Chapitre 3:

.Net Data Entity Framework Core

PLAN DU COURS

1. **Définition**
2. **Historique**
3. **Versions**
4. **Approches**
5. **EF core annotations & Fluent API**

Définition

- ✓ Framework de mapping objet/Database
- ✓ Entity Framework (EF) Core est une version légère, extensible, open source et multiplateforme de Entity Framework.
- ✓ On manipule des objets au lieu des tables
- ✓ Pas besoin d'apprendre le langage SQL => les requêtes sont écrites en C# (LINQ Queries)

Historique Entity framework

Entity Framework 1:

- ✓ Premier release
- ✓ Design-first uniquement
- ✓ EF Designer et outils pour Visual Studio 2008 (avec SP1)

Entity Framework 4:

- ✓ Introduit l'approche « code first »
- ✓ Améliorations des performances, en particulier pour le SQL Server

Historique Entity Framework

Entity Framework 5:

- ✓ L'EF5 est disponible sur NuGet
- ✓ Introduit l'approche «Database First »
- ✓ Types de données spatiales
- ✓ Un certain nombre d'améliorations des performances

Entity Framework 6:

- ✓ Requête et enregistrement asynchrone
- ✓ Personnaliser les conventions « Code First »
- ✓ Mapping pour insérer/mettre à jour/supprimer les procédures stockées
- ✓ Résolution des dépendances

Vers Entity Framework Core

Entity Framework Core:

- ✓ Moderne object-database mapper pour .NET Cross Platform
- ✓ Offre Support de LINQ queries
- ✓ Schémas de migration
- ✓ Support de plusieurs bases de données SQL Server/Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL….
- ✓ Etc….

Approches de génération de la BD

- ✓ Générez un modèle à partir d'une base de données existante.
- ✓ Codez manuellement un modèle pour correspondre à la base de données.
- ✓ Une fois qu'un modèle est créé, utilisez EF Migrations pour créer une base de données à partir du modèle.
- ✓ Les migrations permettent d'évoluer la base de données au fur et à mesure que le modèle change.

Approche database first vs. Code first

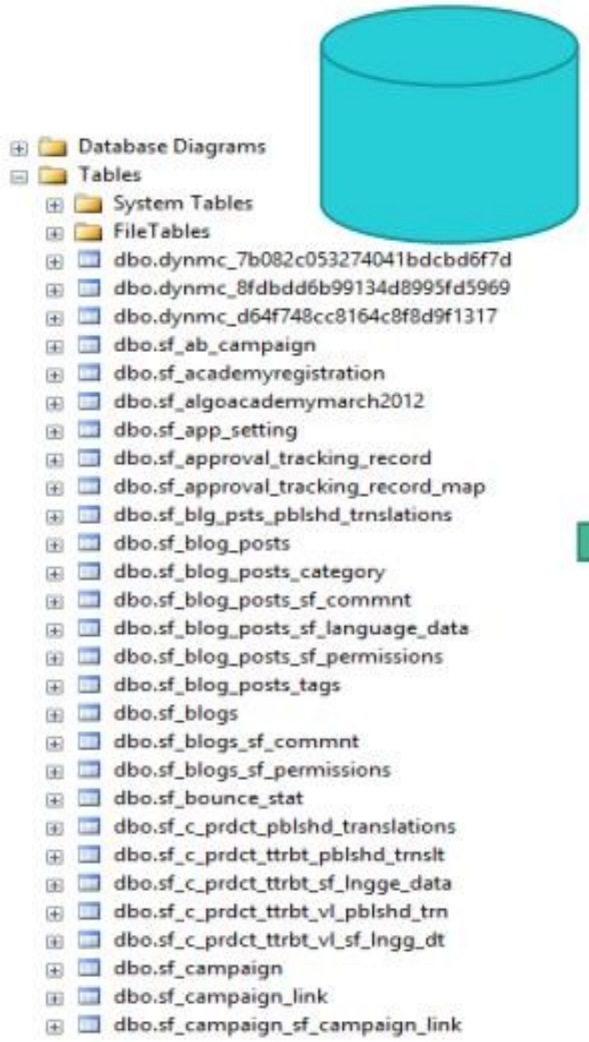
Approche DB first: **Commencer par concevoir la base de données et générer par la suite les classes et les relations entre eux**

