Advanced Lab Exercises: Complex CRUD Operations on a Single Entity Student

Objective:

To enhance proficiency with Entity Framework (EF) Core by implementing and practicing more complex CRUD operations on the Student entity. These exercises will cover advanced querying, data validation, handling large datasets, and working with transactions.

Lab 4: Advanced Querying with LINQ

Step 1: Filter Students by Specific Criteria

1. Modify the Main method to retrieve students who meet specific criteria, such as age greater than 20:

```
using System.Linq;
using (var context = new StudentContext())
{
    var students = context.Students
                          .Where(s => s.Age > 20)
                          .ToList();

    foreach (var student in students)
    {
        Console.WriteLine($"ID: {student.Id}, Name: {student.Name},
Age: {student.Age}, Email: {student.Email}");
    }
}
```

Step 2: Implement Sorting and Pagination

1. Implement sorting by Name and pagination to display students in pages of 5 records each:

```
using (var context = new StudentContext())
{
    int pageNumber = 1;
    int pageSize = 5;

    var students = context.Students
```

```
            .OrderBy(s => s.Name)
                            .Skip((pageNumber - 1) * pageSize)
                            .Take(pageSize)
                            .ToList();

    foreach (var student in students)
    {
        Console.WriteLine($"ID: {student.Id}, Name: {student.Name},
Age: {student.Age}, Email: {student.Email}");
    }
}
```

Lab 5: Data Validation

Step 1: Add Validation to the Student Entity

1. Modify the Student class to include validation attributes:

```
using System.ComponentModel.DataAnnotations;

public class Student
{
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [Range(18, 100)]
    public int Age { get; set; }

    [Required]
    [EmailAddress]
    public string Email { get; set; }
}
```

2. Implement validation logic in the Main method to check for validation errors before saving a student:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;

var student = new Student
{
    Name = "John Doe",
    Age = 17, // Invalid age to trigger validation error
    Email = "john.doe@example.com"
};

var context = new ValidationContext(student, null, null);
var results = new List<ValidationResult>();

bool isValid = Validator.TryValidateObject(student, context, results,
true);

if (isValid)
{
    using (var dbContext = new StudentContext())
    {
        dbContext.Students.Add(student);
        dbContext.SaveChanges();
        Console.WriteLine("Student record added.");
    }
}
else
{
    foreach (var validationResult in results)
    {
   Console.WriteLine(validationResult.ErrorMessage);
```

```
    }
}
```

Lab 6: Bulk Operations

Step 1: Add Multiple Students in a Single Transaction

1. Use a transaction to add multiple student records and roll back if any errors occur:

```
using (var context = new StudentContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            var students = new List<Student>
            {
                new Student { Name = "Alice Smith", Age = 22, Email = "alice.smith@example.com" },
                new Student { Name = "Bob Johnson", Age = 25, Email = "bob.johnson@example.com" },
                new Student { Name = "Charlie Brown", Age = 21, Email = "charlie.brown@example.com" }
            };

            context.Students.AddRange(students);
            context.SaveChanges();

            transaction.Commit();
            Console.WriteLine("Students added successfully.");
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.WriteLine($"Transaction failed: {ex.Message}");
        }
    }
```

}

Step 2: Bulk Update Student Records

1. Implement a bulk update to modify the email domain for all students:

```
using (var context = new StudentContext())
{
    var students = context.Students.ToList();

    foreach (var student in students)
    {
        student.Email = student.Email.Replace("@example.com",
"@newdomain.com");
    }

    context.SaveChanges();
    Console.WriteLine("Bulk update completed.");
}
```

Lab 7: Handling Concurrency

Step 1: Implement Concurrency Token in Student Entity

1. Modify the Student class to include a concurrency token:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public string Email { get; set;}

    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```

Step 2: Handle Concurrency Conflicts

1. Implement logic in the Main method to detect and handle concurrency conflicts:

```
using (var context = new StudentContext())
{
    var student = context.Students.FirstOrDefault(s => s.Name == 
"Alice Smith");

    if (student != null)
    {
        student.Age = 23; // Assume this is an updated value

        try
        {
            context.SaveChanges();
            Console.WriteLine("Student record updated.");
        }
        catch (DbUpdateConcurrencyException ex)
        {
            Console.WriteLine("Concurrency conflict detected. Please 
refresh and try again.");
        }
    }
}
```

Lab Completion

- After completing these advanced exercises, students should be proficient in handling more complex CRUD operations, implementing data validation, working with transactions, and managing concurrency in EF Core.
- Encourage experimentation with different scenarios and more complex queries to solidify understanding.