# Entity Framework

Bibliografie

Site Microsoft

## EF - Prerequisites

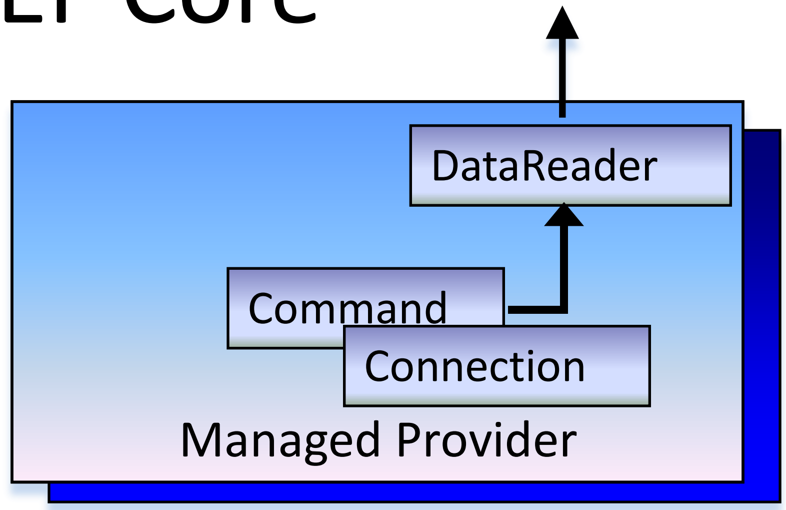
- Cunostinte despre .NET Framework, C#, MS SQL Server, Visual Studio.
- ADO .NET Architecture.

## EF - Tools

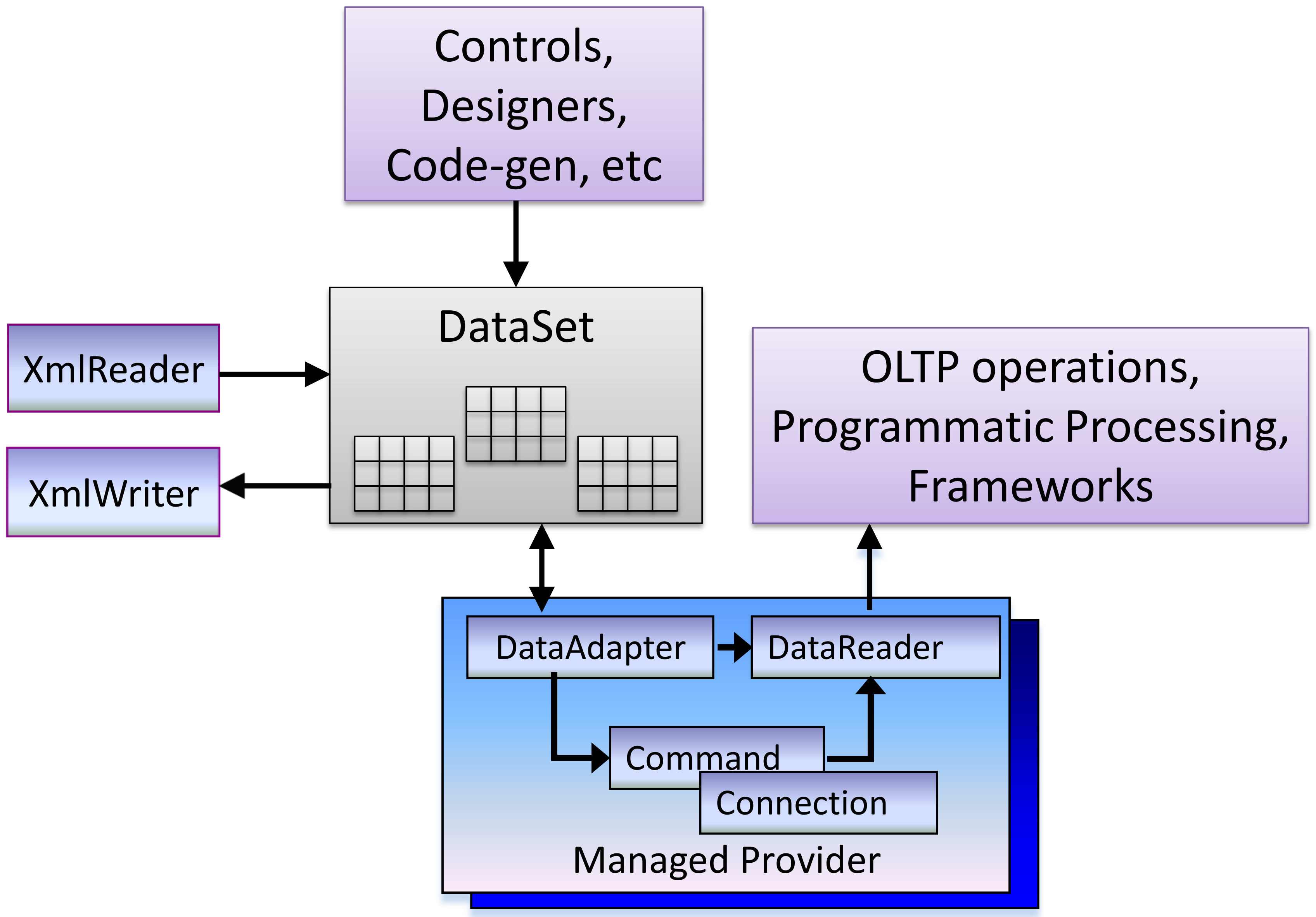
- – Visual Studio 2017 sau 2019.
- – SQL Server … sau SQLExpress Edition.
- – Entity Framework actualizat folosind NuGet Package.

# Entity Framework

- Entity Framework 6
- Entity Framework Core
- EF 6 vs EF Core

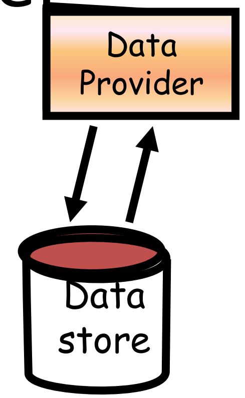


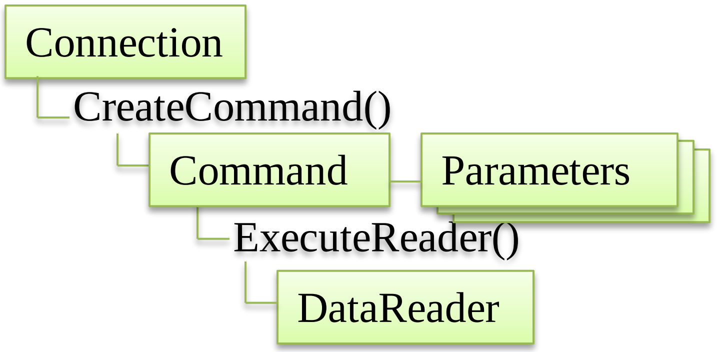
# ADO.NET Architecture


Controls, Designers, Code-gen, etc
DataSet
XmlReader
XmlWriter
OLTP operations, Programmatic Processing, Frameworks
DataAdapter
DataReader
Command
Connection
Managed Provider

# ADO.NET Data Provider

- Specific pentru fiecare DBMS
  - Expune in mod direct interfete catre consumatori.
- ADO.NET DataProvider Object Model
  - Connection
    - Stabileste o conexiune la DataSource
  - Transaction
    - Control explicit al tranzactiei
  - Command
    - Executa instructiuni SQL
  - DataReader
    - Forward-only, Read-Only





## Ce este EF?

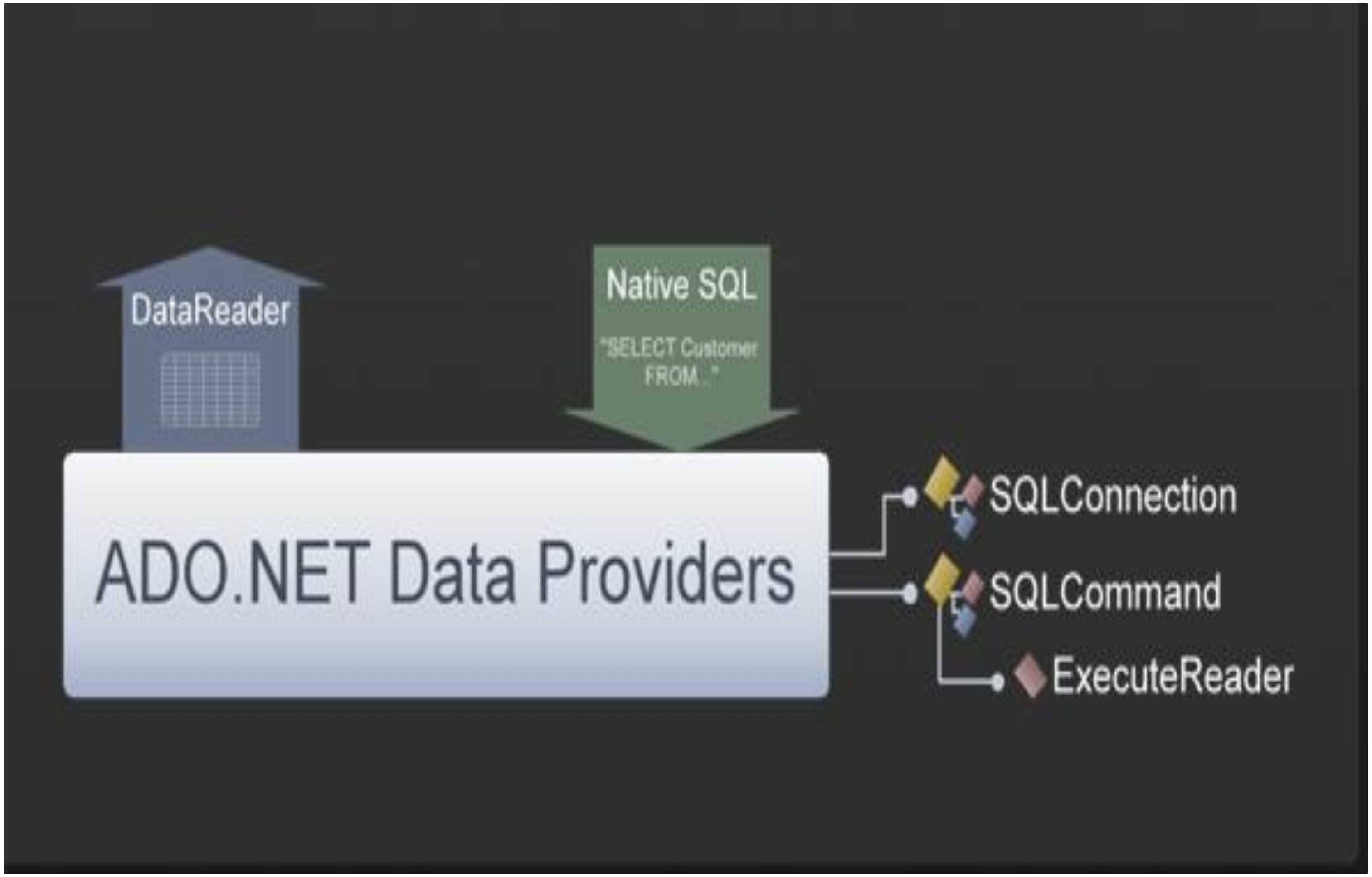
- EF este un “Object Relational Mapping (ORM) framework” ce ofera dezvoltatorilor  un mecanism automat  pentru memorarea si accesarea datelor intr-o baza de date.

## Continuare

- EF apare in 2008. Scop principal: interactiunea dintre aplicatiile .NET si baze de date relationale.
- Roluri ORM:
  - Creaza conexiunea la baza de date.
  - Executa comenzi asupra bazei de date.
  - Transforma rezultatele cererilor ca obiecte ale aplicatiei.
  - Gestioneaza modificarile efectuate asupra obiectelor aplicatiei.
  - Memoreaza aceste modificari in baza de date.