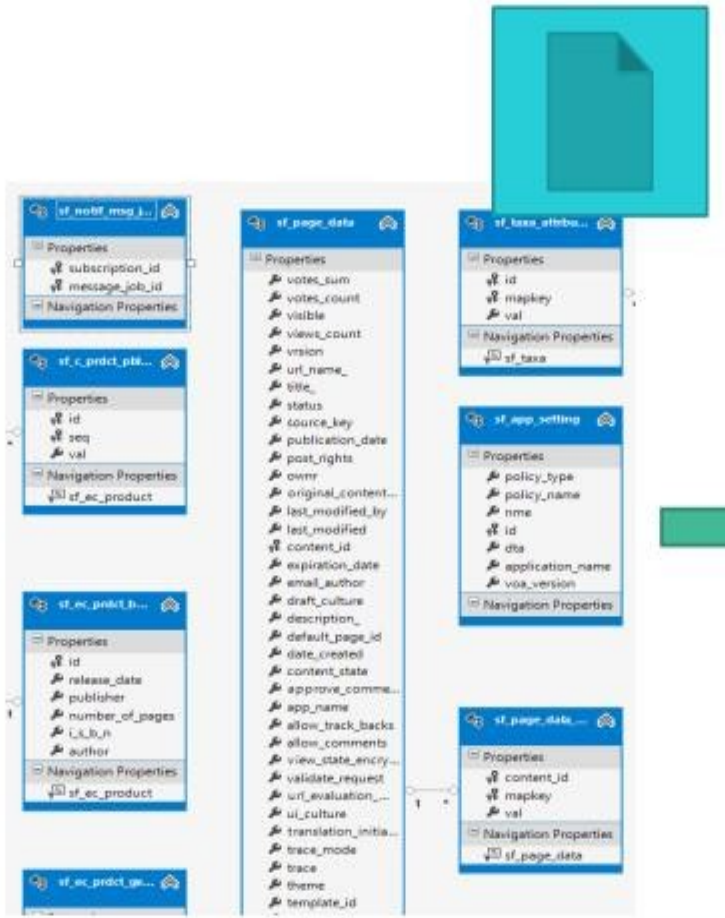
- ✓ Travailler avec une base de données existante
- ✓ Travailler et gérer des données simples
- ✓ Petit projet sans modularité nécessaire

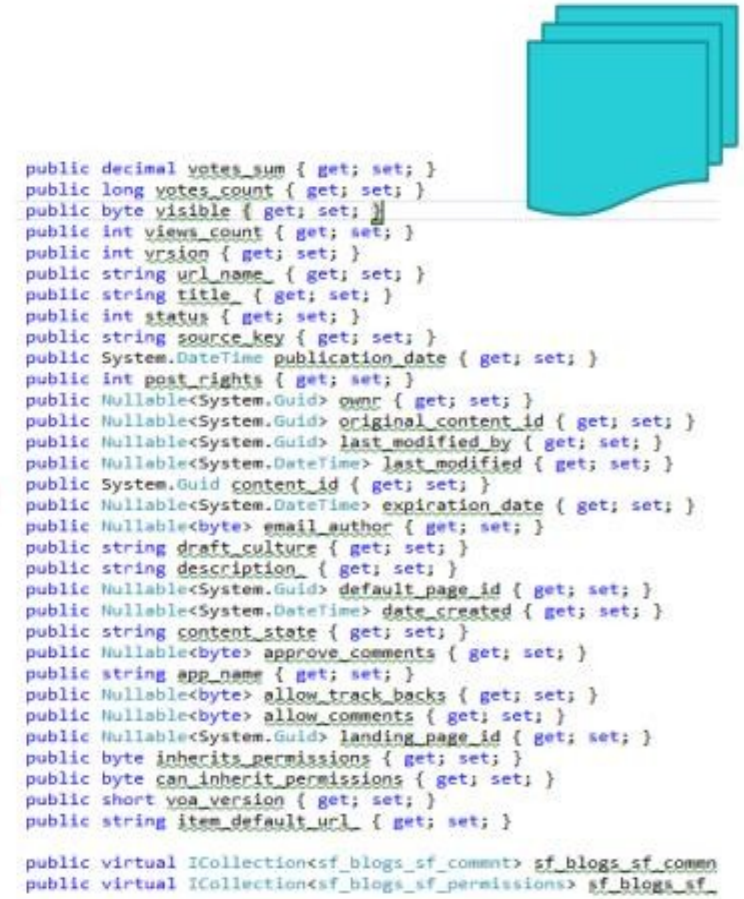
Approche Code first: **Générer les tables à partir des classes**

