

# ADO.NET Entity Framework

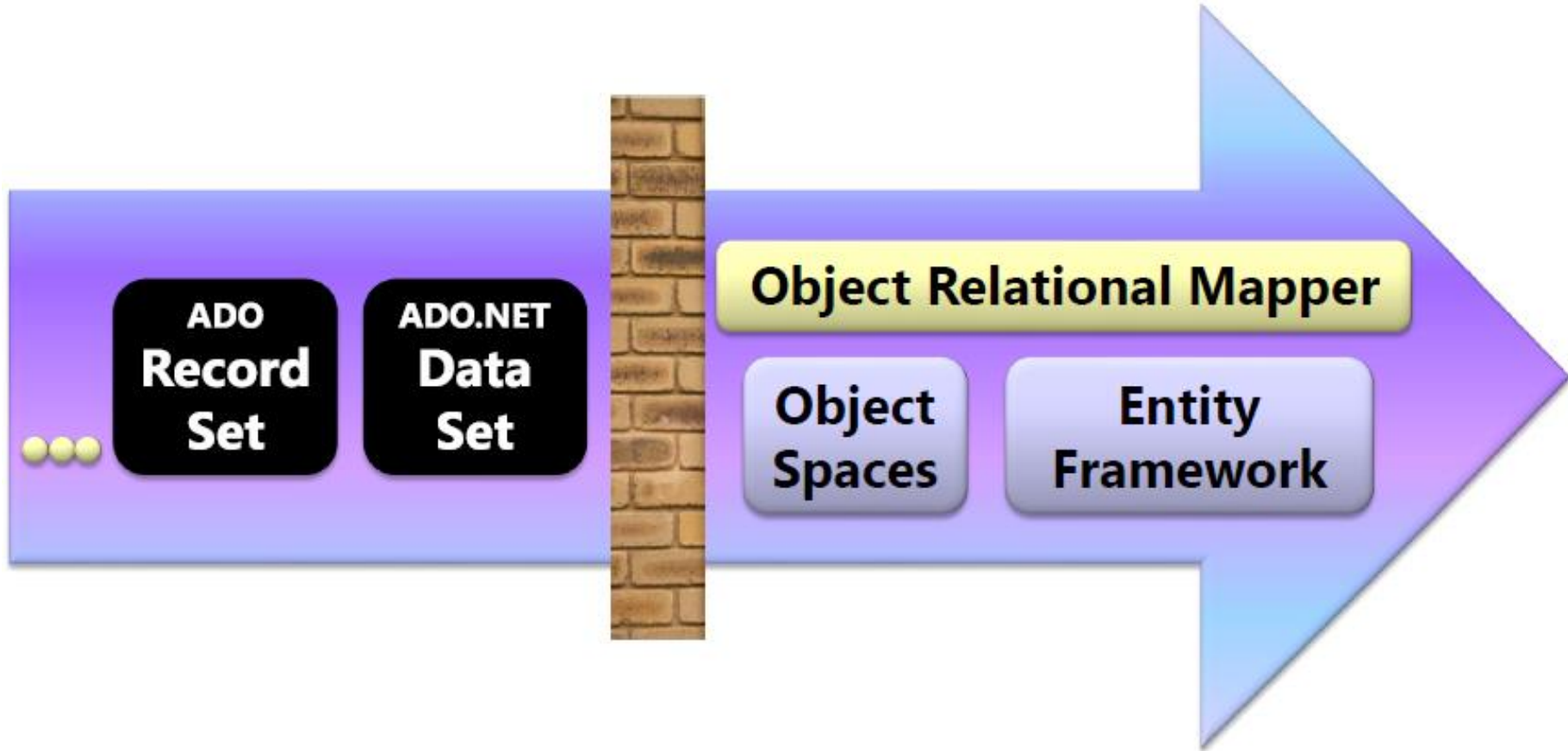
*Temel Kavramlar*

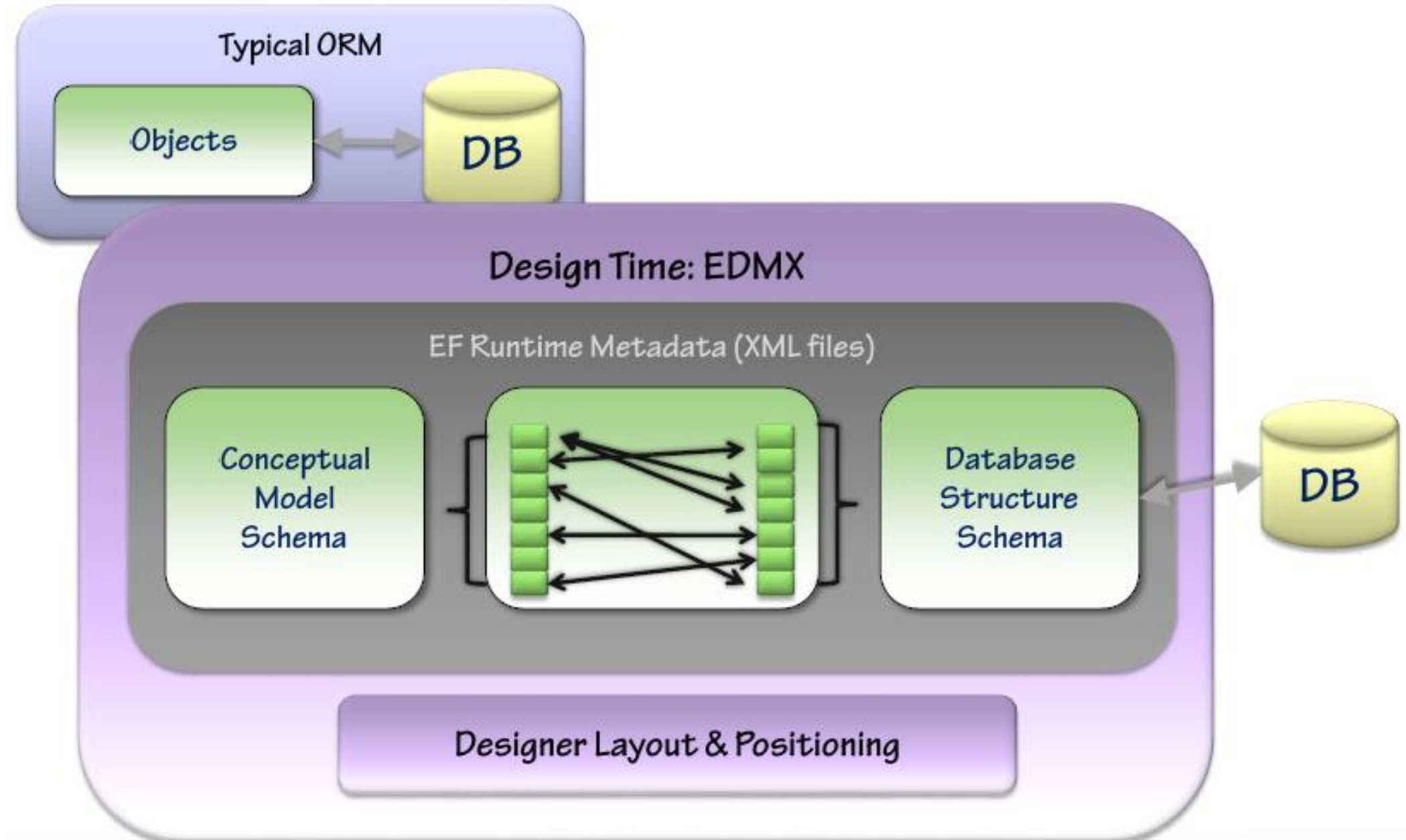
# Ele alınacak başlıklar

- Geçmişteki bilgiler
- Models, Mapping ve Metadata
- Entity SQL ve LINQ to Entities
- Object Services
- EF ile LINQ to SQL'in karşılaştırılması

