Entity Framework

EF Code First

Lesson Objectives

- Data Annotations Attributes
- Fluent API Configurations
- Configure Relationships
- EF Code First Migration

Section 1

DATA ANNOTATIONS ATTRIBUTES

Data Annotations

- Code-First Conventions is default, simple, powerful but not cover advanced cases
  - ✓ More specific data type and constrains
    - Name of user is not exceed 100 characters, I don’t want to create nvarchar(Max)
    - Age of user is in range 0 – 150, let’s set the constrains to check this
  - ✓ Column name, table name in SQL have space
  - ✓ Many-to-Many relationships

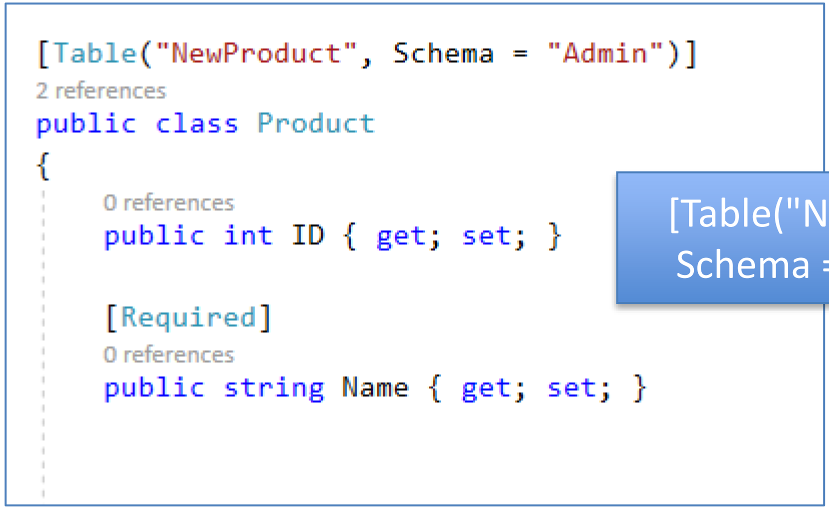
Data Annotations

- A simple attribute based configuration
- Can apply to domain classes and its properties
- Included in a separate namespace
  - ✓ System.ComponentModel.DataAnnotations
  - ✓ System.ComponentModel.DataAnnotations.Schema

Data Annotations - Table Attribute

- The Table attribute can be applied to a class to configure the corresponding table name in the database.
- It overrides the default convention.
  - ✓ What is default schema?
  - ✓ What is convention for table name?
- Format: [Table(string name, Properties:[Schema = string])
  - ✓ name: Name of the Db table.
  - ✓ Schema: Name of the Db Schema in which a specified table should be created. (Optional)

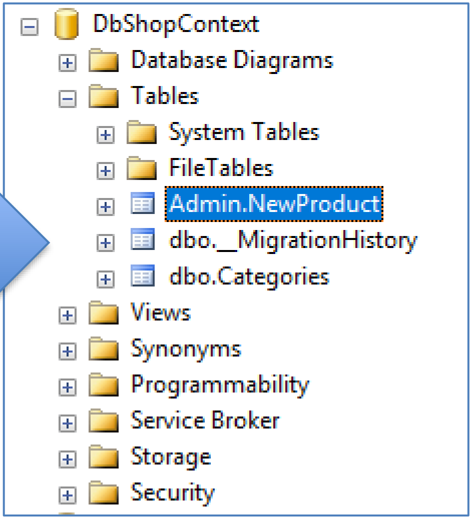
Data Annotations - Table Attribute



```
[Table("NewProduct", Schema = "Admin")]
2 references
public class Product
{
    0 references
    public int ID { get; set; }

    [Required]
    0 references
    public string Name { get; set; }
```

[Table("NewProduct", Schema = "Admin")]



Data Annotations - Column Attribute

- Which are default conventions for column?
  - ✓ Column name
  - ✓ Data type
  - ✓ Order

