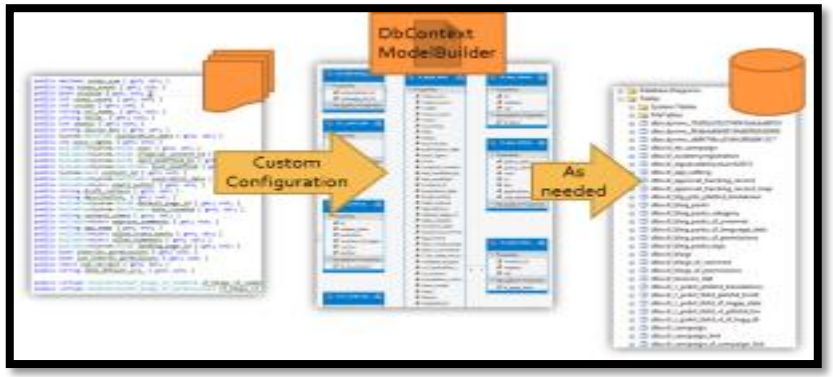
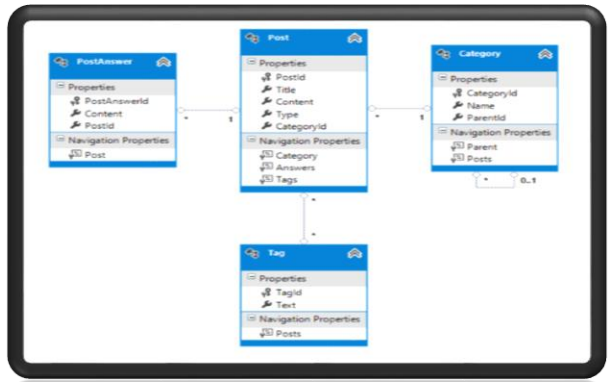




Entity Framework Code First

Telerik Academy

http://academy.telerik.com

Table of Contents

Telerik

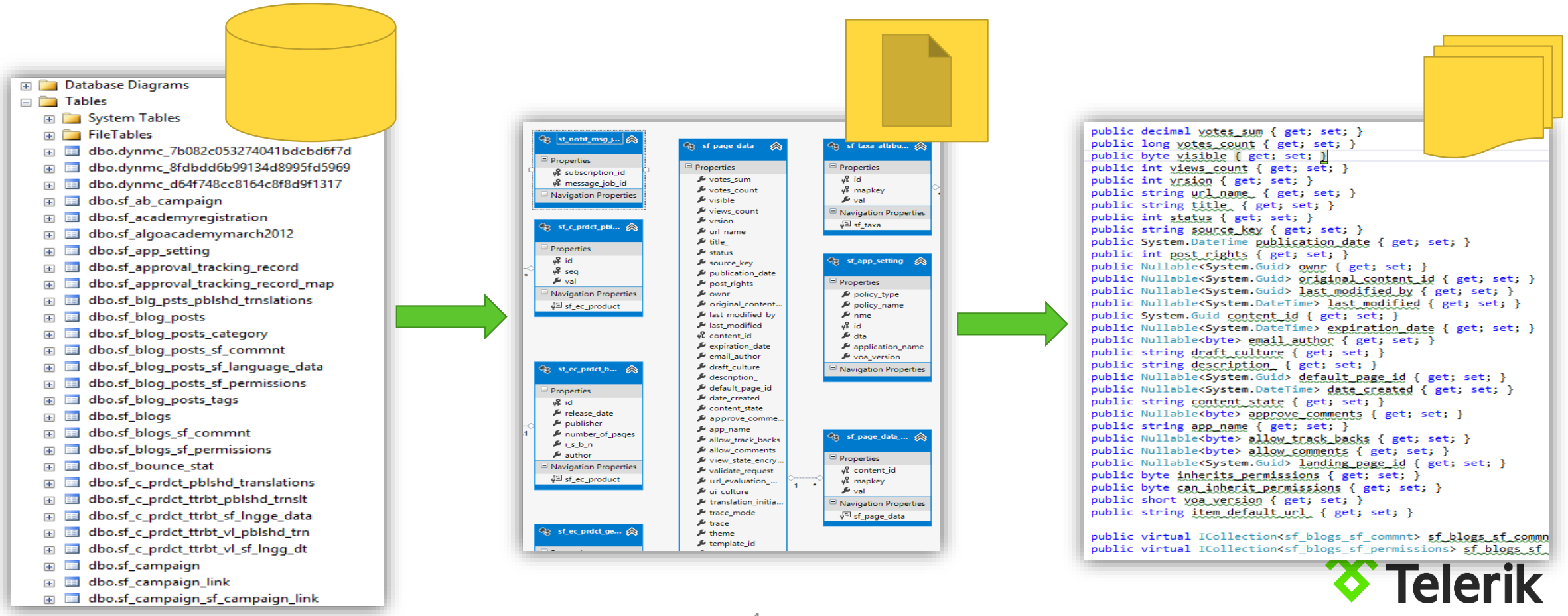
Modeling Workflow

- Entity Framework supports three types of modeling workflow:
  - Database first
    - Create models as database tables
    - Use Management Studio or native SQL queries
  - Model first
    - Create models using visual EF designer in VS
  - Code first
    - Write models and combine them in DbContext

