Kentico

Entity Framework

Petr Svirák

2015

Migrations

- Provides better control over individual versions of code-first database models
- Can be controlled both in Package Manager Console (poweshell console) and code
- Initial migration
  - Created after *Enable-Migrations –ContextTypeName <context>* is called
- How use migrations during development
  - *MigrateDatabaseToLatestVersion* database initializer
  - No real data*/before first deployment:
    - *AutomaticMigrationsEnabled = true;*
    - *AutomaticMigrationDataLossAllowed = true;*
    - Seeding in *DbMigrationsConfiguration.Seed* method
  - 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
  - Re-run migration and re-seed database
- In production, always use *NullDatabaseInitializer* to prevent data-loss.
- If not initializer set, *CreateDatabaseIfNotExists* is used
  - When not using migrations, *DropCreateDatabaseAlways* and *DropCreateDatabaseIfModelChanges* are frequently used

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 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 controller)

* 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 preciously)
- *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 possible 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