Kentico

Entity Framework

Petr Svirák

2016

What is it

- NuGet package @ nuget.org
- Object-relational mapper (ORM) to work with relational data using domain-specific objects
- Eliminates the need for most of the data-access code
- Code-first or Database-first
- Quarable interface to work with DB

http://www.asp.net/entity-framework

What is ORM

- Object-relational mapping
- Technique used to connect two incompatible realms (usually a scalar database and an object-oriented programing language
- New layer of abstraction is introduced that synchronizes scalar values from database with an object structure in memory and vice-versa

EF6 vs. EF Core

- EF6 will be used
  - Plenty is missing in EF Core, for example:
    - many-to-many relations
    - lazy loading
    - simple type conversions
    - joins
    - seeding
    - migrations for contexts in separate libraries
  - The set of currently available features does not make it an option for any larger production project

https://docs.efproject.net/en/latest/efcore-vs-ef6/features.html

Where to start

- Create a class that implements DbContext
  - Base class provides methods to work with object representation of database data
  - Allows further configuration (later)
- Add virtual properties of type DbSet<TModel>
  - IQueryable collection of any given model (code-base class describing a table)
    - LINQ is translated to SQL queries and the query is executed only during materialization (enumeration of the collection), thus not all records needs to be loaded from a database server to memory and trafic between the services is mitigated
  - Virtual so it can be mocked in tests
  - Different models can reference each other directry (property of type Model1 in Model2 class)
    - Even collections are supported (ICollection<T> allows addition)
    - Always make non-scalar properties virtual, so it can be loaded lazily (later)
  - Inheritance can be applied on various models, however, only instantiable classes (with parameterless constructor) can be used as entities

Decorating models

- Applied on models
- *System.ComponentModel* and *System.ComponentModel.DataAnnotations* namespaces
- Model-specific constrains or additional meta-information
  - ~~Display, DisplayName, DisplayFormat, HiddenInput – how is a property presented to a user~~
  - Required, Range, DataType, EnumDataType, Key – property's limitations and DB specifications
- Can be configured separately and in more details (later)

http://www.asp.net/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6
https://msdn.microsoft.com/en-us/library/dd901590(VS.95).aspx

Migrations

- Provides better control over individual versions of code-first database models
- Can be controlled both in Package Manager Console (PoweShell console) or code
  - When using Package Manager Console, use –verbose to get more detailed information on processed tasks
- Multiple context with multiple migrations are permitted atop of single database
- Current state of database is tracked in *__MigrationHistory* table (context name + model hash).
- Current state of code is compared to latest migration during a context initialization

Enabling migrations

- *Enable-Migrations snippet:*
  - *–ContextTypeName <name-of-context-class>*
  - *–MigrationsDirectory <path-replative-to-project-where-migration-configuration-is-stored>*
  - *–ProjectName <name-of-project-migrations-will-be-enabled-(need-EF-reference)>*
  - *–StartUpProjectName <name-of-executable-project-(with-connection-string-in-config)>*
  - *–ContextProjectName <name-of-project-where-context-class-can-be-found>*

- Creates new Configuration class that implements DbMigrationsConfiguration
  - Constuctor allows setting of various aspects of all migration for given context. Most common:
    - *AutomaticMigrationsEnabled* – will try to migrate DB when (code) model changed even without explicit migration
    - *AutomaticMigrationDataLossAllowed* – will try to migrate DB even if a table has to be dropped and re-created
    - *MigrationsDirectory* – folder where explicit migrations are stored
  - Seed method allows creation of testing data

Adding new migration

- *Add-Migration* snippet:
  - *–Name <short-description-of-new-migration-(will-be-extended-with-timestamp)>*
  - *–ConfigurationTypeName <name-of-configuration-class-created-by-Enable-Migrations>*
  - *–ProjectName <name-of-project-migrations-will-be-enabled-(need-EF-reference)>*
  - *–StartUpProjectName <name-of-executable-project-(with-connection-string-in-config)>*
  - *–ConnectionStringName <alternative-to-connection-string-(is-read-from-StartUp-project)>*

- No database change is performed. Only new migration class is generated (implements DbMigration base class).
- Class can be freely renamed and content edited, however, next explicit migration might want to remove some manual changes
- Class can even be deleted with no harm (if it was not yet used to update database)

Updating database

- *Update-Database snippet:*
  - –TargetMigration *<optional-name-of-a-migration-(class-name,-without-time-stamp)>*
  - *–ConfigurationTypeName <name-of-configuration-class-created-by-Enable-Migrations>*
  - *–ProjectName <name-of-project-migrations-will-be-enabled-(need-EF-reference)>*
  - *–StartUpProjectName <name-of-executable-project-(with-connection-string-in-config)>*
  - *–ConnectionStringName <alternative-to-connection-string-(is-read-from-StartUp-project)>*
  - *–Force* – executes migration even if it was already executed and corresponds to current migration

- *No database change is performed. Only new migration class is generated (implements DbMigration base class).*
- *Class can be freely renamed and content edited, however, next explicit migration might want to remove some manual changes*
- *To roll-back to a previous version of DB, use an older migration name as TargetMigration parameter (all migration along the way are roll-backed as well and vice-versa)*
  - *In some cases (usually in combination with automatic migrations), dataseb cannot be updated by the command. For development DB is should always work to delete entire database and let EF recreate it*