Telerik

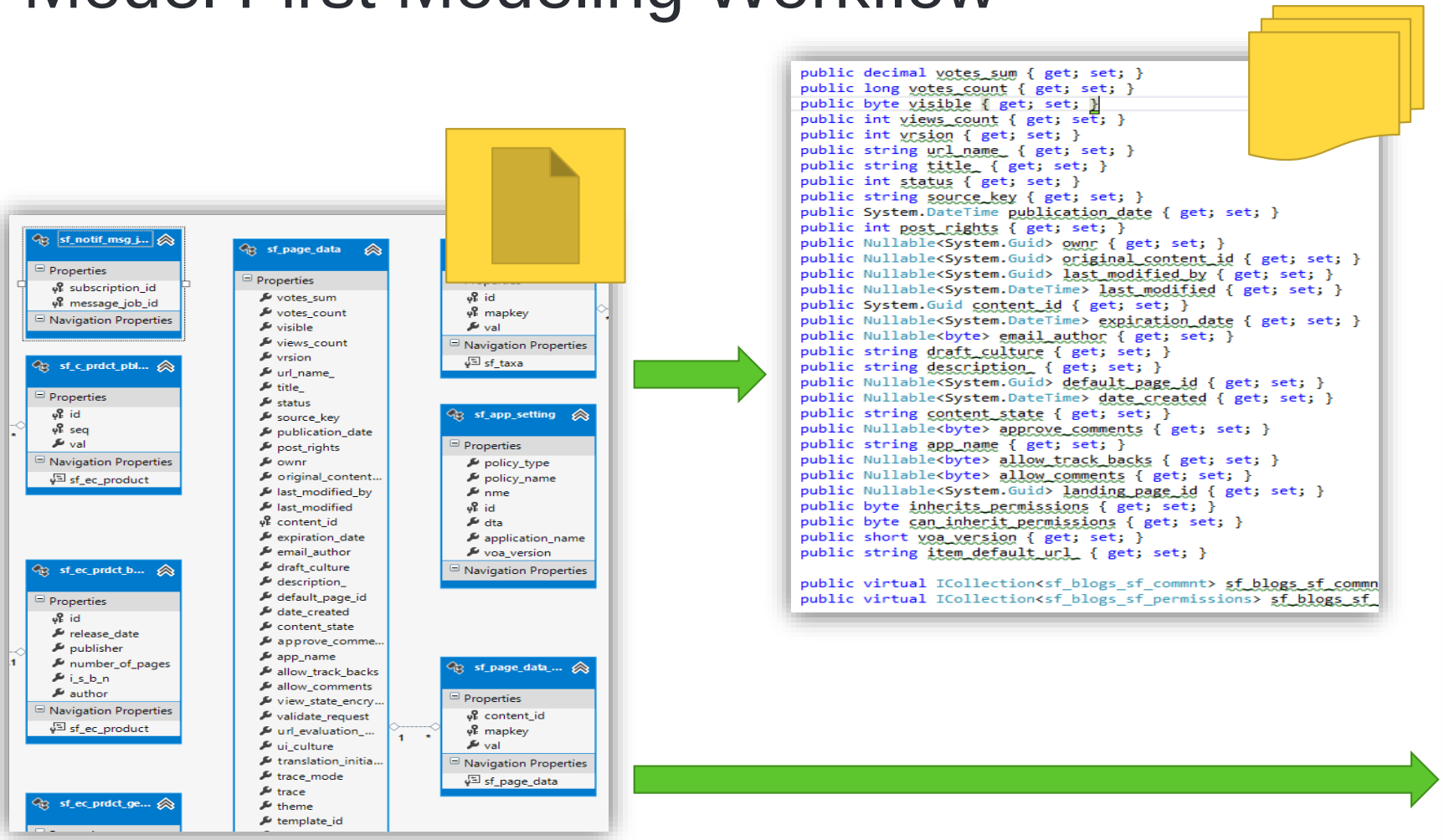
Database First Modeling Workflow

- Create models as database tables and then generate code (models) from them

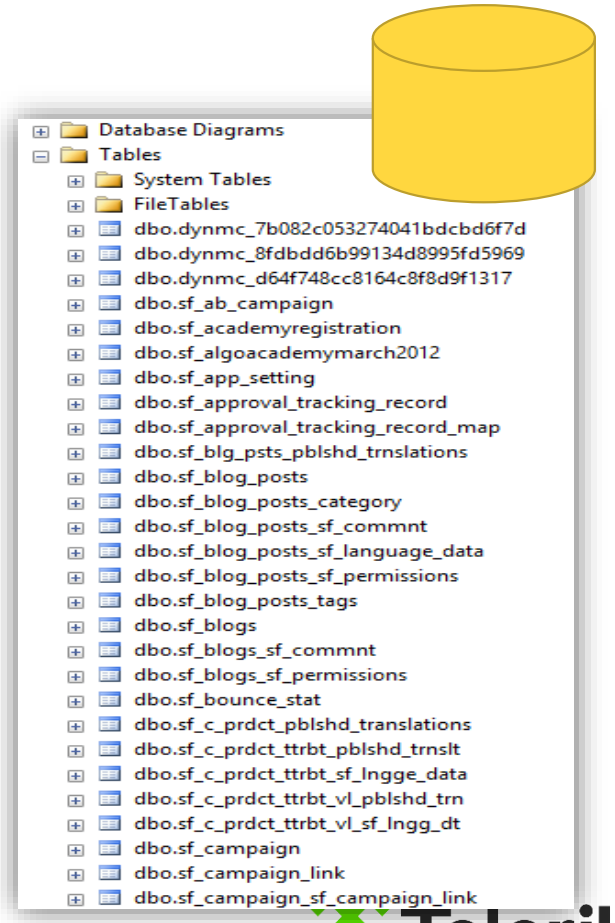


Telerik

Model First Modeling Workflow



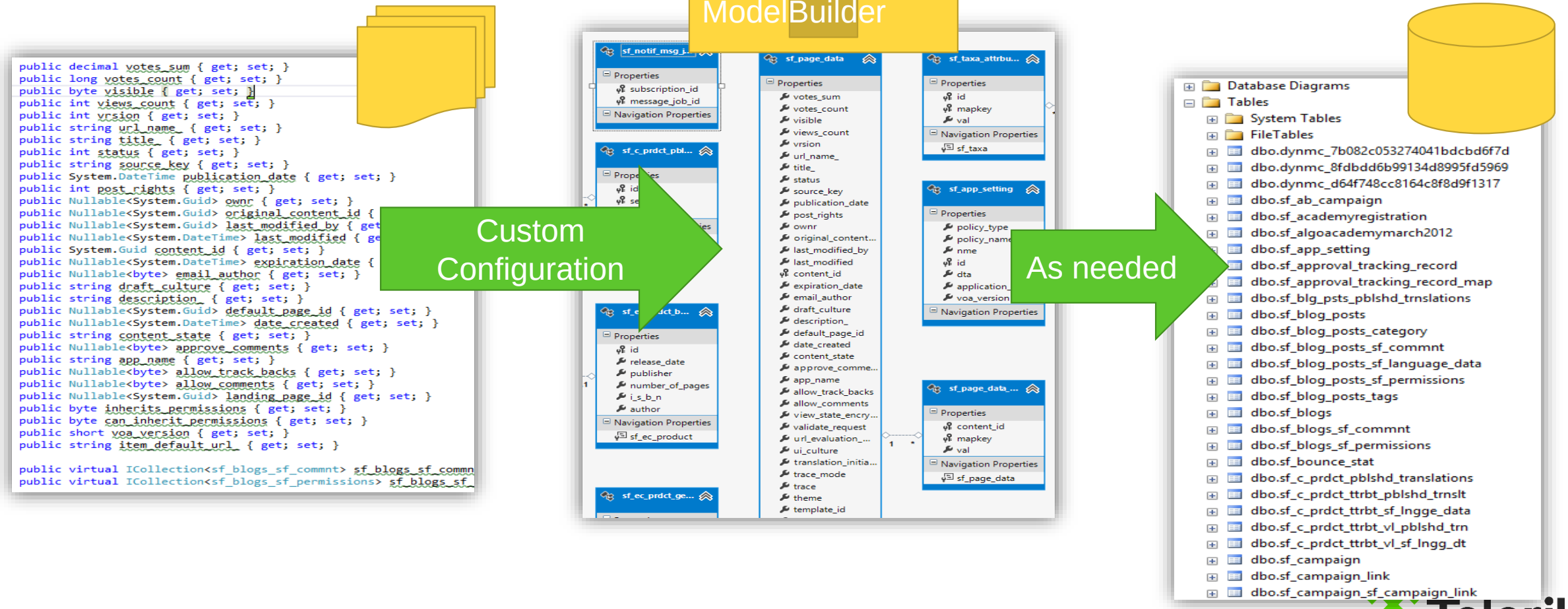

```
public decimal votes_sum { get; set; }
public long votes_count { get; set; }
public byte visible { get; set; }
public int views_count { get; set; }
public int vrsion { get; set; }
public string url_name_ { get; set; }
public string title_ { get; set; }
public int status { get; set; }
public string source_key { get; set; }
public System.DateTime publication_date { get; set; }
public int post_rights { get; set; }
public Nullable<System.Guid> ownr { get; set; }
public Nullable<System.Guid> original_content_id { get; set; }
public Nullable<System.Guid> last_modified_by { get; set; }
public Nullable<System.DateTime> last_modified { get; set; }
public System.Guid content_id { get; set; }
public Nullable<System.DateTime> expiration_date { get; set; }
public Nullable<byte> email_author { get; set; }
public string draft_culture { get; set; }
public string description_ { get; set; }
public Nullable<System.Guid> default_page_id { get; set; }
public Nullable<System.DateTime> date_created { get; set; }
public string content_state { get; set; }
public Nullable<byte> approve_comments { get; set; }
public string app_name { get; set; }
public Nullable<byte> allow_track_backs { get; set; }
public Nullable<byte> allow_comments { get; set; }
public Nullable<System.Guid> landing_page_id { get; set; }
public byte inherits_permissions { get; set; }
public byte can_inherit_permissions { get; set; }
public short voa_version { get; set; }
public string item_default_url_ { get; set; }

public virtual ICollection<sf_blogs_sf_commnt> sf_blogs_sf_commn
public virtual ICollection<sf_blogs_sf_permissions> sf_blogs_sf_
```