- ✓ Commencer l'application à partir de zéro
- ✓ Ne pas avoir de base de données prédéfinie
- ✓ On doit faire beaucoup de manipulations sur la base de données

Workflow approche database first



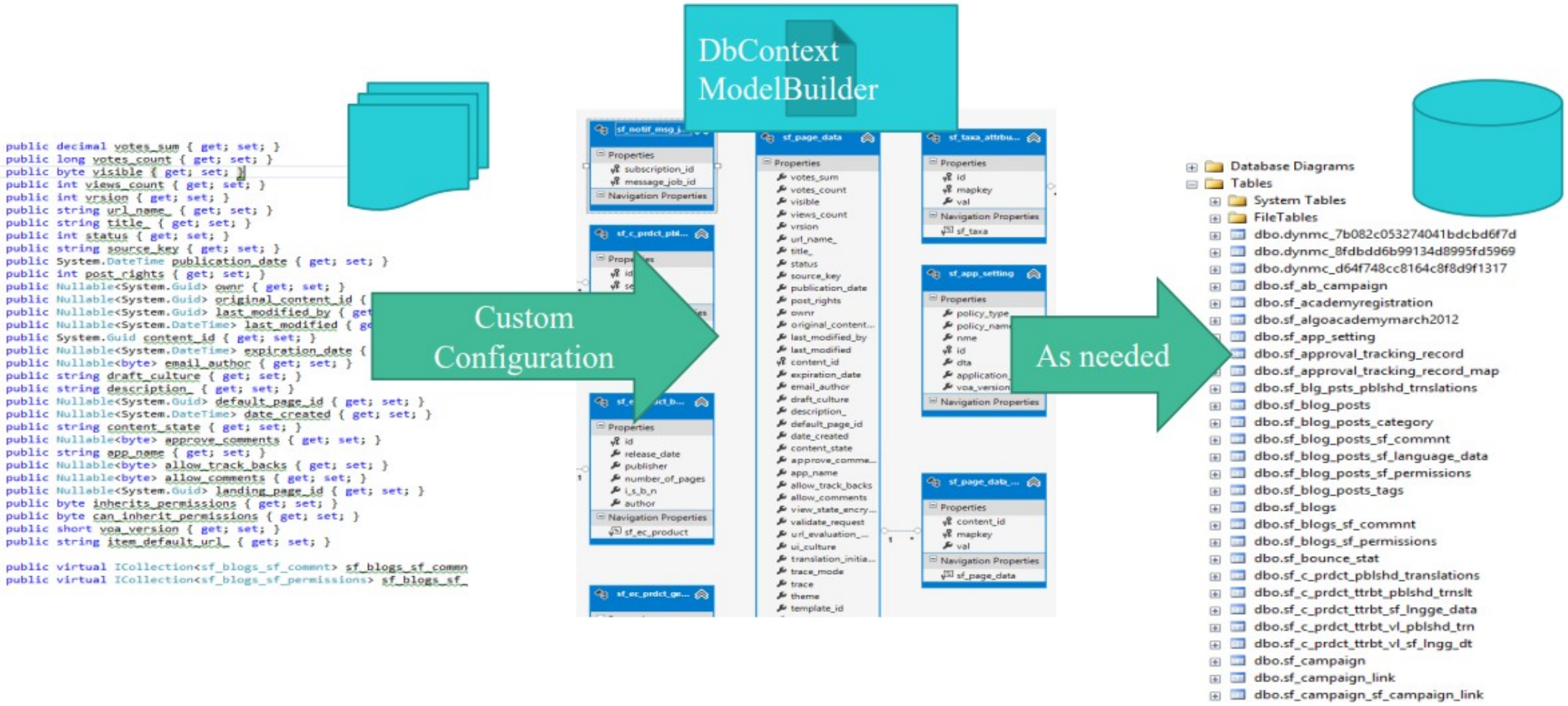




```
public decimal votes_sum { get; set; }
public long votes_count { get; set; }
public byte visible { get; set; }
public int views_count { get; set; }
public int vrsion { get; set; }
public string url_name_ { get; set; }
public string title_ { get; set; }
public int status { get; set; }
public string 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_
```