Data Annotations - Column Attribute

- The Column attribute can be applied to one or more properties in an entity class
- Format: [Column (string name, Properties:[Order = int],[TypeName = string])
  - ✓ name: Name of a column in a SQL table.
  - ✓ Order: Order of a column, starting with zero index. (Optional)
  - ✓ TypeName: Data type of a column. (Optional)

Data Annotations - Column Attribute

- Name:
  - ✓ Can put space in SQL column name
  - ✓ Try to keep naming conventions
  - ✓ Similar to entity field to easy finding/understanding
  - ✓ Keep it simple, meaning full

Data Annotations - Column Attribute

- Order:
  - ✓ Use the zero-based
  - ✓ Difference values
  - ✓ Do not need to continuity
  - ✓ All ordered columns should come first (in order, of course)
  - ✓ All non-ordered columns should come along, in default order as conventions

Data Annotations - Column Attribute

- TypeName:
  - ✓ To change SQL data type for the column
  - ✓ Accept name of data type in SQL as a string

Data Annotations – Index Attribute

- Is used to create an index on a particular column in the database
- Format:[Index(string name, Properties:[IsClustered = bool],[IsUnique = bool] ,[Order = int])]
  - ✓ name: name of Index. Default value is: IX_{property name}
  - ✓ IsClustered: to set index is clustered or non-clustered.
  - ✓ IsUnique: to set constrain data is unique or not
  - ✓ Order: to set order of index

Data Annotations - ForeignKey Attribute

- Is used to configure a foreign key in the relationship between two entities.
- Format: [ForeignKey(name string)]
  - ✓ name: Name of the associated navigation property or the name of the associated foreign key(s).

Data Annotations - NotMapped Attribute

- Used when we do not want to create corresponding columns in the database.
- Format: [NotMapped()]
- Discussion: Give an example to use this attribute?

Data Annotations - InverseProperty Attribute

- Is used when two entities have more than one relationship.
- Format: [InverseProperty(string property)]
  - ✓ property: The navigation property representing the other end of the same relationship.
- The attribute always set on the left-hand (set to property of the parent entity)

Data Annotations - Key Attribute

- What is default convention for PrimaryKey?
- How to set PrimaryKey for non-formatted field?
- How to set PrimaryKey for multiple columns?
- How to use Find method for PrimaryKey with multiple columns?

Data Annotations - Key Attribute

- Applied to a property in an entity class to make it a key property.
- Corresponding column to a PrimaryKey column in the database
- Format: [Key]

Data Annotations - Key Attribute

- EF Core does not support creating a composite key using the Key attribute. You have to use the Fluent API HasKey() function in EF Core.

Data Annotations - Required Attribute

- EF Core will create a NOT NULL column in a database table.
- Can be applied to one or more properties in an entity class.
- Format: [Required]
  - ✓ AllowEmptyStrings: to accept empty string or not
- Required attribute inherit from Validation attribute. Therefor:
  - ✓ ErrorMessage: The error message is thrown when the property associated with the validation control is invalid.

Data Annotations - MaxLength Attribute

- Specifies the maximum length of data value allowed for a property which in turn sets the size of a corresponding column in the database.
- It can be applied to the **string** or **byte[]** properties of an entity.
- Format: [MaxLength(int)]
- MaxLength attribute inherit from Validation attribute

Data Annotations - MaxLength Attribute

Example 1:

```
[MaxLength(50)]
public string ProductName { get; set; }
```

Should create a field ProductName in SQL with data type is: nvarchar(50)

Example 2:

```
[MaxLength(1024)]
public byte[] FileContent { get; set; }
```

Should create a field FileContent in SQL with data type is: varbinary(1024)

Data Annotations - MinLength Attribute

- Same context as MaxLength
- Specifies the minimum length of data value allowed
- Question: How to specify fixed length of the data?
  - ✓ For example: ProductCode is always has 10 characters

Data Annotations - StringLength Attribute

- Can be applied to the string properties of an entity class.
- Specifies the minimum and maximum length of characters that are allowed in a data field.
- User often uses maximum length first, then minimum.
- Format: [StringLength(int, MinimumLength = int)]