Telerik

Code First Modeling Workflow

Domain classes



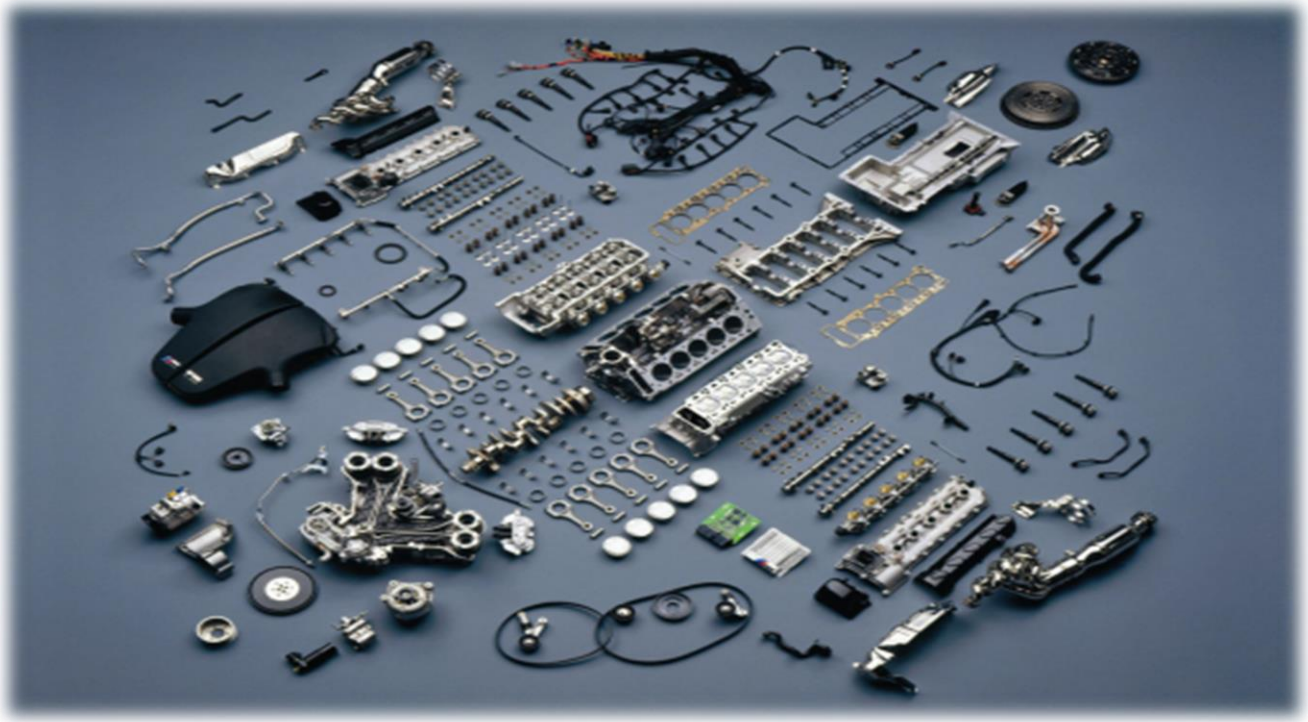
Telerik

Why Use Code First?

- Write code without having to define mappings in XML or create database models
- Define objects in POCO
  - Reuse these models and their attributes
- No base classes required
- Enables database persistence with no configuration
  - Can use automatic migrations
- Can use Data Annotations (Key, Required, etc.)

Telerik

Code First Main Parts



Domain Classes (Models)

- Bunch of normal C# classes (POCO)
  - May contain navigation properties

```
public class PostAnswer
{
    public int PostAnswerId { get; set; }
    public string Content { get; set; }
    public int PostId { get; set; }
    public virtual Post Post { get; set; }
}
```

Primary key

Foreign key

Virtual for lazy loading

Navigation property

- Recommended to be in a separate class library

Telerik

Domain Classes (Models) (2)

```
public class Post
{
    private ICollection<PostAnswer> answers;
    public Post()
    {
        this.answers = new HashSet<PostAnswer>();
    }
    // ...
    public virtual ICollection<PostAnswer> Answers
    {
        get { return this.answers; }
        set { this.answers = value; }
    }
    public PostType Type { get; set; }
}
```

Prevents null reference exception

Navigation property

Enumeration

Demo: Creating Models

DbContext Class

- A class that inherits from DbContext
  - Manages model classes using DbSet type
  - Implements identity tracking, change tracking, and API for CRUD operations
  - Provides LINQ-based data access
- Recommended to be in a separate class library
  - Don't forget to reference the Entity Framework library (using NuGet package manager)
- If you have a lot of models it is recommended to use more than one DbContext

Telerik

DbSet Type

- Collection of single entity type
- Set operations: Add, Attach, Remove, Find
- Use with DbContext to query database

public class **DbSet<TEntity>** :
System.Data.Entity.Infrastructure.DbQuery<TEntity>
where **TEntity** : class
Member of System.Data.Entity

```
public DbSet<Post> Posts { get; set; }
```

Telerik

DbContext Example

```
using System.Data.Entity;

using CodeFirst.Models;

public class ForumContext : DbContext
{
    public DbSet<Category> Categories { get; set; }

    public DbSet<Post> Posts { get; set; }

    public DbSet<PostAnswer> PostAnswers { get; set; }

    public DbSet<Tag> Tags { get; set; }
}
```