Workflow approche code first



EF Core

Chaîne de connexion

Dans ASP.NET Core le système de configuration est très flexible et la chaîne de connexion peut être stockée dans **appsettings.json** ou **Configuration Method** de la classe de contexte

```
{
  "ConnectionStrings": {
    "BloggingDatabase": "Server=(localdb)\
\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewD
b;Trusted_Connection=True;"
  },
}
```

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {

optionsBuilder.UseSqlServer(ConfigurationManager.Connection
Strings["BloggingDatabase"].ConnectionString);
    }
}
```

EF Core

Classe de contexte

- Une instance de la classe de contexte représente une session avec la base de données et peut être utilisé pour créer des requêtes et enregistrer des instances des entités.
- Ef Core ne supporte pas plusieurs opérations parallèles s'exécutant sur une même classe de contexte—> utiliser plusieurs instances de dbcontext.
- La classe de contexte est une classe qui dérive de DbContext et contient DbSet<TEntity> des propriétés pour chaque entité dans le modèle.

public class DbContext : IAsyncDisposable, IDisposable,
Microsoft.EntityFrameworkCore.Infrastructure.IInfrastructure<IServiceProvider
>, Microsoft.EntityFrameworkCore.Internal.IDbContextDependencies,
Microsoft.EntityFrameworkCore.Internal.IDbContextPoolable,
Microsoft.EntityFrameworkCore.Internal.IDbSetCache

https://learn.microsoft.com/fr-fr/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-7.0

EF Core

Classe de contexte : Exemple

```
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace Intro;

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            @"Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True");
    }
}
```

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

EF Core

Classe de contexte : remarques

- Supprimer DbContext après utilisation —> Libérer les ressources non managées et empêcher les fuites de mémoire dans le cas où l'instance reste référencée.
- DbContext n'est pas thread-safe. Ne partagez pas de contextes entre les threads.

Le point de départ de toute configuration DbContext est DbContextOptionsBuilder. Il existe trois façons d'utiliser ce générateur :

1. Dans AddDbContext et les méthodes associées
2. Dans OnConfiguring
3. Construit explicitement avec new

EF Core

Le type DbSet

