Complete SQL and EF Core Setup Guide for C# Developers

A comprehensive guide covering development environment setup, SQL fundamentals, and Entity Framework Core implementation for C# developers targeting Azure SQL Database.

Table of Contents

1. Development Environment Setup
2. SQL Server Installation
3. Azure SQL Database Setup
4. Visual Studio and .NET Setup
5. Entity Framework Core Installation
6. Database Tools Installation
7. SQL Fundamentals
8. EF Core Implementation
9. Azure Deployment
10. Troubleshooting and Best Practices

Part 1: Development Environment Setup

System Requirements

Minimum Hardware Requirements:

- 4 GB RAM (8 GB recommended)
- x64 Processor with 1.4 GHz (2 GHz or higher recommended)
- 20 GB available hard disk space
- Super VGA 800×600 monitor or better
- Internet connection for downloads and Azure services

Supported Operating Systems:

- Windows 10 version 1809 or later
- Windows 11 (all versions)
- Windows Server 2016 or later

Prerequisites Checklist:

- [ ] Administrator privileges on the development machine
- [ ] Stable internet connection

- [ ] Microsoft account for Azure services
- [ ] Valid Azure subscription (free tier available)

Part 2: SQL Server Installation

Option 1: SQL Server 2022 Developer Edition (Recommended)

Step 1: Download SQL Server 2022

1. Visit the official Microsoft SQL Server download page: `https://www.microsoft.com/en-us/sql-server/sql-server-downloads`
2. Click on **"Developer"** under the free editions section
3. Download the installer (approximately 4.2 MB launcher)

Step 2: Run the Installation

1. Double-click the downloaded `SQL2022-SSEI-Dev.exe` file
2. Choose installation type:
   - **Basic**: Default configuration (quick setup)
   - **Custom**: Full control over features and settings
   - **Download Media**: Download full installer for offline installation
3. **Recommended**: Choose **Custom** for learning purposes

Step 3: Feature Selection

Select the following features for development:

- [ ] Database Engine Services (Required)
- [ ] SQL Server Replication
- [ ] Full-Text and Semantic Extractions for Search
- [ ] Data Quality Services
- [ ] Analysis Services
- [ ] Reporting Services - Native
- [ ] Integration Services
- [ ] Management Tools - Complete

Step 4: Instance Configuration

```
Instance Configuration:
- Instance Type: Default instance
- Instance Name: MSSQLSERVER (default)
- Instance ID: MSSQLSERVER
- Instance root directory: C:\Program Files\Microsoft SQL Server\
```

Step 5: Server Configuration

```
Service Accounts:
- SQL Server Database Engine: NT AUTHORITY\SYSTEM
- SQL Server Agent: NT AUTHORITY\SYSTEM

Authentication Mode:
- Mixed Mode (SQL Server Authentication and Windows Authentication)
- Set strong SA password: [YourStrongPassword123!]
- Add Current User as SQL Server Administrator
```

Step 6: Database Engine Configuration

```
Data Directories:
- Data root directory: C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Data
- User database directory: C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\
- Log directory: C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Data
- Backup directory: C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Backup
```

Step 7: Complete Installation

1. Review configuration summary
2. Click **Install** (installation takes 15-30 minutes)
3. Note the installation completion status
4. Record instance names and configuration details

Option 2: SQL Server 2025 Preview (Latest Features)

Download and Setup:

1. Visit: `https://www.microsoft.com/sql-server/sql-server-2025`
2. Fill registration form for preview access
3. Download `SQL2025-SSEI-Eval.exe`
4. Follow similar installation steps as SQL Server 2022

Key Differences in SQL Server 2025:

- Enhanced AI integration capabilities
- Improved performance optimizations
- New security features
- Better cloud integration

Post-Installation Verification

Test SQL Server Installation:

1. Open **SQL Server Configuration Manager**
2. Verify services are running:
    - SQL Server (MSSQLSERVER): Running
    - SQL Server Agent (MSSQLSERVER): Running

3. Test connection using Command Prompt:

```
sqlcmd -S localhost -E
SELECT @@VERSION;
GO
EXIT
```

Part 3: Azure SQL Database Setup

Creating Azure Account and Subscription

Step 1: Azure Account Setup

1. Visit `https://portal.azure.com`
2. Sign up for free Azure account (includes $200 credit)
3. Verify email and phone number
4. Complete identity verification

Step 2: Navigate to SQL Database Service

1. Log into Azure Portal
2. Search for **"SQL databases"** in the top search bar
3. Click **"Create SQL database"**

Creating Azure SQL Database

Step 3: Basic Configuration

```
Project Details:
- Subscription: [Your subscription name]
- Resource Group: Create new "rg-sql-learning-dev"

Database Details:
- Database name: "LearningSQLDB"
- Server: Create new server
```

Step 4: Server Configuration

```
Server Details:
- Server name: "learningsql-server-[unique-suffix]"
- Location: Select nearest region (e.g., East US, West Europe)
- Authentication method: Use SQL authentication

Admin Credentials:
- Server admin login: "sqladmin"
- Password: [Strong password with special characters]
- Confirm password: [Same password]
```

Step 5: Compute and Storage Configuration

```
Service Tier Options:
Development/Testing:
- Basic: 5 DTU, 2GB storage ($5/month)
- Standard S0: 10 DTU, 250GB storage ($15/month)

Production:
- Standard S1: 20 DTU, 250GB storage ($30/month)
- Premium P1: 125 DTU, 500GB storage ($465/month)

Recommended for Learning: Basic or Standard S0
```

Step 6: Additional Settings

```
Data Source:
- Use existing data: None
- Sample: AdventureWorksLT (recommended for learning)
- Backup: None

Networking:
- Connectivity method: Public endpoint
- Allow Azure services: Yes
- Add current client IP: Yes

Security:
- Enable Microsoft Defender for SQL: No (for learning)
- Ledger: Off
```

Step 7: Review and Create

1. Review all configuration settings
2. Add tags if needed:
    - Environment: Development
    - Purpose: Learning
3. Click **"Create"** (deployment takes 5-10 minutes)

Post-Creation Configuration

Step 8: Configure Firewall Rules

1. Navigate to your SQL server in Azure Portal
2. Go to **"Security" > "Networking"**
3. Add firewall rules:
    - Rule name: "MyHomeIP"
    - Start IP: [Your public IP]
    - End IP: [Your public IP]
4. Enable **"Allow Azure services and resources to access this server"**

Step 9: Test Connection

Connection string format:

```
Server=tcp:[your-server-name].database.windows.net,1433;
Initial Catalog=[database-name];
Persist Security Info=False;
User ID=[admin-username];
Password=[admin-password];
MultipleActiveResultSets=False;
Encrypt=True;
TrustServerCertificate=False;
Connection Timeout=30;
```

Part 4: Visual Studio and .NET Setup

Visual Studio 2022 Installation

Step 1: Download Visual Studio

1. Visit `https://visualstudio.microsoft.com/downloads/`
2. Choose **Visual Studio 2022 Community** (free)
3. Download the installer

Step 2: Workload Selection

Select these workloads during installation:

- [ ] ASP.NET **and web development**
- [ ] **.NET desktop development**
- [ ] **Data storage and processing**
- [ ] **Azure development**

Individual Components to Add:

- [ ] SQL Server Data Tools (SSDT)
- [ ] Entity Framework 6 tools
- [ ] NuGet package manager
- [ ] Git for Windows
- [ ] GitHub Copilot (optional)

Step 3: Complete Installation

Installation time: 30-60 minutes depending on selected components

.NET 8 SDK Verification

Check .NET Installation:

Open Command Prompt or PowerShell:

```
dotnet --version
dotnet --list-sdks
dotnet --list-runtimes
```

Expected output should show .NET 8.0 or later.

If .NET 8 is Missing:

1. Visit `https://dotnet.microsoft.com/download`
2. Download .NET 8 SDK
3. Run installer with administrator privileges

VS Code Setup (Alternative)

If Using VS Code:

1. Download from `https://code.visualstudio.com/`
2. Install essential extensions:
    - C# Dev Kit
    - SQL Server (mssql)
    - Azure Account
    - Azure Databases
    - NuGet Package Manager

Part 5: Entity Framework Core Installation

Project Creation and EF Core Setup

Step 1: Create New .NET Project

Using Visual Studio:

1. File → New → Project
2. Select "ASP.NET Core Web API"
3. Project name: "LearningEFCore"
4. Framework: .NET 8.0
5. Authentication: None (for learning)

Using CLI:

```
dotnet new webapi -n LearningEFCore
cd LearningEFCore
```

Step 2: Install EF Core Packages

Via Package Manager Console (Visual Studio):

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.Design
```

Via .NET CLI:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Via NuGet Package Manager (Visual Studio):

1. Right-click project → Manage NuGet Packages
2. Browse tab → Search "Microsoft.EntityFrameworkCore.SqlServer"
3. Install latest stable version
4. Repeat for Tools and Design packages

Step 3: Verify Installation

Check your `.csproj` file should contain:

```
&lt;PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0" /&gt
&lt;PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.0" /&gt;
&lt;PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="8.0.0" /&gt;
```

EF Core Tools Installation

Global Tools Installation:

```
dotnet tool install --global dotnet-ef
```

Verify EF Tools:

```
dotnet ef --version
```

Update EF Tools (if needed):

```
dotnet tool update --global dotnet-ef
```

Part 6: Database Tools Installation

SQL Server Management Studio (SSMS)

Step 1: Download SSMS

1. Visit: `https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms`
2. Download latest version (SSMS-Setup-ENU.exe)
3. Current version: 20.2 or later

Step 2: Install SSMS

1. Run installer as administrator
2. Choose installation location (default recommended)
3. Installation time: 10-15 minutes
4. Restart if prompted

Step 3: Configure SSMS for Local SQL Server

1. Launch SSMS
2. Connect to Database Engine:
   - Server name: `localhost` or `.\MSSQLSERVER`
   - Authentication: Windows Authentication
   - Click Connect

Step 4: Configure SSMS for Azure SQL

1. File → Connect Object Explorer
2. Server name: `[your-server].database.windows.net`
3. Authentication: SQL Server Authentication
4. Login: [your-admin-username]
5. Password: [your-password]

Azure Data Studio (Cross-Platform Alternative)

Step 1: Download and Install

1. Visit: `https://azure.microsoft.com/en-us/products/data-studio/`
2. Download for your operating system
3. Install with default settings

Step 2: Install Extensions

Recommended extensions:

- SQL Database Projects
- Schema Compare
- SQL Server Import
- PowerShell

Step 3: Connect to Databases

1. New Connection
2. Add both local SQL Server and Azure SQL connections
3. Save connection profiles for easy access

Part 7: SQL Fundamentals

Database Design Principles

Understanding Relational Concepts

Entity-Relationship Modeling:

- **Entities**: Objects or things (Customer, Order, Product)
- **Attributes**: Properties of entities (CustomerName, OrderDate, ProductPrice)
- **Relationships**: Connections between entities (Customer places Orders)

Normalization Rules:

1. **First Normal Form (1NF)**: Eliminate repeating groups
2. **Second Normal Form (2NF)**: Eliminate partial dependencies
3. **Third Normal Form (3NF)**: Eliminate transitive dependencies

Example Schema Design:

```
-- Customer entity
CREATE TABLE Customers (
    CustomerID INT IDENTITY(1,1) PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Email NVARCHAR(255) NOT NULL UNIQUE,
    Phone NVARCHAR(20),
    City NVARCHAR(50),
    Country NVARCHAR(50),
    CreatedDate DATETIME2 NOT NULL DEFAULT SYSDATETIME()
);

-- Product entity
CREATE TABLE Products (
    ProductID INT IDENTITY(1,1) PRIMARY KEY,
    ProductName NVARCHAR(100) NOT NULL,
    Description NVARCHAR(MAX),
    CategoryID INT NOT NULL,
    UnitPrice DECIMAL(10,2) NOT NULL CHECK (UnitPrice &gt;= 0),
    UnitsInStock INT NOT NULL DEFAULT 0,
    IsActive BIT NOT NULL DEFAULT 1,
    CreatedDate DATETIME2 NOT NULL DEFAULT SYSDATETIME()
);

-- Order entity
CREATE TABLE Orders (
```

```
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    CustomerID INT NOT NULL,
    OrderDate DATETIME2 NOT NULL DEFAULT SYSDATETIME(),
    OrderNumber AS ('ORD-' + RIGHT('000000' + CAST(OrderID AS VARCHAR(6)), 6)) PERSISTED,
    TotalAmount DECIMAL(12,2) NOT NULL DEFAULT 0,
    OrderStatus NVARCHAR(20) NOT NULL DEFAULT 'Pending',

    CONSTRAINT FK_Orders_Customers
        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

    CONSTRAINT CK_Orders_Status
        CHECK (OrderStatus IN ('Pending', 'Processing', 'Shipped', 'Delivered', 'Cancelled'
);

-- OrderItems junction table (many-to-many relationship)
CREATE TABLE OrderItems (
    OrderItemID INT IDENTITY(1,1) PRIMARY KEY,
    OrderID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity &gt; 0),
    UnitPrice DECIMAL(10,2) NOT NULL CHECK (UnitPrice &gt;= 0),

    CONSTRAINT FK_OrderItems_Orders
        FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

    CONSTRAINT FK_OrderItems_Products
        FOREIGN KEY (ProductID) REFERENCES Products(ProductID),

    -- Prevent duplicate items in same order
    CONSTRAINT UK_OrderItems_Order_Product
        UNIQUE (OrderID, ProductID)
);
```

Data Types and Constraints

Choosing Appropriate Data Types:

```
-- Numeric types
TINYINT         -- 0 to 255 (1 byte)
SMALLINT        -- -32,768 to 32,767 (2 bytes)
INT             -- -2,147,483,648 to 2,147,483,647 (4 bytes)
BIGINT          -- Large integers (8 bytes)
DECIMAL(p,s)    -- Exact numeric (price: DECIMAL(10,2))
FLOAT           -- Approximate numeric (scientific calculations)

-- String types
CHAR(n)         -- Fixed-length (country codes: CHAR(2))
NCHAR(n)        -- Fixed-length Unicode
VARCHAR(n)      -- Variable-length (names: VARCHAR(50))
NVARCHAR(n)     -- Variable-length Unicode (recommended)
NVARCHAR(MAX)   -- Large text (up to 2GB)

-- Date/Time types
DATE            -- Date only (2025-09-10)
TIME            -- Time only (14:30:00.0000000)
```

```
DATETIME2       -- Preferred datetime type
DATETIMEOFFSET  -- With timezone awareness

-- Other types
BIT             -- Boolean (0/1)
UNIQUEIDENTIFIER -- GUID
VARBINARY(MAX)  -- Binary data (files, images)
```

Implementing Constraints:

```
-- Primary Key (clustered by default)
ALTER TABLE Customers
ADD CONSTRAINT PK_Customers PRIMARY KEY (CustomerID);

-- Foreign Key with referential actions
ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customers
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
    ON DELETE RESTRICT  -- Prevent deletion if orders exist
    ON UPDATE CASCADE;  -- Update CustomerID in orders if changed

-- Unique constraints
ALTER TABLE Customers
ADD CONSTRAINT UK_Customers_Email UNIQUE (Email);

-- Check constraints
ALTER TABLE Products
ADD CONSTRAINT CK_Products_Price
    CHECK (UnitPrice &gt;= 0 AND UnitPrice &lt;= 99999.99);

-- Default constraints
ALTER TABLE Orders
ADD CONSTRAINT DF_Orders_OrderDate
    DEFAULT (SYSDATETIME()) FOR OrderDate;

-- Computed columns
ALTER TABLE Orders
ADD OrderYear AS YEAR(OrderDate) PERSISTED;
```

Fundamental Querying

Basic SELECT Operations:

```
-- Simple selection with filtering
SELECT CustomerID, FirstName, LastName, Email
FROM Customers
WHERE Country = 'USA'
    AND City IN ('New York', 'Los Angeles', 'Chicago')
ORDER BY LastName, FirstName;

-- Using wildcards and pattern matching
SELECT ProductName, UnitPrice
FROM Products
WHERE ProductName LIKE '%Phone%'
```

```
    OR ProductName LIKE '%Tablet%'
ORDER BY UnitPrice DESC;

-- Working with dates
SELECT OrderID, CustomerID, OrderDate, TotalAmount
FROM Orders
WHERE OrderDate &gt;= DATEADD(MONTH, -3, GETDATE())
    AND OrderDate &lt; DATEADD(DAY, 1, CAST(GETDATE() AS DATE))
ORDER BY OrderDate DESC;

-- Top N and paging
SELECT TOP 10 CustomerID, FirstName, LastName,
    (SELECT COUNT(*) FROM Orders o WHERE o.CustomerID = c.CustomerID) AS OrderCount
FROM Customers c
ORDER BY OrderCount DESC;

-- Modern paging with OFFSET/FETCH
SELECT CustomerID, FirstName, LastName, Email
FROM Customers
ORDER BY LastName, FirstName
OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;
```

Advanced Filtering and Functions:

```
-- Case expressions
SELECT
    ProductName,
    UnitPrice,
    CASE
        WHEN UnitPrice &lt; 20 THEN 'Budget'
        WHEN UnitPrice &lt; 100 THEN 'Standard'
        ELSE 'Premium'
    END AS PriceCategory,

    -- Date functions
    YEAR(CreatedDate) AS CreatedYear,
    MONTH(CreatedDate) AS CreatedMonth,
    DATEDIFF(DAY, CreatedDate, GETDATE()) AS DaysOld

FROM Products
WHERE IsActive = 1;

-- String functions
SELECT
    CustomerID,
    UPPER(LEFT(FirstName, 1)) + LOWER(SUBSTRING(FirstName, 2, 100)) AS FirstName,
    UPPER(LEFT(LastName, 1)) + LOWER(SUBSTRING(LastName, 2, 100)) AS LastName,
    LEFT(Email, CHARINDEX('@', Email) - 1) AS EmailUsername,
    REPLACE(Phone, '-', '.') AS FormattedPhone
FROM Customers;

-- Null handling
SELECT
    CustomerID,
    FirstName,
    LastName,
```

```
    ISNULL(Phone, 'No phone provided') AS Phone,
    COALESCE(City, Country, 'Unknown location') AS Location
FROM Customers;
```

Joins and Relationships

Inner Joins (Return matching records only):

```
-- Simple inner join
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    o.OrderID,
    o.OrderDate,
    o.TotalAmount
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.OrderDate &gt;= DATEADD(MONTH, -1, GETDATE())
ORDER BY o.OrderDate DESC;

-- Multiple table joins
SELECT
    c.FirstName + ' ' + c.LastName AS CustomerName,
    o.OrderNumber,
    o.OrderDate,
    p.ProductName,
    oi.Quantity,
    oi.UnitPrice,
    (oi.Quantity * oi.UnitPrice) AS LineTotal
FROM Customers c
    INNER JOIN Orders o ON c.CustomerID = o.CustomerID
    INNER JOIN OrderItems oi ON o.OrderID = oi.OrderID
    INNER JOIN Products p ON oi.ProductID = p.ProductID
WHERE o.OrderStatus = 'Delivered'
ORDER BY o.OrderDate DESC, o.OrderID, p.ProductName;
```

Left Joins (Include all records from left table):

```
-- Customers with their order count (including customers with no orders)
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    c.Email,
    COUNT(o.OrderID) AS OrderCount,
    ISNULL(SUM(o.TotalAmount), 0) AS TotalSpent,
    MAX(o.OrderDate) AS LastOrderDate
FROM Customers c
    LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName, c.Email
ORDER BY OrderCount DESC, TotalSpent DESC;

-- Find customers who have never placed an order
SELECT
    c.CustomerID,
```

```
    c.FirstName + ' ' + c.LastName AS CustomerName,
    c.Email,
    c.CreatedDate
FROM Customers c
    LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.CustomerID IS NULL
ORDER BY c.CreatedDate DESC;
```

Right and Full Outer Joins:

```
-- Right join example (rarely used, but good to understand)
SELECT
    p.ProductName,
    ISNULL(SUM(oi.Quantity), 0) AS TotalSold
FROM OrderItems oi
    RIGHT JOIN Products p ON oi.ProductID = p.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY TotalSold DESC;

-- Full outer join (all customers and all orders, even orphaned ones)
SELECT
    ISNULL(c.FirstName + ' ' + c.LastName, 'Unknown Customer') AS CustomerName,
    ISNULL(o.OrderNumber, 'No Order') AS OrderNumber,
    o.OrderDate,
    o.TotalAmount
FROM Customers c
    FULL OUTER JOIN Orders o ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID, o.OrderDate;
```

Self Joins and Advanced Patterns:

```
-- Employee hierarchy example (self-join)
-- First, create an Employees table structure
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50),
    Title NVARCHAR(50),
    ManagerID INT NULL,
    FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)
);

-- Self join to show employee-manager relationships
SELECT
    e.FirstName + ' ' + e.LastName AS Employee,
    e.Title AS EmployeeTitle,
    m.FirstName + ' ' + m.LastName AS Manager,
    m.Title AS ManagerTitle
FROM Employees e
    LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID
ORDER BY m.LastName, e.LastName;
```

Aggregation and Grouping

Basic Aggregations:

```
-- Simple aggregation functions
SELECT
    COUNT(*) AS TotalCustomers,
    COUNT(Phone) AS CustomersWithPhone,  -- Excludes NULLs
    COUNT(DISTINCT Country) AS Countries,
    MIN(CreatedDate) AS FirstCustomer,
    MAX(CreatedDate) AS LatestCustomer
FROM Customers;