Migrations in code

- *Call Database.SetInitializer<TContext>(new MigrateDatabaseToLatestVersion<TContext, TConfiguration>(useSuppliedContext: true));*
  - *useSuppliedContext – will use connection provided to the DbContext*
  - *Always migrates to the latest version of code, custom initializer would be needed for migration to other versions (yet PowerShell snippet can be used)*
  - *It is not suggested to call this code from a DbContext constructor as intializer might differ based on usage*

How use migrations in development

- Pre-production
  - *Use MigrateDatabaseToLatestVersion* database initializer
  - No real data:
    - *AutomaticMigrationsEnabled = (team:* false*, indiviual:* true*);*
    - *AutomaticMigrationDataLossAllowed = (team:* false*, individual:* true*);*
    - Seeding in *DbMigrationsConfiguration.Seed* method
- Production
  - Always backup database and ideally work with a copy
  - Use *NullDatabaseInitializer* database initializer
  - With real data/after deployment:
    - *AutomaticMigrationsEnabled = false;*
    - No seeding
    - Call *Add-Migration* and *Update-Database* manually after each models change iteration

More about migrations

- Named migrations can be manually updated
- Running *Update-Database –TargetMigration <migration> –Script –Force* will
  - Create SQL script to migrate DB to given migration (can be used on production DB)
  - Re-run migration and re-seed database (applies changes to development DB)
- If not initializer set, *CreateDatabaseIfNotExists* is used
  - When not using migrations, *DropCreateDatabaseAlways* and *DropCreateDatabaseIfModelChanges* are frequently used (though permission to create database on server is required in these cases)

https://app.pluralsight.com/library/courses/efmigrations/

Context set-up

- *DbContext* base class has string-parametered constructor with connection string itself or its name (preferably use „name=<connectionStringName>“ in the second case *)

- *Database* property provides access to various aspects of database and its connection.
  - *Database.Log* – allows custom logging of queries and commands executed in the context
  - *Database.CommandTimeout* – amout of time to wait before an command is interupted
  - *Database.Connection.StateChange* – event executed on any change to the state of connection (opened, closed, ...)
- *Configuration* property provides access to various aspects of entity framework behaviour in the context – Each property is important and well described

- Both properties are available publicly in each instance. Always consider whether it is necessary to modify all instances of the context (by using constructor) or individual instances (by amending a property, for example, in a context instance in a given service/repository)

* http://stackoverflow.com/a/25057557/1138663

Context models creation

- Either via DataAnnotations attributes
- Or via overriding OnModelCreating method in a context itself
  - Entity Framework Fluent API
  - Wider variety of posibilities available (than attributes provide)
- *modelBuilder.Configurations* – data anotations stored in separated implementations of *EntityTypeConfiguration<T> base class*.
- *modelBuilder.Conventions* – rules based on properties for (all) models in the context, stored in separate implementations of *IConvention* (or *Convention* base class more precisely)
- *modelBuilder.Properties* – context-wide lightweight conventions
- *modelBuilder.Entity* – relations definition, entity-specific lightweight configurations and conventions

Lazy loading (and proxies)

- If enabled (default), virtual properties representing (other) entities or collections of entities are not queried with the main object itself, but later upon first request/access to given property in code.
- This is done by using automaticaly (run-time) generated proxy types overriding the very virtual properties and replacing their getter with a loading hook.

https://msdn.microsoft.com/en-us/data/jj574232.aspx#lazy

Eager loading

- For queries where it is known that certain properties/related entities will be accessed, eager loading is more suitable as there is only one query to the database, rather than a new query for each virtual property of each object that was accessed for the first time.
- *<DbContext>.<DbSet>.Include(entity => entity.Property)* – notifies Entity Framework to query entity/entities stored in the *Property* along with entity stored in *<DbSet>*.

https://msdn.microsoft.com/en-us/data/jj574232.aspx#eager

Unit of Work

- Design pattern
- Each DbContext acts as a unit of work.
  - Changes made to entities are tracked and persit in memory
  - Each *SaveChanges()* call succeeds fully or nowise (change are persisted only all-together)
  - Bigger the context is, more memory it possibly drains and more responsibilies it has

http://stackoverflow.com/questions/10776121/what-is-the-unit-of-work-pattern-in-ef

Bounded Context

- Design pattern
- Bounded context is context that delimits the applicability of a particular model (one of DDD patterns)
  - Clearer defined boundaries of each entity or entity group
  - Better maintainability and less side-effects on context change

https://msdn.microsoft.com/en-us/magazine/jj883952.aspx

Seeding multiple contexts

- Entity Framework is unable to seed multiple contexts with at least one same entity
- Thus it is necessary to use single seeding context for development/early testing purposes.
  - If there are groups of entities without relation between them, it is possible to have multiple seeding contexts that do not interfere with eath other.
  - Such context(s) should however never been used in any life environment.
- Non-seeding context can overlap freely and may even inherit one another
  - These context should not use any agressive initializer (such as *DropCreateDatabaseAlways*)
  - It is no problem for these contexts to exists in single database

http://stackoverflow.com/a/21538091/1138663

Resources

- https://app.pluralsight.com/library/courses/entity-framework5-getting-started
- https://app.pluralsight.com/library/courses/entity-framework-6-ninja-edition-whats-new