**Лабораторная работа №4. ORM и работа с данными в приложении. Работа с Entity Framework в .Net.**

**Цель работы:** Изучить основы работы с объектно-реляционным маппингом (ORM) и получить практические навыки по интеграции ORM в приложение для работы с базами данных. Научиться создавать модели данных, выполнять основные операции CRUD (Create, Read, Update, Delete), работать с отношениями между таблицами и использовать ORM для взаимодействия с базой данных в контексте предыдущих лабораторных работ.

**Теоретическая часть:**

**1. Что такое ORM?**

Объектно-реляционное маппирование (ORM, Object-Relational Mapping) — это технология программирования, которая позволяет преобразовывать данные между объектно-ориентированными моделями и реляционными базами данных. ORM упрощает работу с базами данных, позволяя разработчикам взаимодействовать с ними через объекты языка программирования, а не напрямую через SQL-запросы.

**Основные задачи ORM:**

* Преобразование объектов приложения в таблицы базы данных и обратно.
* Автоматизация создания SQL-запросов для выполнения CRUD-операций (Create, Read, Update, Delete).
* Управление отношениями между объектами (например, связь "один ко многим", "многие ко многим").
* Предоставление механизма кэширования и оптимизации запросов.

**2. Основные ORM для .NET**

* Entity Framework (EF) : Один из самых популярных ORM для платформы .NET. Поддерживает Code First, Database First и Model First подходы.
* Dapper : Минимальный ORM, который предоставляет только базовые возможности маппинга, но работает очень быстро.
* NHibernate : Порт Hibernate для .NET, поддерживает множество продвинутых функций.

В дальнейшем будет использоваться Entity Framework

Entity Framework — это ORM, предоставляемая Microsoft для работы с базами данных в .NET-приложениях. EF поддерживает три основных подхода к работе с базой данных:

* **Code First** : Создание модели данных на основе классов C# и автоматическое создание базы данных.
* **Database First** : Генерация классов C# на основе существующей базы данных.
* **Model First** : Создание модели данных в графическом редакторе и генерация базы данных на основе этой модели.

**2.1. Настройка проекта**

Чтобы начать работу с Entity Framework, сначала нужно установить необходимые пакеты NuGet. Для этого откройте консоль управления пакетами (Package Manager Console) и выполните следующие команды:

|  |
| --- |
| Install-Package Microsoft.EntityFrameworkCore  Install-Package Microsoft.EntityFrameworkCore.SqlServer # Для работы с SQL Server  Install-Package Microsoft.EntityFrameworkCore.Tools # Для создания миграций |

Если вы используете другую СУБД (например, PostgreSQL или MySQL), замените **Microsoft.EntityFrameworkCore.SqlServer** на соответствующий пакет.

**2.2. Создание модели данных**

Модель данных представляет собой набор классов, которые описывают структуру таблиц базы данных. Каждый класс соответствует таблице, а его свойства — столбцам.

Пример модели:

|  |
| --- |
| public class Blog  {  public int BlogId { get; set; } // Первичный ключ  public string Name { get; set; }  public string Url { get; set; }  // Навигационное свойство для связи "один ко многим"  public List<Post> Posts { get; set; }  }  public class Post  {  public int PostId { get; set; } // Первичный ключ  public string Title { get; set; }  public string Content { get; set; }  // Связь с Blog  public int BlogId { get; set; }  public Blog Blog { get; set; }  } |

**2.3. Создание контекста (DbContext)**

Контекст (**DbContext**) — это главный класс, который управляет взаимодействием с базой данных. Он содержит коллекции (**DbSet<T>**) для каждой сущности.

Пример контекста:

|  |
| --- |
| public class BloggingContext : DbContext  {  public DbSet<Blog> Blogs { get; set; }  public DbSet<Post> Posts { get; set; }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  // Указываем строку подключения к базе данных  optionsBuilder.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=Blogging;Trusted\_Connection=True;");  }  } |

**2.4. Создание миграций**

Entity Framework позволяет автоматически генерировать изменения структуры базы данных через механизм миграций.

Добавление новой миграции может выглядеть:

|  |
| --- |
| Add-Migration InitialCreate |

Применении миграции же выглядит как:

|  |
| --- |
| Update-Database |

После этих шагов EF создаст таблицы **Blogs** и **Posts** в вашей базе данных.

**2.5. CRUD-операции**

**a) Создание данных (Create)**

Добавление новых записей в базу данных:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = new Blog { Name = "My First Blog", Url = "https://example.com" };  context.Blogs.Add(blog);  context.SaveChanges(); // Сохранение изменений  } |

**b) Чтение данных (Read)**

Получение данных из базы данных:

|  |
| --- |
| using (var context = new BloggingContext())  {  // Получить все блоги  var blogs = context.Blogs.ToList();  // Получить конкретный блог по ID  var blog = context.Blogs.FirstOrDefault(b => b.BlogId == 1);  // Получить все посты конкретного блога  var posts = context.Posts.Where(p => p.BlogId == 1).ToList();  } |

**c) Обновление данных (Update)**

Изменение существующих записей:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs.FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  blog.Name = "Updated Blog Name";  context.SaveChanges(); // Сохранение изменений  }  } |

**d) Удаление данных (Delete)**

Удаление записей из базы данных:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs.FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  context.Blogs.Remove(blog);  context.SaveChanges(); // Сохранение изменений  }  } |

**2.6. Работа с отношениями**

Entity Framework поддерживает различные типы отношений между сущностями: "один к одному", "один ко многим" и "многие ко многим".