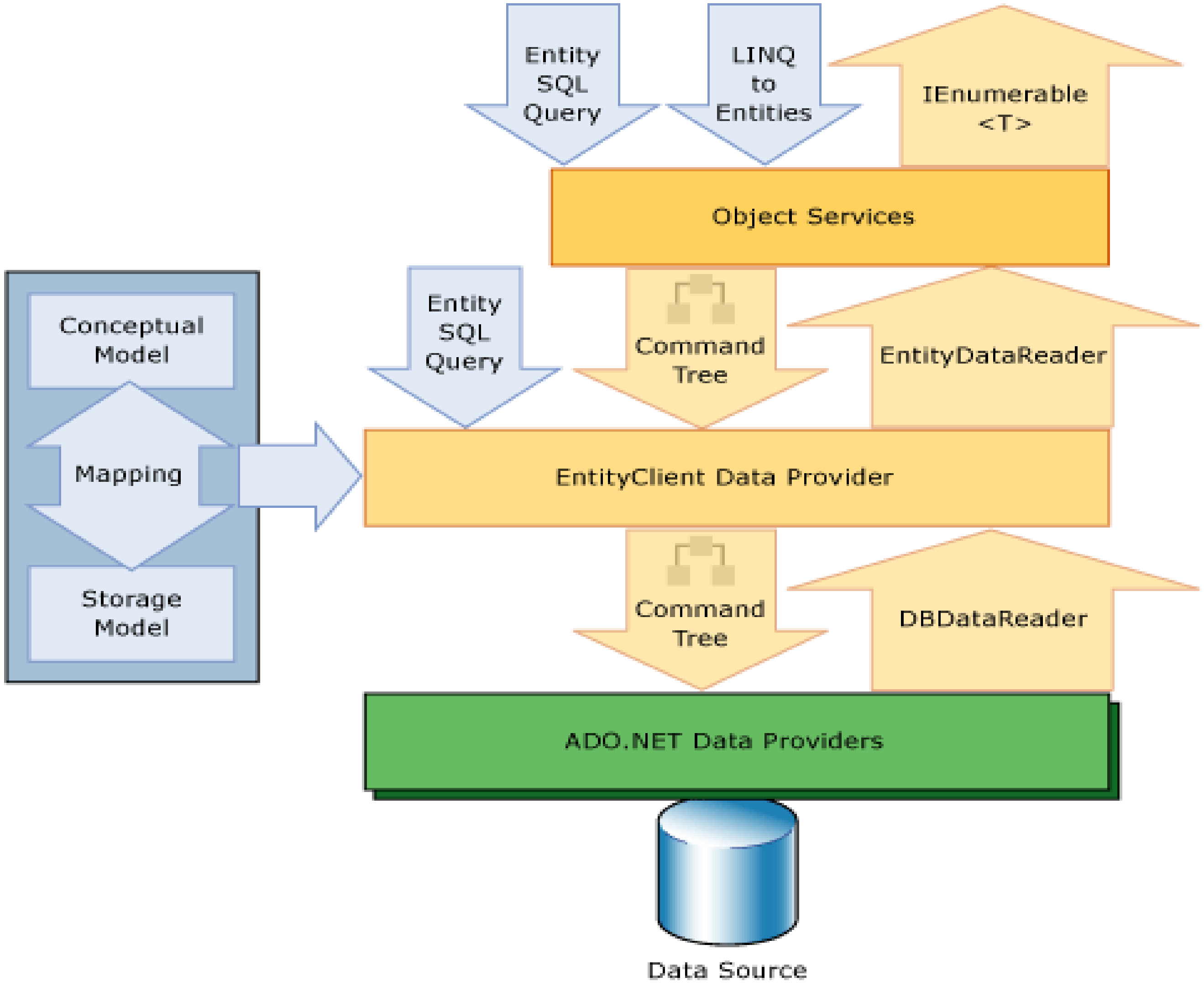
# De ce Entity Framework?

- Scopul unui ORM este de a creste productivitatea prin reducerea task-urilor redundante in ceea ce priveste persistenta datelor folosite in aplicatii.
- EF poate genera si executa comenzile necesare pentru citirea/scrierea datelor in baza de date.
- Se pot efectua cereri asupra *obiectelor* din model folosind “LINQ to Entities”.
- Rezultatul cererilor in EF este materializat in obiecte si nu in scalari ca in ADO.NET.

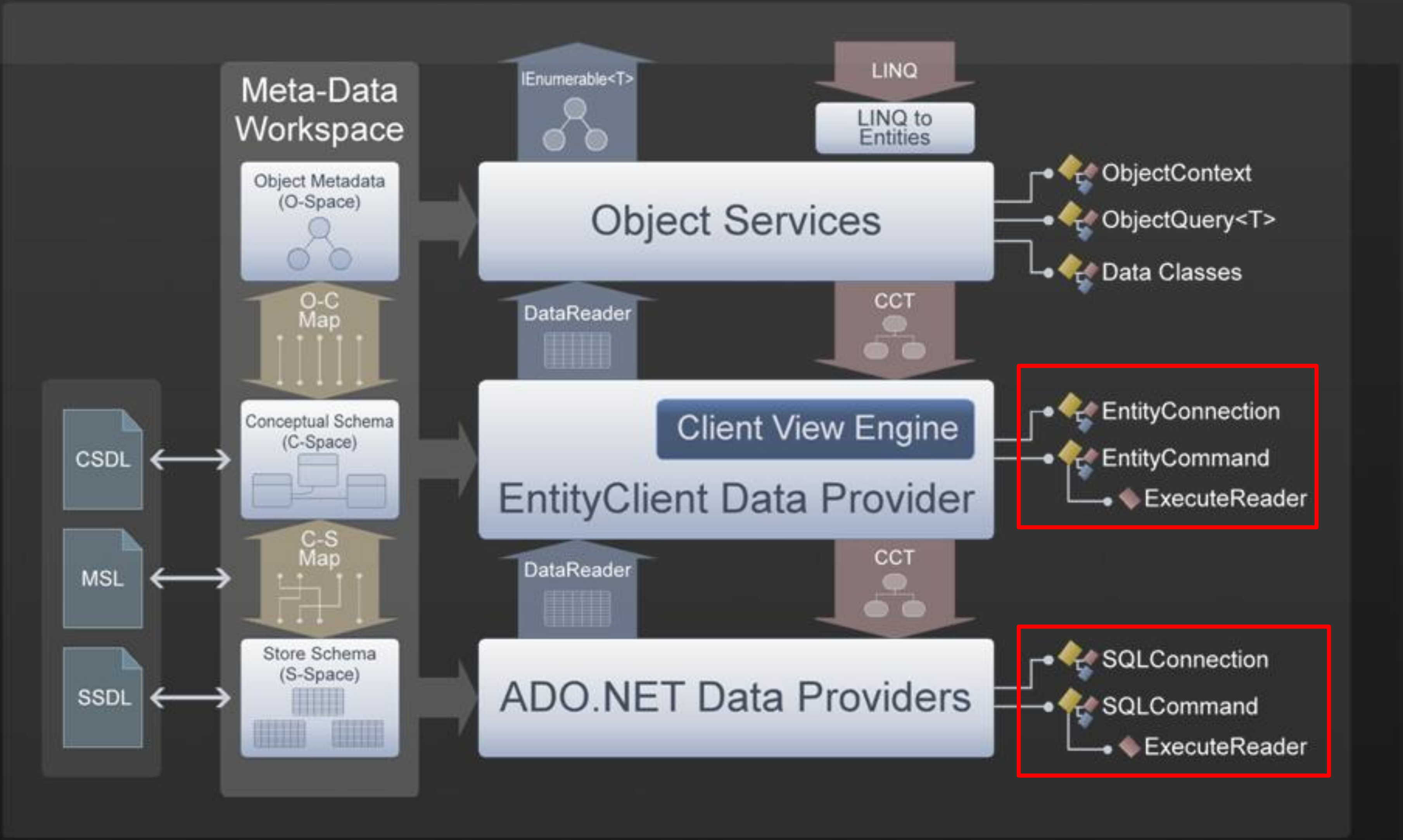

DataReader
Native SQL
"SELECT Customer FROM..."
ADO.NET Data Providers
SQLConnection
SQLCommand
ExecuteReader

## Exemplu utilizare Connection, Command si DataReader

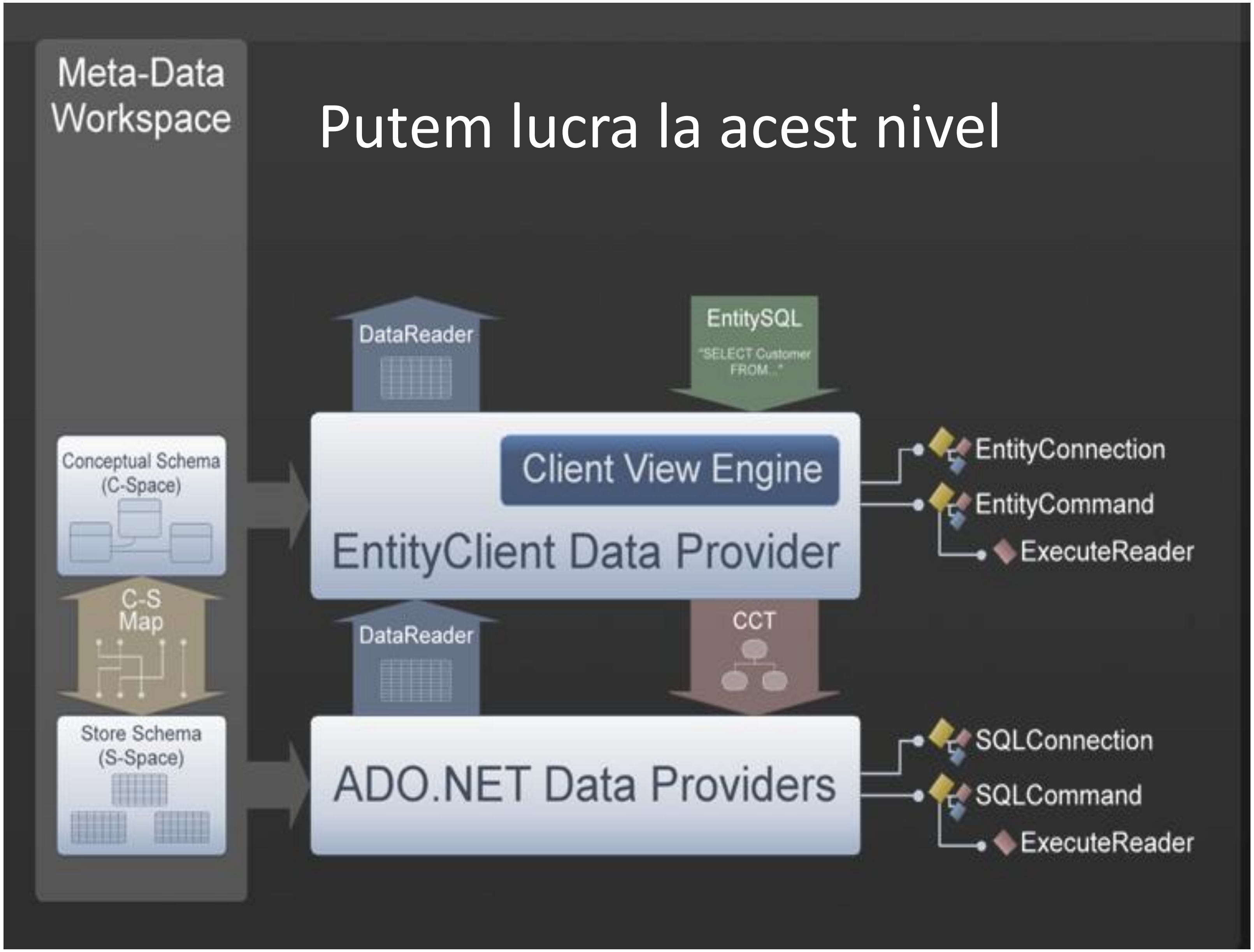
```
// ADO .NET
SqlConnection connection =
       new SqlConnection(connectionString);
connection.Open();
String query = “Select id, Titlu  From Curs”;
SqlCommand command = new SqlCommand(query, connection);
SqlDataReader reader = command.ExecuteReader();
while(reader.Read())
{
    int id = reader.GetInt32(0);
    string titlu = reader.GetString(1);
    // …
}
reader.Close();
command.Dispose();  // command.Close(); !!!
connection.Close();
```


Entity SQL Query
LINQ to Entities
IEnumerable <T>
Object Services
Conceptual Model
Mapping
Storage Model
Entity SQL Query
Command Tree
EntityDataReader
EntityClient Data Provider
Command Tree
DBDataReader
ADO.NET Data Providers
Data Source