Data Annotations - Range Attribute

- Specifies the numeric range constraints for the value of a data field.
- Apply the attribute to a data field of type integer.
  - ✓ [Range(int minimum, int maximum)]
- Apply the attribute to a data field of type double.
  - ✓ [Range(double minimum, double maximum)]

Data Annotations - Range Attribute

- Apply the attribute to a DateTime data field
  - ✓ Range(Type, String, String)
  - ✓ [Range(typeof(DateTime), string minximum, string maximum]
- Apply the attribute to a custom data field
  - ✓ The object to validate must implement the ***IComparable*** interface.
  - ✓ Range(Type, String, String)

Data Annotations - Timestamp Attribute

- It can only be applied once in an entity class to a byte array type property.
- It creates a column with timestamp data type in the SQL Server database.
- Entity Framework API automatically uses this Timestamp column in concurrency check on the UPDATE statement in the database
- Format: [Timestamp]

ConcurrencyCheck Attribute

- The ConcurrencyCheck attribute can be applied to one or more properties in an entity class.
- When applied to a property, the corresponding column in the database table will be used in the optimistic concurrency check using the where clause.
- The ConcurrencyCheck attribute can be applied to any number of properties with any data type.

Data Annotations - All Attributes

- https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=net-5.0

Section 2

FLUENT API CONFIGURATIONS

Fluent API Configurations

- Entity Framework Fluent API is used to configure domain classes to override conventions.
  - ✓ In other words, we can use both Data Annotation attributes and Fluent API at the same time
  - ✓ Fluent API override Data Annotations attributes.

Fluent API Configurations

- EF Fluent API is based on a Fluent API design pattern (a.k.a Fluent Interface) where the result is formulated by method chaining.
- The DbModelBuilder class acts as a Fluent API.
- It provides more options of configurations than Data Annotation attributes.

Fluent API Configurations

- To write Fluent API configurations, override the OnModelCreating() method of DbContext in a context class
  - ✓ protected override void OnModelCreating(ModelBuilder modelBuilder)

Fluent API Configurations

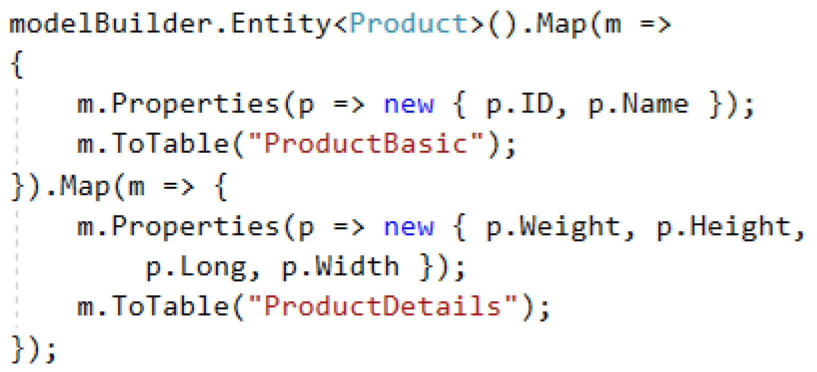
- Model-wide Configuration: Configures the default Schema, entities to be excluded in mapping, etc.
- Entity Configuration: Configures entity to table and relationship mappings
  - ✓ e.g. PrimaryKey, Index, table name, one-to-one, one-to-many, many-to-many etc.
- Property Configuration: Configures property to column mappings
  - ✓ e.g. column name, nullability, Foreignkey, data type, concurrency column, etc.

