Практическая работа №1. Работа с Entity Framework Core

Задание:

1) Создайте БД с помощью технологии Entity Framework, содержащую не менее 4 таблиц. Например, информация по книгам (таблицы «Книги», «Авторы», «Издательства», «Адрес»), информация по автомобилям, БД «Университет» и т.п.

2) Реализуйте основные операции для работы с БД: добавление, чтение, удаление, корректировка объектов.

3) Реализуйте возможность вывода общей статистики по БД (общее количество книг, авторов, среднее количество книг на автора, среднее количество книг на издательство и т.п.)

4) Реализуйте возможность поиска (найти все книги автора, найти авторов с наибольшим количеством книг, найти книгу по названию)

Рекомендации к выполнению:

Для разработки приложения можно использовать проект ***Console (.NET Core)*** и проект с оконным интерфейсом (WinForms, WPF).

Для подключения технологии Entity Framework Core необходимо из NUGet – репозитория установить пакеты:

```
Microsoft.EntityFrameworkCore.SqlServer;

Microsoft.EntityFrameworkCore;
```

Классы сущностей (Book, Author, Publisher и т.п.) можно ввести как простые C#-классы с набором свойств. Н-р,

```
public class Author
{
    public int AuthorId { get; set; }
    public string FullName { get; set; }
     ..
    // Навигационное свойство – используем элементы другой таблицы - Book
    public List<Book> Books { get; set; }
}
```

В классах сущностей рекомендуется использовать стандартные названия свойств-ключей (BookId, AuthorId). Иначе необходимо использовать атрибуты или FluetApi для настройки свойств сущностей.

Связь «1 ко многим» реализуется через наличие свойства-коллекции (н-р, свойство Books типа List<Book> в классе Author).

Для реализации связи «многие ко многим» необходимо использовать три таблицы. Например, для связи «Авторы – Книги» необходимо создать промежуточную сущность:

```
public class AuthorBook {
```

```
      public int AuthorId { get; set; }
      public Author Author { get; set; }
      public int BookId { get; set; }
      public Book Book { get; set; }
}
```

Затем необходимо ввести класс контекста:

```
    public class ApplicationContext : DbContext
    {
        public DbSet<Author> Authors { get; set; }
        public DbSet<Book> Books { get; set; }
          . . .
        public ApplicationContext()
        {
            // Удаляем БД, если она существовала
            // после отладки строку можно убрать
            Database.EnsureDeleted();
            Database.EnsureCreated();
        }
        protected override void OnConfiguring(
                              DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder
.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=BooksDb;Trusted_Connec
tion=True;");

        }
    }
```

В классе контекста определяем коллекции сущностей (DbSet<>) и методы подготовки контекста. В OnConfiguring задаем строку подключения к БД.

Для заполнения БД предварительно создаем экземпляры сущностей:

```
using (ApplicationContext context = new ApplicationContext())
{

      Author a1 = new Author { Name = "Richter" };
      Author a2 = new Author { Name = "Petzold" };
      context.Authors.AddRange(a1, a2);
      context.SaveChanges();


      Book b1 = new Book { Author = a1, Title = "CLR via C#" };
      Book b2 = new Book { Author = a1, Title = "Programming Windows" };
      Book b3 = new Book { Author = a2, Title = "WPF" };
      Book b4 = new Book { Author = a2, Title = "Windows Forms" };
      Book b5 = new Book { Author = a2, Title = "UWP" };

      context.Books.AddRange(b1, b2, b3, b4, b5);
      context.SaveChanges();
      . . .
}
```

Для манипуляций с данными (поиск элементов, отвечающих заданным условиям, группировка, сортировка и т.п.) используем методы LINQ с прямой или явной загрузкой данных:

```
var authorsWithManyBooks = db.Authors.Include(a => a.Books)
                        .Where(a => a.Books.Count > 5).ToList();
```