# ADO.NET Entity Framework



## Putem lucra la acest nivel



# Exemplu: EntityConnection, EntityCommand, ExecuteReader

```
// Specify the provider name, server and database.
string providerName = "System.Data.SqlClient";
string serverName = ".";
string databaseName = "AdventureWorks";
SqlConnectionStringBuilder sqlBuilder = new
                                SqlConnectionStringBuilder();
sqlBuilder.DataSource = serverName;
sqlBuilder.InitialCatalog = databaseName;
sqlBuilder.IntegratedSecurity = true;
string providerString = sqlBuilder.ToString();
EntityConnectionStringBuilder entityBuilder =
                                new EntityConnectionStringBuilder();
```

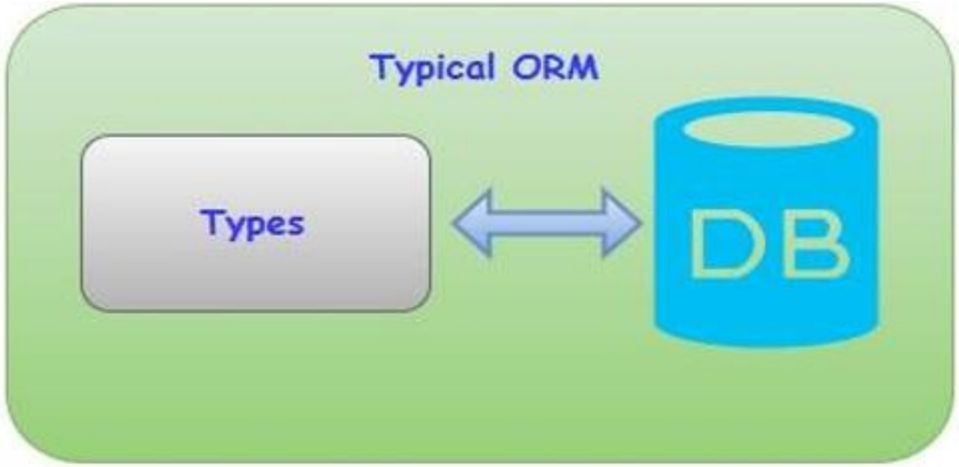
# Continuare

```
//Set the provider name.
entityBuilder.Provider = providerName;
// Set the provider-specific connection string.
entityBuilder.ProviderConnectionString =
    providerString;
// Set the Metadata location. New!!!
entityBuilder.Metadata =
    @"res://*/AdventureWorksModel.csdl|
    res://*/AdventureWorksModel.ssdl|
    res://*/AdventureWorksModel.msl";
```

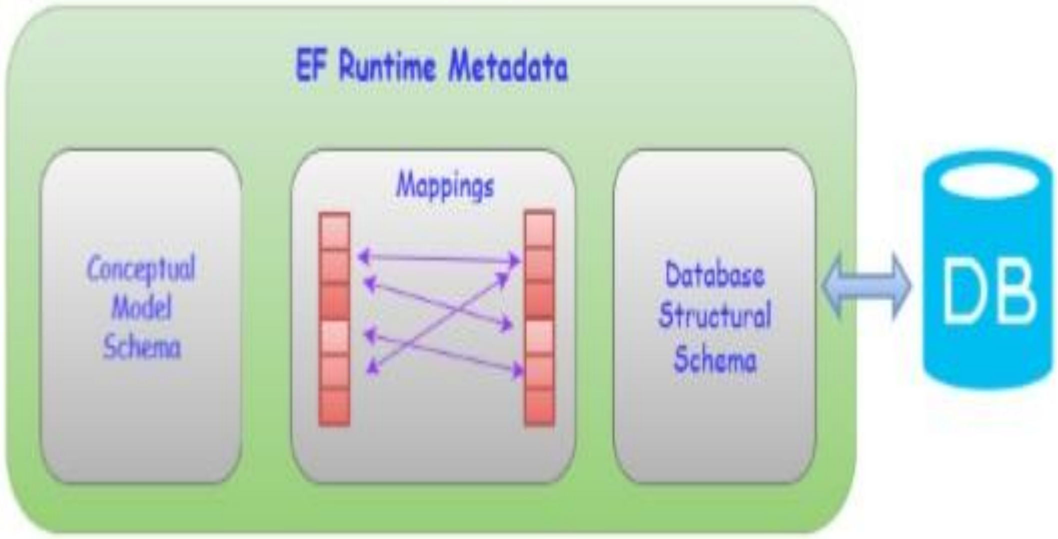
# Continuare

```
using (EntityConnection conn =
                    new EntityConnection(entityBuilder.ToString()))
{
    conn.Open();
    EntityCommand cmd = new EntityCommand(query, conn);
    using (EntityDataReader reader =    cmd.ExecuteReader())
{
     while (reader.Read())
     {
         // ...
      }
     conn.Close();
 }
```

# De ce Entity Framework?



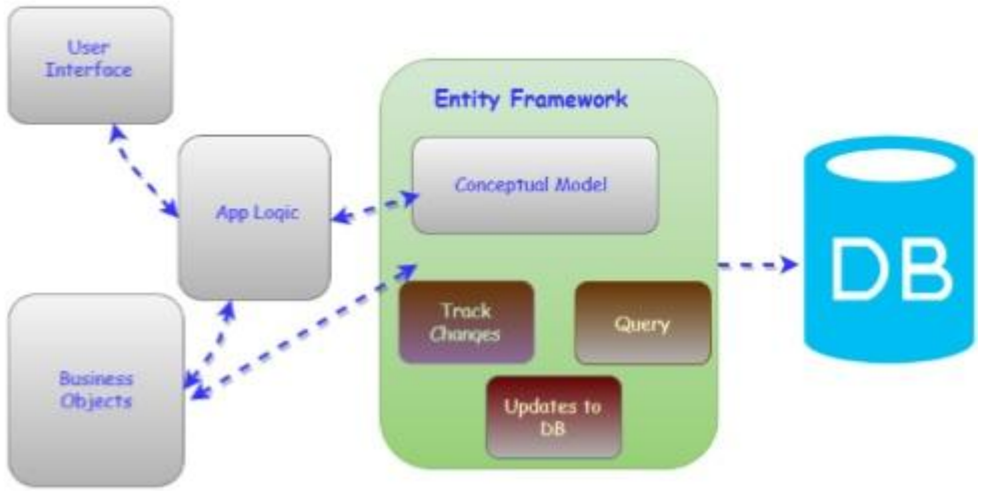
# De ce Entity Framework?



## EF - modele

- Model Conceptual.
- Model memorare.
- Model mapare.

# EF – model conceptual



## Continuare

- Cererile se executa direct asupra *obiectelor* din modelul conceptual.
- Se foloseste o singura metoda – **SaveChanges()** - pentru a salva datele in baza de date.

# Trasaturi

- Entity Framework este un tool Microsoft.
- EF exista in doua versiuni: V6 si EF Core
- EF Core este dezvoltat ca un produs Open Source.
- EF lucreaza cu bd ce au un furnizor de EF.
- EF genereaza comenzi SQL din LINQ to Entities.
- EF creaza cereri parametrizate.
- EF gestioneaza evidenta modificarilor obiectelor din memorie (obiecte din model).
- EF permite generarea comenzilor insert, update si delete.
- EF are suport pentru proceduri catalogate.

# Entity Framework - Architecture

- EF – Data Provider.
- EF – Entity Client.
- EF - Object Service.

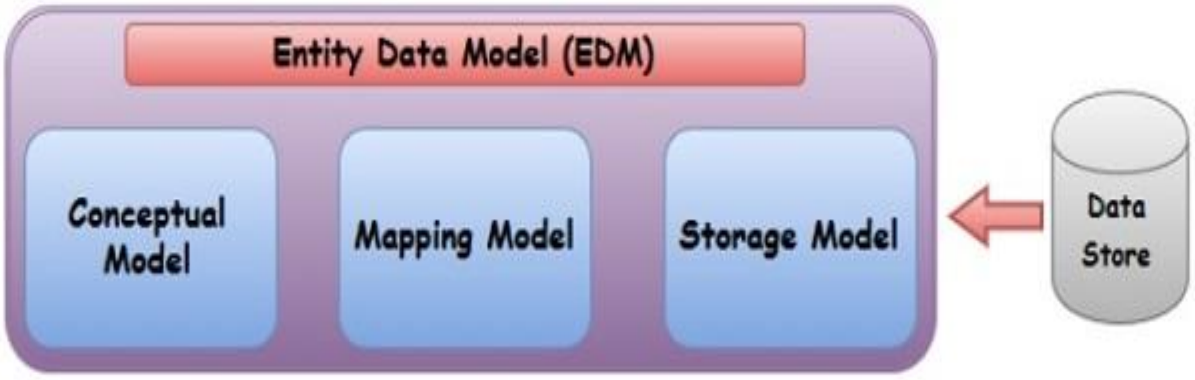
Aceste niveluri au fost evidentiate pe un slide anterior.

## Continuare - Data Provider

- Data Provider
  - Specific fiecarui furnizor de baze de date, ce abstractizeaza interfetele ADO.NET pentru conectare la baza de date cand lucram cu schema conceptuala.
  - Translateaza LINQ to entities la expresii SQL valide si le executa asupra bazei de date.

# Continuare – Entity Client

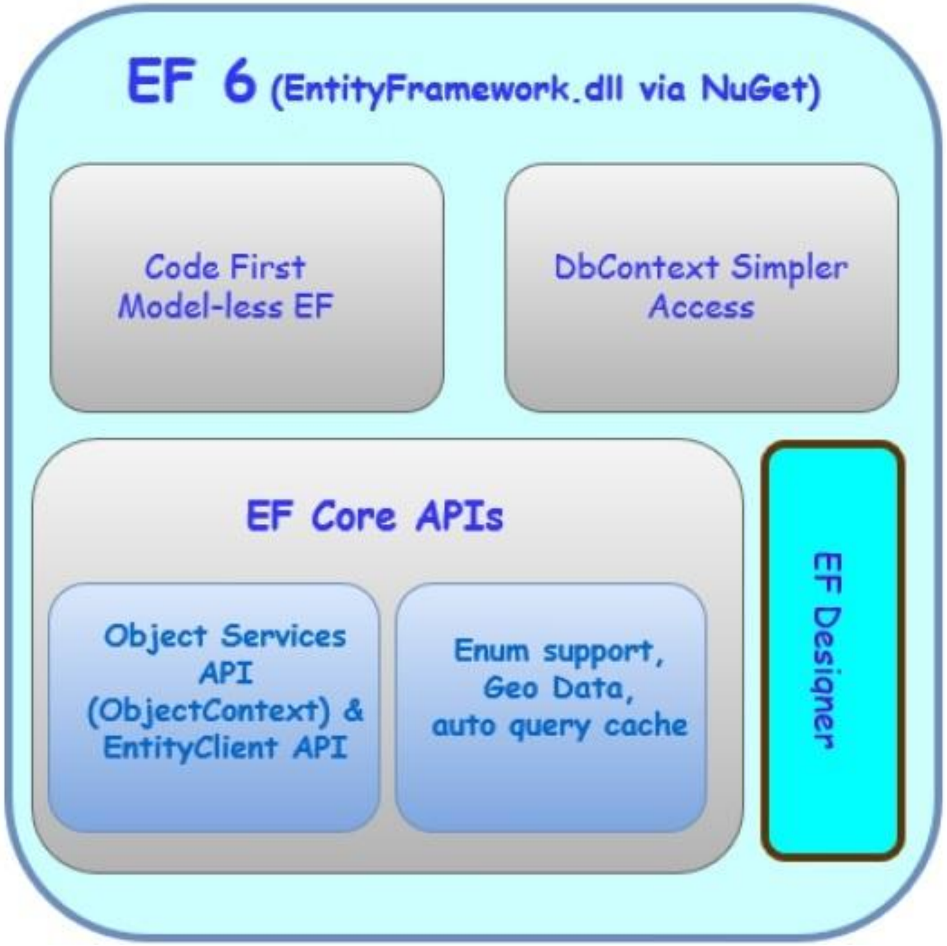
- Entity Client
  - Model Conceptual
  - Model Mapare
  - Model Memorare