Nesne	Veritabanı
OOP prensiplerine göre oluşturulurlar.	İlişkisel cebire göre oluşturulurlar.
Miras alma ön plandadır.	Normalizasyon gerektirir.
Linkler, referanslarla verilir.	Linkler, foreign key'lerle verilir.
Bellek adresleri ile tanımlanırlar.	Primary key'ler ile tanımlanırlar.
Çalışma zamanında tanımlanan veri türlerini kullanırlar.	Veritabanı tarafından tanımlanan veri türlerini kullanırlar.
Veriyi listeler ve ağaçlar şeklinde saklayabilirler.	Veriyi tuple'lar şeklinde saklayabilirler.
Şu an için transactional değildir.	Ağırlıklı olarak transactional'dır.

# EF' ü Neden Kullanmalıyım?





# Nerelerde Kullanılabilir?

## Client/n-Tier

- WinForms
- WPF

## Distributed

- ASP.NET EntityDataSource
- ASP.NET Dynamic Data
- ASP.NET MVC
- WCF & Self-Tracking Entities
- WCF RIA Services (for Silverlight)
- WCF Data Services (RESTful data)

## Cloud

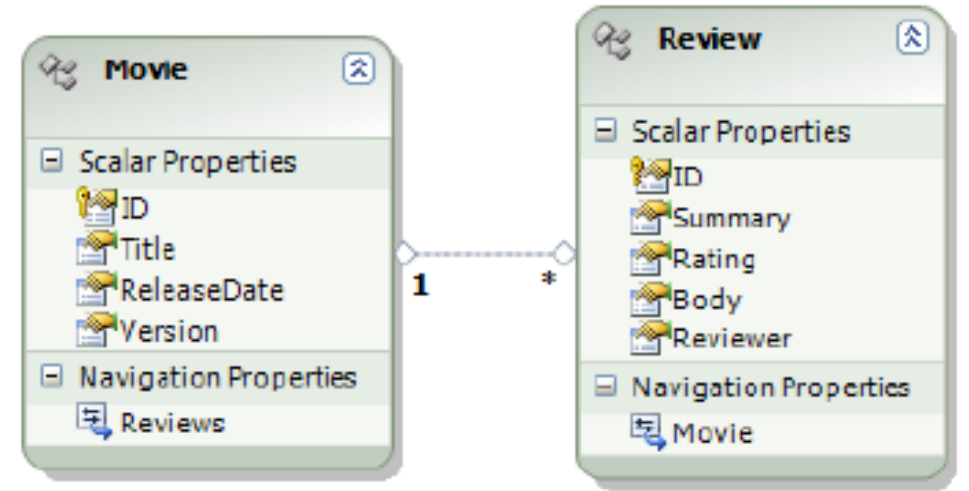
- SQL Azure
- AppFabric (Velocity/caching)

# Entity Framework

- Yeni ADO.NET olarak adlandırılabilir.
  - ADO.NET'den daha üst düzey bir soyutlama (abstraction) sağlar.
  - Yeni olarak Entity Data Model kavramı ön plana çıkar.
  - Geleneksel ORM araçlarının ötesinde DataSevisleri sağlayabilir.
- Özellikleri
  - LINQ Provider
  - Visual Studio Designer desteği bulunmaktadır.
  - Esnek mapping
  - Değişik data providerları ile çalışabilmektedir. (Oracle, DB2, vd.)