Entity Mappings

- Configure Default Schema
  - ✓ Specifies the default database schema.
  - ✓ modelBuilder.HasDefaultSchema("Admin");
- Map Entity to Table
  - ✓ Specifies the table name instead of default convention.
  - ✓ modelBuilder.Entity<Product>().ToTable(“NewProduct");
  - ✓ modelBuilder.Entity<Product>().ToTable("NewProduct", "Admin");

Map Entity to Multiple Tables

- Map set of properties to each table
- Used when:
  - ✓ Has many properties/columns
  - ✓ Separate table for difference used purposes



```
modelBuilder.Entity<Product>().Map(m =>
{
    m.Properties(p => new { p.ID, p.Name });
    m.ToTable("ProductBasic");
}).Map(m => {
    m.Properties(p => new { p.Weight, p.Height,
        p.Long, p.Width });
    m.ToTable("ProductDetails");
});
```

Property Mappings

- HasKey
  - ✓ To configure a key property.
  - ✓ Override default convention, ID or <Entity>Id becomes normal column in SQL
  - ✓ modelBuilder.Entity<Product>().HasKey(p => p.ProductKey);
- HasColumnName/Order/Type
  - ✓ To configure column name, order, data type in SQL
- IsOptional/IsRequired
  - ✓ To configure Null or NotNull Column

Property Mappings

- HasMaxLength
  - ✓ To configure maxlength
- IsFixedLength
  - ✓ To change datatype from nvarchar to nchar
- HasPrecision
  - ✓ To set size decimal

Section 3

CONFIGURE RELATIONSHIPS

Configure One-to-Many Relationships

- Step 1/4: start configuring with any one entity class.
- Step 2/4: use **.HasRequired()** to specifies required property. This will create a NotNull foreign key column in the DB.

Configure One-to-Many Relationships

- Step 3/4: use **.WithMany()** to specifies that the parent entity class includes many children entities. Here, many infers the ICollection type property.
- Step 4/4: use .**HasForeignKey()** to specify the name of the foreign key

Configure One-to-One Relationship

- Configure a One-to-Zero-or-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasOptional() to another entity
  - ✓ Use WithRequired() this entity

Configure One-to-One Relationship

- Configure a One-to-Zero-or-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasOptional() to another entity
  - ✓ Use WithRequired() this entity
- Configure a One-to-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasRequired() to another entity
  - ✓ Use WithRequiredPrincipal() this entity

Configure a Many-to-Many Relationship

- Get Entity from any side
- Use HasMany() to another entity
- Use WithMany() to this entity
- [Optional] Use Map
  - ✓ Use MapLeftKey
  - ✓ Use MapRightKey
  - ✓ Use ToTable

Section 4

CODE-FIRST MIGRATION

Migration in EF Code-First

- When you don’t have permission to drop database
- When you have important data, so you cannot drop database
- Migration: movement from one part of something to another.
  - ✓ Add or Remove entity from DbContext
  - ✓ Add or Remove or Update entity property

Code-Based Migration

- Provides more control on the migration
- Allows to configure additional things
  - ✓ setting a default value of a column,
  - ✓ configure a computed column
  - ✓ ….
- You may consider to backup database first, before run migration command