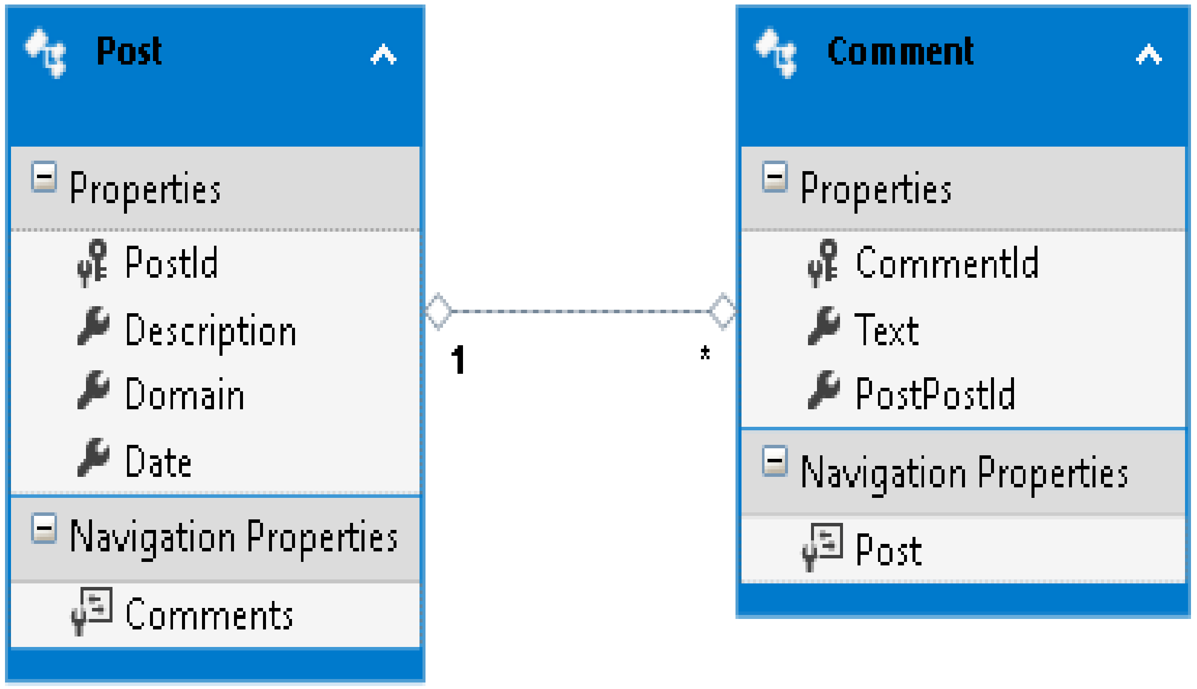
# Continuare – Object Service

- Object Service
  - Object Context, reprezinta sesiunea interactiunii intre aplicatii si sursa de date.
  - Object Context  - executa diverse operatii asupra entitatilor (add, delete, update, save).
  - Gestioneaza starea modificarilor  entitatilor (Track Changes).
  - Gestioneaza rezultatele cererilor.

# EF 6


EF 6 (EntityFramework.dll via NuGet)
Code First Model-less EF
DbContext Simpler Access
EF Core APIs
Object Services API (ObjectContext) & EntityClient API
Enum support, Geo Data, auto query cache
EF Designer

# Exemplu


Post
Properties
PostId
Description
Domain
Date
Navigation Properties
Comments
1
*
Comment
Properties
CommentId
Text
PostPostId
Navigation Properties
Post

## Modelul

```
public partial class Post
{
  public Post()   {  this.Comments = new HashSet<Comment>();  }
        [DataMember]
        public int PostId { get; set; }
        [DataMember]
        public string Description { get; set; }
        [DataMember]
        public string Domain { get; set; }
        [DataMember]
        public System.DateTime Date { get; set; }
        [DataMember]
        public virtual ICollection<Comment> Comments { get; set; }
   }
```

# Continuare

```
public partial class Comment
    {
        [DataMember]
        public int CommentId { get; set; }
        [DataMember]
        public string Text { get; set; }
        [DataMember]
        public int PostPostId { get; set; }

        [DataMember]
        public virtual Post Post { get; set; }
    }
```

# Continuare - Contextul

```
public partial class ModelPostCommentContainer  : DbContext
   {
      public ModelPostCommentContainer()  :
                        base("name=ModelPostCommentContainer")
      {  Configuration.LazyLoadingEnabled = false;
         Configuration.ProxyCreationEnabled = false;
      }
      protected override void OnModelCreating(
                                    DbModelBuilder modelBuilder)
      {  throw new UnintentionalCodeFirstException();   }

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

# CSDL generat

- VS 2019 deschide solutie din

D:\Documente\Cursuri\Curs special NET\WCF\Code\PostComment

- Se vor vizualiza csdl, msl, ssdl

# Continuare

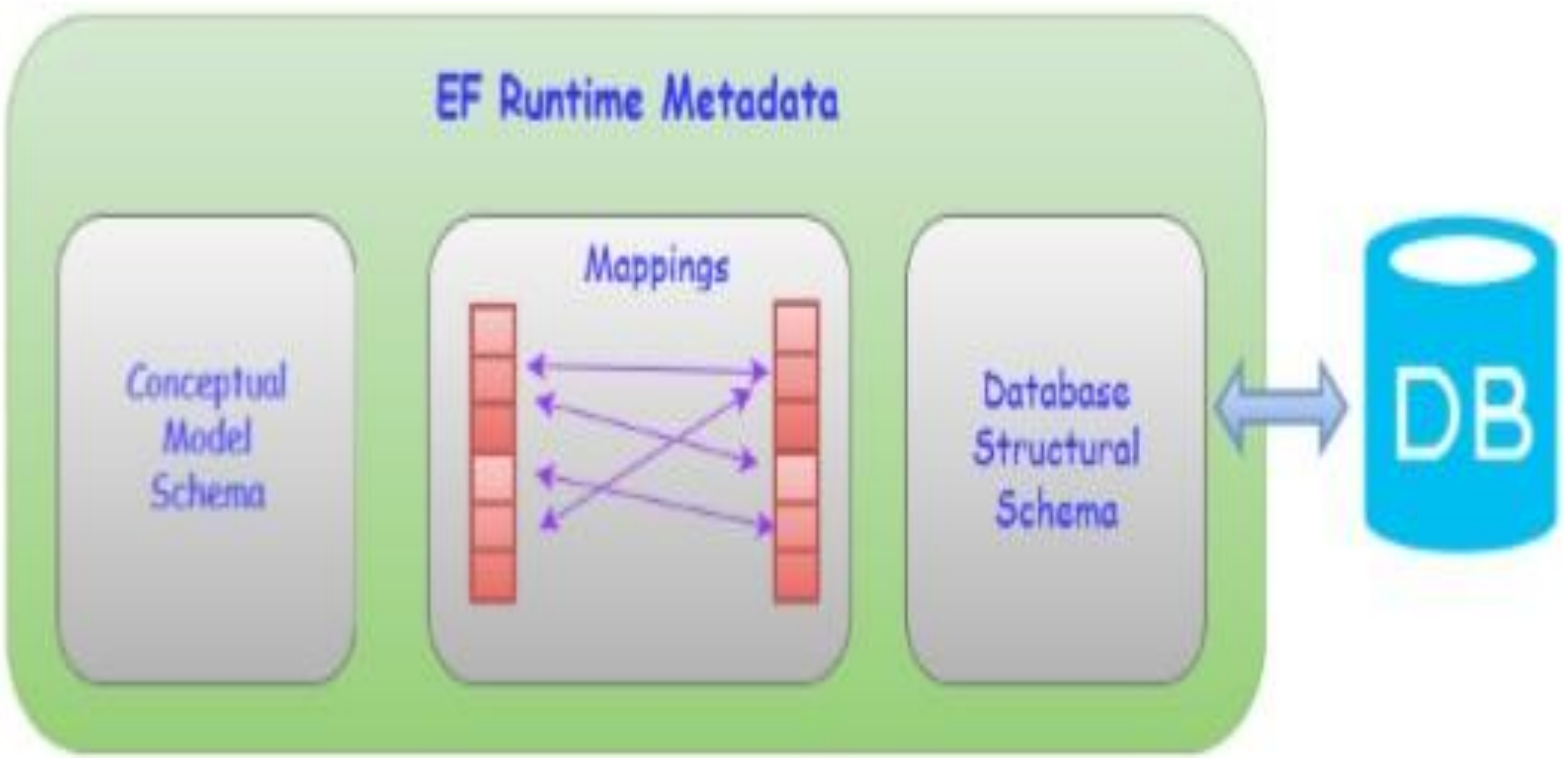
- Structura tabele

```
CREATE TABLE [dbo].[Posts] (
    [PostId]        INT  IDENTITY (1, 1) NOT NULL,
    [Description] NVARCHAR (MAX) NOT NULL,
    [Domain]       NVARCHAR (MAX) NOT NULL,
    [Date]            DATETIME       NOT NULL,
    CONSTRAINT [PK_Posts] PRIMARY KEY
    CLUSTERED ([PostId] ASC)
);
```

## Continuare

```
CREATE TABLE [dbo].[Comments] (
    [CommentId]  INT   IDENTITY (1, 1) NOT NULL,
    [Text]            NVARCHAR (MAX) NOT NULL,
    [PostPostId]     INT            NOT NULL,
    CONSTRAINT [PK_Comments] PRIMARY KEY
    CLUSTERED ([CommentId] ASC),
    CONSTRAINT [FK_CommentPost] FOREIGN KEY
    ([PostPostId]) REFERENCES [dbo].[Posts] ([PostId])
);
```

# Rezumat


EF Runtime Metadata
Conceptual Model Schema
Mappings
Database Structural Schema
DB

## Exemple cod

- In solutia deschisa se vor exemplifica cateva metode.

# ObjectContext

- ***Context object*** reprezinta accesul la serviciile din EF.

- Expune obiecte entitate.
- Gestioneaza conexiunile la baza de date.
- Genereaza SQL parametrizat.
- Transfera date in / din baza de date.
- Realizeaza cache la obiecte.
- Jurnalizeaza modificarile efectuate.
- Materializeaza sau transforma o multime rezultat fara tip intr-o colectie de obiecte puternic tipizate.

# ObjectContext - metode

- AcceptAllChanges
- AddObject
- CreateDatabase
- DeleteDatabase
- DeleteObject
- DetectChanges
- ExecuteStoreCommand
- SaveChanges(SaveOptions)
- Attach

## ObjectSet - Metode

- AddObject
- ApplyCurrentValues
- Attach
- DeleteObject
- Detach
- Execute
- Include

# DbContext

- Conceptual aceasta clasa este similara cu **ObjectContext**. Se foloseste pentru a executa cereri asupra EDM si a efectua actualizarea bazei de date (insert, update, delete, cereri). Aceasta clasa este folosita in versiunile EF mai mari sau egale cu 4.1.