Telerik

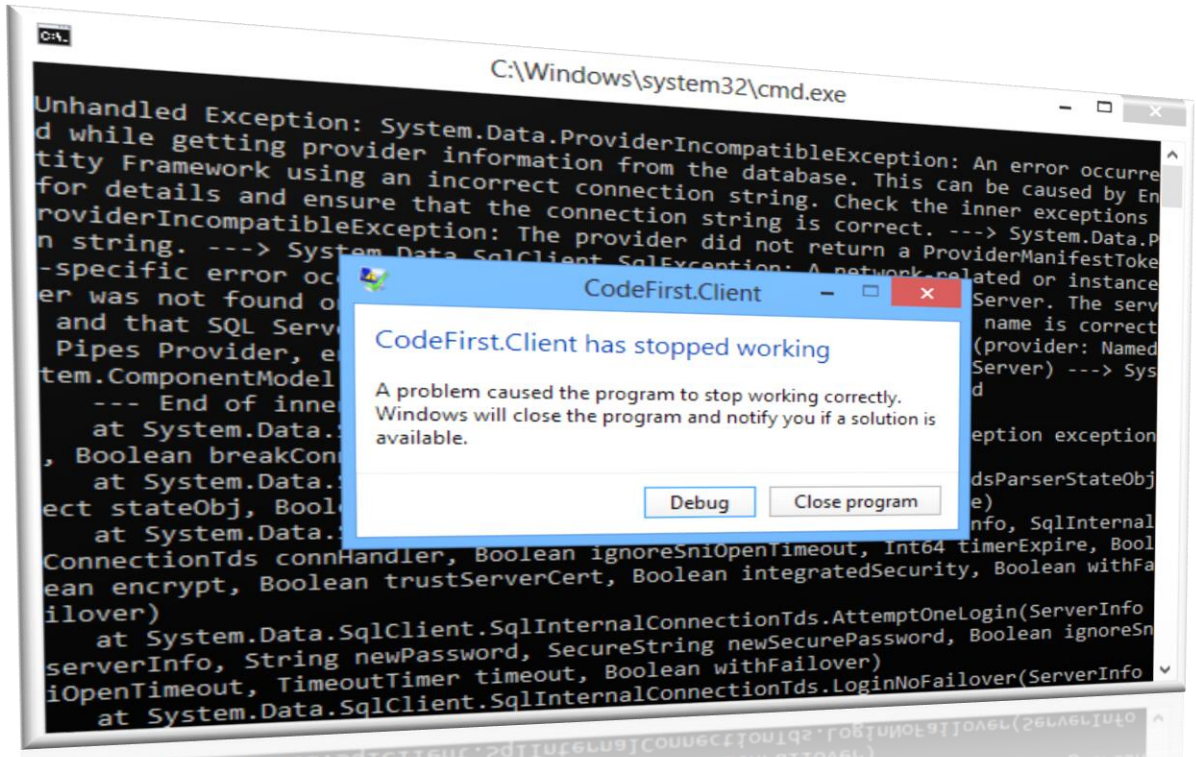
Demo: Creating DbContext

How to Interact With the Data?

- In the same way as when we use database first or model first approach

```
var db = new ForumContext();
var category = new Category { Parent = null, Name = "Database course", };
db.Categories.Add(category);

var post = new Post();
post.Title = "Срока на домашните";
post.Content = "Моля удължете срока на домашните";
post.Type = PostType.Normal;
post.Category = category;
post.Tags.Add(new Tag { Text = "домашни" });
post.Tags.Add(new Tag { Text = "срок" });
db.Posts.Add(post);
db.SaveChanges();
```

Telerik



Demo: Using The Data

Where is My Data?

- By default app.config file contains link to default connection factory that creates local db

```
<entityFramework>
  <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
    <parameters>
      <parameter value="v11.0" />
    </parameters>
  </defaultConnectionFactory>
</entityFramework>
```

- Server name by default: (localdb)\v11.0 or .\SQLEXPRESS
- We can use VS server explorer to view database

Telerik

How to Connect to SQL Server?

- First, create context constructor that calls base constructor with appropriate connection name

```
public class ForumContext : DbContext
{
    public ForumContext()
        : base("ForumDb")
    { }
    // ...
}
```

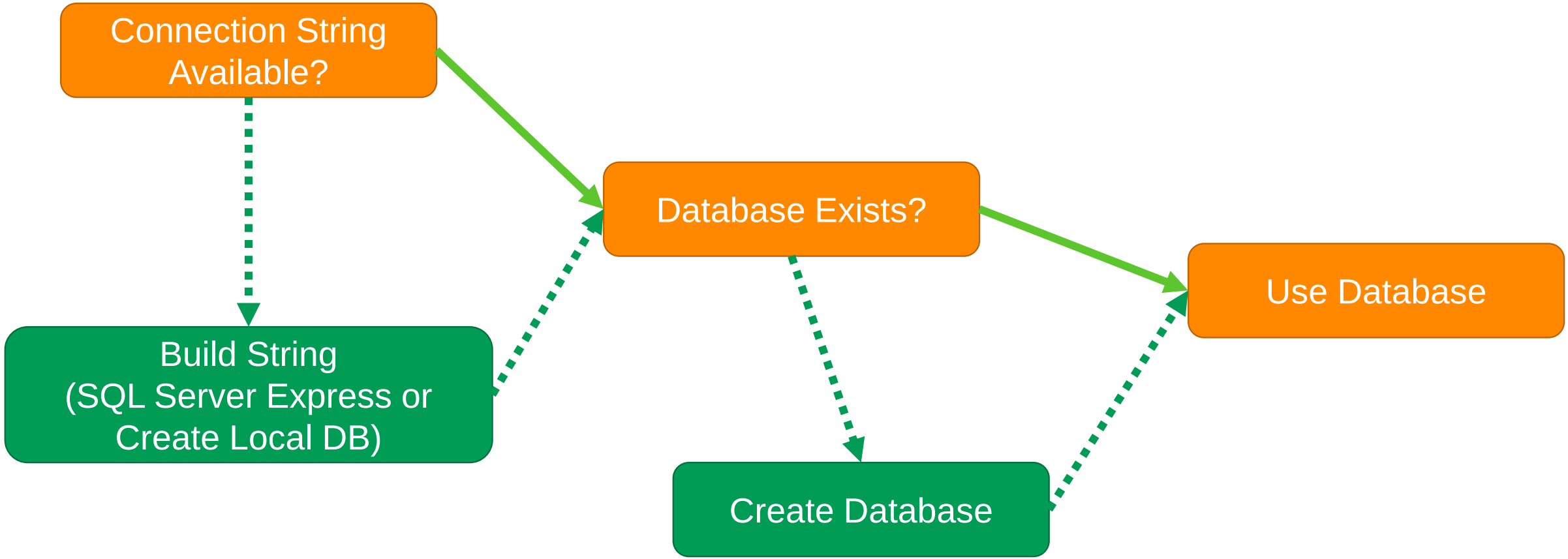
- Then add [illegible] in **app.config**