-- Aggregate with filtering
SELECT
    COUNT(*) AS TotalOrders,
    SUM(TotalAmount) AS GrandTotal,
    AVG(TotalAmount) AS AverageOrderValue,
    MIN(TotalAmount) AS SmallestOrder,
    MAX(TotalAmount) AS LargestOrder,
    STDEV(TotalAmount) AS StandardDeviation
FROM Orders
WHERE OrderDate &gt;= DATEADD(YEAR, -1, GETDATE())
    AND OrderStatus = 'Delivered';
```

GROUP BY Operations:

```
-- Sales by month
SELECT
    YEAR(OrderDate) AS OrderYear,
    MONTH(OrderDate) AS OrderMonth,
    DATENAME(MONTH, OrderDate) AS MonthName,
    COUNT(*) AS OrderCount,
    SUM(TotalAmount) AS MonthlyRevenue,
    AVG(TotalAmount) AS AvgOrderValue
FROM Orders
WHERE OrderDate &gt;= DATEADD(YEAR, -2, GETDATE())
GROUP BY YEAR(OrderDate), MONTH(OrderDate), DATENAME(MONTH, OrderDate)
ORDER BY OrderYear DESC, OrderMonth DESC;

-- Customer analysis
SELECT
    c.Country,
    c.City,
    COUNT(DISTINCT c.CustomerID) AS CustomerCount,
    COUNT(o.OrderID) AS TotalOrders,
    SUM(o.TotalAmount) AS TotalRevenue,
    AVG(o.TotalAmount) AS AvgOrderValue
FROM Customers c
    LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.Country, c.City
HAVING COUNT(DISTINCT c.CustomerID) &gt;= 5  -- Cities with at least 5 customers
ORDER BY TotalRevenue DESC;
```

HAVING Clause and Advanced Grouping:

```
-- Find top customers (those with more than $1000 in total purchases)
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    COUNT(o.OrderID) AS OrderCount,
    SUM(o.TotalAmount) AS TotalPurchases,
    AVG(o.TotalAmount) AS AvgOrderValue,
    MIN(o.OrderDate) AS FirstOrder,
    MAX(o.OrderDate) AS LastOrder
FROM Customers c
    INNER JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.OrderStatus = 'Delivered'
GROUP BY c.CustomerID, c.FirstName, c.LastName
HAVING SUM(o.TotalAmount) &gt; 1000
    AND COUNT(o.OrderID) &gt;= 3
ORDER BY TotalPurchases DESC, OrderCount DESC;

-- Product performance analysis
SELECT
    p.ProductID,
    p.ProductName,
    COUNT(DISTINCT oi.OrderID) AS OrdersWithProduct,
    SUM(oi.Quantity) AS TotalQuantitySold,
    SUM(oi.Quantity * oi.UnitPrice) AS TotalRevenue,
    AVG(oi.UnitPrice) AS AvgSellingPrice,
    p.UnitPrice AS CurrentPrice
FROM Products p
    LEFT JOIN OrderItems oi ON p.ProductID = oi.ProductID
    LEFT JOIN Orders o ON oi.OrderID = o.OrderID AND o.OrderStatus = 'Delivered'
GROUP BY p.ProductID, p.ProductName, p.UnitPrice
HAVING COUNT(DISTINCT oi.OrderID) &gt; 0  -- Only products that have been sold
ORDER BY TotalRevenue DESC;
```

Window Functions and Advanced Analytics

Ranking Functions:

```
-- Rank customers by total purchases
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    SUM(o.TotalAmount) AS TotalPurchases,
    RANK() OVER (ORDER BY SUM(o.TotalAmount) DESC) AS PurchaseRank,
    ROW_NUMBER() OVER (ORDER BY SUM(o.TotalAmount) DESC) AS RowNum,
    DENSE_RANK() OVER (ORDER BY SUM(o.TotalAmount) DESC) AS DenseRank
FROM Customers c
    INNER JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.OrderStatus = 'Delivered'
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY TotalPurchases DESC;

-- Partition by category
SELECT
    p.ProductID,
```

```
    p.ProductName,
    p.CategoryID,
    p.UnitPrice,
    RANK() OVER (PARTITION BY p.CategoryID ORDER BY p.UnitPrice DESC) AS PriceRankInCategor
    AVG(p.UnitPrice) OVER (PARTITION BY p.CategoryID) AS CategoryAvgPrice
FROM Products p
WHERE p.IsActive = 1
ORDER BY p.CategoryID, PriceRankInCategory;
```

Running Totals and Moving Averages:

```
-- Daily sales with running totals
SELECT
    CAST(o.OrderDate AS DATE) AS OrderDate,
    COUNT(*) AS DailyOrders,
    SUM(o.TotalAmount) AS DailyRevenue,

    -- Running totals
    SUM(COUNT(*)) OVER (ORDER BY CAST(o.OrderDate AS DATE)) AS RunningOrderCount,
    SUM(SUM(o.TotalAmount)) OVER (ORDER BY CAST(o.OrderDate AS DATE)) AS RunningRevenue,

    -- Moving averages (7-day window)
    AVG(SUM(o.TotalAmount)) OVER (
        ORDER BY CAST(o.OrderDate AS DATE)
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) AS SevenDayAvgRevenue

FROM Orders o
WHERE o.OrderDate &gt;= DATEADD(MONTH, -3, GETDATE())
    AND o.OrderStatus = 'Delivered'
GROUP BY CAST(o.OrderDate AS DATE)
ORDER BY OrderDate;
```

Lead/Lag Functions:

```
-- Customer purchase patterns
WITH CustomerOrders AS (
    SELECT
        c.CustomerID,
        c.FirstName + ' ' + c.LastName AS CustomerName,
        o.OrderID,
        o.OrderDate,
        o.TotalAmount,
        ROW_NUMBER() OVER (PARTITION BY c.CustomerID ORDER BY o.OrderDate) AS OrderSequence
    FROM Customers c
        INNER JOIN Orders o ON c.CustomerID = o.CustomerID
    WHERE o.OrderStatus = 'Delivered'
)
SELECT
    CustomerID,
    CustomerName,
    OrderID,
    OrderDate,
    TotalAmount,
    OrderSequence,
```

```
    -- Previous order information
    LAG(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS PreviousOrderDate,
    LAG(TotalAmount) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS PreviousOrderAmou

    -- Next order information
    LEAD(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS NextOrderDate,

    -- Days between orders
    DATEDIFF(DAY,
        LAG(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate),
        OrderDate
    ) AS DaysSinceLastOrder

FROM CustomerOrders
ORDER BY CustomerID, OrderDate;
```

Subqueries and Common Table Expressions

Scalar Subqueries:

```
-- Add customer's total order count to each order
SELECT
    o.OrderID,
    o.CustomerID,
    (SELECT FirstName + ' ' + LastName
     FROM Customers c
     WHERE c.CustomerID = o.CustomerID) AS CustomerName,
    o.OrderDate,
    o.TotalAmount,
    (SELECT COUNT(*)
     FROM Orders o2
     WHERE o2.CustomerID = o.CustomerID) AS CustomerTotalOrders
FROM Orders o
WHERE o.OrderDate &gt;= DATEADD(MONTH, -1, GETDATE())
ORDER BY o.OrderDate DESC;
```

Correlated Subqueries:

```
-- Find customers whose last order was more than 6 months ago
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    c.Email,
    (SELECT MAX(o.OrderDate)
     FROM Orders o
     WHERE o.CustomerID = c.CustomerID) AS LastOrderDate
FROM Customers c
WHERE EXISTS (
    SELECT 1 FROM Orders o
    WHERE o.CustomerID = c.CustomerID
)
AND (
    SELECT MAX(o.OrderDate)
```

```
    FROM Orders o
    WHERE o.CustomerID = c.CustomerID
) &lt; DATEADD(MONTH, -6, GETDATE())
ORDER BY LastOrderDate DESC;

-- Products never ordered
SELECT
    p.ProductID,
    p.ProductName,
    p.UnitPrice,
    p.UnitsInStock
FROM Products p
WHERE NOT EXISTS (
    SELECT 1 FROM OrderItems oi
    WHERE oi.ProductID = p.ProductID
)
AND p.IsActive = 1
ORDER BY p.UnitPrice DESC;
```

Common Table Expressions (CTEs):

```
-- Recursive CTE for employee hierarchy
WITH EmployeeHierarchy AS (
    -- Anchor: Top-level managers (no manager)
    SELECT
        EmployeeID,
        FirstName + ' ' + LastName AS EmployeeName,
        Title,
        ManagerID,
        0 AS Level,
        CAST(FirstName + ' ' + LastName AS NVARCHAR(1000)) AS HierarchyPath
    FROM Employees
    WHERE ManagerID IS NULL

    UNION ALL

    -- Recursive: Employees with managers
    SELECT
        e.EmployeeID,
        e.FirstName + ' ' + e.LastName,
        e.Title,
        e.ManagerID,
        eh.Level + 1,
        CAST(eh.HierarchyPath + ' -&gt; ' + e.FirstName + ' ' + e.LastName AS NVARCHAR(1000
    FROM Employees e
        INNER JOIN EmployeeHierarchy eh ON e.ManagerID = eh.EmployeeID
)
SELECT
    EmployeeID,
    EmployeeName,
    Title,
    Level,
    HierarchyPath
FROM EmployeeHierarchy
ORDER BY Level, EmployeeName;
```

```
-- Multiple CTEs for complex analysis
WITH MonthlySales AS (
    SELECT
        YEAR(OrderDate) AS SalesYear,
        MONTH(OrderDate) AS SalesMonth,
        SUM(TotalAmount) AS MonthlyRevenue,
        COUNT(*) AS MonthlyOrders
    FROM Orders
    WHERE OrderStatus = 'Delivered'
        AND OrderDate &gt;= DATEADD(YEAR, -2, GETDATE())
    GROUP BY YEAR(OrderDate), MONTH(OrderDate)
),
SalesGrowth AS (
    SELECT
        SalesYear,
        SalesMonth,
        MonthlyRevenue,
        MonthlyOrders,
        LAG(MonthlyRevenue) OVER (ORDER BY SalesYear, SalesMonth) AS PreviousMonthRevenue,
        MonthlyRevenue - LAG(MonthlyRevenue) OVER (ORDER BY SalesYear, SalesMonth) AS Reven
    FROM MonthlySales
)
SELECT
    SalesYear,
    SalesMonth,
    DATENAME(MONTH, DATEFROMPARTS(SalesYear, SalesMonth, 1)) AS MonthName,
    MonthlyRevenue,
    MonthlyOrders,
    PreviousMonthRevenue,
    RevenueGrowth,
    CASE
        WHEN PreviousMonthRevenue &gt; 0
        THEN (RevenueGrowth * 100.0 / PreviousMonthRevenue)
        ELSE NULL
    END AS GrowthPercent
FROM SalesGrowth
ORDER BY SalesYear, SalesMonth;
```

Part 8: EF Core Implementation

Entity Classes and DbContext Setup

Creating Entity Classes:

```
// Customer entity with proper conventions
public class Customer
{
    public int CustomerID { get; set; }

    [Required]
    [MaxLength(50)]
    public string FirstName { get; set; } = string.Empty;
```

```
    [Required]
    [MaxLength(50)]
    public string LastName { get; set; } = string.Empty;

    [Required]
    [MaxLength(255)]
    [EmailAddress]
    public string Email { get; set; } = string.Empty;

    [MaxLength(20)]
    public string? Phone { get; set; }

    [MaxLength(50)]
    public string? City { get; set; }

    [MaxLength(50)]
    public string? Country { get; set; }

    public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

    // Navigation properties
    public virtual ICollection&lt;Order&gt; Orders { get; set; } = new List&lt;Order&gt;();
}

// Product entity
public class Product
{
    public int ProductID { get; set; }

    [Required]
    [MaxLength(100)]
    public string ProductName { get; set; } = string.Empty;

    public string? Description { get; set; }

    public int CategoryID { get; set; }

    [Column(TypeName = "decimal(10,2)")]
    [Range(0, 99999.99)]
    public decimal UnitPrice { get; set; }

    public int UnitsInStock { get; set; } = 0;

    public bool IsActive { get; set; } = true;

    public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

    // Navigation properties
    public virtual ICollection&lt;OrderItem&gt; OrderItems { get; set; } = new List&lt;Orde
}

// Order entity
public class Order
{
    public int OrderID { get; set; }
```

```
    public int CustomerID { get; set; }

    public DateTime OrderDate { get; set; } = DateTime.UtcNow;

    [MaxLength(20)]
    public string OrderNumber { get; set; } = string.Empty;

    [Column(TypeName = "decimal(12,2)")]
    public decimal TotalAmount { get; set; }

    [MaxLength(20)]
    public string OrderStatus { get; set; } = "Pending";

    // Navigation properties
    public virtual Customer Customer { get; set; } = null!;
    public virtual ICollection&lt;OrderItem&gt; OrderItems { get; set; } = new List&lt;Orde
}

// OrderItem entity (junction table)
public class OrderItem
{
    public int OrderItemID { get; set; }

    public int OrderID { get; set; }

    public int ProductID { get; set; }

    [Range(1, int.MaxValue)]
    public int Quantity { get; set; }

    [Column(TypeName = "decimal(10,2)")]
    [Range(0, 99999.99)]
    public decimal UnitPrice { get; set; }

    // Navigation properties
    public virtual Order Order { get; set; } = null!;
    public virtual Product Product { get; set; } = null!;
}
```

Creating the DbContext:

```
public class LearningDbContext : DbContext
{
    public LearningDbContext(DbContextOptions&lt;LearningDbContext&gt; options) : base(opti
    {
    }

    // DbSets
    public DbSet&lt;Customer&gt; Customers =&gt; Set&lt;Customer&gt;();
    public DbSet&lt;Product&gt; Products =&gt; Set&lt;Product&gt;();
    public DbSet&lt;Order&gt; Orders =&gt; Set&lt;Order&gt;();
    public DbSet&lt;OrderItem&gt; OrderItems =&gt; Set&lt;OrderItem&gt;();

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
```

```
        // Customer configuration
        modelBuilder.Entity&lt;Customer&gt;(entity =&gt;
        {
            entity.HasKey(e =&gt; e.CustomerID);

            entity.Property(e =&gt; e.CustomerID)
                .UseIdentityColumn();

            entity.HasIndex(e =&gt; e.Email)
                .IsUnique()
                .HasDatabaseName("IX_Customers_Email");

            entity.Property(e =&gt; e.CreatedDate)
                .HasDefaultValueSql("SYSDATETIME()");
        });

        // Product configuration
        modelBuilder.Entity&lt;Product&gt;(entity =&gt;
        {
            entity.HasKey(e =&gt; e.ProductID);

            entity.Property(e =&gt; e.ProductID)
                .UseIdentityColumn();

            entity.Property(e =&gt; e.UnitPrice)
                .HasColumnType("decimal(10,2)");

            entity.Property(e =&gt; e.CreatedDate)
                .HasDefaultValueSql("SYSDATETIME()");

            entity.HasCheckConstraint("CK_Products_UnitPrice",
                "[UnitPrice] &gt;= 0 AND [UnitPrice] &lt;= 99999.99");
        });

        // Order configuration
        modelBuilder.Entity&lt;Order&gt;(entity =&gt;
        {
            entity.HasKey(e =&gt; e.OrderID);

            entity.Property(e =&gt; e.OrderID)
                .UseIdentityColumn();

            entity.Property(e =&gt; e.TotalAmount)
                .HasColumnType("decimal(12,2)");

            entity.Property(e =&gt; e.OrderDate)
                .HasDefaultValueSql("SYSDATETIME()");

            // Computed column for OrderNumber
            entity.Property(e =&gt; e.OrderNumber)
                .HasComputedColumnSql("('ORD-' + RIGHT('000000' + CAST([OrderID] AS VARCHAR

            entity.HasCheckConstraint("CK_Orders_Status",
                "[OrderStatus] IN ('Pending', 'Processing', 'Shipped', 'Delivered', 'Cancel
```

```
            // Foreign key relationship
            entity.HasOne(e =&gt; e.Customer)
                .WithMany(c =&gt; c.Orders)
                .HasForeignKey(e =&gt; e.CustomerID)
                .OnDelete(DeleteBehavior.Restrict)
                .HasConstraintName("FK_Orders_Customers");
        });

        // OrderItem configuration
        modelBuilder.Entity&lt;OrderItem&gt;(entity =&gt;
        {
            entity.HasKey(e =&gt; e.OrderItemID);

            entity.Property(e =&gt; e.OrderItemID)
                .UseIdentityColumn();

            entity.Property(e =&gt; e.UnitPrice)
                .HasColumnType("decimal(10,2)");

            entity.HasCheckConstraint("CK_OrderItems_Quantity", "[Quantity] &gt; 0");
            entity.HasCheckConstraint("CK_OrderItems_UnitPrice", "[UnitPrice] &gt;= 0");

            // Composite unique constraint
            entity.HasIndex(e =&gt; new { e.OrderID, e.ProductID })
                .IsUnique()
                .HasDatabaseName("IX_OrderItems_Order_Product");

            // Foreign key relationships
            entity.HasOne(e =&gt; e.Order)
                .WithMany(o =&gt; o.OrderItems)
                .HasForeignKey(e =&gt; e.OrderID)
                .OnDelete(DeleteBehavior.Cascade)
                .HasConstraintName("FK_OrderItems_Orders");

            entity.HasOne(e =&gt; e.Product)
                .WithMany(p =&gt; p.OrderItems)
                .HasForeignKey(e =&gt; e.ProductID)
                .OnDelete(DeleteBehavior.Restrict)
                .HasConstraintName("FK_OrderItems_Products");
        });

        // Seed data (optional)
        SeedData(modelBuilder);
    }

    private static void SeedData(ModelBuilder modelBuilder)
    {
        // Seed customers
        modelBuilder.Entity&lt;Customer&gt;().HasData(
            new Customer
            {
                CustomerID = 1,
                FirstName = "John",
                LastName = "Doe",
                Email = "john.doe@example.com",
                City = "New York",
```

```
                Country = "USA",
                CreatedDate = new DateTime(2024, 1, 1)
            },
            new Customer
            {
                CustomerID = 2,
                FirstName = "Jane",
                LastName = "Smith",
                Email = "jane.smith@example.com",
                City = "London",
                Country = "UK",
                CreatedDate = new DateTime(2024, 1, 2)
            }
        );

        // Seed products
        modelBuilder.Entity&lt;Product&gt;().HasData(
            new Product
            {
                ProductID = 1,
                ProductName = "Laptop Pro",
                CategoryID = 1,
                UnitPrice = 1299.99m,
                UnitsInStock = 50,
                CreatedDate = new DateTime(2024, 1, 1)
            },
            new Product
            {
                ProductID = 2,
                ProductName = "Wireless Mouse",
                CategoryID = 1,
                UnitPrice = 29.99m,
                UnitsInStock = 200,
                CreatedDate = new DateTime(2024, 1, 1)
            }
        );
    }
}
```

Configuration and Dependency Injection

Program.cs Configuration (.NET 8):

```
using LearningEFCore.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure Entity Framework
builder.Services.AddDbContext&lt;LearningDbContext&gt;(options =&gt;
```

```
{
    // For local SQL Server
    // options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))

    // For Azure SQL Database
    options.UseSqlServer(builder.Configuration.GetConnectionString("AzureSqlConnection"));

    // Development configuration
    if (builder.Environment.IsDevelopment())
    {
        options.EnableSensitiveDataLogging();
        options.EnableDetailedErrors();
        options.LogTo(Console.WriteLine);
    }
});

// Register application services
builder.Services.AddScoped&lt;ICustomerService, CustomerService&gt;();
builder.Services.AddScoped&lt;IOrderService, OrderService&gt;();

var app = builder.Build();

// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();

    // Auto-migrate database in development
    using (var scope = app.Services.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService&lt;LearningDbContext&gt;();
        try
        {
            context.Database.Migrate();
        }
        catch (Exception ex)
        {
            var logger = scope.ServiceProvider.GetRequiredService&lt;ILogger&lt;Program&gt;
            logger.LogError(ex, "An error occurred creating the DB.");
        }
    }
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

appsettings.json Configuration:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=LearningEFCore;Trusted_Connection=True;
    "AzureSqlConnection": "Server=tcp:your-server.database.windows.net,1433;Initial Catalog
```

```
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.EntityFrameworkCore": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Migrations and Database Management

Creating and Managing Migrations:

```
# Create initial migration
dotnet ef migrations add InitialCreate

# View migration SQL without applying
dotnet ef migrations script

# Apply migrations to database
dotnet ef database update

# Create migration for specific changes
dotnet ef migrations add AddProductCategories

# Rollback to specific migration
dotnet ef database update PreviousMigrationName

# Remove last migration (if not applied)
dotnet ef migrations remove

# Generate SQL script for production deployment
dotnet ef migrations script InitialCreate AddProductCategories -o migration.sql

# Update to specific migration
dotnet ef database update AddProductCategories

# Reset database (remove and recreate)
dotnet ef database drop --force
dotnet ef database update
```

Custom Migration Example:

```
public partial class AddCustomIndexes : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        // Create custom indexes for performance
        migrationBuilder.Sql(@"
            CREATE INDEX IX_Orders_CustomerID_OrderDate
            ON Orders(CustomerID, OrderDate DESC)
            INCLUDE (TotalAmount, OrderStatus);
```

```
        ");

        migrationBuilder.Sql(@"
            CREATE INDEX IX_OrderItems_ProductID
            ON OrderItems(ProductID)
            INCLUDE (Quantity, UnitPrice);
        ");

        // Add computed column
        migrationBuilder.Sql(@"
            ALTER TABLE Orders
            ADD OrderYear AS YEAR(OrderDate) PERSISTED;
        ");
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.Sql("DROP INDEX IX_Orders_CustomerID_OrderDate ON Orders;");
        migrationBuilder.Sql("DROP INDEX IX_OrderItems_ProductID ON OrderItems;");
        migrationBuilder.Sql("ALTER TABLE Orders DROP COLUMN OrderYear;");
    }
}
```

Repository Pattern and Service Layer

Repository Interface and Implementation:

```
// Generic repository interface
public interface IRepository&lt;T&gt; where T : class
{
    Task&lt;T?&gt; GetByIdAsync(int id);
    Task&lt;IEnumerable&lt;T&gt;&gt; GetAllAsync();
    Task&lt;IEnumerable&lt;T&gt;&gt; FindAsync(Expression&lt;Func&lt;T, bool&gt;&gt; predic
    Task&lt;T&gt; AddAsync(T entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(int id);
    Task&lt;int&gt; CountAsync(Expression&lt;Func&lt;T, bool&gt;&gt;? predicate = null);
    Task&lt;bool&gt; ExistsAsync(int id);
}

// Customer-specific repository interface
public interface ICustomerRepository : IRepository&lt;Customer&gt;
{
    Task&lt;Customer?&gt; GetByEmailAsync(string email);
    Task&lt;IEnumerable&lt;Customer&gt;&gt; GetCustomersWithOrdersAsync();
    Task&lt;PaginatedResult&lt;Customer&gt;&gt; GetPagedAsync(int page, int pageSize, strin
    Task&lt;IEnumerable&lt;Customer&gt;&gt; GetTopCustomersByRevenueAsync(int count);
}

// Repository implementation
public class CustomerRepository : ICustomerRepository
{
    private readonly LearningDbContext _context;

    public CustomerRepository(LearningDbContext context)
```

```
    {
        _context = context;
    }

    public async Task&lt;Customer?&gt; GetByIdAsync(int id)
    {
        return await _context.Customers
            .Include(c =&gt; c.Orders)
            .FirstOrDefaultAsync(c =&gt; c.CustomerID == id);
    }

    public async Task&lt;IEnumerable&lt;Customer&gt;&gt; GetAllAsync()
    {
        return await _context.Customers
            .AsNoTracking()
            .OrderBy(c =&gt; c.LastName)
            .ThenBy(c =&gt; c.FirstName)
            .ToListAsync();
    }

    public async Task&lt;IEnumerable&lt;Customer&gt;&gt; FindAsync(Expression&lt;Func&lt;Cu
    {
        return await _context.Customers
            .AsNoTracking()
            .Where(predicate)
            .ToListAsync();
    }

    public async Task&lt;Customer&gt; AddAsync(Customer entity)
    {
        _context.Customers.Add(entity);
        await _context.SaveChangesAsync();
        return entity;
    }

    public async Task UpdateAsync(Customer entity)
    {
        _context.Entry(entity).State = EntityState.Modified;
        await _context.SaveChangesAsync();
    }

    public async Task DeleteAsync(int id)
    {
        var customer = await _context.Customers.FindAsync(id);
        if (customer != null)
        {
            _context.Customers.Remove(customer);
            await _context.SaveChangesAsync();
        }
    }

    public async Task&lt;int&gt; CountAsync(Expression&lt;Func&lt;Customer, bool&gt;&gt;? p
    {
        return predicate == null
            ? await _context.Customers.CountAsync()
            : await _context.Customers.CountAsync(predicate);
```

```
    }

    public async Task&lt;bool&gt; ExistsAsync(int id)
    {
        return await _context.Customers.AnyAsync(c =&gt; c.CustomerID == id);
    }

    public async Task&lt;Customer?&gt; GetByEmailAsync(string email)
    {
        return await _context.Customers
            .FirstOrDefaultAsync(c =&gt; c.Email == email);
    }

    public async Task&lt;IEnumerable&lt;Customer&gt;&gt; GetCustomersWithOrdersAsync()
    {
        return await _context.Customers
            .Include(c =&gt; c.Orders)
            .Where(c =&gt; c.Orders.Any())
            .AsNoTracking()
            .ToListAsync();
    }

    public async Task&lt;PaginatedResult&lt;Customer&gt;&gt; GetPagedAsync(int page, int pa
    {
        var query = _context.Customers.AsQueryable();

        if (!string.IsNullOrEmpty(search))
        {
            query = query.Where(c =&gt;
                c.FirstName.Contains(search) ||
                c.LastName.Contains(search) ||
                c.Email.Contains(search));
        }

        var totalCount = await query.CountAsync();

        var customers = await query
            .OrderBy(c =&gt; c.LastName)
            .ThenBy(c =&gt; c.FirstName)
            .Skip((page - 1) * pageSize)
            .Take(pageSize)
            .AsNoTracking()
            .ToListAsync();

        return new PaginatedResult&lt;Customer&gt;(customers, totalCount, page, pageSize);
    }

    public async Task&lt;IEnumerable&lt;Customer&gt;&gt; GetTopCustomersByRevenueAsync(int
    {
        return await _context.Customers
            .Select(c =&gt; new {
                Customer = c,
                TotalRevenue = c.Orders.Where(o =&gt; o.OrderStatus == "Delivered").Sum(o =
            })
            .OrderByDescending(x =&gt; x.TotalRevenue)
            .Take(count)
```

```
            .Select(x =&gt; x.Customer)
            .AsNoTracking()
            .ToListAsync();
    }
}

// Pagination helper class
public class PaginatedResult&lt;T&gt;
{
    public IEnumerable&lt;T&gt; Data { get; }
    public int TotalCount { get; }
    public int Page { get; }
    public int PageSize { get; }
    public int TotalPages =&gt; (int)Math.Ceiling((double)TotalCount / PageSize);
    public bool HasPreviousPage =&gt; Page &gt; 1;
    public bool HasNextPage =&gt; Page &lt; TotalPages;

    public PaginatedResult(IEnumerable&lt;T&gt; data, int totalCount, int page, int pageSiz
    {
        Data = data;
        TotalCount = totalCount;
        Page = page;
        PageSize = pageSize;
    }
}
```

Service Layer Implementation:

```
// Customer service interface
public interface ICustomerService
{
    Task&lt;CustomerDto?&gt; GetCustomerAsync(int id);
    Task&lt;IEnumerable&lt;CustomerListDto&gt;&gt; GetAllCustomersAsync();
    Task&lt;PaginatedResult&lt;CustomerListDto&gt;&gt; GetPagedCustomersAsync(int page, int
    Task&lt;CustomerDto&gt; CreateCustomerAsync(CreateCustomerDto dto);
    Task&lt;bool&gt; UpdateCustomerAsync(int id, UpdateCustomerDto dto);
    Task&lt;bool&gt; DeleteCustomerAsync(int id);
    Task&lt;CustomerStatsDto&gt; GetCustomerStatsAsync(int id);
}

// DTOs for data transfer
public record CustomerDto(
    int CustomerID,
    string FirstName,
    string LastName,
    string Email,
    string? Phone,
    string? City,
    string? Country,
    DateTime CreatedDate,
    int OrderCount,
    decimal TotalSpent
);

public record CustomerListDto(
    int CustomerID,
```

```
    string FullName,
    string Email,
    string? City,
    string? Country,
    int OrderCount
);

public record CreateCustomerDto(
    string FirstName,
    string LastName,
    string Email,
    string? Phone,
    string? City,
    string? Country
);

public record UpdateCustomerDto(
    string FirstName,
    string LastName,
    string? Phone,
    string? City,
    string? Country
);

public record CustomerStatsDto(
    int CustomerID,
    string FullName,
    int TotalOrders,
    decimal TotalSpent,
    decimal AverageOrderValue,
    DateTime? LastOrderDate,
    DateTime? FirstOrderDate
);

// Customer service implementation
public class CustomerService : ICustomerService
{
    private readonly ICustomerRepository _customerRepository;
    private readonly ILogger&lt;CustomerService&gt; _logger;

    public CustomerService(ICustomerRepository customerRepository, ILogger&lt;CustomerServi
    {
        _customerRepository = customerRepository;
        _logger = logger;
    }

    public async Task&lt;CustomerDto?&gt; GetCustomerAsync(int id)
    {
        var customer = await _customerRepository.GetByIdAsync(id);
        return customer == null ? null : MapToDto(customer);
    }

    public async Task&lt;IEnumerable&lt;CustomerListDto&gt;&gt; GetAllCustomersAsync()
    {
        var customers = await _customerRepository.GetAllAsync();
        return customers.Select(MapToListDto);
```

```
    }

    public async Task&lt;PaginatedResult&lt;CustomerListDto&gt;&gt; GetPagedCustomersAsync(
    {
        var result = await _customerRepository.GetPagedAsync(page, pageSize, search);
        var dtos = result.Data.Select(MapToListDto);
        return new PaginatedResult&lt;CustomerListDto&gt;(dtos, result.TotalCount, result.P
    }

    public async Task&lt;CustomerDto&gt; CreateCustomerAsync(CreateCustomerDto dto)
    {
        // Validate email uniqueness
        var existing = await _customerRepository.GetByEmailAsync(dto.Email);
        if (existing != null)
        {
            throw new InvalidOperationException("Email already exists");
        }

        var customer = new Customer
        {
            FirstName = dto.FirstName,
            LastName = dto.LastName,
            Email = dto.Email,
            Phone = dto.Phone,
            City = dto.City,
            Country = dto.Country
        };

        var created = await _customerRepository.AddAsync(customer);
        _logger.LogInformation("Created customer {CustomerId} with email {Email}", created.

        return MapToDto(created);
    }

    public async Task&lt;bool&gt; UpdateCustomerAsync(int id, UpdateCustomerDto dto)
    {
        var customer = await _customerRepository.GetByIdAsync(id);
        if (customer == null) return false;

        customer.FirstName = dto.FirstName;
        customer.LastName = dto.LastName;
        customer.Phone = dto.Phone;
        customer.City = dto.City;
        customer.Country = dto.Country;

        await _customerRepository.UpdateAsync(customer);
        _logger.LogInformation("Updated customer {CustomerId}", id);

        return true;
    }

    public async Task&lt;bool&gt; DeleteCustomerAsync(int id)
    {
        var exists = await _customerRepository.ExistsAsync(id);
        if (!exists) return false;
```

```
        await _customerRepository.DeleteAsync(id);
        _logger.LogInformation("Deleted customer {CustomerId}", id);

        return true;
    }

    public async Task&lt;CustomerStatsDto&gt; GetCustomerStatsAsync(int id)
    {
        var customer = await _customerRepository.GetByIdAsync(id);
        if (customer == null)
            throw new ArgumentException("Customer not found", nameof(id));

        var deliveredOrders = customer.Orders.Where(o =&gt; o.OrderStatus == "Delivered").T

        return new CustomerStatsDto(
            customer.CustomerID,
            $"{customer.FirstName} {customer.LastName}",
            deliveredOrders.Count,
            deliveredOrders.Sum(o =&gt; o.TotalAmount),
            deliveredOrders.Any() ? deliveredOrders.Average(o =&gt; o.TotalAmount) : 0,
            deliveredOrders.Any() ? deliveredOrders.Max(o =&gt; o.OrderDate) : null,
            deliveredOrders.Any() ? deliveredOrders.Min(o =&gt; o.OrderDate) : null
        );
    }

    private static CustomerDto MapToDto(Customer customer)
    {
        var deliveredOrders = customer.Orders.Where(o =&gt; o.OrderStatus == "Delivered").T
        return new CustomerDto(
            customer.CustomerID,
            customer.FirstName,
            customer.LastName,
            customer.Email,
            customer.Phone,
            customer.City,
            customer.Country,
            customer.CreatedDate,
            deliveredOrders.Count,
            deliveredOrders.Sum(o =&gt; o.TotalAmount)
        );
    }

    private static CustomerListDto MapToListDto(Customer customer)
    {
        return new CustomerListDto(
            customer.CustomerID,
            $"{customer.FirstName} {customer.LastName}",
            customer.Email,
            customer.City,
            customer.Country,
            customer.Orders.Count
        );
    }
}
```

