Подход 0: Model First

То как все работает в данный момент:

Автогенерация классов с момощью визарда.

Подход 1: Code First

Для класса:

public class Anketa {

public int ID { get; set; }

public int FileId { get; set; }

[StringLength(150)]

public string FIO { get; set; }

public DateTime Birthday { get; set; }

[StringLength(50)]

public string Email { get; set; }

[StringLength(25)]

public string Phone { get; set; }

public bool Zuls { get; set; }

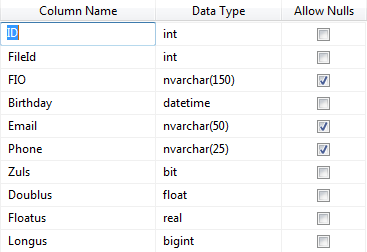
public double Doublus { get; set; }

public float Floatus { get; set; }

public long Longus { get; set; }

}

Создается таблица вида:



После того как приложение впервые мигрировало таблицы из модели в SQL создается файл \_\_MigrationHistory. Если нужно обновить дизайн таблиц – удаляем все таблицы, удаляем этот файл и все создастся с нуля.

Если миграция была выполнена и добавили новое поле – будет ошибка.

Нужно включить механизм поддержки миграций:

https://docs.microsoft.com/ru-ru/ef/ef6/modeling/code-first/migrations/index

Если хотим чтобы был именно CodeFirst то в консоли нугет пакетов выполняем комманду

Enable-Migrations -ProjectName ToMars.Model

Контекст должен поддерживать IDbContextFactory<DbContext>

public DbContext Create() { return new DbContext("connectionStringName"); }

Add-Migration -ProjectName ToMars.Model – запросит название для миграции

Add-Migration будет автоматически формировать следующую миграцию на основе изменений, внесенных в модель с момента создания последней миграции.

Update-Database будет применять ожидающие обработки миграции к базе данных.

Подход 2: Отаказ от edmx и автогенеренных классов

- Таблицы создаем на MS SQL

- В каком либо промежуточном проекте, с помошью визарда создаем EDMX файлы

- Из этих файлов вытаскием в рабочий проект POCO классы сделанные визардом

- Из данных визарда (файла \*.context.cs) в свой контекст вытаскиваем объявления

public DbSet<BindingRule> BindingRules { get; set; }

и хранимки

public virtual int BulkInsertDeposits(string xmlString, Nullable<bool> cleanup, Nullable<System.DateTime> startDate, Nullable<System.DateTime> endDate)

При внесении изменений в таблицы SQL изменения синхронизируем вручную.

Минусы Model First:

* При изменении таблицы на SQL нужно обновлять модель в коде. Иногда возникают проблемы с именованием классов и полей.
* Одни и те же таблицы на разных проектах нужно синхронизировать визардом.
* Нет возможности менять класс контекста (поставить большой таймаут в конструкторе).
* POCO классы можно только расширять.

Плюсы Model First:

* Легко вытаскивать в код большие таблицы и хранимки.

Минусы Code First:

* Необходимость синхронизации модели и БД с помошью команд, в ручном режиме
* Сложно вытаскивать хранимки и большие таблицы. Нужен визард и левый проект
* Необходимость переделки нескольких проектов

Плюсы Code First:

* Класс контекста можно менять как угодно т.к. нет автогенерации
* POCO классы можно таскать их проекта в проект Copy+Paste

Минусы подхода 2:

* При изменении таблицы на SQL нужно обновлять модель в коде. Просто добавляем поля вручную (если мало) или вытаскивая из визарда (в левом проекте) если много полей.
* Сложно вытаскивать хранимки и большие таблицы. Нужен визард и левый проект
* Необходимость переделки нескольких проектов

Плюсы подхода 2:

* Класс контекста можно менять как угодно т.к. нет автогенерации
* POCO классы можно таскать их проекта в проект Copy+Paste