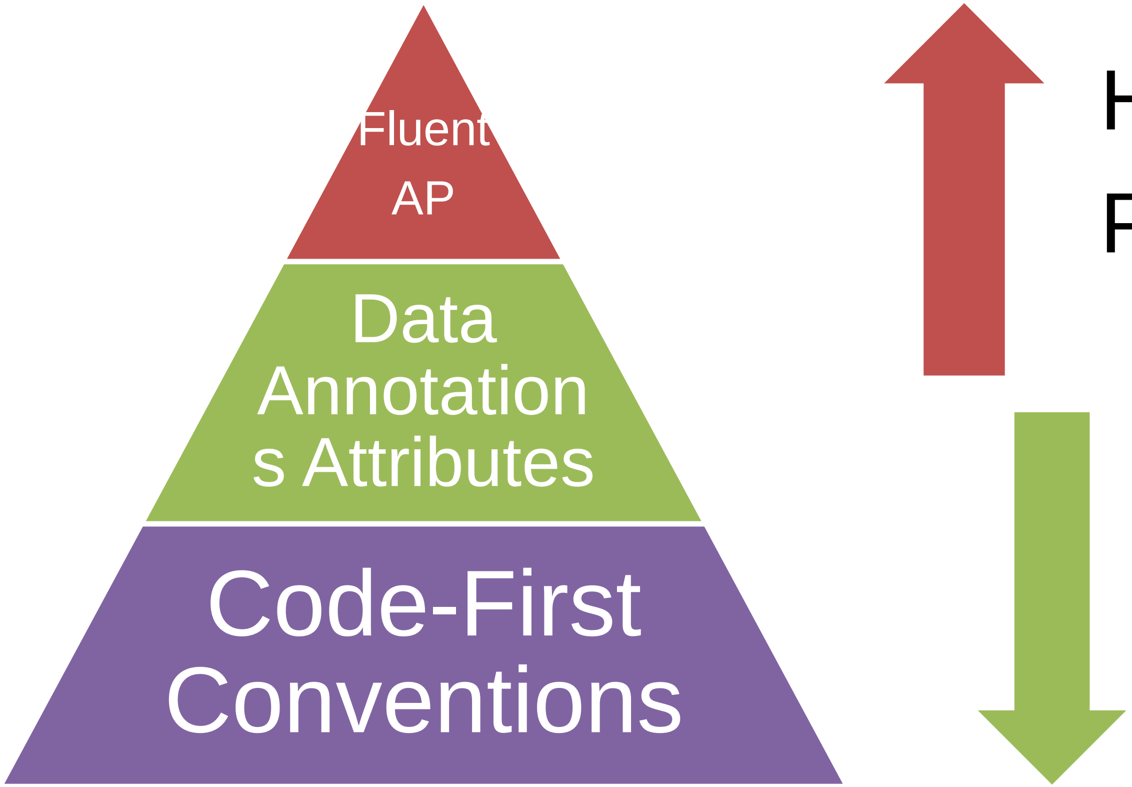
Code-Based Migration

- Process to migrate:
  - ✓ Step 1.x: Add-Migration:
    - Creates a new migration class as per specified name with the Up() and Down() methods.
  - ✓ Step 2.x: Update-Database:
    - Executes the last migration file created by the Add-Migration command and applies changes to the database schema.

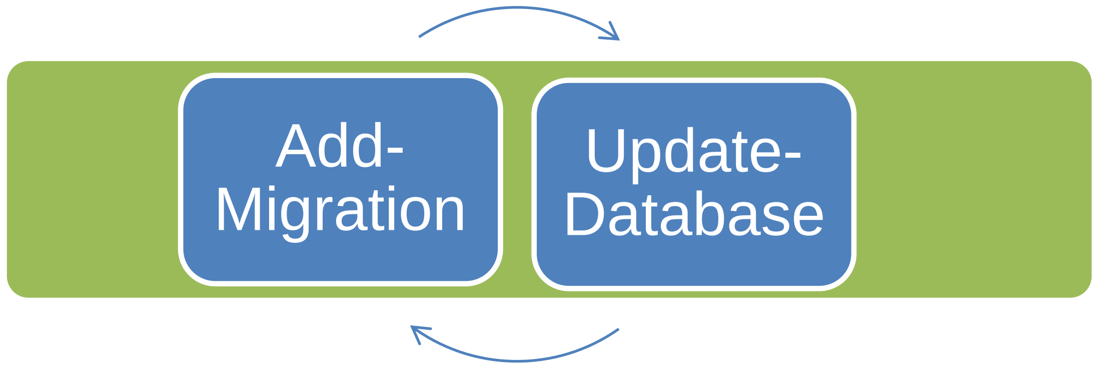
Rollback Migration

- Rollback to a previous migration due to an error in the current migration or wanting to rewind and start over
- PM> update-database -TargetMigration:<name without timestamp>
- Notices:
  - ✓ The migration cannot rollback deleted data
  - ✓ The migrations which were rolled back were **not deleted**.
  - ✓ The next time perform an update-database command, the migrations will be re-executed.
  - ✓ To complete delete, we need to delete the migration from project entirely

Summary


Fluent
AP
Data Annotations Attributes
Code-First Conventions
High Priority
Powerful
Most used

Summary



Thank you