```
public partial class SchoolDBEntities : DbContext
{
    public SchoolDBEntities()
        : base("name=SchoolDBEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<Course> Courses { get; set; }
    public virtual DbSet<Standard> Standards { get; set; }
    public virtual DbSet<Student> Students { get; set; }
    public virtual DbSet<StudentAddress> StudentAddresses { get; set; }
    public virtual DbSet<Teacher> Teachers { get; set; }
    public virtual DbSet<View_StudentCourse> View_StudentCourse { get; set; }
```

- ✓ DbSet représente un ensemble d'entité qui peuvent être utilisées pour des opérations de création, lecture et suppression.
- ✓ La classe de contexte (dérivée de DbContext) doit inclure des DbSet pour les entités pour assurer le mapping entre les tables de la base de données et les vues.

EF Core
LINQ Query sur les données

On peut créer des instances de classes d'entités pour traiter les données

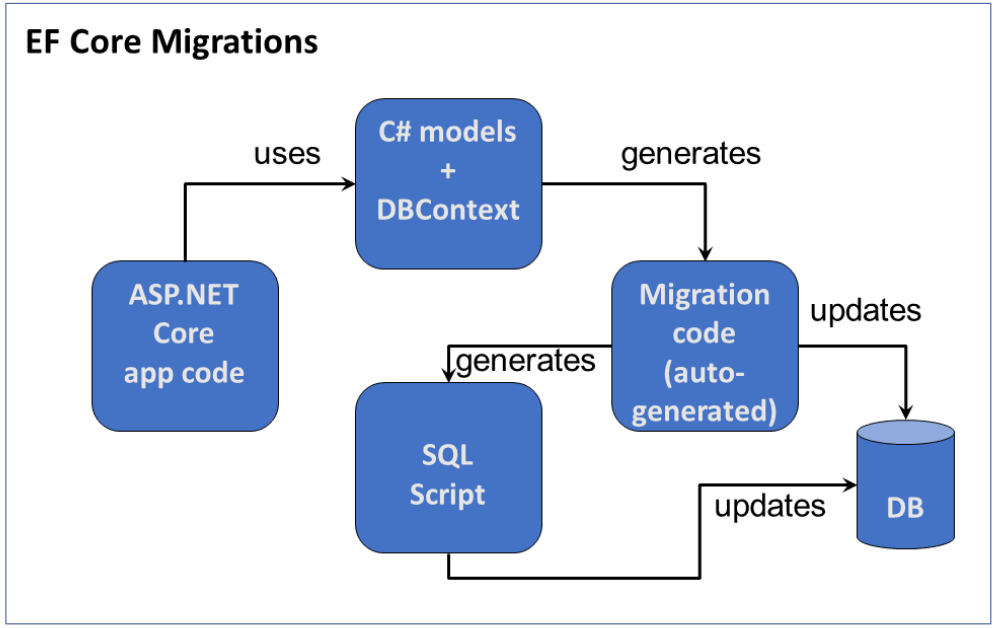
```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

EF Core

Migration en EF Core

- Préservation du schéma de la base de données
- Une migration compare un ancien modèle à un autre
- EF Core enregistre toutes les migrations appliquées dans une table d'historique spéciale, ce qui lui permet de savoir quelles migrations ont été appliquées.



EF Core
Migration en EF Core

dotnet ef migrations add InitialCreate

Add-Migration InitialCreate

EF Core créera un répertoire nommé **Migrations** dans votre projet et générera des fichiers.

