Лабораторная работа №10

«Entity Framework, Microsoft SQL Server»

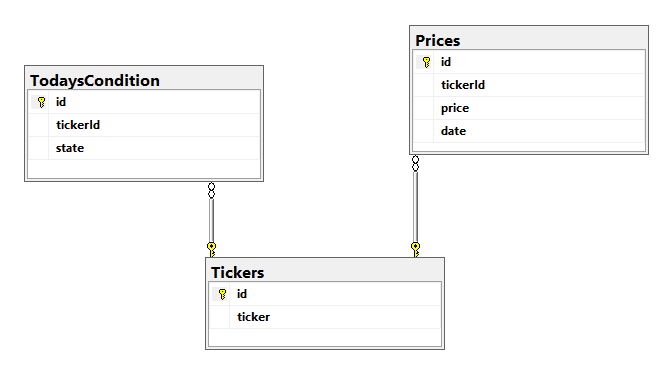
**Цели работы:**

1. Научиться работать c ORM - фреймворком Entity Framework средствами языка C#.
2. Получить базовые навыки работы с базами данных на примере Microsoft SQL Server.

**Задание№1**

Создайте приложение для загрузки цен акций из Лабораторной работы №9 и сохранения их в базу данных. По запросу пользователя необходимо выводить состояние акции на сегодняшний день по сравнению с предыдущим (выросла/упала).

Приложение должно заполнять таблицы в БД данными с сайта <https://finance.yahoo.com/>, используйте следующую схему БД или разработайте свою:



Для удобства используйте образ Docker c Microsoft SQL Server либо, при желании, воспользуйтесь любой SQL базой данных.

После заполнения базы данных приложение анализирует последнюю цену и цену на день раньше, в результате чего заполняет таблицу TodaysCondition.

Пользователь вводит строковый тикер и в ответ получает результат изменения цены – выросла/упала.

Графический интерфейс реализуйте по желанию.

**Теоретические сведения**

**Entity Framework**

Entity Framework представляет ORM-технологию (object-relational mapping - отображения данных на реальные объекты) от компании Microsoft для доступа к данным. Entity Framework Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Как технология доступа к данным Entity Framework Core работает поверх платформы .NET и поэтому может использоваться на различных платформах стека .NET. Это и стандартные платформы типа Windows Forms, консольные приложения, WPF, UWP и ASP.NET Core. При этом кроссплатформенная природа EF Core позволяет задействовать ее не только на ОС Windows, но и на Linux и Mac OS X.

Поскольку Entity Framework Core работает на основе платформы .NET, то он развивается вместе с данной платформой. Текущая версия EF Core - 8.0 была выпущена в ноябре 2023 года вместе с .NET 8. И технология продолжает развиваться.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер. По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с MS SQL Server, для SQLite, для PostgreSQL. Также имеются провайдеры от сторонних поставщиков, например, для MySQL.

Стоит отметить, что Entity Framework Core развивается как opensourse-проект, все файлы которого можно найти в репозитории на github по адресу <https://github.com/dotnet/efcore>.

Также стоит отметить, что EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнение запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

Основная функциональность Entity Framework Core сосредоточена в следующих пакетах:

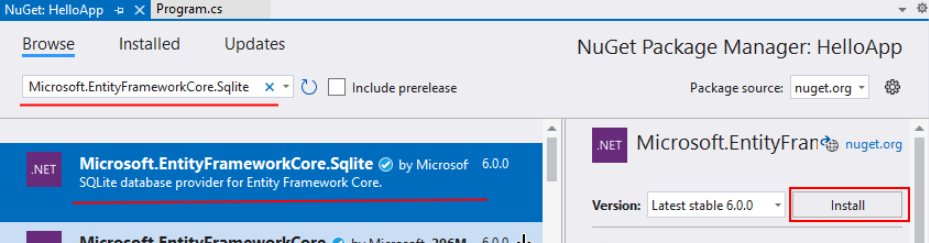
* Microsoft.EntityFrameworkCore:основной пакет EF Core
* Microsoft.EntityFrameworkCore.SqlServer: представляет функциональность провайдера для Microsoft SQL Server и SQL Azure
* Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQL Server
* Microsoft.EntityFrameworkCore.Sqlite: представляет функциональность провайдера для SQLite и включает нативные бинарные файлы для движка базы данных
* Microsoft.EntityFrameworkCore.Sqlite.Core: представляет функциональность провайдера для SQLite, но в отличие от предыдущего пакета не содержит нативные бинарные файлы для движка базы данных
* Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQLite
* Microsoft.EntityFrameworkCore.Cosmos: представляет функциональность провайдера для Azure Cosmos DB
* Microsoft.EntityFrameworkCore.InMemory: представляет функциональность провайдера базы данных в памяти
* Microsoft.EntityFrameworkCore.Tools: содержит команды EF Core PowerShell для Visual Studio Package Manager Console; применяется в Visual Studio для миграций и генерации классов по готовой бд
* Microsoft.EntityFrameworkCore.Design: содержит вспомогательные компоненты EF Core, применяемые в процессе разработки
* Microsoft.EntityFrameworkCore.Proxies: хранит функциональность для так называемой "ленивой загрузки" (lazy-loading) и прокси остлеживания изменений
* Microsoft.EntityFrameworkCore.Abstractions: содержит набор абстракций EF Core, которые не зависят от конкретной СУБД
* Microsoft.EntityFrameworkCore.Relational: хранит компоненты EF Core для провайдеров реляционных СУБД
* Microsoft.EntityFrameworkCore.Analyzers: содержит функционал анализаторов C# для EF Core

