Assignment 3:

### 1. SQL Server

Best Way to Copy an SQL Server database “base” to database “Copied”: we can achieve this functionality by below scenarios.

* detaching and attaching database;
* comparing and synchronizing schemas and data of source and target databases;
* Restoring a backed up source database to a target database.

Each of them has its strong and weak points. To sum up, we would like to list pros and cons of each method.

**The Detach and Attach Method**

**We can perform this functionality through Wizards.**

Pros:

* The fastest way to copy a “base” database.

Cons:

* Source database should be offline;
* All connections will be lost;
* shared folder is required;
* SQL Server version on the target must be the same or higher than that on the source;

**Backup and Restore Method**

**We can perform this functionality through Wizards as well as in SQL Scripts.**

Pros:

* source database may remain online;

Cons:

* a bit slower, then the previous method;
* SQL Server version on the target must be the same or higher that on the source;
* shared folder is required;

**Schema and Data Compare& Sync Method**

**We can pick any schema under “base” database and will move into “Copied” database once compare and validation is completed. Basically it will achieve through wizards.**

Pros:

* the source database may remain online;
* does not require a shared folder;
* method does not depend on the SQL Server version;

Cons:

* time consuming method.

### 2. Entity Framework Migrations

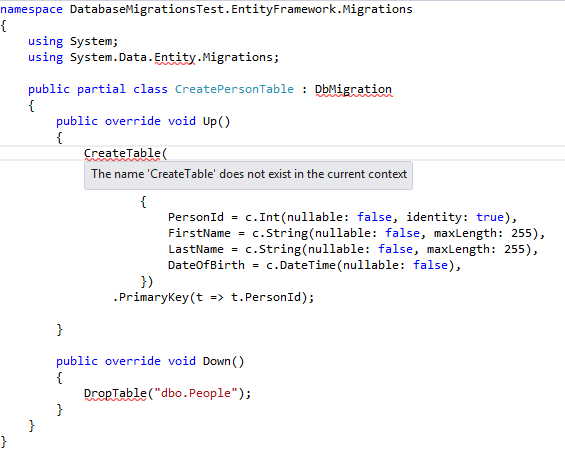
If your project already relies on [Entity Framework](https://docs.microsoft.com/en-us/ef/), then [EF Migrations](https://msdn.microsoft.com/en-us/library/jj591621.aspx) is best way to achieve this functionality.

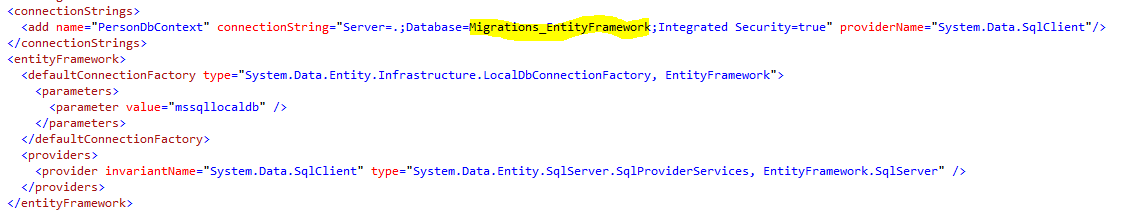
Steps:

* 1. Running Enable-Migrations in Package Manager Console will configure the project with migrations support
  2. Any modification to the project’s DbContext you can run Add-Migration, which detects the changes in the code and creates a new migration file

For automation purposes, Entity Framework NuGet package also includes **migrate.exe** utility, which allows applying migrations from a specified DLL to a database with a specified connection string.

Sample screenscot:





We can execute through BAT file.

copy ..\packages\EntityFramework.6.1.3\tools\**migrate.exe** bin\Debug\

bin\Debug\**migrate.exe** DatabaseMigrationsTest.EntityFramework.dll /connectionString="Server=.;Database=Migrations\_EntityFramework;Integrated Security=true" /connectionProviderName="System.Data.SqlClient"

### 3.  DbUp

DBUp is open source product to copy the SQL Schema from Target data source.

The migration files are created as plain SQL files in the project, but then the build action for them needs to be set to Embedded Resource, effectively making them part of the assembly

Example:

static void Main(string[] args)

{

var connectionString = ConfigurationManager.ConnectionStrings["Database"].ConnectionString;

// creates the database if not yet exists

EnsureDatabase.For.SqlDatabase(connectionString);

var upgrader = DeployChanges.To.SqlDatabase(connectionString)

.WithScriptsEmbeddedInAssembly(Assembly.GetExecutingAssembly())

.LogToConsole()

.Build();

var result = upgrader.PerformUpgrade();

Console.Out.WriteLine(result.Successful ? "Success" : "Failure");

}

Plenty of tools can be used to manage database migrations, but each will have its own pros and cons. For our project, it seems unreasonable to go for Entity Framework since its stores as code, not plain SQL files.