Advanced Querying Techniques

Complex LINQ Queries:

```
public class OrderService : IOrderService
{
    private readonly LearningDbContext _context;

    public OrderService(LearningDbContext context)
    {
        _context = context;
    }

    // Get orders with customer and product details
    public async Task&lt;IEnumerable&lt;OrderDetailDto&gt;&gt; GetOrdersWithDetailsAsync(in
    {
        var query = _context.Orders
            .Include(o =&gt; o.Customer)
            .Include(o =&gt; o.OrderItems)
                .ThenInclude(oi =&gt; oi.Product)
            .AsQueryable();

        if (customerId.HasValue)
        {
            query = query.Where(o =&gt; o.CustomerID == customerId.Value);
        }

        return await query
            .Select(o =&gt; new OrderDetailDto
            {
                OrderID = o.OrderID,
                OrderNumber = o.OrderNumber,
                OrderDate = o.OrderDate,
                CustomerName = $"{o.Customer.FirstName} {o.Customer.LastName}",
                CustomerEmail = o.Customer.Email,
                TotalAmount = o.TotalAmount,
                OrderStatus = o.OrderStatus,
                Items = o.OrderItems.Select(oi =&gt; new OrderItemDto
                {
                    ProductName = oi.Product.ProductName,
                    Quantity = oi.Quantity,
                    UnitPrice = oi.UnitPrice,
                    LineTotal = oi.Quantity * oi.UnitPrice
                }).ToList()
            })
            .OrderByDescending(o =&gt; o.OrderDate)
            .AsNoTracking()
            .ToListAsync();
    }

    // Monthly sales report with growth calculation
    public async Task&lt;IEnumerable&lt;MonthlySalesDto&gt;&gt; GetMonthlySalesReportAsync(
    {
        var startDate = DateTime.UtcNow.AddMonths(-months);

        return await _context.Orders
```

```
            .Where(o =&gt; o.OrderDate &gt;= startDate &amp;&amp; o.OrderStatus == "Deliver
            .GroupBy(o =&gt; new { Year = o.OrderDate.Year, Month = o.OrderDate.Month })
            .Select(g =&gt; new MonthlySalesDto
            {
                Year = g.Key.Year,
                Month = g.Key.Month,
                OrderCount = g.Count(),
                TotalRevenue = g.Sum(o =&gt; o.TotalAmount),
                AverageOrderValue = g.Average(o =&gt; o.TotalAmount)
            })
            .OrderBy(m =&gt; m.Year)
            .ThenBy(m =&gt; m.Month)
            .AsNoTracking()
            .ToListAsync();
    }

    // Product performance analysis
    public async Task&lt;IEnumerable&lt;ProductPerformanceDto&gt;&gt; GetProductPerformance
    {
        return await _context.Products
            .Select(p =&gt; new ProductPerformanceDto
            {
                ProductID = p.ProductID,
                ProductName = p.ProductName,
                CurrentPrice = p.UnitPrice,
                UnitsInStock = p.UnitsInStock,
                TotalOrders = p.OrderItems.Count(oi =&gt; oi.Order.OrderStatus == "Delivere
                TotalQuantitySold = p.OrderItems
                    .Where(oi =&gt; oi.Order.OrderStatus == "Delivered")
                    .Sum(oi =&gt; oi.Quantity),
                TotalRevenue = p.OrderItems
                    .Where(oi =&gt; oi.Order.OrderStatus == "Delivered")
                    .Sum(oi =&gt; oi.Quantity * oi.UnitPrice),
                AverageSellingPrice = p.OrderItems
                    .Where(oi =&gt; oi.Order.OrderStatus == "Delivered")
                    .Average(oi =&gt; (decimal?)oi.UnitPrice) ?? 0
            })
            .Where(p =&gt; p.TotalOrders &gt; 0)
            .OrderByDescending(p =&gt; p.TotalRevenue)
            .AsNoTracking()
            .ToListAsync();
    }

    // Customer purchasing patterns
    public async Task&lt;IEnumerable&lt;CustomerPurchasePatternDto&gt;&gt; GetCustomerPurch
    {
        return await _context.Customers
            .Where(c =&gt; c.Orders.Any(o =&gt; o.OrderStatus == "Delivered"))
            .Select(c =&gt; new CustomerPurchasePatternDto
            {
                CustomerID = c.CustomerID,
                CustomerName = $"{c.FirstName} {c.LastName}",
                FirstOrderDate = c.Orders
                    .Where(o =&gt; o.OrderStatus == "Delivered")
                    .Min(o =&gt; o.OrderDate),
                LastOrderDate = c.Orders
```

```
                    .Where(o =&gt; o.OrderStatus == "Delivered")
                    .Max(o =&gt; o.OrderDate),
                TotalOrders = c.Orders.Count(o =&gt; o.OrderStatus == "Delivered"),
                TotalSpent = c.Orders
                    .Where(o =&gt; o.OrderStatus == "Delivered")
                    .Sum(o =&gt; o.TotalAmount),
                AverageOrderValue = c.Orders
                    .Where(o =&gt; o.OrderStatus == "Delivered")
                    .Average(o =&gt; o.TotalAmount),
                DaysBetweenFirstAndLast = EF.Functions.DateDiffDay(
                    c.Orders.Where(o =&gt; o.OrderStatus == "Delivered").Min(o =&gt; o.Orde
                    c.Orders.Where(o =&gt; o.OrderStatus == "Delivered").Max(o =&gt; o.Orde
                )
            })
            .OrderByDescending(c =&gt; c.TotalSpent)
            .AsNoTracking()
            .ToListAsync();
    }
}
```

Performance Optimization

Query Optimization Techniques:

```
public class OptimizedQueryService
{
    private readonly LearningDbContext _context;

    public OptimizedQueryService(LearningDbContext context)
    {
        _context = context;
    }

    // Compiled queries for frequently used operations
    private static readonly Func&lt;LearningDbContext, int, Task&lt;Customer?&gt;&gt; GetCu
        EF.CompileAsyncQuery((LearningDbContext context, int id) =&gt;
            context.Customers
                .Include(c =&gt; c.Orders)
                .FirstOrDefault(c =&gt; c.CustomerID == id));

    private static readonly Func&lt;LearningDbContext, string, Task&lt;Customer?&gt;&gt; Ge
        EF.CompileAsyncQuery((LearningDbContext context, string email) =&gt;
            context.Customers.FirstOrDefault(c =&gt; c.Email == email));

    public async Task&lt;Customer?&gt; GetCustomerByIdOptimizedAsync(int id)
    {
        return await GetCustomerByIdCompiled(_context, id);
    }

    public async Task&lt;Customer?&gt; GetCustomerByEmailOptimizedAsync(string email)
    {
        return await GetCustomerByEmailCompiled(_context, email);
    }

    // Projection to avoid loading unnecessary data
```

```
public async Task&lt;IEnumerable&lt;CustomerSummaryDto&gt;&gt; GetCustomerSummariesAsyn
{
    return await _context.Customers
        .Select(c =&gt; new CustomerSummaryDto
        {
            CustomerID = c.CustomerID,
            FullName = c.FirstName + " " + c.LastName,
            Email = c.Email,
            OrderCount = c.Orders.Count(),
            LastOrderDate = c.Orders.Max(o =&gt; (DateTime?)o.OrderDate)
        })
        .AsNoTracking()
        .ToListAsync();
}

// Split queries for large includes
public async Task&lt;IEnumerable&lt;Order&gt;&gt; GetOrdersWithAllDataAsync(int custome
{
    return await _context.Orders
        .AsSplitQuery()
        .Include(o =&gt; o.Customer)
        .Include(o =&gt; o.OrderItems)
            .ThenInclude(oi =&gt; oi.Product)
        .Where(o =&gt; o.CustomerID == customerId)
        .AsNoTracking()
        .ToListAsync();
}

// Bulk operations (EF Core 7+)
public async Task&lt;int&gt; BulkUpdateOrderStatusAsync(string fromStatus, string toSta
{
    return await _context.Orders
        .Where(o =&gt; o.OrderStatus == fromStatus)
        .ExecuteUpdateAsync(s =&gt; s.SetProperty(o =&gt; o.OrderStatus, toStatus));
}

public async Task&lt;int&gt; BulkDeleteOldOrdersAsync(DateTime cutoffDate)
{
    return await _context.Orders
        .Where(o =&gt; o.OrderDate &lt; cutoffDate &amp;&amp; o.OrderStatus == "Cancell
        .ExecuteDeleteAsync();
}

// Raw SQL for complex operations
public async Task&lt;IEnumerable&lt;MonthlySalesReportDto&gt;&gt; GetMonthlySalesReport
{
    return await _context.Database
        .SqlQueryRaw&lt;MonthlySalesReportDto&gt;(@"
            SELECT
                YEAR(OrderDate) as Year,
                MONTH(OrderDate) as Month,
                DATENAME(MONTH, OrderDate) as MonthName,
                COUNT(*) as OrderCount,
                SUM(TotalAmount) as Revenue,
                AVG(TotalAmount) as AverageOrderValue
            FROM Orders
```

```
                WHERE OrderStatus = 'Delivered'
                    AND OrderDate &gt;= DATEADD(YEAR, -2, GETDATE())
                GROUP BY YEAR(OrderDate), MONTH(OrderDate), DATENAME(MONTH, OrderDate)
                ORDER BY Year DESC, Month DESC")
            .ToListAsync();
    }

    // Stored procedure execution
    public async Task&lt;int&gt; ExecuteMonthlyMaintenanceAsync()
    {
        return await _context.Database.ExecuteSqlRawAsync(
            "EXEC sp_MonthlyMaintenance @ProcessDate = {0}",
            DateTime.UtcNow);
    }
}
```