dotnet ef database update

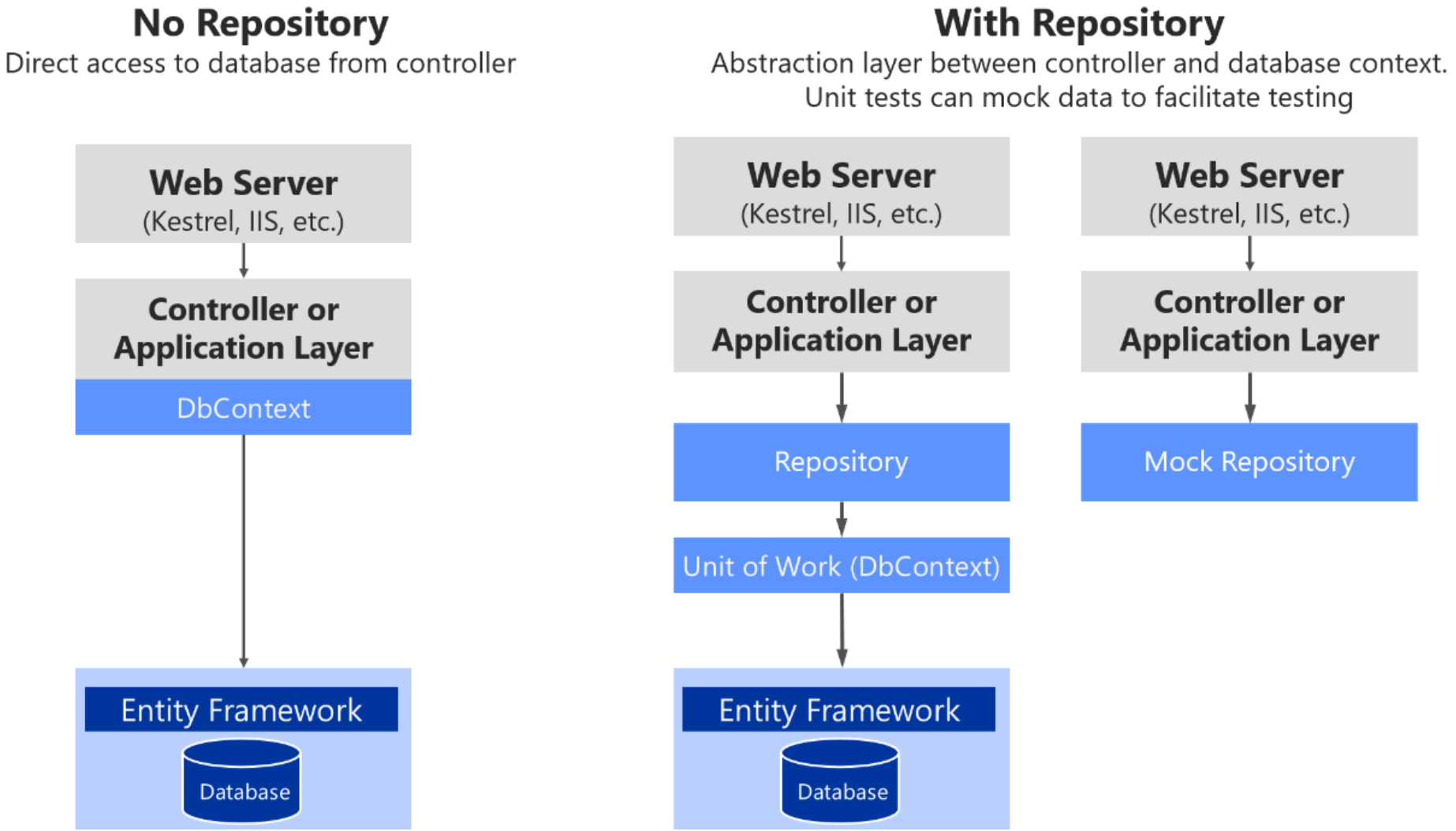
Update-Database

EF Core créera le schéma de la base de données

Référencer des types à partir d'un autre DbContext: Cela peut entraîner des conflits de migration. Pour éviter cela, excluez le type des migrations de l'un des DbContexts Via **API fluent.**

