**Импорт данных из JSON файла**

**JSON**

Самый долгий и сложный формат для импорта. С ним не умеет работать ни Excel, ни SQL Server. Единственное решение – код.

C# не умеет по умолчанию работать с JSON, для этого ему необходима библиотека, которая поставляется с проектами ASP.NET. Данные проекты представляют собой веб-решения, которые можно использовать для создания REST Api на C#. Данный процесс нас пока не интересует, нам необходима только 1 библиотека - Newtonsoft.Json.

Данную библиотеку можно найти в Windows, но это долго и не сильно оправданно, тем более могут появиться ошибки по типу «Невозможно импортировать библиотеку», хотя это dll и вроде бы все должно работать.

Самый надежный и простой вариант – создать ASP.NET (.NET Framework) проект – зайти в ссылки – свойства Newtonsoft.Json – скопировать путь до dll – вернуться в первоначальное приложение – ссылки – добавить – обзор – вставить путь – Ок. Библиотека скопируется в папку с .exe файлом приложения.

Теперь можно приступать к импорту JSON файла.

Первым делом необходимо создать шаблон (классы) на основании которых C# сам разделит JSON строку. Данный процесс, при котором из строки создаются объекты, называется – десериализация.

Для быстрого создания класса под десериализацию, необходимо воспользоваться инструментами Visual Studio. Скопировать элемент JSON (если считать, что файл – массив JSON элементов). Создать новый класс – правка – специальная вставка – вставить JSON как класс. Данный метод позволяет сразу создавать вложенные классы, для вложенных данных, и не тратить на это время.

Затем необходимо прочитать данные из файла в строку и десериализовать их в объекты. В итоге получается список объектов определенного класса.

Для продолжения работы необходимо чтобы имелось точное представление, что такое Entity Framework и модель ADO.NET.

Если брать объяснение из википедии:

**ADO.NET Entity Framework (EF)** — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL.

Модель ADO.NET является частью EF, она позволяет подключиться к БД и предоставляет инструменты для удобной работы с таблицами и хранящимися в них данными.

Согласно модели ADO.NET

* каждая таблица – список строк
* строка – класс
* столбцы таблицы – свойства класса (get/set)

**Пример модели ADO.NET.**

// Таблица Users

public DbSet<User> Users { get; set; }

// В таблице Users хранятся такие данные как

// id – primary key

// name – имя пользователя

// age – возраст пользователя

public class User

{

    public int Id { get; set; }

    public string Name { get; set; }

    public int Age { get; set; }

}

Модель ADO.NET предоставляет 2 метода:

* Code First
* DataBase First

Как следует из названий, первый – делает возможным создавать таблицы не открывая SSMS, второй – не тратить время на написание кода для хранения и связи таблиц.

Запомнив все это, можно продолжать импорт JSON файла.

Мы десериализовали данные, теперь у нас есть классы с данными, можно сохранять их в модель ADO.NET.

Порядок действий будет зависеть от самих данных:

* Если в данных представлена информация из Главной и Зависимой таблицы, и больше нет внешних связей, все данные и так уже представлены. Тогда лучше использовать CodeFirst и не тратить много времени на создание этих 2х таблиц.
* Если есть информация из внешней таблицы, лучше потратить немного времени на ручное создание таблицы, т.к. если кодом, то можно легко запутаться со связями + вам все равно придется переопределять данные, т.к. нельзя будет десериализовать JSON по таблице из БД.

1й случай – нет связи с уже существующими таблицами (Code First)

Файл:

{

"Address": "г. Санкт-Петербург, Богатырский проспект, 23",

"Station\_ID": 1,

"data": [{

"Name": "92",

"Price": 44.09,

"AmountOfFuel": 43360

}, {

"Name": "95",

"Price": 47.67,

"AmountOfFuel": 50149

}, {

"Name": "98",

"Price": 56.2,

"AmountOfFuel": 17943

}, {

"Name": "Disel Fuel",

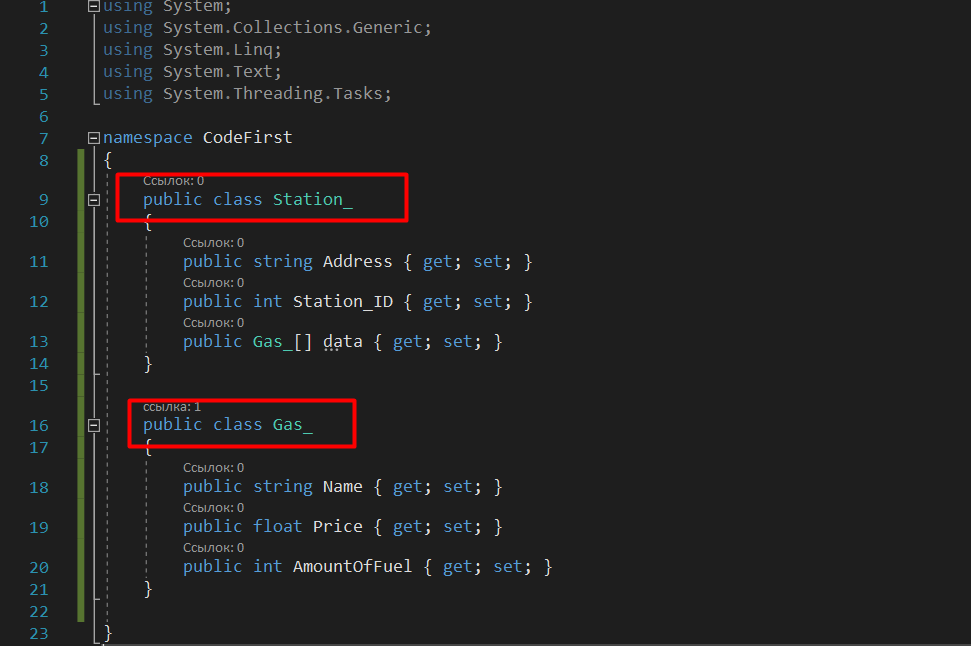
"Price": 48.5,

"AmountOfFuel": 30075

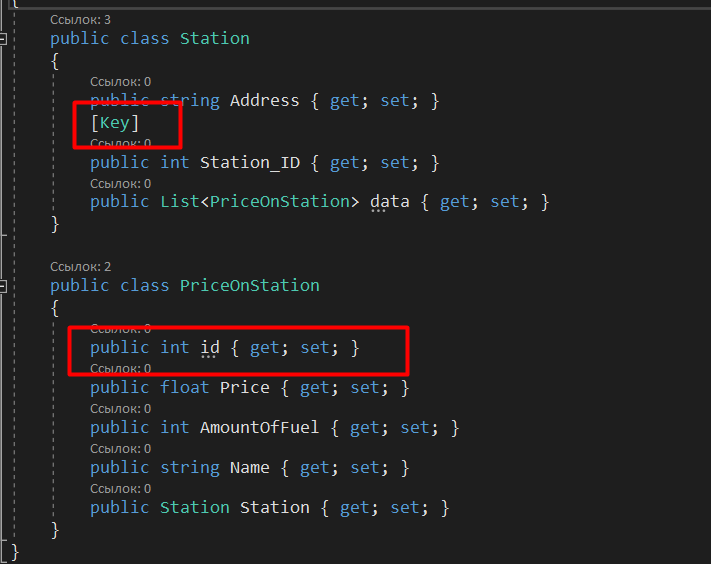
}]

}

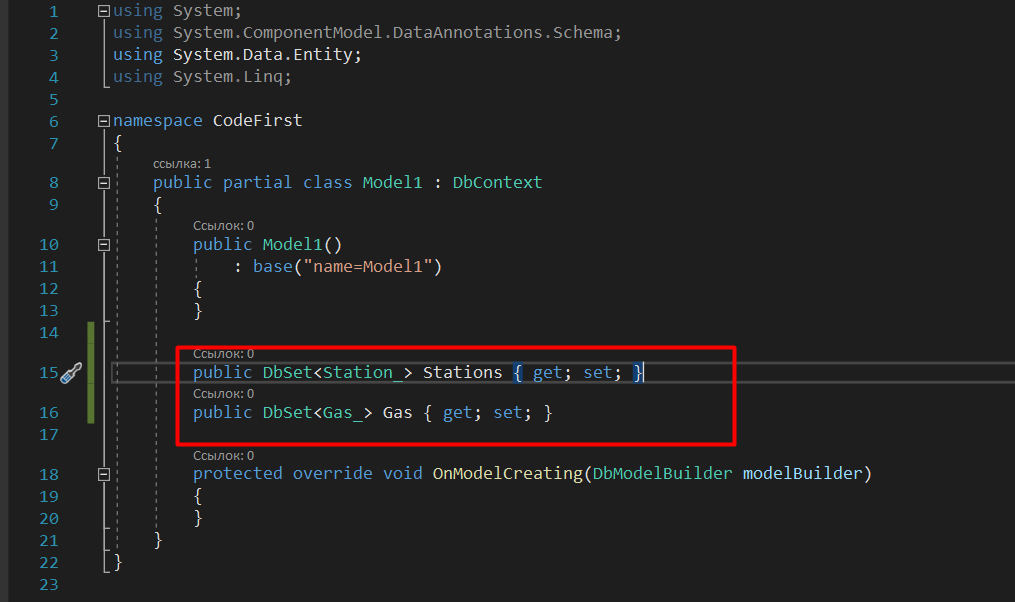
После специальной вставки и изменения названия классов:



Необходимо добавить [Key] перед свойствами, тем самым добавляется информация о том, какое поле считать Первичным Ключом.



Добавить Модель ASO.NET (Code First) – зарегистрировать таблицы в БД.



Осталось только выполнить миграции в БД.