Server address might be .\SQLEXPRESS

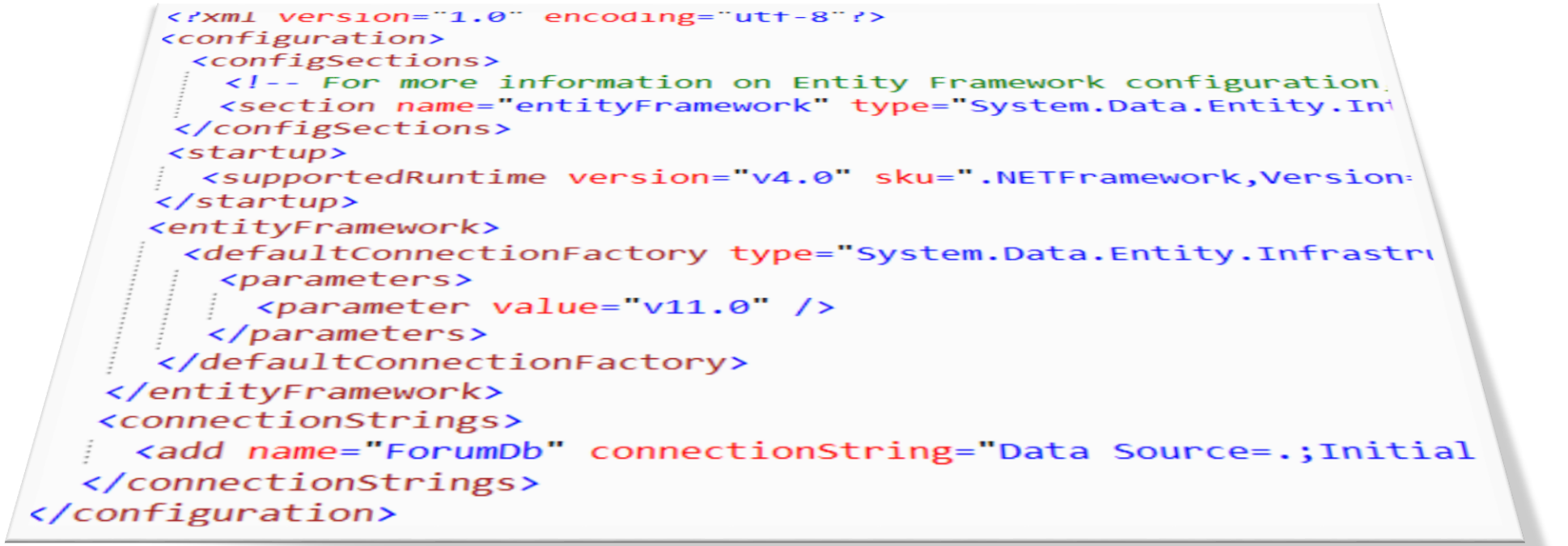
```
<connectionS[illegible]
   <add name="[illegible]" connectionString="Data Source=.;Initial
Catalog=ForumDb;Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Telerik

Database Connection Workflow

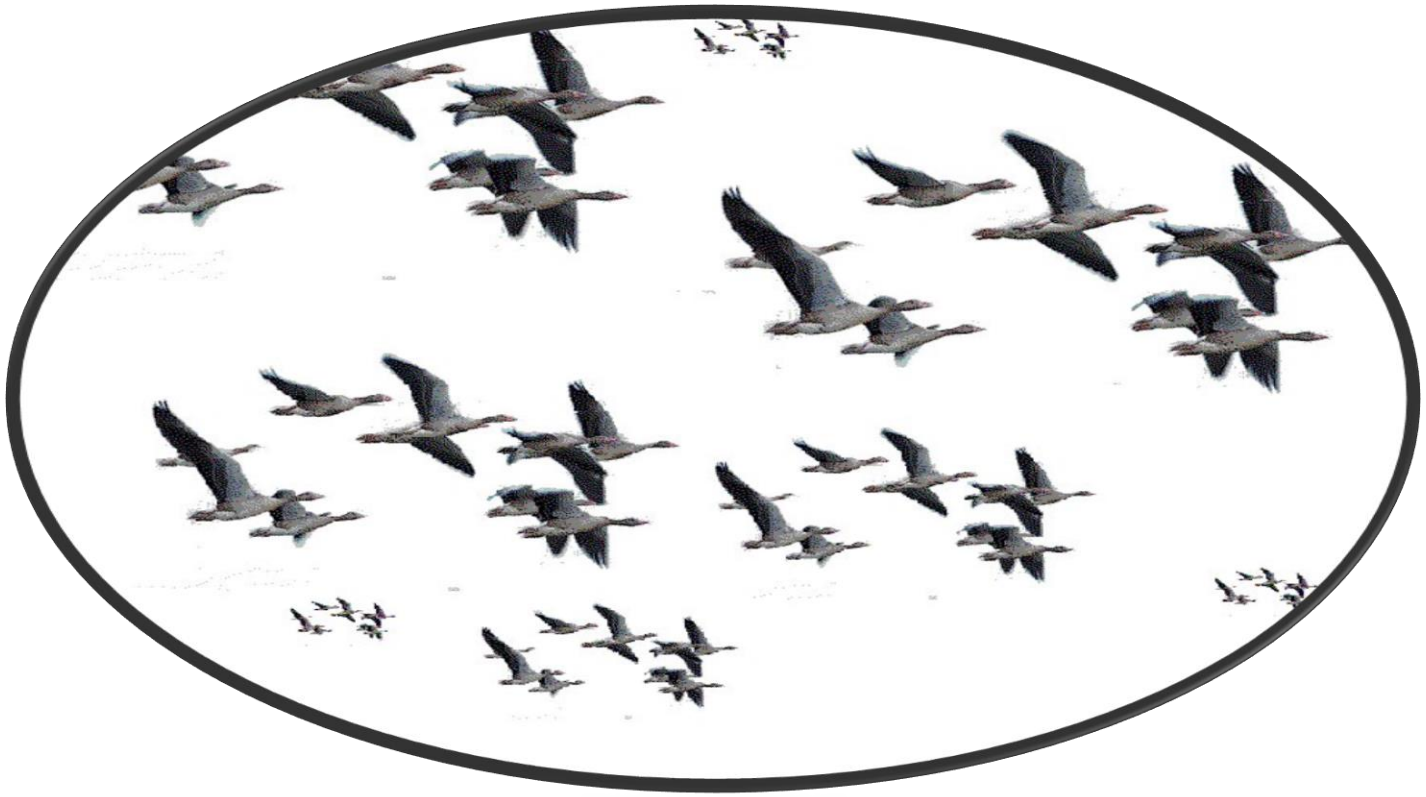


Telerik



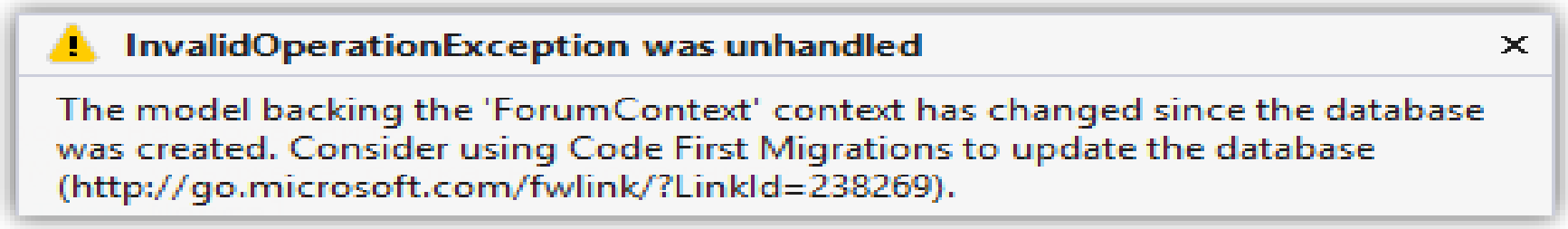
Demo: Change Database Connection

Using Code First Migrations



Changes in Domain Classes

- What happens when we change our models?
  - Entity Framework compares our model with the model in __MigrationHistory table
- By default Entity Framework only creates the database and don't do any changes after that
- Using Code First Migrations we can manage differences between models and database



Telerik

Code First Migrations

- Enable Code First Migrations
  - Open Package Manager Console
  - Run Enable-Migrations command
    - This will create some initial jumpstart code
    - -EnableAutomaticMigrations for auto migrations
- Two types of migrations
  - Automatic migrations
    - Set AutomaticMigrationsEnabled = true;
  - Code-based (providing full control)
    - Separate C# code file for every migration

Telerik

Database Migration Strategies

- CreateDatabaseIfNotExists (default)
- DropCreateDatabaseIfModelChanges
  - We loose all the data when change the model
- DropCreateDatabaseAlways
  - Great for automated integration testing
- MigrateDatabaseToLatestVersion
  - This option uses our migrations
- We can implement IDatabaseInitializer if we want custom migration strategy

Telerik

Use Code First Migrations

- First, enable code first migrations
- Second, we need to tell to Entity Framework to use our migrations with code (or app.config)