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<IdentityUser>()
        .ToTable("AspNetUsers", t =>
t.ExcludeFromMigrations());
}
```

Persistence avec EF Core : Architecture



EF CORE Conventions & Annotations

EF Core

- Lorsque vous utilisez Entity Framework Code First, le comportement par défaut consiste à mapper vos classes POCO à des tables à l'aide d'un ensemble de conventions intégrées à EF.
- Parfois, vous ne pouvez pas ou ne souhaitez pas suivre ces conventions → mapper des entités à autre chose que ce que les conventions dictent.

Data annotations

API Fluent

- Les annotations couvrent uniquement un sous-ensemble des fonctionnalités de l'API Fluent.
- **Il existe donc des scénarios de mappage qui ne peuvent pas être réalisés à l'aide d'annotations.**

Conventions EF Core

EF Core utilise un *modèle* de métadonnées pour décrire comment les types d'entités de l'application sont mappés à la base de données sous-jacente. Ce modèle est conçu à l'aide d'un ensemble de ***conventions***

Si une propriété s'appelle Id ou <EntitéId>, elle sera configurée comme étant une clé primaire —> Une classe contenant les deux privilégie Id

Les conventions pour la clé étrangère :

- **<navigation property name><principal primary key property name>Id**
- **<principal class name><primary key property name>Id**
- **<principal primary key property name>Id**

Si on ne souhaite pas inclure explicitement la clé étrangère, EF Core va créer une propriété shadow en utilisant le pattern <principal primary key property name>Id

Conventions EF Core

Reference Property Name in Dependent Entity	Foreign Key Property Name in Dependent Entity	Principal Primary Key Property Name	Foreign Key Column Name in DB
Grade	GradeId	GradeId	GradeId
Grade	-	GradeId	GradeId
Grade	-	Id	GradeId
CurrentGrade	CurrentGradeId	GradeId	CurrentGradeId
CurrentGrade	-	GradeId	CurrentGradeGradeId
CurrentGrade	-	Id	CurrentGradeId
CurrentGrade	GradeId	Id	GradeId

Conventions EF Core

- EF Core mappe une entité à une table ayant le même nom définit dans la propriété DbSet<TEntity>.
- EF Core assure aussi le mapping des entités n'ayant pas de DbSet<TEntity> correspondant si cette dernière fait partie d'une relation
- EF Core mappe les propriétés des entités aux colonnes de la base de données en leurs attribuant les même noms.
- EF Core reconnaît les types complexes

```
public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}
```

Conventions EF Core

Vous pouvez ignorer les conventions:

```
public class SchoolEntities : DbContext
{
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention, the generated tables
        // will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

Si vous souhaitez ignorer une entité utilisez l'attribut **NotMapped** ou l'API **DbModelBuilder.Ignore** Fluent.

Conventions EF Core

Vous pouvez utiliser des conventions personnalisées

```
public class ProductContext : DbContext
   {
      static ProductContext()
      {
         Database.SetInitializer(new DropCreateDatabaseIfModelChanges<ProductContext>());
      }
      public DbSet<Product> Products { get; set; }
     protected override void OnModelCreating(DbModelBuilder modelBuilder)
      {
         modelBuilder.Properties()
                  .Where(p => p.Name == "Key")
                  .Configure(p => p.IsKey());
      }
   }
```

Conventions EF Core

- Vous pouvez utiliser des classes de convention

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Properties<int>()
                .Where(p => p.Name.EndsWith("Key"))
                .Configure(p => p.IsKey());

    modelBuilder.Conventions.Add(new DateTime2Convention());
}
```

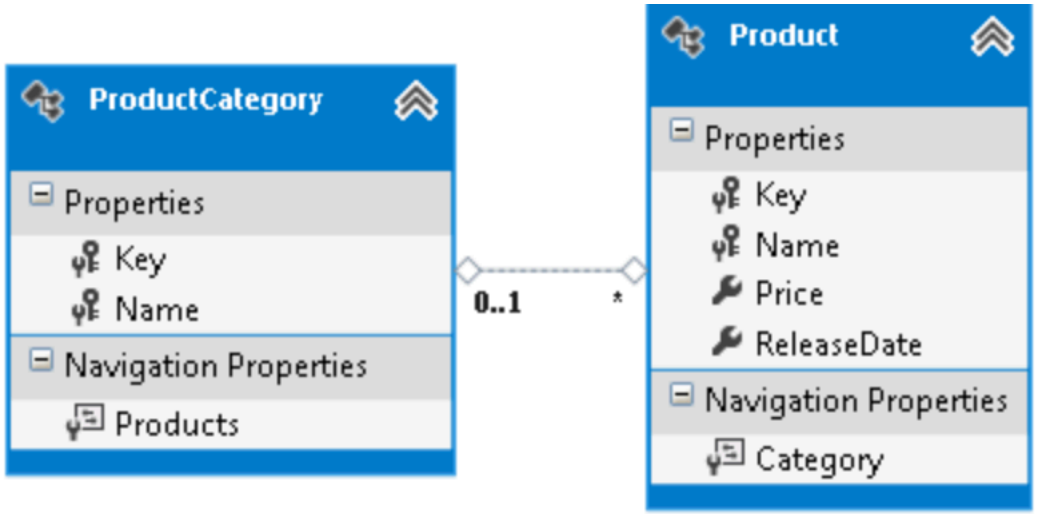
```
public class DateTime2Convention : Convention
{
    public DateTime2Convention()
    {
        this.Properties<DateTime>()
            .Configure(c => c.HasColumnType("datetime2"));
    }
}
```

Conventions EF Core

Que fait ce code?

