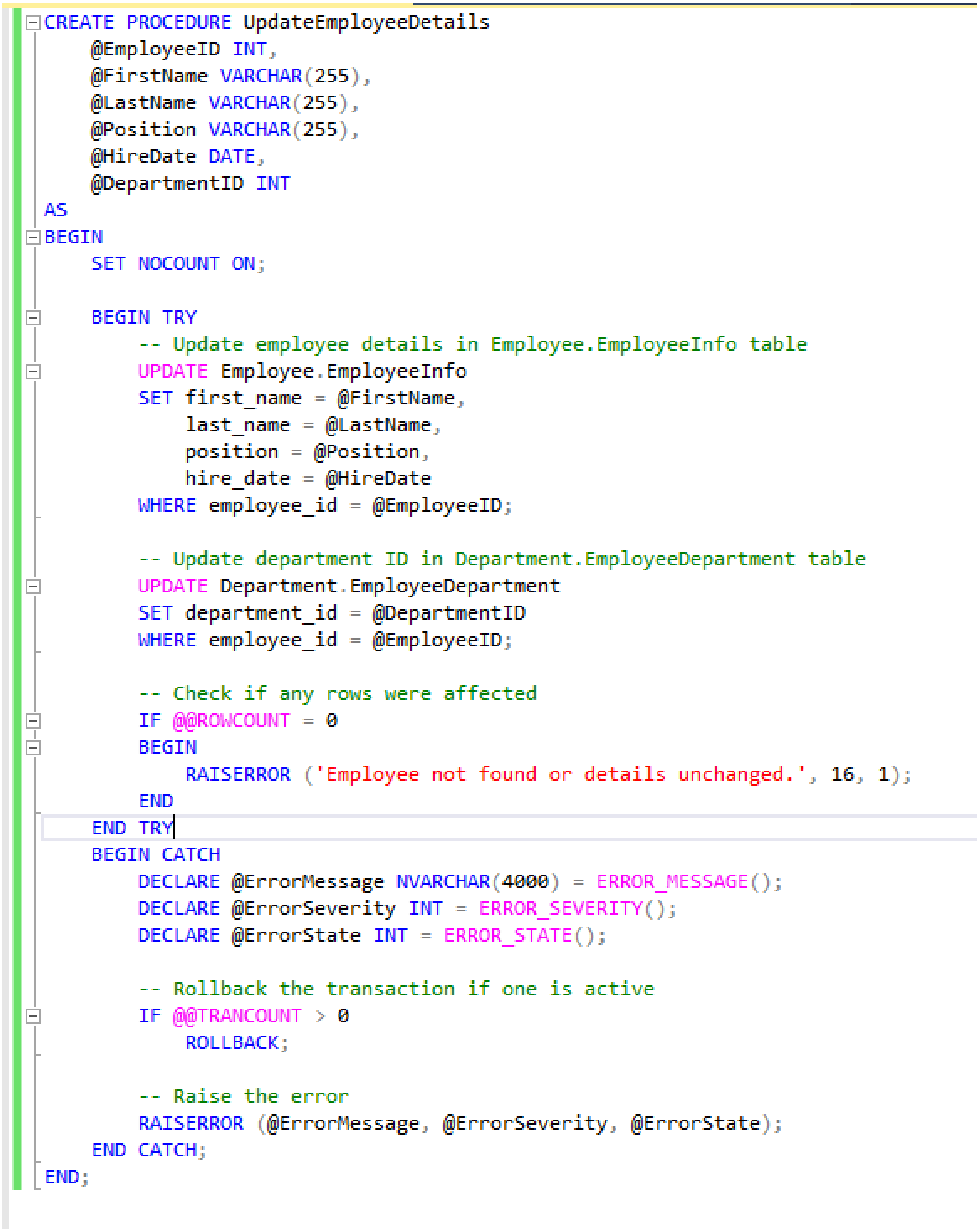
Add New Employee

When adding a new employee I need to make it add entries to both the EmployeeInfo and the EmployeeDepartment. To create an new employee entry and connect it to the department.