```
Database.SetInitializer(
    new MigrateDatabaseToLatestVersion
        <ForumContext, Configuration>());
```

- We can configure automatic migration

```
public Configuration()
{
    this.AutomaticMigrationsEnabled = true;
    this.AutomaticMigrationDataLossAllowed = true;
}
```

This will allow us to delete or change properties

Telerik

Seeding the Database

- During a migration we can seed the database with some data using the Seed method

```
protected override void Seed(ForumContext context)
{
    /* This method will be called after migrating to
       the latest version. You can use the
       DbSet<T>.AddOrUpdate() helper extension method
       to avoid creating duplicate seed data. E.g. */

    context.Tags.AddOrUpdate(new Tag { Text = "срок" });
    context.Tags.AddOrUpdate(new Tag { Text = "форум" });
}
```

- This method will be run every time (since EF 5)

Telerik

Demo: Code First Migrations

Configure Mappings

Configure Mappings

- Entity Framework respects mapping details from two sources
  - Data annotation attributes in the models
    - Can be reused for validation purposes
  - Fluent API code mapping configuration
    - By overriding OnModelCreating method
    - By using custom configuration classes
- Use one approach or the other

Telerik

Data Annotations

- There is a bunch of data annotation attributes in System.ComponentModel.DataAnnotations
  - [Key] – specifies the primary key of the table
  - For validation: [StringLength], [MaxLength], [MinLength], [Required]
  - Schema: [Column], [Table], [ComplexType], [ComplexType], [InverseProperty], [ForeignKey], [DatabaseGenerated], [NotMapped], [Index]
- In EF 6 we are able to add custom attributes by using custom conventions

Telerik

Fluent API for Mappings

- By overriding OnModelCreating method in DbContext class we can specify mapping configurations

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<Tag>().HasKey(x => x.TagId);
    modelBuilder.Entity<Tag>().Property(x => x.Text).IsUnicode(true);
    modelBuilder.Entity<Tag>().Property(x => x.Text).HasMaxLength(255);
    // modelBuilder.Entity<Tag>().Property(x => x.Text).IsFixedLength();
    base.OnModelCreating(modelBuilder);
}
```

Telerik

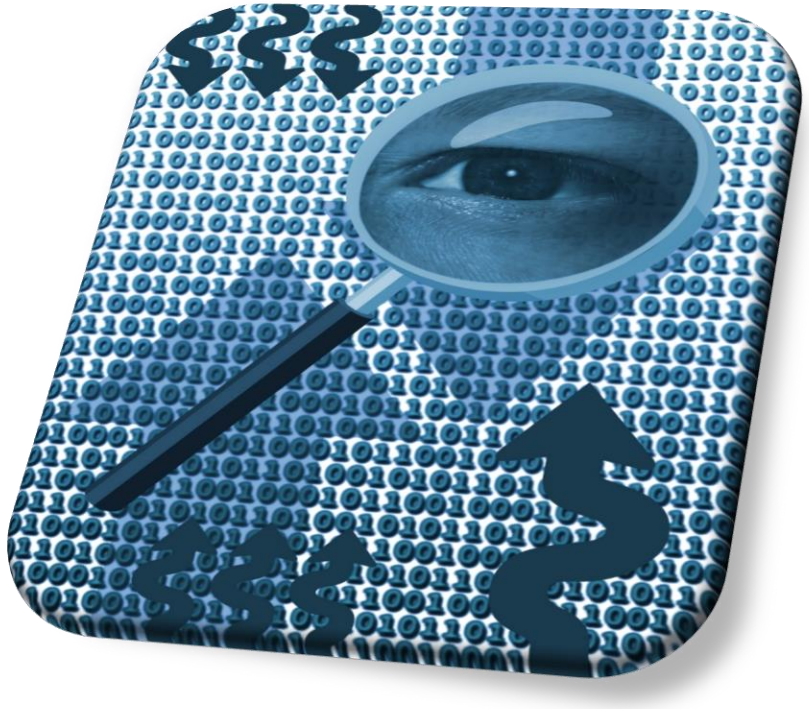
Fluent API Configurations

- .Entity()
  - Map: Table Name, Schema
  - Inheritance Hierarchies, Complex Types
  - Entity -> Multiple Tables
  - Table -> Multiple Entities
  - Specify Key (including Composite Keys)
- .Property()
  - Attributes (and Validation)
  - Map: Column Name, Type, Order
  - Relationships
  - Concurrency

Telerik

Demo: Configure Mappings

Working with Data



Reading Data with LINQ Query

- We can use extension methods (fluent API)

```
using (var context = new NorthwindEntities())
{
  var customerPhoness = context.Customers
                               .Select(c => c.Phone)
                               ere(c => c.City == "London")
                               ist();
}
```