1. Вид – другие окна – консоль диспетчера пакетов
2. Enable-migrations
3. Add-migration “Title migration”
4. Update-database

**Синглтон для подключения к БД**

class Instances

{

private static DemoEntities \_db = null;

public static DemoEntities db

{

get

{

if (\_db == null)

\_db = new DemoEntities();

return \_db;

}

}

}

**Листинг импорта данных (Code First).**

// Импорт файла

var file = File.ReadAllLines(@"K:\5 КУРС\Demo\Gas\_Station\_Log.csv");

var strData = string.Join("", file);

// Десериализация

var data = JsonConvert.DeserializeObject<List<Gas\_Station\_>>($"[{strData}]");

// Добавить все данные в таблицу + автоматически данные добавятся и в связанные таблицы

Instances.db.Gas\_Station\_.AddRange(data);

// Сохранить все изменения

Instances.db.SaveChanges();

Для импорта через DataBase First необходимо 2 класса:

* Первый – чтобы прочитать данные из файла
* Второй – таблица в БД

**Листинг импорта данных (DataBase First).**

// Импорт файла

var file = File.ReadAllLines(@"K:\5 КУРС\Demo\Gas\_Station\_Log.csv");

// CSV файл не содержит знаков между строками,

// чтобы эти элементы превратились в массив а не в одну большую строку

// соединение происходит через ','

var strData = string.Join(",", file);

// Десериализация

var data = JsonConvert.DeserializeObject<List<Gas\_Station\_LogRead>>($"[{strData}]");

// Импорт файла

foreach (Gas\_Station\_LogRead item in data)

{

// Переназначение данных, т.к. в файле идет Address, а внешний ключ должен быть Station\_ID

Instances.db.Gas\_Station\_.FirstOrDefault(q =>

// Поиск необходимой АЗС по адресу

q.Address.ToLower().Contains(item.ASZ.ToLower()))

// Добавление по связи, новой записи в таблицу

.Gas\_Station\_Logs.Add(new Gas\_Station\_Log

{

Date = item.Date,

FuelType = item.FuelType,

LitersCount = item.LitersCount,

Price = item.Price,

PaymentMethod = item.PaymentMethod,

});

// Сохранение изменений

Instances.db.SaveChanges();

}

}

**Для импортирования данных:**

1. Создайте таблицу в базе данных, в которую необходимо заполнить данные, со всеми полями из импортируемой таблицы (рис.1)

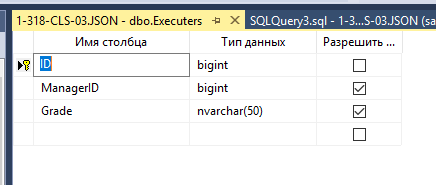
****

Рисунок 1. Создание БД в SQL сервере

1. Создайте проект в VS.Net, создайте соединение Entity с БД
2. Подключите к проекту библиотеку «Newtonsoft.JSON» (рис.2)

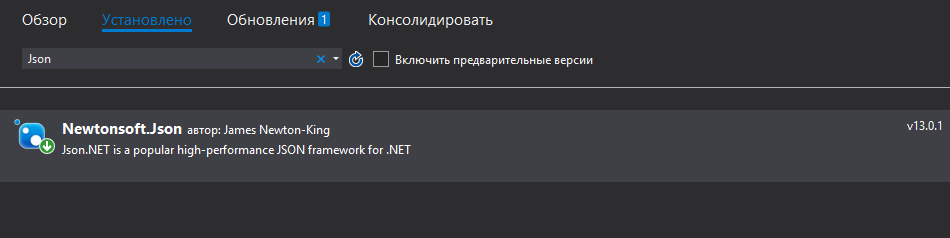


Рисунок 2. Добавление нового пакета

1. Создайте в проекте класс, название полей, которого соответствует названиям полей json файла (рис.3)

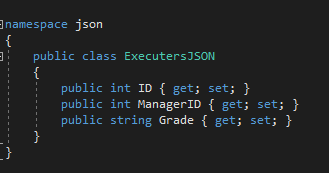
****

Рисунок 3. Класс json

1. Затем в проект добавьте модель данных, содержащую в себе импортируемую таблицу. (рис. 4).

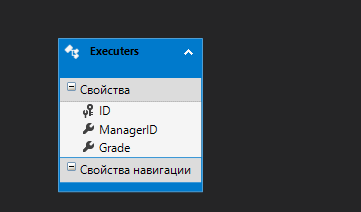
****

Рисунок 4. Модель данных

1. В классе MainWindow напишите код для парсинга json файла и добавления данных в таблицу БД (Листинг 1.).

***Листинг 1.* Код класса MainWindow**

public partial class MainWindow : Window

{ public MainWindow()

{ InitializeComponent();

**//парсинг json файла**

var users = JsonConvert.DeserializeObject<ExecutersJSON[]>(File.ReadAllText(@"C:\Users\ \Desktop\Executers.json"));

TestEntities1 ctx = new TestEntities1();

**// перебор всех элементов json**

foreach(var user in users)

{// **создание объекта базы данных**

var dbUsers = new Executers() { ID = user.ID, ManagerID = user.ManagerID, Grade = user.Grade };

ctx.Executers.Add(dbUsers);

}

ctx.SaveChanges();

}

}

1. После запуска программы на выполнение в таблицу БД вносятся новые данные