**1. Introduction**

* Define **One-to-Many** and **Many-to-Many** relationships between tables.
* Understand **Navigation Properties** in EF Core.
* Use **LINQ queries** for filtering, joining, and grouping data.
* Perform complex data retrieval using EF Core’s query syntax.

**Part 1 – Entity Relationships**

**2. What Are Relationships in Databases?**

A **relationship** defines how two tables are connected to each other.

Entity Framework Core allows you to represent these connections using **navigation properties** in your C# classes.

**3. Types of Relationships**

|  |  |  |
| --- | --- | --- |
| **Relationship Type** | **Description** | **Example** |
| **One-to-One** | A single record in one table relates to a single record in another. | Each student has one address. |
| **One-to-Many** | One record relates to multiple records in another table. | One course has many students. |
| **Many-to-Many** | Multiple records relate to multiple records in another table. | Students can enroll in multiple courses. |

**4. One-to-Many Example**

In this example:

* One **Course** can have multiple **Students**.
* Each **Student** belongs to one **Course**.

**Course.cs**

using System.ComponentModel.DataAnnotations;

namespace StudentApi.Models

{

public class Course

{

[Key]

public int CourseId { get; set; }

[Required]

[StringLength(100)]

public string CourseName { get; set; } = string.Empty;

public ICollection<Student> Students { get; set; } = new List<Student>();

}

}

**Student.cs**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace StudentApi.Models

{

public class Student

{

[Key]

public int Id { get; set; }

[Required, StringLength(50)]

public string Name { get; set; } = string.Empty;

[Range(1, 100)]

public int Age { get; set; }

[StringLength(10)]

public string? Grade { get; set; }

// Foreign Key

[ForeignKey("Course")]

public int CourseId { get; set; }

// Navigation Property

public Course? Course { get; set; }

}

}

**Explanation:**

* CourseId in Student acts as a **foreign key**.
* Each Student references a Course.
* The Course class has a collection of Students.

**5. Updating DbContext**

Update your context to include the new model:

public DbSet<Course> Courses { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Course>()

.HasMany(c => c.Students)

.WithOne(s => s.Course)

.HasForeignKey(s => s.CourseId)

.OnDelete(DeleteBehavior.Cascade);

}

This ensures **referential integrity**:  
If a course is deleted, all its students are deleted automatically.

**6. SQL Representation**

When you run the migration or scaffold, EF Core will create two tables:

CREATE TABLE Courses (

CourseId INT IDENTITY(1,1) PRIMARY KEY,

CourseName NVARCHAR(100) NOT NULL

);

CREATE TABLE Students (

Id INT IDENTITY(1,1) PRIMARY KEY,

Name NVARCHAR(50) NOT NULL,

Age INT NOT NULL,

Grade NVARCHAR(10),

CourseId INT FOREIGN KEY REFERENCES Courses(CourseId)

);

**Part 2 – LINQ (Language Integrated Query)**

**7. Introduction to LINQ**

**LINQ** allows you to query collections (like Lists, Arrays, or DbSets) using C# syntax instead of SQL.  
EF Core translates LINQ into SQL automatically.

**8. LINQ Query Types**

|  |  |
| --- | --- |
| **Query Type** | **Description** |
| **Method Syntax** | Uses methods like .Where(), .Select(), .OrderBy() |
| **Query Syntax** | Uses SQL-like keywords (from, where, select) |

Both produce the same result.

**9. Basic LINQ Examples**

**1. Retrieve all students**

var students = \_context.Students.ToList();

**2. Filter by Grade**

var topStudents = \_context.Students

.Where(s => s.Grade == "A")

.ToList();

**3. Sort Students by Age**

var sorted = \_context.Students.OrderBy(s => s.Age).ToList();

**4. Select Specific Columns**

var names = \_context.Students.Select(s => s.Name).ToList();

**10. LINQ with Relationships (Join Queries)**

**Join Students and Courses**

var studentCourses = from s in \_context.Students

join c in \_context.Courses

on s.CourseId equals c.CourseId

select new

{

StudentName = s.Name,

CourseName = c.CourseName,

Grade = s.Grade

};

**Equivalent Method Syntax:**

var result = \_context.Students

.Include(s => s.Course)

.Select(s => new

{

s.Name,

s.Course.CourseName,

s.Grade

})

.ToList();

**11. LINQ Aggregation and Grouping**

**Group by Course**

var groupByCourse = \_context.Students

.GroupBy(s => s.Course.CourseName)

.Select(g => new

{

Course = g.Key,

StudentCount = g.Count()

})

.ToList();

**Aggregate Functions:**

var stats = new

{

TotalStudents = \_context.Students.Count(),

MaxAge = \_context.Students.Max(s => s.Age),

AvgAge = \_context.Students.Average(s => s.Age)

};

**12. LINQ Query Translation**

EF Core converts LINQ to SQL automatically.  
For example:

\_context.Students.Where(s => s.Grade == "A");

Will generate:

SELECT \* FROM Students WHERE Grade = 'A';

This abstraction allows developers to focus on **C# logic** instead of SQL.

**13. Eager Loading vs Lazy Loading**

|  |  |
| --- | --- |
| **Type** | **Description** |
| **Eager Loading** | Loads related data immediately using .Include() |
| **Lazy Loading** | Loads related data only when accessed (requires setup) |

Example:

var students = \_context.Students.Include(s => s.Course).ToList();

This retrieves both Students and their Course data in one query.

**14. LINQ with Anonymous Types**

You can project specific fields into new anonymous objects:

var studentInfo = \_context.Students

.Select(s => new { s.Name, s.Grade })

.ToList();

This reduces data transfer and improves performance.

**15. LINQ Performance Tips**

* Use .ToList() **only at the end** of a query chain.
* Prefer **async methods** (ToListAsync()).
* Avoid querying in loops.
* Use **projection** to return only needed fields.
* Use **indexes** in SQL Server for large datasets.

**Mini Task for Day 5**

**Objective:**  
Create a Course table and link it to Student (One-to-Many relationship).

**Tasks:**

1. Update your model and StudentDBContext.
2. Create new endpoints in CoursesController:
   * GET /api/courses → List all courses
   * GET /api/courses/{id} → List students in that course
3. Write LINQ queries to:
   * List all students per course
   * Count students per course

**Snapshots :**

A screenshot of a computer

AI-generated content may be incorrect.

StudentCourseDB created and tables verified in SSMS

A screenshot of a computer

AI-generated content may be incorrect.

Project structure in VS Code with required folders

A screenshot of a computer

AI-generated content may be incorrect.

Connection string setup in appsettings.json

A computer screen shot of a program

AI-generated content may be incorrect.

DbContext class with DbSet<Student> and DbSet<Course>

A screen shot of a computer program

AI-generated content may be incorrect.

EF Core migration command executed successfully

A screen shot of a computer program

AI-generated content may be incorrect.

EF Core migration Files Created

A screenshot of a computer

AI-generated content may be incorrect.

Tables created in SQL Server after migration

A screenshot of a computer

AI-generated content may be incorrect.

Swagger UI to test API

GET /api/students tested successfully in SwaggerA screenshot of a computer

AI-generated content may be incorrect.