Платформу Entity Framework Core можно применять в различных технологиях стека .NET - консольных приложениях, программах на WinForms, WPF, UWP, веб-приложения ASP.NET и так далее. В данном случае мы будем рассматривать базовые моменты платформы на примере консольных приложений, как наиболее простых и не содержащих никакого лишнего кода.

Кроме того, EF Core может работать с различными системами баз данных. Здесь мы будем рассмотривать общие возможности на примере бд SQLite, как самой простой и удобной СУБД. И поскольку Entity Framework позволяет в большой степени писать универсальный код для подключения к разным СУБД, этот код можно будет применять и к другим СУБД. Однако, в последствии также затронем работу с другими СУБД в тех аспектах, где есть отличия.

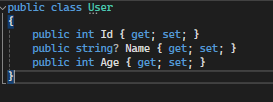
**Простое приложение**

Чтобы начать работать с EntityFramework Core, нам необходимо вначале добавить в проект Nuget-пакет EntityFramework Core. Однако в данном случае нам нужен не общий пакет для Entity Framework Core, а пакет для конкретной СУБД. Так, в данном случае мы будем использовать SQLite в качестве СУБД, поэтому нам надо добавить пакет Microsoft.EntityFrameworkCore.Sqlite. Например, с помощью .NET CLI или через инструменты работы с Nuget в Visual Studio:



Итак, необходимые пакеты добавлены. Теперь мы можем их использовать.

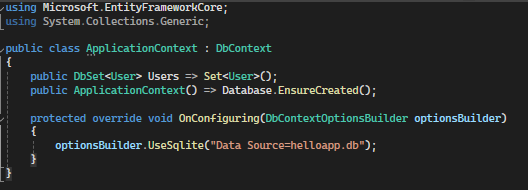
Далее нам надо определить модель, которая будет описывать данные. Пусть наше приложение будет посвящено работе с пользователями. Поэтому добавим в проект новый класс User:



Это обычный класс, который содержит несколько свойств. Каждое свойство будет сопоставляться с отдельным столбцом в таблице из бд.

Надо отметить, что Entity Framework требует определения ключа элемента для создания первичного ключа в таблице в бд. По умолчанию при генерации бд EF в качестве первичных ключей будет рассматривать свойства с именами Id или [Имя\_класса]Id (то есть UserId).

Взаимодействие с базой данных в Entity Framework Core происходит посредством специального класса - контекста данных. Поэтому добавим в наш проект новый класс, который назовем ApplicationContext и который будет иметь следующий код:



Основу функциональности Entity Framework Core составляют классы, которые располагаются в пространстве имен Microsoft.EntityFrameworkCore. Среди всего набора классов этого пространства имен следует выделить следующие:

* DbContext: определяет контекст данных, используемый для взаимодействия с базой данных
* DbSet/DbSet<TEntity>: представляет набор объектов, которые хранятся в базе данных
* DbContextOptionsBuilder: устанавливает параметры подключения

Для работы приложения с базой данной через Entity Framework необходим контекст данных - класс производный от DbContext. В данном случае таким контекстом является класс ApplicationContext.

И также в классе определено одно свойство Users, которое будет хранить набор объектов User. В классе контекста данных набор объектов представляет класс DbSet<T>. Через это свойство будет осуществляться связь с таблицей, где будут храниться данные объектов User.

Кроме того, для настройки подключения нам надо переопределить метод OnConfiguring. Передаваемый в него параметр класса DbContextOptionsBuilder с помощью метода UseSqlite позволяет настроить строку подключения для соединения с базой данных SQLite.

В качестве параметра в метод передается строка подключения, которая в данном случае имеет только один параметр - Data Source. Он определяет файл базы данных - в данном случае "helloapp.db".

И также стоит отметить, что по умолчанию у нас нет базы данных. Поэтому в конструкторе класса контекста определен вызов метода Database.EnsureCreated(), который при создании контекста автоматически проверит наличие базы данных и, если она отсутствует, создаст ее.

Теперь определим сам код программы, который будет взаимодействовать с созданной БД. Для этого изменим файл Program.cs следующим образом:

