Entity Framework Code First

Difference between code first and DB first

In code first approach we have

- entity to tables mapping in the C# code instead of .edmx XML file
- use a general connection string. We don't need to specify configuration files in the connection string
- We create all entities and mapping manually instead of code generation in DB first
- EF can update DB with using our mapping

Entities and mapping

We create entities as general C# objects. Most of them are the same as
Mapping will bind entities to the DB tables. We can apply mapping in two ways:

- Use code annotations attributes
- Use fluent API with DbModelBuilder
- We can combine these two approaches in the same context. (we apply this approach in the Utopia)
- Use separately classes with mapping for each entity. (This approach we used in workflows.)

mapping example:

Mapping by attributes

```
public partial class DbNode
    {
        [Key]
        public long Id { get; set; }


        [Required]
        [StringLength(255)]
        public string Name { get; set; }
```

Mapping by modelBuilder

```
 public partial class UtopiaDB : DbContext
    {
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // map foreign key.One to many
            modelBuilder.Entity<DbNode>()
                        .HasMany(e => e.DbEmailQueueAttachmentNodes)
                        .WithRequired(e => e.DbNode)
                        .HasForeignKey(e => e.NodeID)
                        .WillCascadeOnDelete(false);

            // map foreign key. many to many
            modelBuilder.Entity<DbSearchCriteria>()
                .HasMany(e => e.DbTriggers)
                .WithMany(e => e.DbSearchCriterias)
                .Map(m => m.ToTable("DbTriggerToSearchCriteria")
                           .MapLeftKey("SearchCriteriaId")
                           .MapRightKey("TriggerId"));

            // map complex primary key
            modelBuilder.Entity<DbTriggerInfoNodes>()
                .HasKey(e => new {e.TriggerID, e.NodeInfoType});
        }
    }
```

Mapping by mapping class (used in WF)

```
public class DbWorkflowMap : EntityTypeConfiguration<Workflow>
    {
        public DbWorkflowMap(string tablePrefix)
        {
            ToTable(tablePrefix + "DbWorkflows");

            HasKey(t => t.Id);
            Property(t => t.Id).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
            HasMany(t => t.Owners).WithMany().Map(a => a.MapLeftKey("WorkflowID")
                                              .MapRightKey("UserID")
                                              .ToTable(tablePrefix + "DbWorkflowOwners"));
```

Migration

To update the DB structure we use migration.

- Migration is a C# class that contains updating instruction.
- EF can generate migrations automatically

How to change structure

1. Add changes to entities and mapping if it needed
2. Create a migration for this changes. To make it automatically create a migration, run this command in the Package Manager Console:

```
Add-Migration "MigrationNameHere" -ProjectName DataAccess -ConnectionString "data source=(localdb)\MSSQLLocalDB;initial catalog=UtopiaDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" -ConnectionProviderName System.Data.SqlClient -ConfigurationTypeName UtopiaDBMigrationConfiguration
```

1. After this command EF will generate C# class for this migration. Please check it

```
        public partial class InitialMigration : DbMigration
            {
                public override void Up()
                {
                   //Some instructions
                }


                public override void Down()
                {
                }
            }
```

4. To apply migration to the DB you can run next command:

Update DB to the last state

```
update-database -verbose -ProjectName DataAccess -ConnectionString "data source=(localdb)\MSSQLLocalDB;initial catalog=UtopiaDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName System.Data.SqlClient
```

Update/Downgrade db to a specific migration.

```
update-database -TargetMigration:<Insert migration name> -verbose -ProjectName DataAccess -ConnectionString "data source=(localdb)\MSSQLLocalDB;initial catalog=UtopiaDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName System.Data.SqlClient
```

Update for updating remote DB's/connections that require authentication:

```
update-database -verbose -ProjectName DataAccess -ConnectionString "data source={databaseserveraddress};User={databaseUsername};password={password};initial catalog={schemaName};integrated security=False;MultipleActiveResultSets=True;App=EntityFramework" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName System.Data.SqlClient
```

5. Applied migrations are stored in `__MigrationHistory` table

PostgreSQL process

There are several things that are different during PostgreSQL Code-First Migration process:

1. Update DB to the last state

```
update-database -verbose -ProjectName DataAccess -ConnectionString "Host=localhost;Database=UtopiaDB;Integrated Security=False;Username=postgres;Password=pass@word1;Port=5432" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName Npgsql
```

2. If Package Manager Console cannot update database, DbMigratorEF project can be used. Pay attention to constants and main method before running it, main methods are UpdateDatabase(config) and ScaffoldNewMigration(config)
3. During master merging into feature/OnPremiseMaster new migrations should be deleted and rescafolded. If "dbo" scheme appears after migration, that means one or more migrations are not deleted.

1 visit in last 30 days

Quinn GodfreyImage not found or type unknown

Quinn Godfreycommented Jul 6, 2020 (edited)

Generate SQL for migration:

```
Update-Database -Script -SourceMigration: [CurrentAppliedMigrationName] -TargetMigration: [MigrationToUpdateToName]
```

Trevor ChadwickImage not found or type unknown

Trevor Chadwickcommented Jul 21, 2020 (edited)

Update for updating remote DB's/connections that require authentication:

```
update-database -verbose -ProjectName DataAccess -ConnectionString "data source={databaseserveraddress};User={databaseUsername};password={password};initial catalog={schemaName};integrated security=False;MultipleActiveResultSets=True;App=EntityFramework" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName System.Data.SqlClient
```

Chayston WoodImage not found or type unknown

Chayston Woodcommented Nov 5, 2020

Update DB to the last state - NO VERBOSE

```
update-database -ProjectName DataAccess -ConnectionString "data source=(localdb)\MSSQLLocalDB;initial catalog=UtopiaDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework" -ConfigurationTypeName UtopiaDbMigrationConfiguration -ConnectionProviderName System.Data.SqlClient
```

Bryan Christensen Image not found or type unknown

Bryan Christensen[commented Nov 23, 2020]

This message comes through when I went and changed DbNodeClosures ID, needing to drop the restraint and change the Primary Key name. We were forcing users to Manual to change these, but we can use this script (or a variant of it) to do the rename automatically.
From Roman Alekhin
@BryanC this script searching constraint name in sys.key_constraints which contains constrains names for all tables. We just specify table name and constraint type (PK or UK) and receive for example primary key name for specific DB. As result we will receive correct name for any person's DB.
This will find PK or UK by table name in sys.key_constraints and generate SQL script for dropping it

```
private const string BasicScript = @"DECLARE @TableName NVARCHAR(128)
            SELECT @TableName = '<tableName>'
            DECLARE @SQL NVARCHAR(MAX)
            SELECT
                @SQL = 'ALTER TABLE ' + @TableName + ' DROP CONSTRAINT [' + kc.name + ']'
            FROM
            sys.key_constraints kc
            WHERE
            kc.[type] = '<constraintType>'
                AND kc.parent_object_id = OBJECT_ID(@TableName)
            EXEC SP_EXECUTESQL @SQL";
```

This just replaces <constraitType> macro to PK

```
private const string ConstraintTypePlaceholder = "<constraintType>";
private const string PrimaryKeyConstraintType = "PK";
private readonly string _dropPrimaryKeyScript =
            BasicScript.Replace(ConstraintTypePlaceholder, PrimaryKeyConstraintType);
```

And this drops PK for all tables in list:

```
private const string TableNamePlaceholder = "<tableName>";
private void DropPrimaryKeyConstraint(string[] tableNames)
        {
            foreach (var tableName in tableNames)
            {
                Sql(_dropPrimaryKeyScript.Replace(TableNamePlaceholder, tableName));
            }
        }
```

Revision #3
Created 18 July 2022 23:50:45 by Bryce Holloway
Updated 11 October 2023 14:38:28 by Abe Austin