```
public class ProductContext : DbContext
{   // cod lipsa ...
        public DbSet<Post> Posts { get; set; }
        public DbSet<Comment> Comments { get; set; }
}
```

# DbContext - Continuare

- Accesarea unei proprietati **DbSet** din cadrul contextului reprezinta definitia unei cereri ce returneaza (toate) entitatile de tipul specificat. Faptul ca accesam o proprietate nu inseamna ca cererea se executa. O cerere este executata cand:
- Este enumerata in *foreach*.
- Este enumerata de o operatiune din colectie cum ar fi **ToArray**, **ToDictionary** sau **ToList**.
- Sunt specificati in cadrul cererii operatorii LINQ **First** sau **Any** .
- Sunt apelate urmatoarele metode: metoda extinsa **Load** pe un DbSet, **DbEntityEntry.Reload** si **Database.ExecuteSqlCommand**.

## Proprietati pentru DbContext

- **ChangeTracker**
- **Configuration**
- **Database**

## DbContext - Metode

- **Entry(Object)**; **Entry<TEntity> (TEntity)**
- Gets a DbEntityEntry object for the given entity providing access to information about the entity and the ability to perform actions on the entity.
- **OnModelCreating()**
- This method is called when the model for a derived context has been initialized, but before the model has been locked down and used to initialize the context.
- **SaveChanges()**; **SaveChangesAsync()**; **SaveChangesAsync( CancellationToken)**
- **Set(Type)**; **Set<TEntity>()**
- Returns a DbSet instance for access to entities of the given type in the context, the ObjectStateManager, and the underlying store.

## Proprietatea Database

- Prin proprietatea **Database** obtinem un obiect **Database**. Acest obiect expune metode ce pot executa comenzi SQL (DDL/DML) la nivel de baza de date.
- Clasa **Database** expune proprietatea **Connection** ce permite recrearea unui obiect **DbConnection** daca acesta nu exista.

# DbSet, DbSet<Tentity>

- **Add** ; **AddRange**
- **Attach**
- **Find**
- **Include** - Specifies the related objects to include in the query results. (Inherited from DbQuery<TResult>.)
- **Remove** ; **RemoveRange**
- **SqlQuery**
- Creates a raw SQL query that will return entities in this set. By default, the entities returned are tracked by the context; this can be changed by calling ***AsNoTracking*** on the DbSqlQuery<TEntity> returned. Note that the entities returned are always of the type for this set and never of a derived type. If the table or tables queried may contain data for other entity types, then the SQL query must be written appropriately to ensure that only entities of the correct type are returned.

# Cereri  pe Entity Data Model - EDM

- EF foloseste informatiile din model si fisierele de mapare pentru a translata cererile asupra obiectelor din modelul conceptual in cereri specifice sursei de date.
- EF furnizeaza urmatoarele modalitati de a interoga modelul conceptual si a returna obiecte:

o **LINQ to Entities**. Furnizeaza suport pentru Language-Integrated Query (LINQ) in vederea interogarii tipurilor de entitati definite in modelul conceptual.

o **Entity SQL**. Un dialect SQL independent de mediul de memorare ce lucreaza direct cu entitatile in modelul conceptual.

o Metode de constructie a cererilor folosind metodele din **LINQ to Objects**.

o **SQL** nativ.

## Incarcare date relationate

- La ce se refera?
- Proprietati de navigare:
  - Tip referinta (Reference).
  - Tip colectie (Collection).

- Lazy loading
- Eager loading
- Explicit loading

## Lazy loading

- *Implicit*. Cerintele pe care trebuie sa le indeplineasca clasele POCO definite sunt:
- Sa fie **public** dar **nu sealed**.
- Proprietatile de navigare - **virtual**.
- **DbContext.Configuration.LazyLoadingEnabled**. Implicit este setata pe *true*.
- Proxy dinamic creat intern.

# Eager loading

- Include :
- var query = context.Posts.Include(“Comments").Take(5);
- var query = from item in context.Posts.Include(ct=>ct.Comments)

  where ct.Text == “gRPC”

  select ct;

## Explicit loading

- **Entry**()- stabilire intrare pentru entitatea data.
- **Collection();**
- **Reference();**
- **Property(); CurrentValue;**
- **Load();**

# Exemplu - Collection

```
//Disable Lazy loading
  ctx.Configuration.LazyLoadingEnabled = false;

  var itemsPost= from post in ctx.Posts
        where post.PostId  == 1
        select post;
        var single = itemsPost.Single();
        ctx.Entry(single)
              .Collection(d =>  d.Comments)
              .Load();
```

# Exemplu - Reference

```
//Disable Lazy loading
   ctx.Configuration.LazyLoadingEnabled = false;

   var itemsCom= from com in ctx.Comments
         where post.CommentId  == 10
         select com;
         var single = itemsCom.Single();
         ctx.Entry(single)
               .Reference(d =>  d.Posts)
               .Load();
```

# Property – doua prototipuri

```
using (var context = new BloggingContext())
{
   var blog = context.Blogs.Find(3);
   // Read the current value of the Name property
   string currentName1 = context.Entry(blog)
                                         .Property(u => u.Name).CurrentValue;
   // Set the Name property to a new value
   context.Entry(name).Property(u => u.Name).CurrentValue = "My Fancy Blog";

   // Read the current value of the Name property
   //using a string for the property name
   object currentName2 = context.Entry(blog).Property("Name").CurrentValue;
   // Set the Name property to a new value using
   // a string for the property name
     context.Entry(blog).Property("Name").CurrentValue = "My Boring Blog";
}
```

## Explicit loading - continuare

- **IsModified** – proprietate scalara.
- **IsLoaded** – continut proprietate de navigare incarcat sau nu.

# Entity Framework si aplicatii N-tier

- Jurnalizare:
  - ***Snapshot** change tracking (fara notificare context).*
    - *DetectChanges() din ChangeTracker.*
  - *Change tracking **proxies** (cu notificare context).*

    Clasele sa fie *publice* si sa *nu* fie *sealed*.

    Proprietatile sa fie *virtuale*. Getter-i si setter-i sa fie publici.

    Proprietatile de navigare de tip colectie sa fie de tipul **ICollection<T>.**

# Exemplu

```
public partial class CustomerType
{
    public CustomerType()
    {
        this.Customers = new HashSet<Customer>();
    }
    // Proprietati scalare
    public virtual int CustomerTypeId { get; set; }
    public virtual string Description { get; set; }
    // Entitate colectie
    public virtual ICollection<Customer> Customers { get; set; }
}
```

# Detectie modifificari

- Metode implicate:
  - **DbSet.Add**
  - **DbSet.Find**
  - **DbSet.Remove**
  - **DbSet.Local**
  - **DbContext.SaveChanges**
  - **Rulare cerere LINQ asupra unui DbSet**
  - **DbSet.Attach**
  - **DbContext.GetValidationErrors**
  - **DbContext.Entry**
  - **DbChangeTracker.Entries**

- Detectarea modificarilor poate fi setata pe ***on*** sau ***off*** astfel:
  **context.Configuration.AutoDetectChangesEnabled = false; // true = on**

# DetectChanges() si …

- Metoda **DetectChanges**() este responsabila si pentru a sincroniza relatiile existente intre entitati, *entity reference* sau *entity collection*. Acest lucru este vizibil atunci cand folosim *data binding* in aplicatii GUI si facem modificari asupra proprietatilor de navigare. Modificarile efectuate trebuie sa se reflecte in interfata GUI a aplicatiei.
- AsNoTracking() ???

# DbContext - State

Proprietatea **State**.

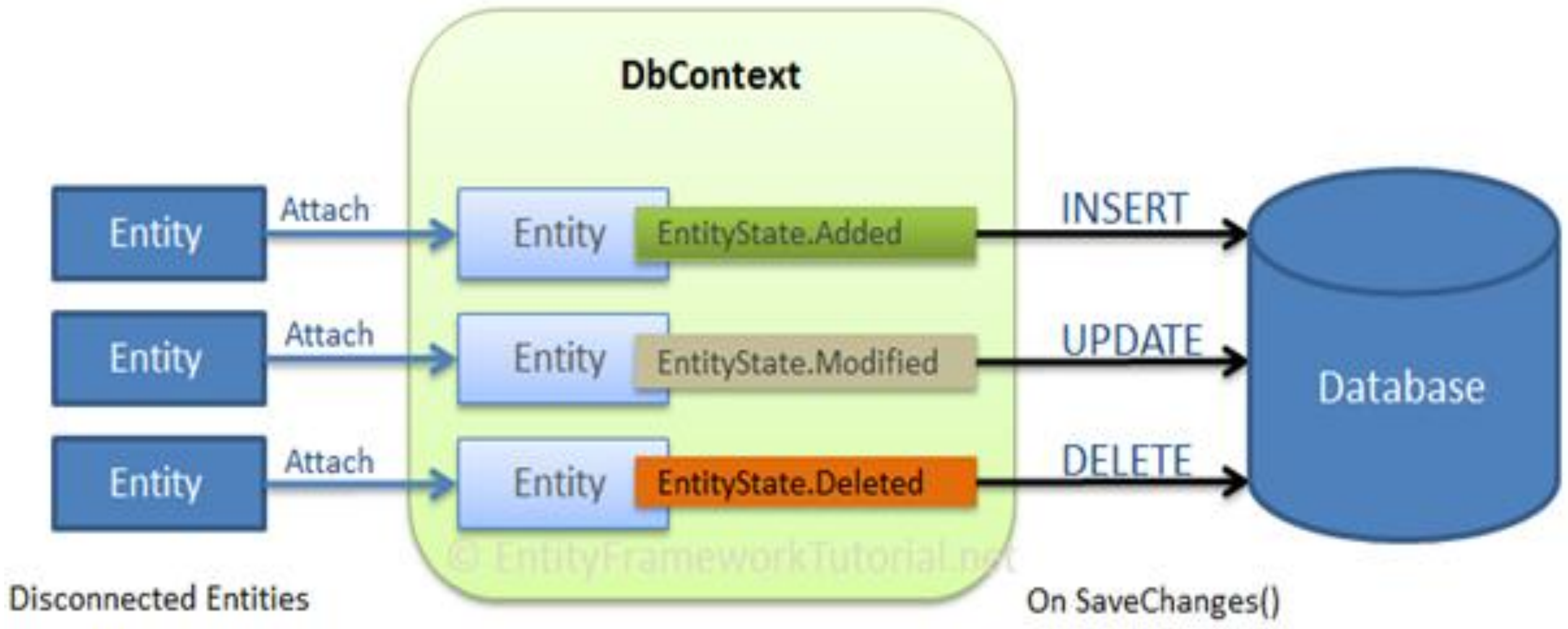
Enumerarea **EntityState**

```
{
Unchanged,
Added,
Modified,
Deleted,
Detached
}
```

# EntityState

```
context.Entry(item).State = EntityState.Added;
context.SaveChanges();

context.Entry(item).State = EntityState.Unchanged;
context.SaveChanges();

context.Entry(item).State = EntityState.Modified;
context.SaveChanges();