This is called projection

ToList() method executes the query

- Find element by id

```
using (var context = new NorthwindEntities())
{
  var customer = context.Customers.Find(2);
  Console.WriteLine(customer.ContactTitle);
}
```

Telerik

Logging the Native SQL Queries

- To print the native database SQL commands executed on the server use the following:

```
var query = context.Countries;
Console.WriteLine(query.ToString());
```

Telerik

Creating New Data

- To create a new database row use the method Add(…) of the corresponding collection:

```
// Create new order object
Order order = new Order()
{
  OrderDate = DateTime.Now, ShipName = "Titanic",
  ShippedDate = new DateTime(1912, 4, 15),
  ShipCity = "Bottom Of The Ocean"
};
// Mark the object for inserting
context.Orders.Add(order);
context.SaveChanges();
```

This will execute an SQL INSERT

- SaveChanges() method call is required to post the SQL commands to the database

Cascading Inserts

- We can also add cascading entities to the database:

```
Country spain = new Country();
spain.Name = "Spain";
spain.Population = "46 030 10";
spain.Cities.Add(new City { Name = "Barcelona"} );
spain.Cities.Add(new City { Name = "Madrid"} );
countryEntities.Countries.Add(spain);
countryEntities.SaveChanges();
```

- This way we don't have to add each City individually
  - They will be added when the Country entity (Spain) is inserted to the database

Telerik

Updating Existing Data

- `DbContext` allows modifying entity properties and persisting them in the database
  - Just load an entity, modify it and call `SaveChanges()`
- The `DbContext` automatically tracks all changes made on its entity objects

```
Order order = northwindEntities.Orders.First();
order.OrderDate = DateTime.Now;
context.SaveChanges();
```

This will execute an SQL UPDATE

This will execute an SQL SELECT to load the first order

Deleting Existing Data

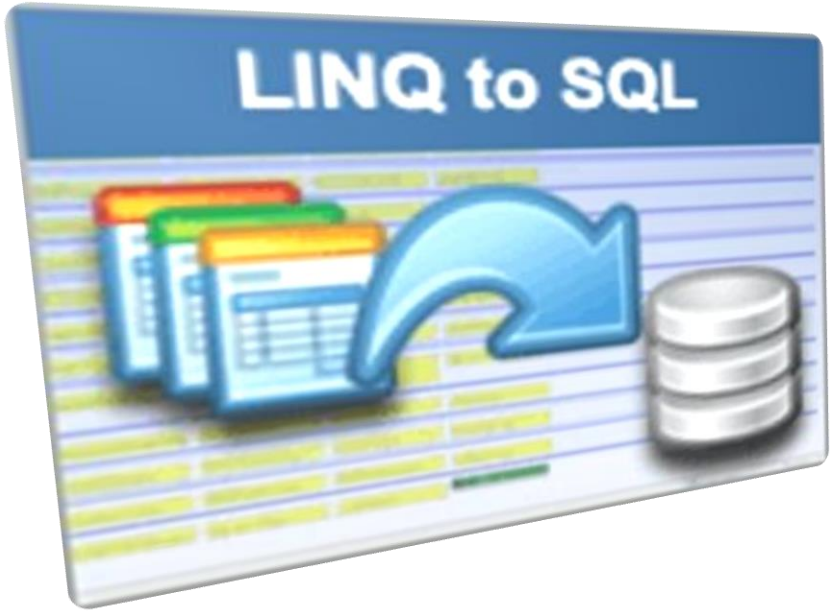
- Delete is done by `Remove()` on the specified entity collection
- `SaveChanges()` method performs the delete action in the database

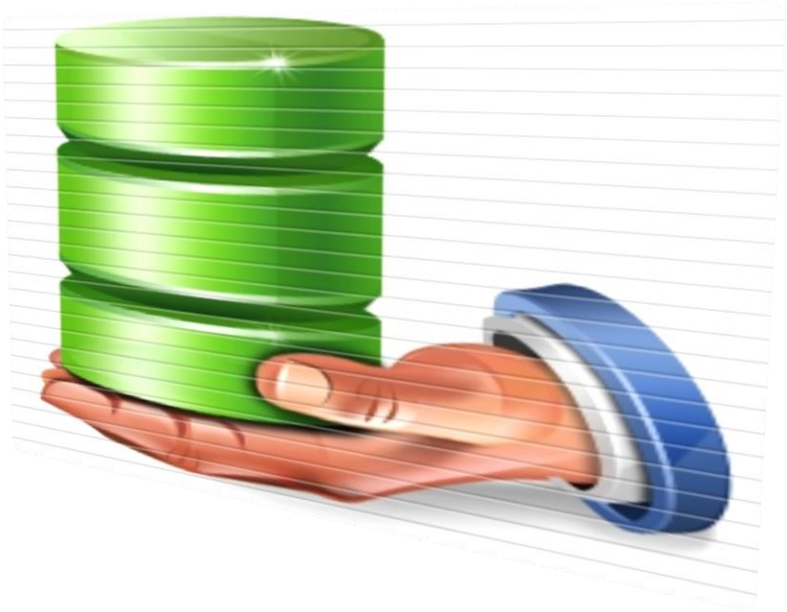
```
Order order = northwindEntities.Orders.First();

// Mark the entity for deleting on the next save
northwindEntities.Orders.Remove(order);
northwindEntities.SaveChanges();
```

This will execute an SQL DELETE command

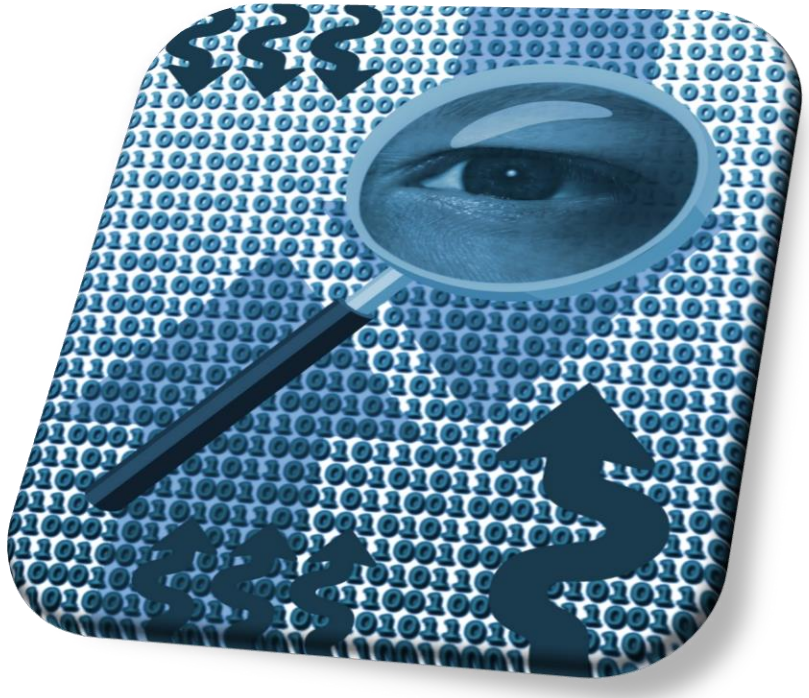
Telerik

Demo: CRUD Operations with Entity Framework





Entity Framework ++



Entity Framework

- EntityFramework is not perfect
  - Slow inserts, no update without selecting the entity, no delete by predicate
- You can use
  - EntityFramework.Extended - https://github.com/loresoft/EntityFramework.Extended
  - EntityFramework.Utilities - https://github.com/MikaelEliasson/EntityFramework.Utilities
- Performance test
  - http://blog.credera.com/technology-insights/microsoft-solutions/entity-framework-batch-operations-using-ef-utilities/