**a) Один ко многим**

В приведенной выше модели **Blog** имеет множество **Post**. Чтобы получить все посты конкретного блога:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs.Include(b => b.Posts).FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  foreach (var post in blog.Posts)  {  Console.WriteLine(post.Title);  }  }  } |

Здесь используется метод **Include**, чтобы загрузить связанные данные.

**b) Многие ко многим**

Для реализации отношения "многие ко многим" добавьте дополнительную сущность:

|  |
| --- |
| public class Tag  {  public int TagId { get; set; }  public string Name { get; set; }  public List<PostTag> PostTags { get; set; }  }  public class PostTag  {  public int PostId { get; set; }  public Post Post { get; set; }  public int TagId { get; set; }  public Tag Tag { get; set; }  } |

Теперь можно связывать посты с тегами через сущность **PostTag**.

**2.7. Ленивая загрузка (Lazy Loading)**

По умолчанию EF выполняет ленивую загрузку связанных данных. Например, если вы получаете блог, связанные посты будут загружены только при обращении к ним:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs.FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  foreach (var post in blog.Posts) // Здесь происходит ленивая загрузка  {  Console.WriteLine(post.Title);  }  }  } |

**2.8. Жадная загрузка (Eager Loading)**

Жадная загрузка позволяет загрузить связанные данные заранее:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs  .Include(b => b.Posts)  .FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  foreach (var post in blog.Posts)  {  Console.WriteLine(post.Title);  }  }  } |

**2.9. Явная загрузка (Explicit Loading)**

Явная загрузка позволяет контролировать момент загрузки связанных данных:

|  |
| --- |
| using (var context = new BloggingContext())  {  var blog = context.Blogs.FirstOrDefault(b => b.BlogId == 1);  if (blog != null)  {  context.Entry(blog).Collection(b => b.Posts).Load();  foreach (var post in blog.Posts)  {  Console.WriteLine(post.Title);  }  }  } |

**2.10. Работа с хранимыми процедурами**

Entity Framework также поддерживает вызов хранимых процедур:

|  |
| --- |
| using (var context = new BloggingContext())  {  var result = context.Database.SqlQuery<Blog>("EXEC GetBlogs").ToList();  } |

**2.11. Транзакции**

Для выполнения нескольких операций как единого целого используйте транзакции:

|  |
| --- |
| using (var context = new BloggingContext())  {  using (var transaction = context.Database.BeginTransaction())  {  try  {  var blog = new Blog { Name = "New Blog", Url = "https://newblog.com" };  context.Blogs.Add(blog);  var post = new Post { Title = "First Post", Content = "Hello World!", BlogId = blog.BlogId };  context.Posts.Add(post);  context.SaveChanges();  transaction.Commit(); // Фиксация изменений  }  catch  {  transaction.Rollback(); // Откат изменений  throw;  }  }  } |

**Практическое задание:**

**Добавление поддержки Entity Framework в проект RESTful API**

* Создайте новую базу данных или используйте существующую.
* Настройте DbContext для работы с таблицами/моделями, которые уже были реализованы в RESTful API.
* Используйте подход Code First , чтобы создать миграции и автоматически сгенерировать структуру базы данных на основе ваших моделей.
* Примените миграции для создания базы данных.

**Реализация CRUD-операций через Entity Framework в контроллерах RESTful API**

* Обновите контроллеры RESTful API, чтобы они использовали ApplicationDbContext вместо ручных SQL-запросов.
* Реализуйте методы для выполнения операций Create, Read, Update и Delete (CRUD) с использованием Entity Framework.

**Тестирование RESTful API с использованием Postman или Swagger**

* + Убедитесь, что все методы работают корректно.
  + Проверьте, что данные успешно сохраняются в базе данных и извлекаются из неё.

**Обновление клиента WPF для работы с обновленным RESTful API**

* + В клиентском приложении WPF обновите логику взаимодействия с RESTful API, чтобы она соответствовала новым методам.
  + Добавьте новые функциональные возможности:
    - **Создание новых записей:** Реализуйте форму для добавления новых данных.
    - **Изменение существующих записей:** Добавьте возможность редактирования данных.
    - **Удаление записей:** Реализуйте функцию удаления выбранных записей.
    - **Поиск записей:** Добавьте простой механизм фильтрации данных (например, поиск по имени или другому полю).
* Добавьте поддержку транзакций для сложных операций, где требуется изменение нескольких связанных сущностей одновременно.