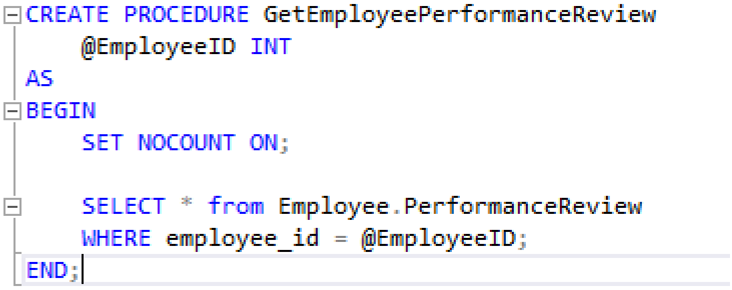
```
CREATE PROCEDURE AddNewEmployee
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Position VARCHAR(255),
    @HireDate DATE,
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Insert the new employee
    INSERT INTO Employee.EmployeeInfo (first_name, last_name, position, hire_date)
    VALUES (@FirstName, @LastName, @Position, @HireDate);

    -- Declare variable to hold the newly inserted employee ID
    DECLARE @NewEmployeeID INT;

    -- Retrieve the newly inserted employee ID
    SELECT @NewEmployeeID = SCOPE_IDENTITY();

    -- Insert the entry in the EmployeeDepartment table
    INSERT INTO Department.EmployeeDepartment (employee_id, department_id)
    VALUES (@NewEmployeeID, @DepartmentID);

    -- Return the newly inserted employee ID
    SELECT @NewEmployeeID AS NewEmployeeID;
END;
```

Update Employee Details



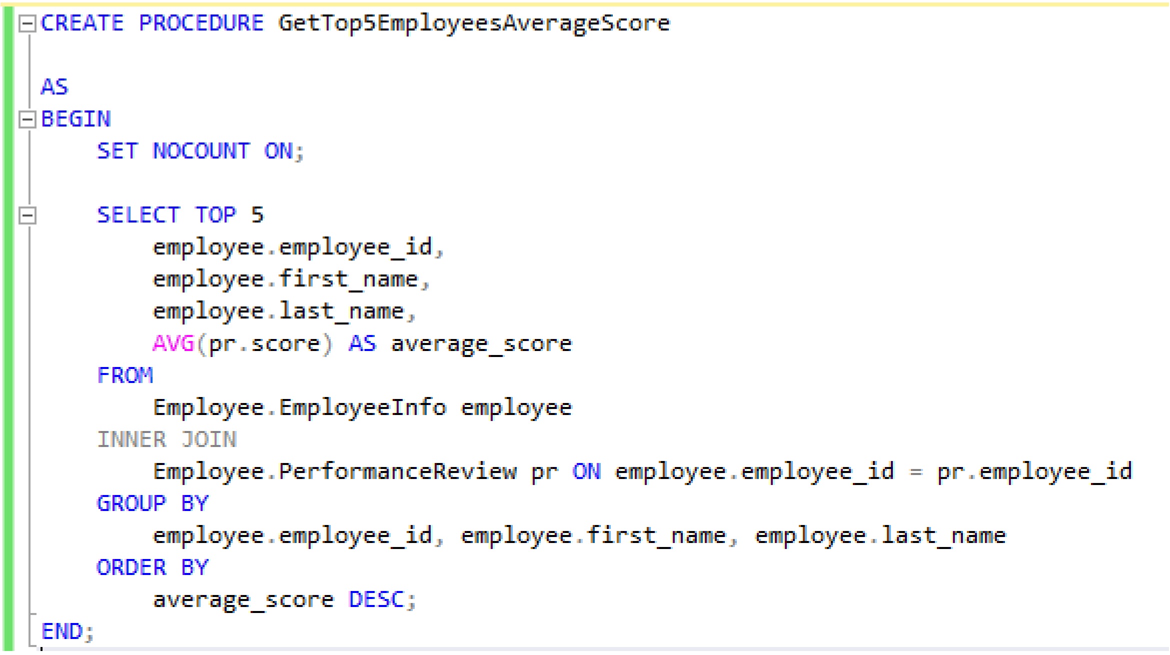
```
CREATE PROCEDURE UpdateEmployeeDetails
    @EmployeeID INT,
    @FirstName VARCHAR(255),
    @LastName VARCHAR(255),
    @Position VARCHAR(255),
    @HireDate DATE,
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Update employee details in Employee.EmployeeInfo table
        UPDATE Employee.EmployeeInfo
        SET first_name = @FirstName,
            last_name = @LastName,
            position = @Position,
            hire_date = @HireDate
        WHERE employee_id = @EmployeeID;

        -- Update department ID in Department.EmployeeDepartment table
        UPDATE Department.EmployeeDepartment
        SET department_id = @DepartmentID
        WHERE employee_id = @EmployeeID;

        -- Check if any rows were affected
        IF @@ROWCOUNT = 0
        BEGIN
            RAISERROR ('Employee not found or details unchanged.', 16, 1);
        END
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        -- Rollback the transaction if one is active
        IF @@TRANCOUNT > 0
            ROLLBACK;

        -- Raise the error
        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH;
END;
```