```
modelBuilder.Properties<int>()
                .Where(x => x.Name == "Key")
                .Configure(x => x.IsKey().HasColumnOrder(1));

    modelBuilder.Properties()
                .Where(x => x.Name == "Name")
                .Configure(x => x.IsKey().HasColumnOrder(2));
```



Data Annotations

Clé
Requis
MaxLength et MinLength
NotMapped
ComplexType
ConcurrencyCheck
TimeStamp
Table et colonne
DatabaseGenerated
Index
Attributs de relation : InverseProperty et ForeignKey

Data Annotations

L'attribut **[PrimaryKey(nameof(State), nameof(LicensePlate))]** a été introduit dans EF Core 7.0. Utilisez l'API Fluent dans les versions antérieures.

```
internal class Car
{
[PrimaryKey(nameof(State), nameof(LicensePlate))]
    public string State { get; set; }
    public string LicensePlate { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

```
[Key]
    public string LicensePlate { get; set; }
```

Clé Composite Vs Clé simple

Data Annotations

NotMapped

Par convention, toutes les propriétés publiques avec un getter et un setter seront incluses dans le modèle.
Des propriétés spécifiques peuvent être exclues comme suit :

```
NotMapped]
  public DateTime LoadedFromDatabase { get; set; }
```

Column

Lors de l'utilisation d'une base de données relationnelle, les propriétés d'entité sont mappées aux colonnes de table portant le même nom que la propriété .

```
Column("blog_id")]
  public int BlogId { get; set; }
```

Data Annotations

Column(TypeName)

Pour configurer vos colonnes pour spécifier un type de données exact pour une colonne. Par exemple, le code suivant se configure `Url` en tant que chaîne non unicode avec une longueur maximale de `200` et `Rating` comme décimal avec précision et `5` échelle de `2`

```
 [Column(TypeName = "varchar(200)")]
    public string Url { get; set; }

    [Column(TypeName = "decimal(5, 2)")]
    public decimal Rating { get; set; }
```

MaxLength

La configuration d'une longueur maximale de 500 entraîne la création d'une colonne de type `nvarchar(500)` sur SQL Server

```
[MaxLength(500)]
```

Data Annotations

Required

Par convention, une propriété dont le type .NET peut contenir null est configurée comme facultative, tandis que les propriétés dont le type .NET ne peut pas contenir null sont configurées selon les besoins.

```
[Required] // Data annotations needed to configure as required
public string FirstName { get; set; }
[Required] // Data annotations needed to configure as required
public string LastName { get; set; }
public string MiddleName { get; set; } // Optional by convention
```

Sans NRT

Avec NRT

```
public string FirstName { get;       set; } // Required by convention
public string LastName { get;        set ; } // Required by convention
public string? MiddleName { get ; set } // Optional by convention
```

Data Annotations

Comment

Vous pouvez définir un commentaire texte arbitraire qui est défini sur la colonne de base de données, ce qui vous permet de documenter votre schéma dans la base de données :

```
public class Blog
{
    public int BlogId { get; set; }

    [Comment("The URL of the blog")]
    public string Url { get; set; }
}
```

DatabaseGenerated

Si vous mappez vos classes Code First aux tables qui contiennent des colonnes calculées, vous ne souhaitez pas que Entity Framework essaie de mettre à jour ces colonnes.

```
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public DateTime DateCreated { get; set; }
```