Monitoring and Logging:

```
// Custom logging for EF Core queries
public class QueryLoggingInterceptor : IDbCommandInterceptor
{
    private readonly ILogger&lt;QueryLoggingInterceptor&gt; _logger;

    public QueryLoggingInterceptor(ILogger&lt;QueryLoggingInterceptor&gt; logger)
    {
        _logger = logger;
    }

    public override ValueTask&lt;DbDataReader&gt; ReaderExecutedAsync(
        DbCommand command, CommandExecutedEventData eventData,
        DbDataReader result, CancellationToken cancellationToken = default)
    {
        if (eventData.Duration &gt; TimeSpan.FromSeconds(1))
        {
            _logger.LogWarning(
                "Slow query detected: {Duration}ms - {CommandText}",
                eventData.Duration.TotalMilliseconds,
                command.CommandText);
        }

        return new ValueTask&lt;DbDataReader&gt;(result);
    }
}

// Performance monitoring service
public class PerformanceMonitoringService
{
    private readonly LearningDbContext _context;
    private readonly ILogger&lt;PerformanceMonitoringService&gt; _logger;

    public PerformanceMonitoringService(LearningDbContext context, ILogger&lt;PerformanceMo
    {
        _context = context;
        _logger = logger;
    }
```

```
    public async Task&lt;QueryPerformanceReport&gt; GeneratePerformanceReportAsync()
    {
        var stopwatch = System.Diagnostics.Stopwatch.StartNew();

        // Test various query patterns
        var customerCount = await _context.Customers.CountAsync();
        var customerTime = stopwatch.ElapsedMilliseconds;

        stopwatch.Restart();
        var orderCount = await _context.Orders.CountAsync();
        var orderTime = stopwatch.ElapsedMilliseconds;

        stopwatch.Restart();
        var complexQuery = await _context.Customers
            .Include(c =&gt; c.Orders)
                .ThenInclude(o =&gt; o.OrderItems)
            .Take(10)
            .ToListAsync();
        var complexTime = stopwatch.ElapsedMilliseconds;

        return new QueryPerformanceReport
        {
            CustomerCountQuery = customerTime,
            OrderCountQuery = orderTime,
            ComplexIncludeQuery = complexTime,
            TotalCustomers = customerCount,
            TotalOrders = orderCount,
            GeneratedAt = DateTime.UtcNow
        };
    }
}
```

Part 9: Azure Deployment

Preparing for Production Deployment

Environment-Specific Configuration:

```
// Production-ready Program.cs
var builder = WebApplication.CreateBuilder(args);

// Add services
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure Entity Framework with different strategies per environment
if (builder.Environment.IsProduction())
{
    // Production configuration
    builder.Services.AddDbContext&lt;LearningDbContext&gt;(options =&gt;
    {
        var connectionString = builder.Configuration.GetConnectionString("Production");
```

```
        options.UseSqlServer(connectionString, sqlOptions =&gt;
        {
            sqlOptions.CommandTimeout(30);
            sqlOptions.EnableRetryOnFailure(
                maxRetryCount: 3,
                maxRetryDelay: TimeSpan.FromSeconds(10),
                errorNumbersToAdd: null);
        });

        // Production optimizations
        options.EnableServiceProviderCaching();
        options.EnableSensitiveDataLogging(false);
    });
}
else if (builder.Environment.IsStaging())
{
    // Staging configuration
    builder.Services.AddDbContext&lt;LearningDbContext&gt;(options =&gt;
    {
        var connectionString = builder.Configuration.GetConnectionString("Staging");
        options.UseSqlServer(connectionString);
        options.EnableDetailedErrors();
    });
}
else
{
    // Development configuration
    builder.Services.AddDbContext&lt;LearningDbContext&gt;(options =&gt;
    {
        var connectionString = builder.Configuration.GetConnectionString("Development");
        options.UseSqlServer(connectionString);
        options.EnableSensitiveDataLogging();
        options.EnableDetailedErrors();
        options.LogTo(Console.WriteLine);
    });
}

// Add health checks
builder.Services.AddHealthChecks()
    .AddDbContextCheck&lt;LearningDbContext&gt;();

// Register application services
builder.Services.AddScoped&lt;ICustomerRepository, CustomerRepository&gt;();
builder.Services.AddScoped&lt;ICustomerService, CustomerService&gt;();

var app = builder.Build();

// Configure pipeline
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();

    // Auto-migrate in development only
    await EnsureDatabaseAsync(app.Services);
}
```

```
app.UseHttpsRedirection();
app.UseHealthChecks("/health");
app.UseAuthorization();
app.MapControllers();

await app.RunAsync();

// Helper method for database initialization
static async Task EnsureDatabaseAsync(IServiceProvider services)
{
    using var scope = services.CreateScope();
    var context = scope.ServiceProvider.GetRequiredService&lt;LearningDbContext&gt;();
    var logger = scope.ServiceProvider.GetRequiredService&lt;ILogger&lt;Program&gt;&gt;();

    try
    {
        await context.Database.MigrateAsync();
        logger.LogInformation("Database migration completed successfully");
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "An error occurred while migrating the database");
        throw;
    }
}
```

Azure SQL Configuration and Security

Secure Connection String Management:

```
{
  "ConnectionStrings": {
    "Development": "Server=localhost;Database=LearningEFCore_Dev;Trusted_Connection=True;Tr
    "Staging": "Server=tcp:learningsql-staging.database.windows.net,1433;Initial Catalog=Le
    "Production": "Server=tcp:learningsql-prod.database.windows.net,1433;Initial Catalog=Le
  },
  "KeyVault": {
    "VaultUrl": "https://learningsql-kv.vault.azure.net/"
  }
}
```

Using Azure Key Vault for Secrets:

```
// Configure Key Vault in Program.cs
if (builder.Environment.IsProduction() || builder.Environment.IsStaging())
{
    var keyVaultUrl = builder.Configuration["KeyVault:VaultUrl"];
    if (!string.IsNullOrEmpty(keyVaultUrl))
    {
        builder.Configuration.AddAzureKeyVault(
            new Uri(keyVaultUrl),
            new DefaultAzureCredential());
    }
```

```
}

// Custom configuration class
public class DatabaseConfiguration
{
    public string ConnectionString { get; set; } = string.Empty;
    public int CommandTimeout { get; set; } = 30;
    public int MaxRetryCount { get; set; } = 3;
    public bool EnableRetryOnFailure { get; set; } = true;
    public bool EnableSensitiveDataLogging { get; set; } = false;
}

// Configure with options pattern
builder.Services.Configure&lt;DatabaseConfiguration&gt;(
    builder.Configuration.GetSection("Database"));
```

CI/CD Pipeline Configuration

GitHub Actions Workflow (.github/workflows/deploy.yml):

```
name: Deploy to Azure

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

env:
  AZURE_WEBAPP_NAME: learningefc-api
  AZURE_WEBAPP_PACKAGE_PATH: '.'
  DOTNET_VERSION: '8.x'

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Restore dependencies
      run: dotnet restore

    - name: Build
      run: dotnet build --no-restore --configuration Release

    - name: Test
      run: dotnet test --no-build --configuration Release --verbosity normal

    - name: Generate migration scripts
```

```
      run: |
        dotnet ef migrations script --startup-project src/LearningEFCore \
          --project src/LearningEFCore \
          --output migration.sql \
          --idempotent

    - name: Upload migration script
      uses: actions/upload-artifact@v3
      with:
        name: migration-script
        path: migration.sql

    - name: Publish
      run: dotnet publish --no-build --configuration Release --output ./publish

    - name: Upload artifact
      uses: actions/upload-artifact@v3
      with:
        name: webapp
        path: ./publish

  deploy-staging:
    needs: build-and-test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: staging

    steps:
    - name: Download artifact
      uses: actions/download-artifact@v3
      with:
        name: webapp

    - name: Download migration script
      uses: actions/download-artifact@v3
      with:
        name: migration-script

    - name: Deploy to staging
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}-staging
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE_STAGING }}
        package: .

    - name: Run database migration
      uses: azure/sql-action@v1
      with:
        server-name: learningsql-staging.database.windows.net
        connection-string: ${{ secrets.AZURE_SQL_CONNECTION_STRING_STAGING }}
        sql-file: './migration.sql'

  deploy-production:
    needs: [build-and-test, deploy-staging]
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
```

```
    environment: production

    steps:
    - name: Download artifact
      uses: actions/download-artifact@v3
      with:
        name: webapp

    - name: Download migration script
      uses: actions/download-artifact@v3
      with:
        name: migration-script

    - name: Deploy to production
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: .

    - name: Run database migration
      uses: azure/sql-action@v1
      with:
        server-name: learningsql-prod.database.windows.net
        connection-string: ${{ secrets.AZURE_SQL_CONNECTION_STRING }}
        sql-file: './migration.sql'
```

Part 10: Troubleshooting and Best Practices

Common Issues and Solutions

Connection String Issues:

```
// Connection string troubleshooting helper
public class ConnectionStringValidator
{
    public static async Task&lt;bool&gt; TestConnectionAsync(string connectionString)
    {
        try
        {
            using var connection = new SqlConnection(connectionString);
            await connection.OpenAsync();

            using var command = connection.CreateCommand();
            command.CommandText = "SELECT 1";
            var result = await command.ExecuteScalarAsync();

            return result?.ToString() == "1";
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Connection test failed: {ex.Message}");
            return false;
```

```
        }
    }

    public static void ValidateConnectionString(string connectionString)
    {
        if (string.IsNullOrWhiteSpace(connectionString))
            throw new ArgumentException("Connection string cannot be null or empty");

        var builder = new SqlConnectionStringBuilder(connectionString);

        if (string.IsNullOrWhiteSpace(builder.DataSource))
            throw new ArgumentException("Server name is required");

        if (string.IsNullOrWhiteSpace(builder.InitialCatalog))
            throw new ArgumentException("Database name is required");

        Console.WriteLine($"Server: {builder.DataSource}");
        Console.WriteLine($"Database: {builder.InitialCatalog}");
        Console.WriteLine($"Authentication: {(builder.IntegratedSecurity ? "Windows" : "SQL
    }
}
```

Migration Issues:

```
# Common EF Core troubleshooting commands

# Check current migration status
dotnet ef migrations list

# View what changes would be applied
dotnet ef migrations script --from LastAppliedMigration

# Reset migrations (development only)
dotnet ef database drop --force
dotnet ef migrations remove --force
dotnet ef migrations add InitialCreate
dotnet ef database update

# Fix "pending migration" errors
dotnet ef database update --verbose

# Generate idempotent scripts for production
dotnet ef migrations script --idempotent --output deploy.sql
```