Telerik

The N+1 Query Problem

The N+1 Query Problem

- What is the N+1 Query Problem?
  - Imagine a database that contains tables `Products`, `Suppliers` and `Categories`
    - Each product has a supplier and a category
  - We want to print each `Product` along with its `Supplier` and `Category`:

```
foreach (var product in context.Products)
{
  Console.WriteLine("Product: {0}; {1}; {2}",
    product.ProductName, product.Supplier.CompanyName,
    product.Category.CategoryName);
}
```

Telerik

The N+1 Query Problem (2)

- This code will execute N+1 SQL queries:
  - Imagine we have 100 products in the database
    - That's ~ 201 SQL queries → very slow!
    - We could do the same with a single SQL query

```
foreach (var product in context.Products)
{
  Console.WriteLine("Product: {0}; {1}; {2}",
    product.ProductName, product.Supplier.CompanyName,
    product.Category.CategoryName);
}
```

One query to retrive the products

Additional N queries to retrieve the supplier for each product

Additional N queries to retrieve the category for each product

Telerik

Solution to the N+1 Query Problem

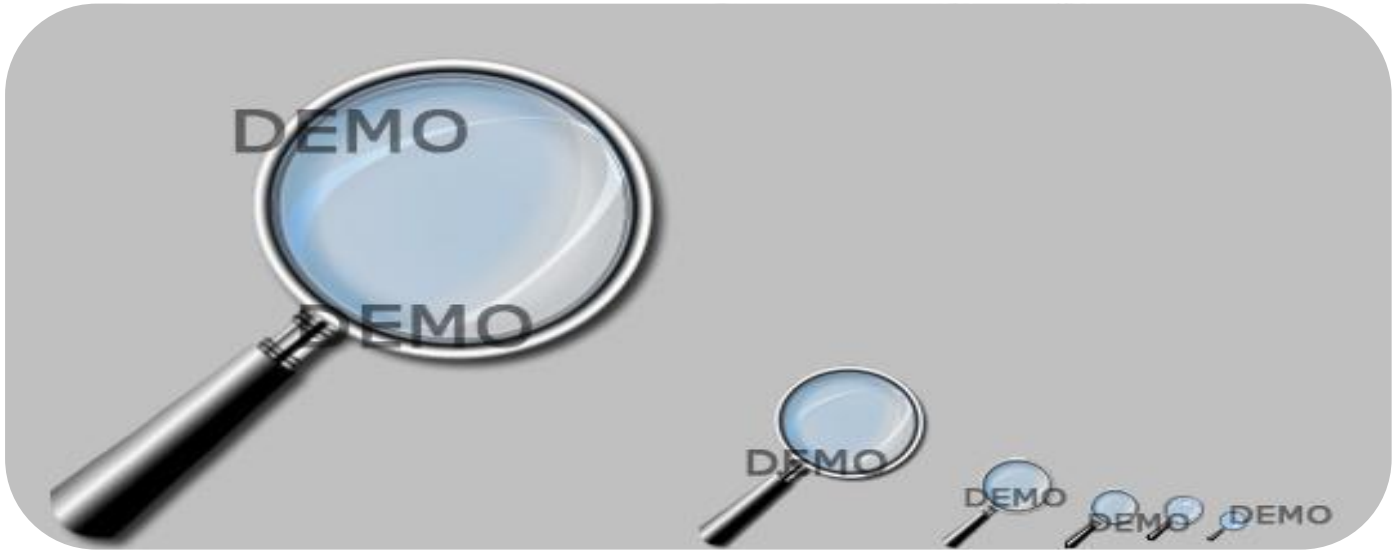
- Fortunately there is an easy way in EF to avoid the N+1 query problem:

Using `Include(…)` method only one SQL query with join is made to get the related entities

```
foreach (var product in context.Products.
  Include(p => p.Supplier).Include(p => p.Category))
{
  Console.WriteLine("Product: {0}; {1}; {2}",
    product.ProductName, product.Supplier.CompanyName,
    product.Category.CategoryName);
}
```

No additional SQL queries are made here for the related entities

Solution to the N+1 Query Problem



Incorrect Use of ToList()

Incorrect Use of ToList()

- In EF invoking `ToList()` executes the underlying SQL query in the database
  - Transforms `IQueryable<T>` to `List<T>`
  - Invoke `ToList()` as late as possible, after all filtering, joins and groupings
- Avoid such code:
  - This will cause all order details to come from the database and to be filtered later in the memory

```
List<Order_Detail> orderItemsFromTokyo =
  northwindEntities.Order_Details.ToList().
  Where(od => od.Product.Supplier.City == "Tokyo").ToList();
```

Telerik

Demo: Incorrect Use of ToList()

Incorrect Use of SELECT *

Deleting Entities Faster with Native SQL Query

Deleting Entities

- Deleting entities (slower):
  - Executes SELECT + DELETE commands

```
NorthwindEntities northwindEntities = new NorthwindEntities();
var category = northwindEntities.Categories.Find(46);
northwindEntities.Categories.Remove(category);
northwindEntities.SaveChanges();
```

- Deleting entities with native SQL (faster):
  - Executes a single DELETE command

```
NorthwindEntities northwindEntities = new NorthwindEntities();
northwindEntities.Database.ExecuteSqlCommand(
  "DELETE FROM Categories WHERE CategoryID = {0}", 46);
```

Telerik

Deleting Entities Faster with Native SQL Query

Performance optimization

- You can remove some of the default options

```
public DbContext()
{
        this.Configuration.AutoDetectChangesEnabled = false;
        this.Configuration.LazyLoadingEnabled = false;
        this.Configuration.ProxyCreationEnabled = false;
        this.Configuration.ValidateOnSaveEnabled = false;
}
```

Telerik

Entity Framework Code First

http://academy.telerik.com

Workshop

1. Using code first approach, create database for a forum system with the following tables:
    1. Forum Posts – title, content, author, date created, views, tags, category
        1. Questions and answers should be the same database type
    2. Forum Categories – title, description, posts, URL
    3. Forum Tags – title, posts
2. Write a console application that uses the data
3. Seed the data with random values

Telerik