context.Entry(item).State = EntityState.Deleted;
context.SaveChanges();
```

# Scenariu cu entitati deconectate



# EF Core

- Configurare – OnConfiguring()

  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

- Creare model – OnModelCreating()

  protected override void OnModelCreating(ModelBuilder modelBuilder)

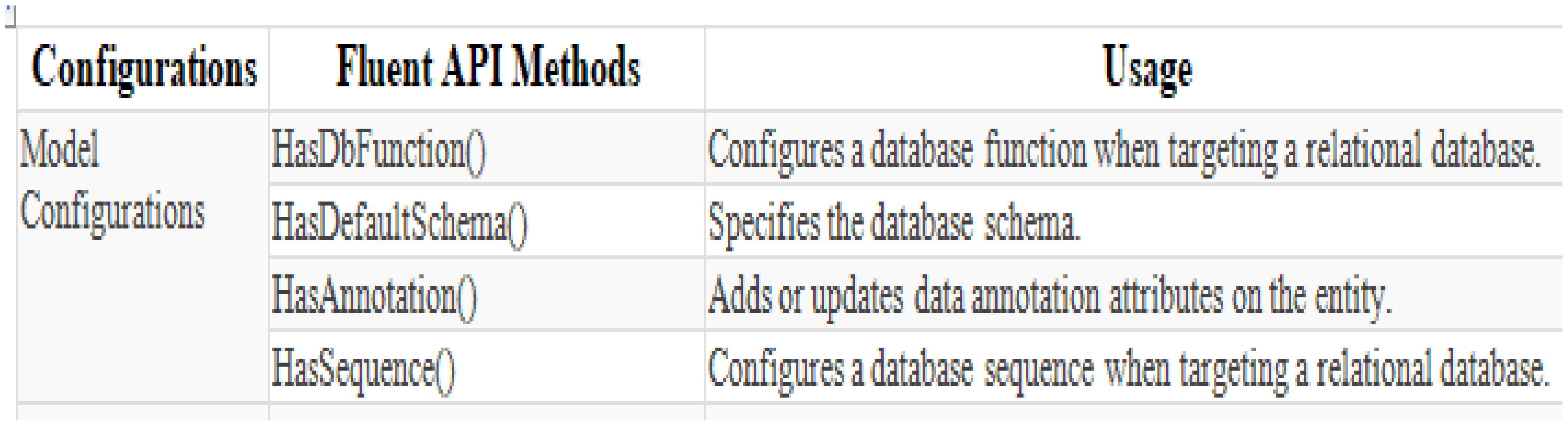
- Salvare date – SaveChanges()

# EF Core Fluent API

Entity Framework Core Fluent API configures the following aspects of a model:

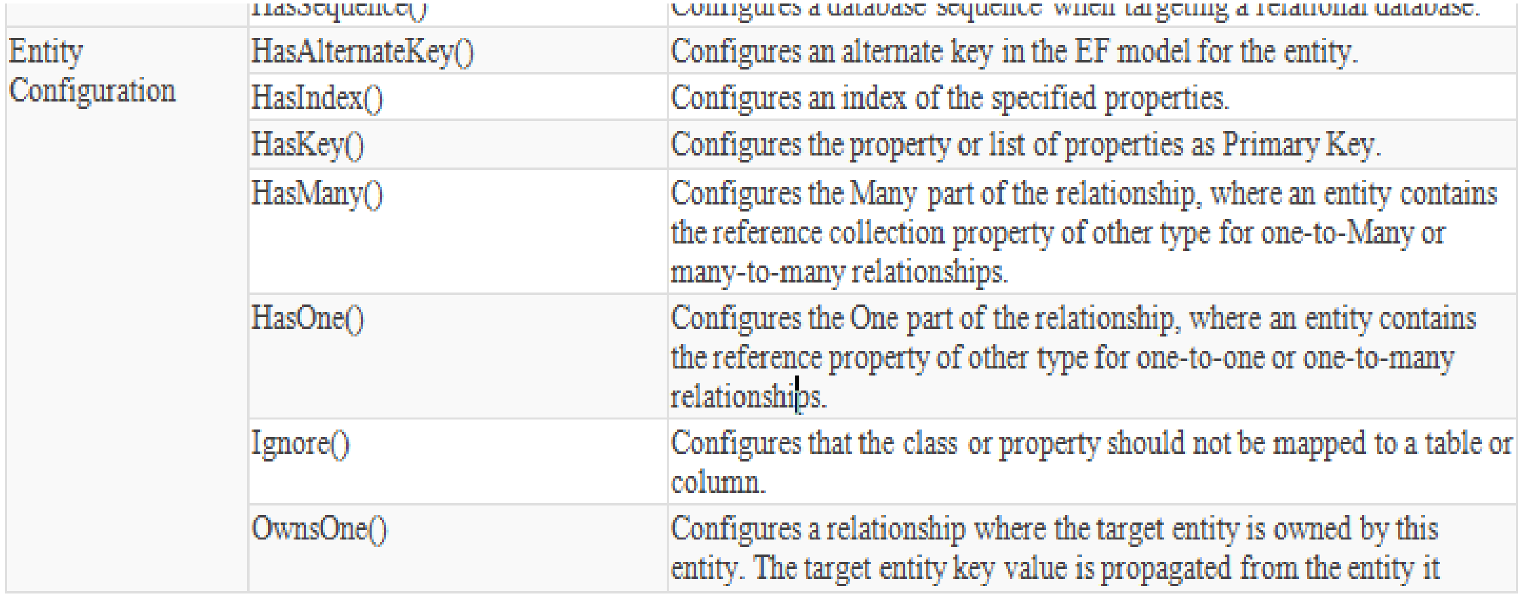
- **Model Configuration**: Configures an EF model to database mappings. Configures the default Schema, DB functions, additional data annotation attributes and entities to be excluded from mapping.
- **Entity Configuration**: Configures entity to table and relationships mapping e.g. PrimaryKey, AlternateKey, Index, table name, one-to-one, one-to-many, many-to-many relationships etc.
- **Property Configuration**: Configures property to column mapping e.g. column name, default value, nullability, Foreignkey, data type, concurrency column etc.

# Model configurations



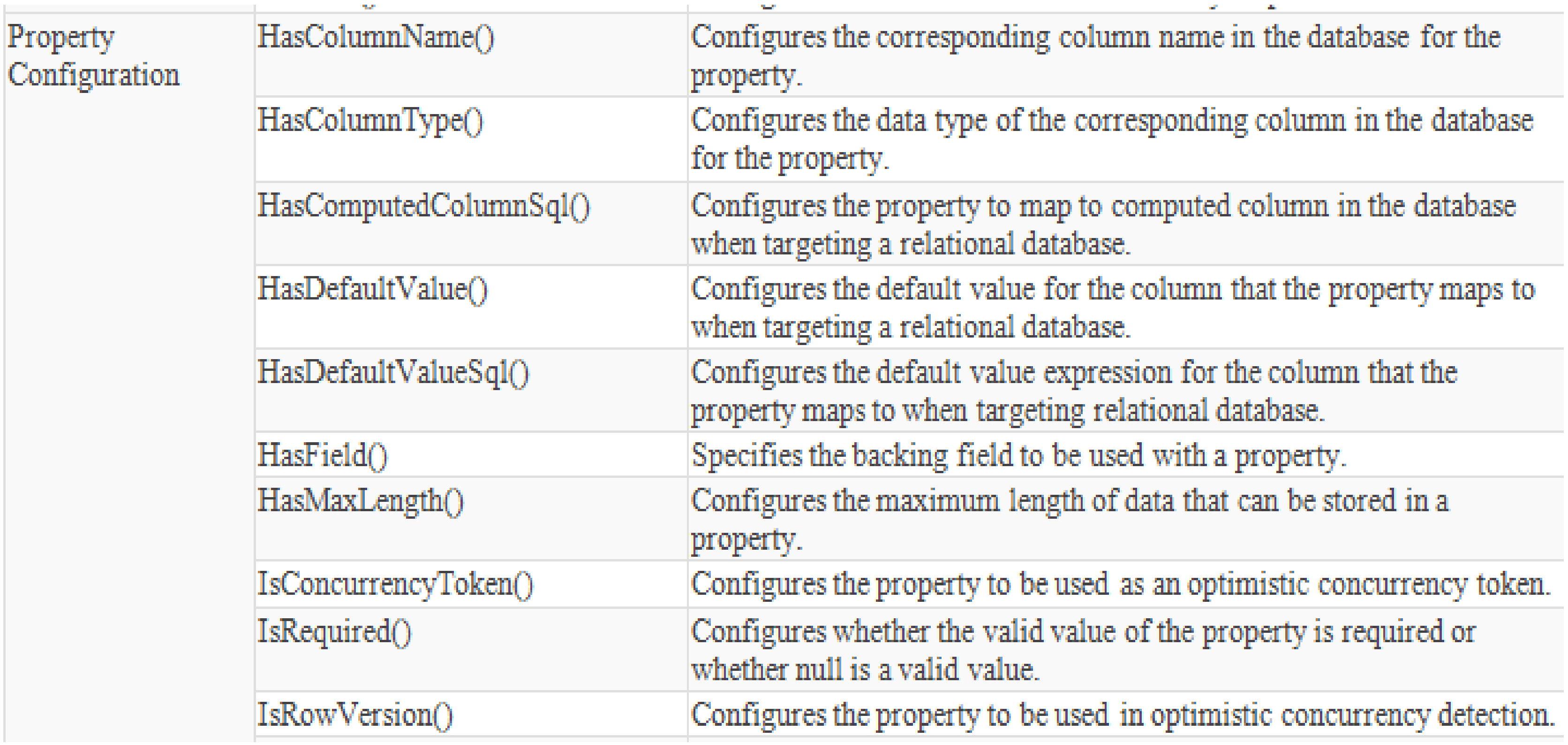
| Configurations | Fluent API Methods | Usage |
|---|---|---|
| Model Configurations | HasDbFunction() | Configures a database function when targeting a relational database. |
| | HasDefaultSchema() | Specifies the database schema. |
| | HasAnnotation() | Adds or updates data annotation attributes on the entity. |
| | HasSequence() | Configures a database sequence when targeting a relational database. |

# Entity configuration



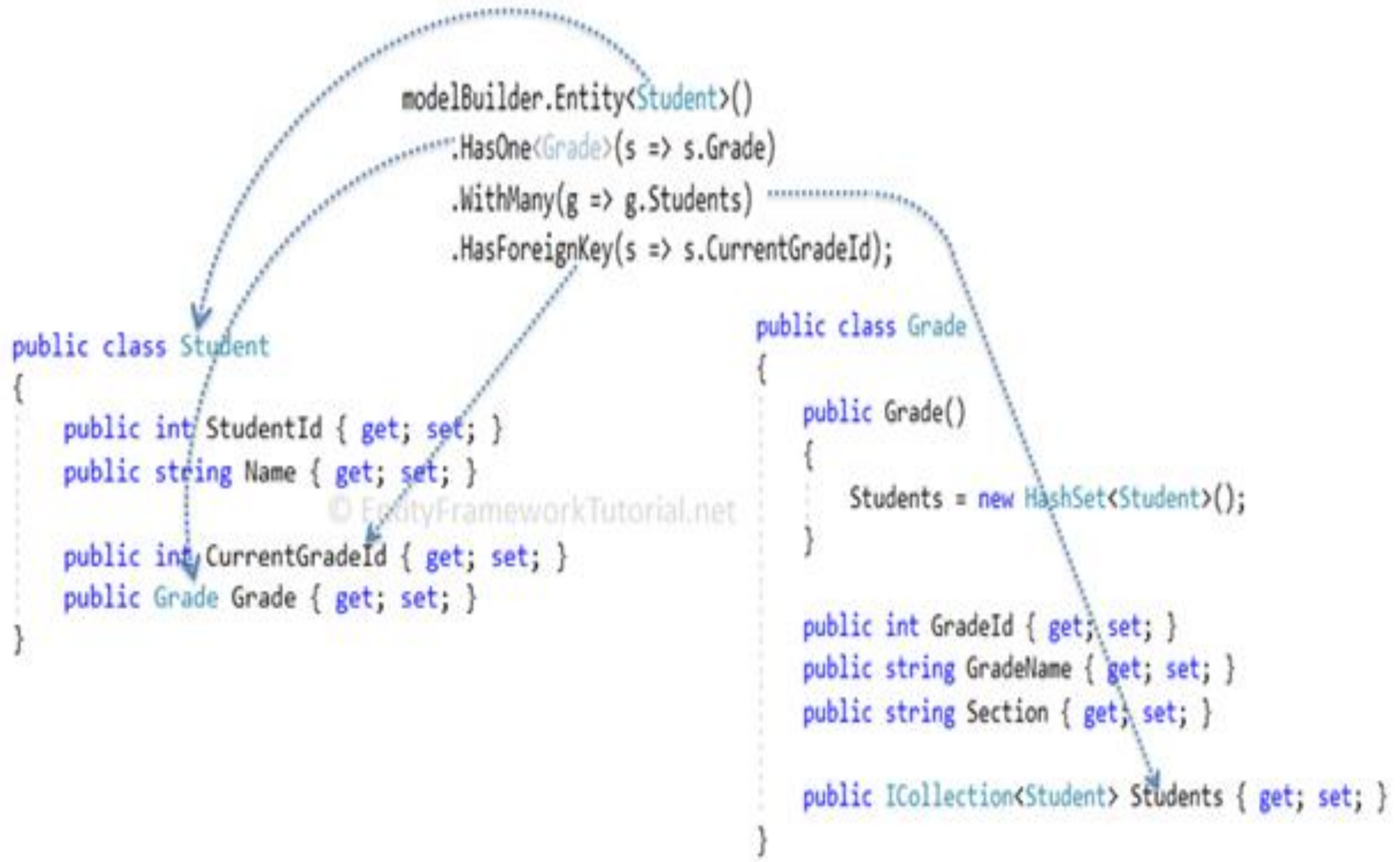
| | | |
|---|---|---|
| | HasSequence() | Configures a database sequence when targeting a relational database. |
| Entity Configuration | HasAlternateKey() | Configures an alternate key in the EF model for the entity. |
| | HasIndex() | Configures an index of the specified properties. |
| | HasKey() | Configures the property or list of properties as Primary Key. |
| | HasMany() | Configures the Many part of the relationship, where an entity contains the reference collection property of other type for one-to-Many or many-to-many relationships. |
| | HasOne() | Configures the One part of the relationship, where an entity contains the reference property of other type for one-to-one or one-to-many relationships. |
| | Ignore() | Configures that the class or property should not be mapped to a table or column. |
| | OwnsOne() | Configures a relationship where the target entity is owned by this entity. The target entity key value is propagated from the entity it |

# Property configuration



| | | |
|---|---|---|
| Property Configuration | HasColumnName() | Configures the corresponding column name in the database for the property. |
| | HasColumnType() | Configures the data type of the corresponding column in the database for the property. |
| | HasComputedColumnSql() | Configures the property to map to computed column in the database when targeting a relational database. |
| | HasDefaultValue() | Configures the default value for the column that the property maps to when targeting a relational database. |
| | HasDefaultValueSql() | Configures the default value expression for the column that the property maps to when targeting relational database. |
| | HasField() | Specifies the backing field to be used with a property. |
| | HasMaxLength() | Configures the maximum length of data that can be stored in a property. |
| | IsConcurrencyToken() | Configures the property to be used as an optimistic concurrency token. |
| | IsRequired() | Configures whether the valid value of the property is required or whether null is a valid value. |
| | IsRowVersion() | Configures the property to be used in optimistic concurrency detection. |

# Exemplu (1 -> *)

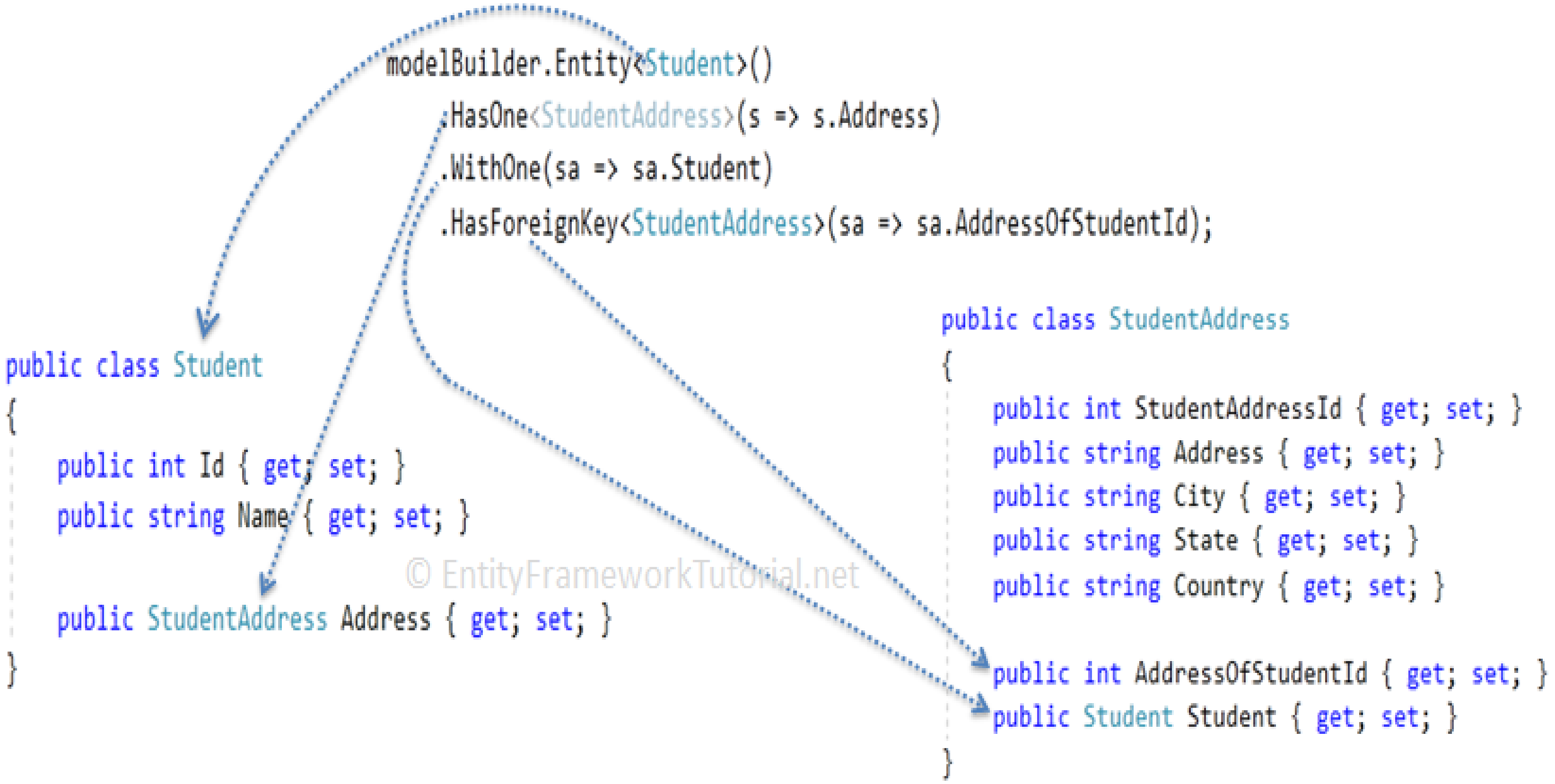