Performance Troubleshooting:

```
// Query performance analyzer
public class QueryPerformanceAnalyzer
{
    private readonly LearningDbContext _context;
    private readonly ILogger&lt;QueryPerformanceAnalyzer&gt; _logger;

    public QueryPerformanceAnalyzer(LearningDbContext context, ILogger&lt;QueryPerformanceA
    {
        _context = context;
```

```
        _logger = logger;
    }

    public async Task&lt;string&gt; AnalyzeQueryPlanAsync(string sql)
    {
        try
        {
            // Enable actual execution plan
            await _context.Database.ExecuteSqlRawAsync("SET STATISTICS IO ON");
            await _context.Database.ExecuteSqlRawAsync("SET STATISTICS TIME ON");

            var planQuery = $@"
                SET SHOWPLAN_ALL ON
                {sql}
                SET SHOWPLAN_ALL OFF";

            var plan = await _context.Database
                .SqlQueryRaw&lt;string&gt;(planQuery)
                .ToListAsync();

            return string.Join("\n", plan);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error analyzing query plan for: {Sql}", sql);
            return $"Error: {ex.Message}";
        }
    }

    public async Task&lt;Dictionary&lt;string, object&gt;&gt; GetDatabaseStatsAsync()
    {
        var stats = new Dictionary&lt;string, object&gt;();

        try
        {
            // Get table sizes
            var tableSizes = await _context.Database
                .SqlQueryRaw&lt;TableSizeInfo&gt;(@"
                    SELECT
                        t.name AS TableName,
                        p.rows AS RowCount,
                        SUM(a.total_pages) * 8 AS TotalSpaceKB
                    FROM sys.tables t
                    INNER JOIN sys.indexes i ON t.object_id = i.object_id
                    INNER JOIN sys.partitions p ON i.object_id = p.object_id AND i.index_id
                    INNER JOIN sys.allocation_units a ON p.partition_id = a.container_id
                    WHERE t.name IN ('Customers', 'Orders', 'OrderItems', 'Products')
                    GROUP BY t.name, p.rows
                    ORDER BY p.rows DESC")
                .ToListAsync();

            stats["TableSizes"] = tableSizes;

            // Get index usage
            var indexUsage = await _context.Database
                .SqlQueryRaw&lt;IndexUsageInfo&gt;(@"
```

```
                    SELECT
                        OBJECT_NAME(s.object_id) AS TableName,
                        i.name AS IndexName,
                        s.user_seeks,
                        s.user_scans,
                        s.user_lookups,
                        s.user_updates
                    FROM sys.dm_db_index_usage_stats s
                    INNER JOIN sys.indexes i ON s.object_id = i.object_id AND s.index_id =
                    WHERE OBJECT_NAME(s.object_id) IN ('Customers', 'Orders', 'OrderItems',
                .ToListAsync();

            stats["IndexUsage"] = indexUsage;

        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error getting database stats");
        }

        return stats;
    }
}

public class TableSizeInfo
{
    public string TableName { get; set; } = string.Empty;
    public long RowCount { get; set; }
    public long TotalSpaceKB { get; set; }
}

public class IndexUsageInfo
{
    public string TableName { get; set; } = string.Empty;
    public string IndexName { get; set; } = string.Empty;
    public long UserSeeks { get; set; }
    public long UserScans { get; set; }
    public long UserLookups { get; set; }
    public long UserUpdates { get; set; }
}
```

Production Best Practices

Error Handling and Resilience:

```
// Resilient database service wrapper
public class ResilientDatabaseService&lt;T&gt; where T : class
{
    private readonly IRepository&lt;T&gt; _repository;
    private readonly ILogger&lt;ResilientDatabaseService&lt;T&gt;&gt; _logger;
    private readonly RetryPolicy _retryPolicy;

    public ResilientDatabaseService(
        IRepository&lt;T&gt; repository,
        ILogger&lt;ResilientDatabaseService&lt;T&gt;&gt; logger)
```

```
    {
        _repository = repository;
        _logger = logger;
        _retryPolicy = new RetryPolicy(3, TimeSpan.FromSeconds(2));
    }

    public async Task&lt;TResult&gt; ExecuteWithRetryAsync&lt;TResult&gt;(
        Func&lt;Task&lt;TResult&gt;&gt; operation,
        string operationName)
    {
        return await _retryPolicy.ExecuteAsync(async () =&gt;
        {
            try
            {
                _logger.LogDebug("Executing {OperationName}", operationName);
                var result = await operation();
                _logger.LogDebug("Successfully executed {OperationName}", operationName);
                return result;
            }
            catch (SqlException ex) when (IsTransientError(ex))
            {
                _logger.LogWarning(ex, "Transient error in {OperationName}, will retry", op
                throw;
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Non-transient error in {OperationName}", operationNam
                throw;
            }
        });
    }

    private static bool IsTransientError(SqlException ex)
    {
        // Common transient error codes
        var transientErrors = new[] { 2, 53, 121, 233, 997, 11001, 40197, 40501, 40613 };
        return transientErrors.Contains(ex.Number);
    }
}

public class RetryPolicy
{
    private readonly int _maxRetries;
    private readonly TimeSpan _baseDelay;

    public RetryPolicy(int maxRetries, TimeSpan baseDelay)
    {
        _maxRetries = maxRetries;
        _baseDelay = baseDelay;
    }

    public async Task&lt;T&gt; ExecuteAsync&lt;T&gt;(Func&lt;Task&lt;T&gt;&gt; operation)
    {
        var attempt = 0;
        while (true)
        {
```

```
            try
            {
                return await operation();
            }
            catch (Exception) when (attempt &lt; _maxRetries)
            {
                attempt++;
                var delay = TimeSpan.FromMilliseconds(_baseDelay.TotalMilliseconds * Math.P
                await Task.Delay(delay);
            }
        }
    }
}
```

Monitoring and Health Checks:

```
// Custom health check for database
public class DatabaseHealthCheck : IHealthCheck
{
    private readonly LearningDbContext _context;
    private readonly ILogger&lt;DatabaseHealthCheck&gt; _logger;

    public DatabaseHealthCheck(LearningDbContext context, ILogger&lt;DatabaseHealthCheck&gt
    {
        _context = context;
        _logger = logger;
    }

    public async Task&lt;HealthCheckResult&gt; CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default)
    {
        try
        {
            // Simple connectivity test
            await _context.Database.ExecuteSqlRawAsync("SELECT 1", cancellationToken);

            // Test table access
            var customerCount = await _context.Customers.CountAsync(cancellationToken);

            var data = new Dictionary&lt;string, object&gt;
            {
                ["CustomerCount"] = customerCount,
                ["DatabaseName"] = _context.Database.GetDbConnection().Database,
                ["ServerVersion"] = _context.Database.GetDbConnection().ServerVersion
            };

            return HealthCheckResult.Healthy("Database is accessible", data);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Database health check failed");
            return HealthCheckResult.Unhealthy("Database is not accessible", ex);
        }
    }
}
```

```
// Register health checks
builder.Services.AddHealthChecks()
    .AddCheck&lt;DatabaseHealthCheck&gt;("database")
    .AddDbContextCheck&lt;LearningDbContext&gt;("efcore");
```

Security Best Practices

Secure Configuration:

```
// Secure database configuration
public class SecureDatabaseConfiguration
{
    public static void ConfigureSecureDefaults(DbContextOptionsBuilder options, string conn
    {
        options.UseSqlServer(connectionString, sqlOptions =&gt;
        {
            // Security settings
            sqlOptions.CommandTimeout(30);

            // Connection resilience
            sqlOptions.EnableRetryOnFailure(
                maxRetryCount: 3,
                maxRetryDelay: TimeSpan.FromSeconds(10),
                errorNumbersToAdd: null);
        });

        // Disable sensitive data logging in production
        options.EnableSensitiveDataLogging(false);

        // Enable service provider caching for performance
        options.EnableServiceProviderCaching();

        // Configure warnings
        options.ConfigureWarnings(warnings =&gt;
        {
            warnings.Throw(RelationalEventId.QueryClientEvaluationWarning);
            warnings.Throw(CoreEventId.IncludeIgnoredWarning);
        });
    }
}

// Input validation and SQL injection prevention
public class SecureQueryService
{
    private readonly LearningDbContext _context;

    public SecureQueryService(LearningDbContext context)
    {
        _context = context;
    }

    // Always use parameterized queries
    public async Task&lt;IEnumerable&lt;Customer&gt;&gt; SearchCustomersSecureAsync(string
    {
```

```
        // Validate input
        if (string.IsNullOrWhiteSpace(searchTerm))
            return Enumerable.Empty&lt;Customer&gt;();

        // Sanitize input
        searchTerm = searchTerm.Trim();
        if (searchTerm.Length &gt; 50)
            searchTerm = searchTerm.Substring(0, 50);

        // Use parameterized query (EF Core handles this automatically)
        return await _context.Customers
            .Where(c =&gt; c.FirstName.Contains(searchTerm) ||
                       c.LastName.Contains(searchTerm) ||
                       c.Email.Contains(searchTerm))
            .Take(100) // Limit results
            .AsNoTracking()
            .ToListAsync();
    }

    // When using raw SQL, always parameterize
    public async Task&lt;IEnumerable&lt;CustomerRevenueDto&gt;&gt; GetCustomerRevenueSecure
    {
        // Validate date range
        if (endDate &lt; startDate)
            throw new ArgumentException("End date must be after start date");

        if ((endDate - startDate).TotalDays &gt; 365)
            throw new ArgumentException("Date range cannot exceed 365 days");

        return await _context.Database
            .SqlQueryRaw&lt;CustomerRevenueDto&gt;(@"
                SELECT
                    c.CustomerID,
                    c.FirstName + ' ' + c.LastName AS CustomerName,
                    SUM(o.TotalAmount) AS TotalRevenue
                FROM Customers c
                    INNER JOIN Orders o ON c.CustomerID = o.CustomerID
                WHERE o.OrderDate &gt;= @startDate
                    AND o.OrderDate &lt; @endDate
                    AND o.OrderStatus = 'Delivered'
                GROUP BY c.CustomerID, c.FirstName, c.LastName
                ORDER BY TotalRevenue DESC",
                new SqlParameter("@startDate", startDate),
                new SqlParameter("@endDate", endDate))
            .ToListAsync();
    }
}
```

Conclusion

This comprehensive guide covers everything needed to master SQL and Entity Framework Core for C# development, from initial setup through production deployment on Azure. The combination of solid SQL fundamentals with modern EF Core patterns provides a robust foundation for building scalable, maintainable data-driven applications.

Key takeaways include understanding relational design principles, mastering T-SQL for Azure SQL Database, implementing clean architecture patterns with EF Core, and following security and performance best practices for production

deployments.

The 8-week learning plan provides a structured approach to building expertise incrementally, while the troubleshooting section ensures you can handle common challenges effectively. With this knowledge, you'll be well-equipped to build professional-grade applications that leverage both the productivity of EF Core and the performance capabilities of well-designed SQL databases.