Лабораторная работа №15
Применение ORM для доступа к данным БД

1Цель работы

1.1 Научиться создавать приложение C# для организации доступа к БД.

1.2 Научиться проектировать приложение, использующее паттерн репозиторий и Dapper.

2 Литература

2.1 https://metanit.com/sharp/aspnet3.26.1.php

2.2 https://learn.microsoft.com/ru-ru/dotnet/framework/data/adonet/asynchronous-programming

3 Задание

3.1 Создание классов для работы с БД

3.1.1 Создать открытый класс DatabaseContext, управляющий подключением к БД:

```
public class DatabaseContext
{
    private readonly string _connectionString;

    public DatabaseContext(string server, string database, string login, string password)
    {
        // составление строки подключения _connectionString
    }

    public IDbConnection CreateConnection() => new SqlConnection(_connectionString);
}
```

3.1.2 Создать классы репозитория *Посетитель*Repository и *Жанр*Repository (будут обеспечивать чтение и запись данных таблиц Посетитель и Жанр).

Добавить в них поле _dbContext и конструктор, присваивающий значение полю:

```
private readonly DatabaseContext _dbContext;

public UserRepository(DatabaseContext dbContext)
{
    _dbContext = dbContext;
}
```

3.1.3 Создать в консольном приложении объекты типа DatabaseContext и классов репозитория.

3.2 Создание и реализация интерфейса репозитория

3.2.1 Создать обобщенный интерфейс репозитория, определяющий основные операции для работы с сущностями в БД:

```
public interface IRepository<T> where T : class
{
    Task<T> GetByIdAsync(int id);
    Task<IEnumerable<T>> GetAllAsync();
    Task<int> AddAsync(T entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(int id);
}
```

3.2.2 Добавить классы модели данных для представления сущностей Посетитель и Жанр с открытыми автосвойствами.

3.2.3 Указать, что классы репозитория реализуют интерфейс IRepository<T>. Используя рефакторинг, сгенерировать реализацию методов интерфейса.

3.3 Реализация методов на чтение данных

3.3.1 Реализовать метод GetByIdAsync в классах репозиториев. Метод должен возвращать объект по идентификатору.

Для реализации использовать метод Dapper QuerySingleOrDefaultAsync<Тип>.

3.3.2 Реализовать метод GetAllAsync в классах репозиториев. Метод должен возвращать список всех объектов.

Для реализации использовать метод Dapper QueryAsync<Тип>.

3.3.3 Проверить работу созданных методов, вызвав их в консольном приложении.

3.4 Реализация методов на добавление данных

3.4.1 Реализовать метод AddAsync в классах репозиториев. Метод должен возвращать идентификатор добавленного объекта.

Для реализации использовать метод Dapper ExecuteScalarAsync<int>.

3.4.2 Проверить работу созданных методов, вызвав их в консольном приложении.

3.5 Реализация методов на редактирование и удаление данных

3.5.1 Реализовать метод DeleteAsync в классах репозиториев. Метод должен удалять объект из БД по идентификатору.

Для реализации использовать метод Dapper ExecuteAsync.

3.5.2 Реализовать метод UpdateAsync в классах репозиториев. Метод должен изменять данные объекта с идентификатором объекта.

Для реализации использовать метод Dapper ExecuteAsync.

3.5.3 Проверить работу созданных методов, вызвав их в консольном приложении.

4 Порядок выполнения работы

4.1 Выполнить все задания из п.3 в консольном приложении на C#.

4.2 Ответить на контрольные вопросы.

5 Содержание отчета

5.1 Титульный лист

5.2 Цель работы

5.3 Ответы на контрольные вопросы

5.4 Вывод

6 Контрольные вопросы

6.1 Что такое Dapper и для чего используется?

6.2 Какие методы Dapper позволяют извлечь данные из БД?

6.3 Какие методы Dapper позволяют изменить данные в БД?

6.4 Зачем используетсяя паттерн «репозиторий»?