```
modelBuilder.Entity<Student>()
    .HasOne<Grade>(s => s.Grade)
    .WithMany(g => g.Students)
    .HasForeignKey(s => s.CurrentGradeId);
```

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }

    public int CurrentGradeId { get; set; }
    public Grade Grade { get; set; }
}
```

```
public class Grade
{
    public Grade()
    {
        Students = new HashSet<Student>();
    }

    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

# Exemplu (1 -> 1)

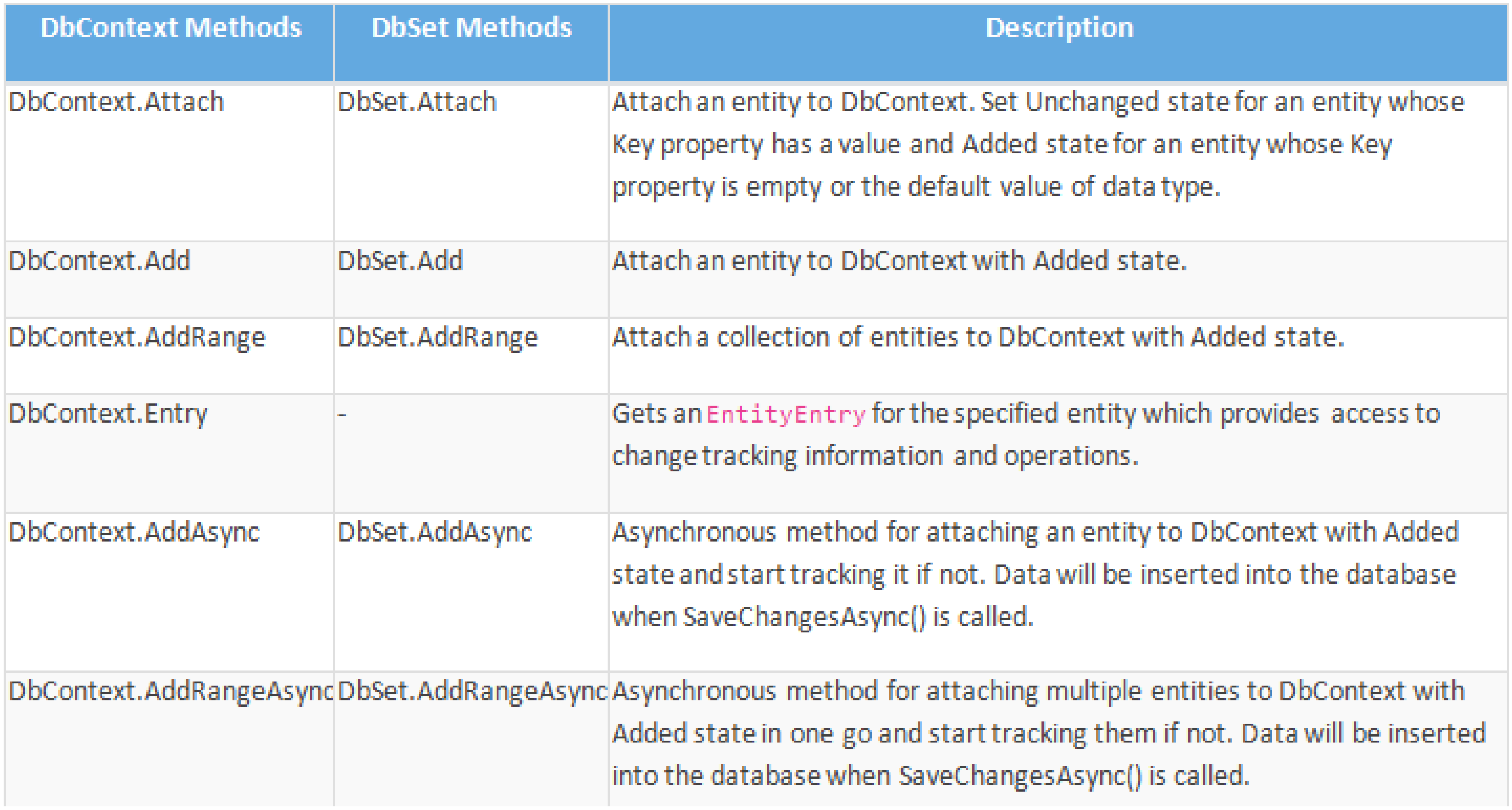


```
modelBuilder.Entity<Student>()
    .HasOne<StudentAddress>(s => s.Address)
    .WithOne(sa => sa.Student)
    .HasForeignKey<StudentAddress>(sa => sa.AddressOfStudentId);
```

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }

    public StudentAddress Address { get; set; }
}
```

```
public class StudentAddress
{
    public int StudentAddressId { get; set; }
    public string Address { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public string Country { get; set; }

    public int AddressOfStudentId { get; set; }
    public Student Student { get; set; }
}
```

# DbContext - DbSet

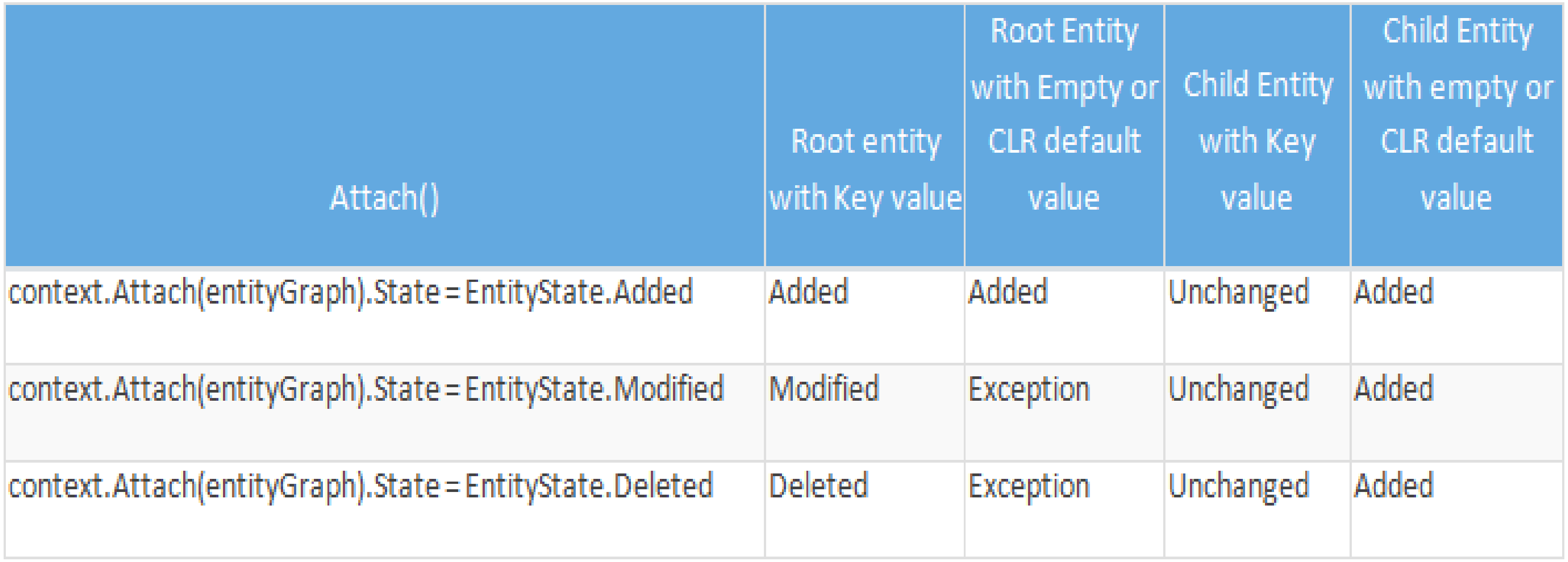


| DbContext Methods | DbSet Methods | Description |
|---|---|---|
| DbContext.Attach | DbSet.Attach | Attach an entity to DbContext. Set Unchanged state for an entity whose Key property has a value and Added state for an entity whose Key property is empty or the default value of data type. |
| DbContext.Add | DbSet.Add | Attach an entity to DbContext with Added state. |
| DbContext.AddRange | DbSet.AddRange | Attach a collection of entities to DbContext with Added state. |
| DbContext.Entry | - | Gets an `EntityEntry` for the specified entity which provides access to change tracking information and operations. |
| DbContext.AddAsync | DbSet.AddAsync | Asynchronous method for attaching an entity to DbContext with Added state and start tracking it if not. Data will be inserted into the database when SaveChangesAsync() is called. |
| DbContext.AddRangeAsync | DbSet.AddRangeAsync | Asynchronous method for attaching multiple entities to DbContext with Added state in one go and start tracking them if not. Data will be inserted into the database when SaveChangesAsync() is called. |