Get Employee Performance Reviews



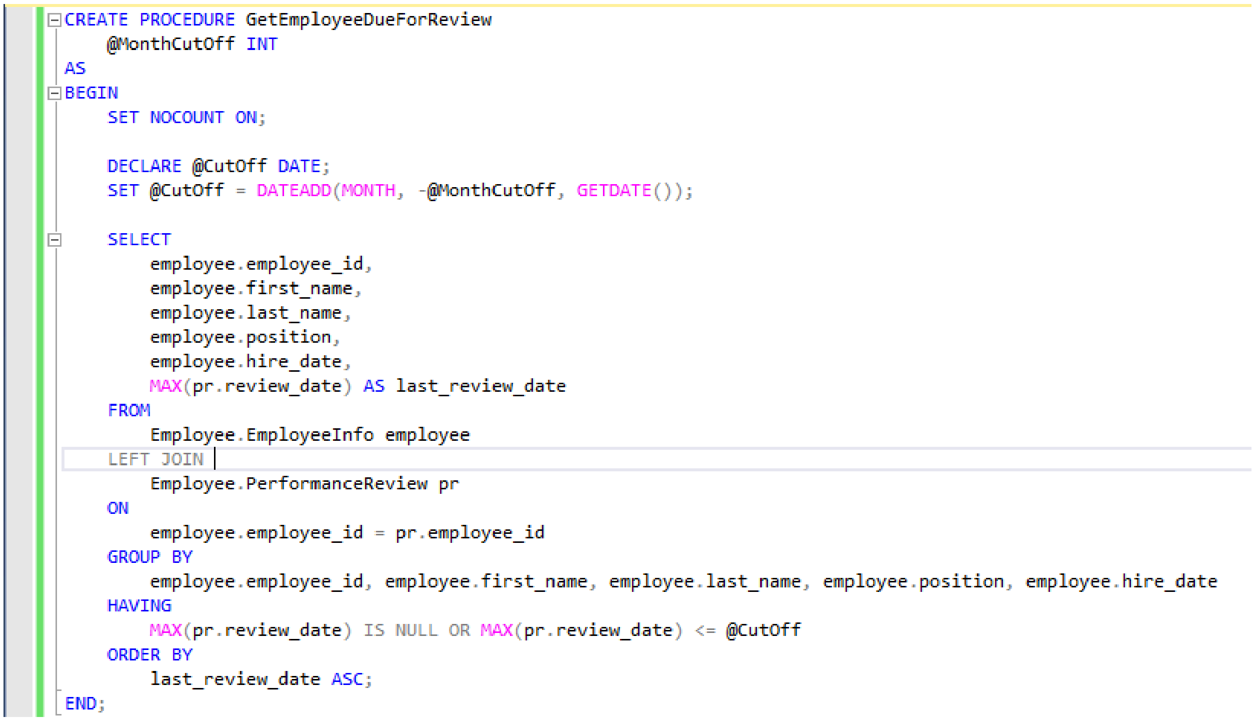
```sql
CREATE PROCEDURE GetEmployeePerformanceReview
    @EmployeeID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT * from Employee.PerformanceReview
    WHERE employee_id = @EmployeeID;
END;
```

Get Top 5 Employees



```sql
CREATE PROCEDURE GetTop5EmployeesAverageScore

AS
BEGIN
    SET NOCOUNT ON;

    SELECT TOP 5
        employee.employee_id,
        employee.first_name,
        employee.last_name,
        AVG(pr.score) AS average_score
    FROM
        Employee.EmployeeInfo employee
    INNER JOIN
        Employee.PerformanceReview pr ON employee.employee_id = pr.employee_id
    GROUP BY
        employee.employee_id, employee.first_name, employee.last_name
    ORDER BY
        average_score DESC;
END;
```

Get Employees That Need a Performance Review



```sql
CREATE PROCEDURE GetEmployeeDueForReview
    @MonthCutOff INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CutOff DATE;
    SET @CutOff = DATEADD(MONTH, -@MonthCutOff, GETDATE());

    SELECT
        employee.employee_id,
        employee.first_name,
        employee.last_name,
        employee.position,
        employee.hire_date,
        MAX(pr.review_date) AS last_review_date
    FROM
        Employee.EmployeeInfo employee
    LEFT JOIN
        Employee.PerformanceReview pr
    ON
        employee.employee_id = pr.employee_id
    GROUP BY
        employee.employee_id, employee.first_name, employee.last_name, employee.position, employee.hire_date
    HAVING
        MAX(pr.review_date) IS NULL OR MAX(pr.review_date) <= @CutOff
    ORDER BY
        last_review_date ASC;
END;
```

Components

The