## Working with Disconnected Entity Graph in Entity Framework Core

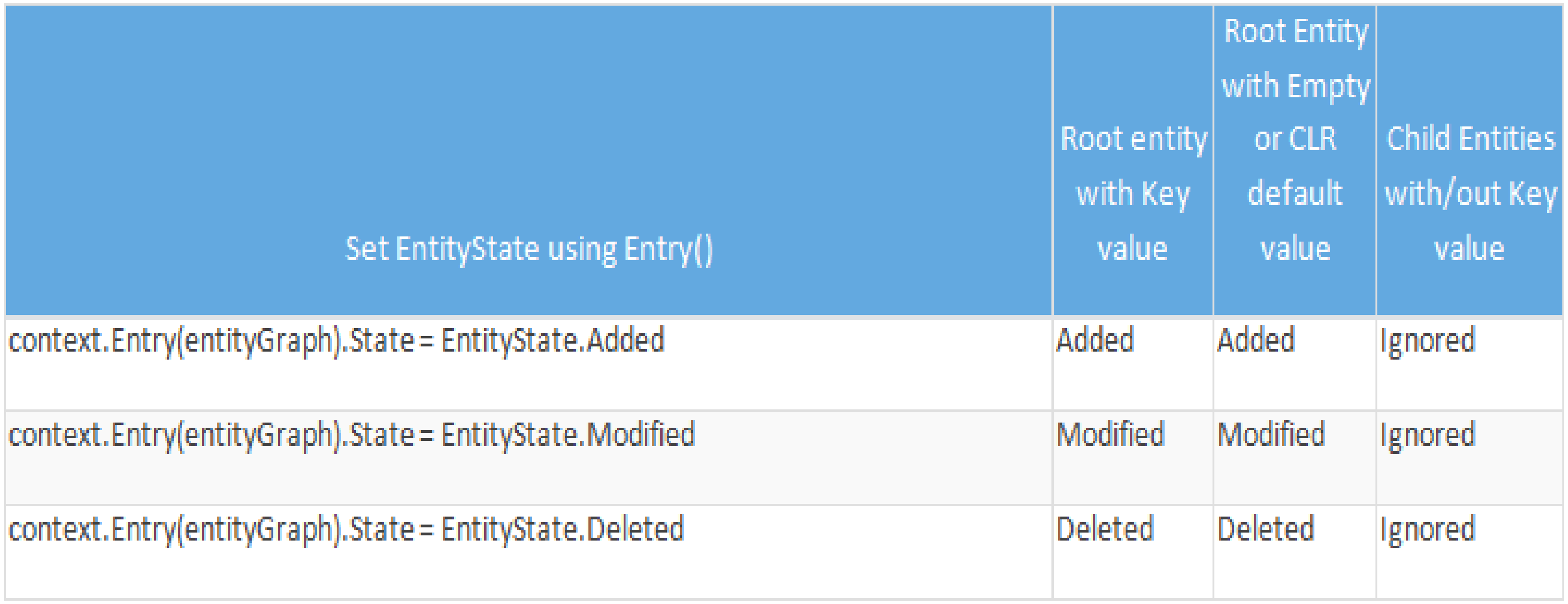
- Attach()
- Entry()
- Add()
- Update()
- Remove()

# Attach()



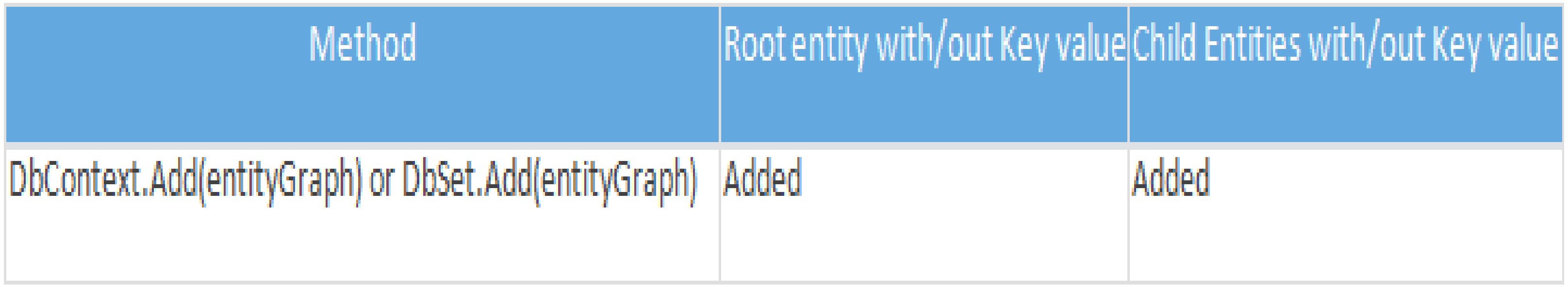
| Attach() | Root entity with Key value | Root Entity with Empty or CLR default value | Child Entity with Key value | Child Entity with empty or CLR default value |
|---|---|---|---|---|
| context.Attach(entityGraph).State = EntityState.Added | Added | Added | Unchanged | Added |
| context.Attach(entityGraph).State = EntityState.Modified | Modified | Exception | Unchanged | Added |
| context.Attach(entityGraph).State = EntityState.Deleted | Deleted | Exception | Unchanged | Added |

# Entry()



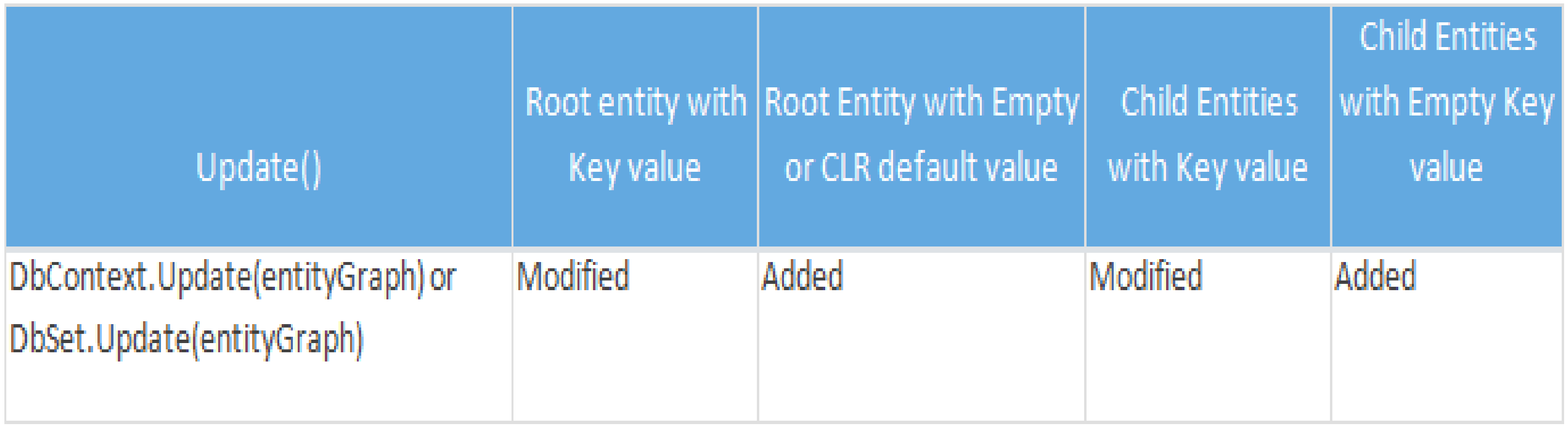
| Set EntityState using Entry() | Root entity with Key value | Root Entity with Empty or CLR default value | Child Entities with/out Key value |
|---|---|---|---|
| context.Entry(entityGraph).State = EntityState.Added | Added | Added | Ignored |
| context.Entry(entityGraph).State = EntityState.Modified | Modified | Modified | Ignored |
| context.Entry(entityGraph).State = EntityState.Deleted | Deleted | Deleted | Ignored |

# Add()



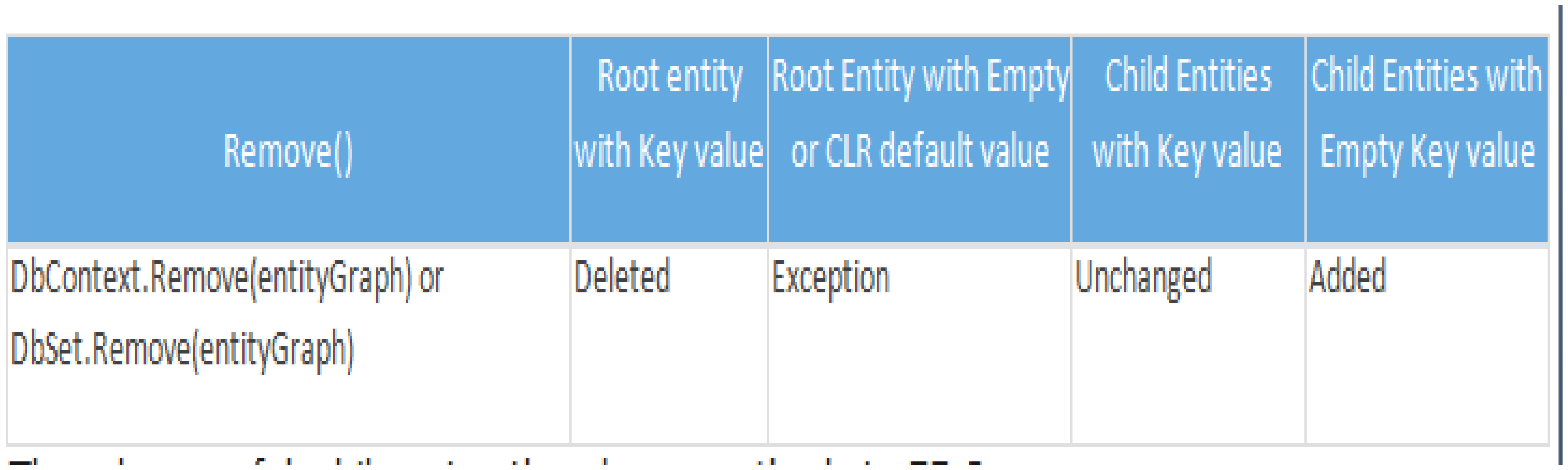
| Method | Root entity with/out Key value | Child Entities with/out Key value |
|---|---|---|
| DbContext.Add(entityGraph) or DbSet.Add(entityGraph) | Added | Added |

# Update()



| Update() | Root entity with Key value | Root Entity with Empty or CLR default value | Child Entities with Key value | Child Entities with Empty Key value |
|---|---|---|---|---|
| DbContext.Update(entityGraph) or DbSet.Update(entityGraph) | Modified | Added | Modified | Added |

# Remove()



| Remove() | Root entity with Key value | Root Entity with Empty or CLR default value | Child Entities with Key value | Child Entities with Empty Key value |
|---|---|---|---|---|
| DbContext.Remove(entityGraph) or DbSet.Remove(entityGraph) | Deleted | Exception | Unchanged | Added |

# ChangeTracker.TrackGraph() in Entity Framework Core

Purpose: To track the entire entity graph and set custom entity states to each entity in a graph.

The ChangeTracker.TrackGraph() method begins tracking an entity and any entities that are reachable by traversing it's navigation properties. The specified callback is called for each discovered entity and an appropriate EntityState must be set for each entity. The callback function allows us to implement a custom logic to set the appropriate state. If no state is set, the entity remains untracked.

## Execute Raw SQL Queries in Entity Framework Core

- Metode redenumite: FromSql, ExecuteSql, ExecuteSqlAsync in
  - FromSqlRaw,
  - ExecuteSqlRaw, si
  - ExecuteSqlRawAsync.

## Metode cu substitutie (interpolare)

- Use FromSqlInterpolated, ExecuteSqlInterpolated, and ExecuteSqlInterpolatedAsync to create a parameterized query where the parameters are passed as part of an interpolated query string.