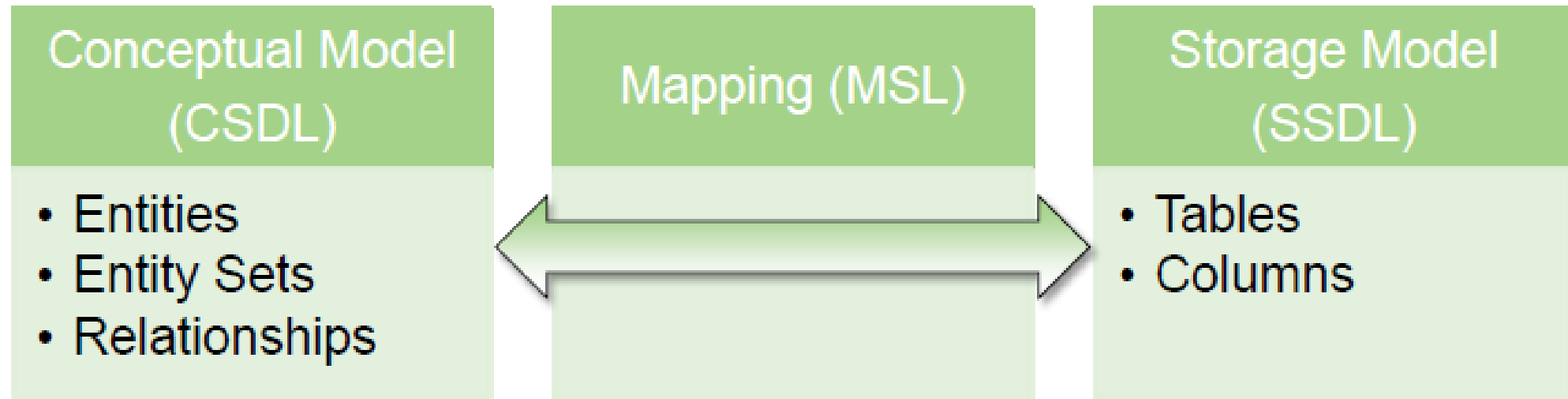
# Entity

- Varlığı devam eden her obje entity'dir.
- Bir Entity,
  - Bir veya daha fazla karmaşık özellikleri olabilir.
  - Diğer entityyle ile bir veya daha fazla bir ilişki içinde olabilir.
- "Entity key", entitylerin birbirinden eşsiz olarak ayrılmasını sağlar.



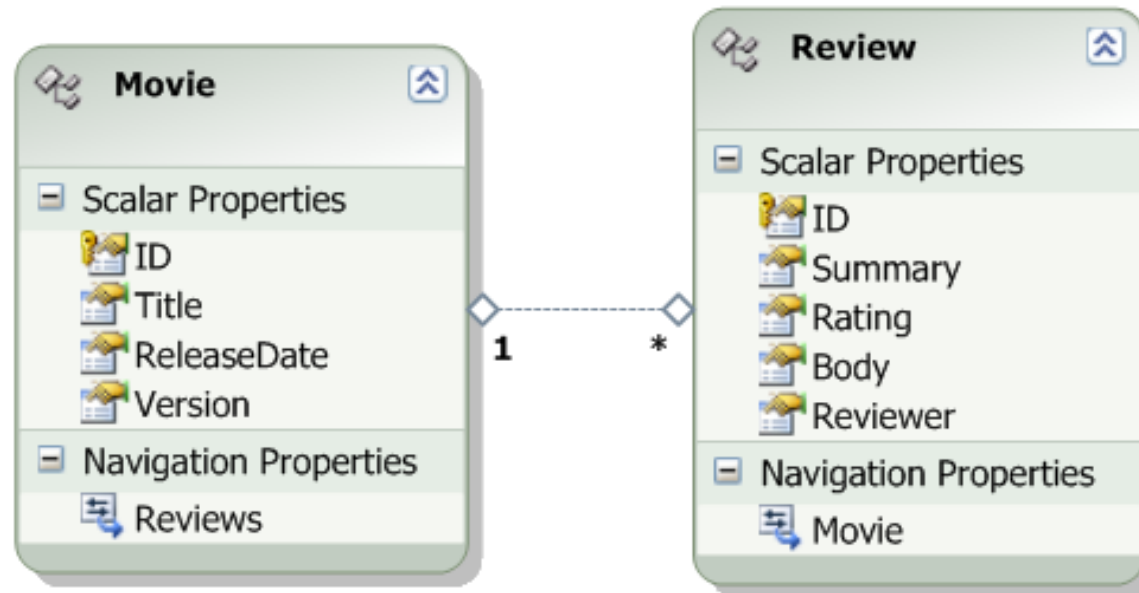


# Entity Data Model



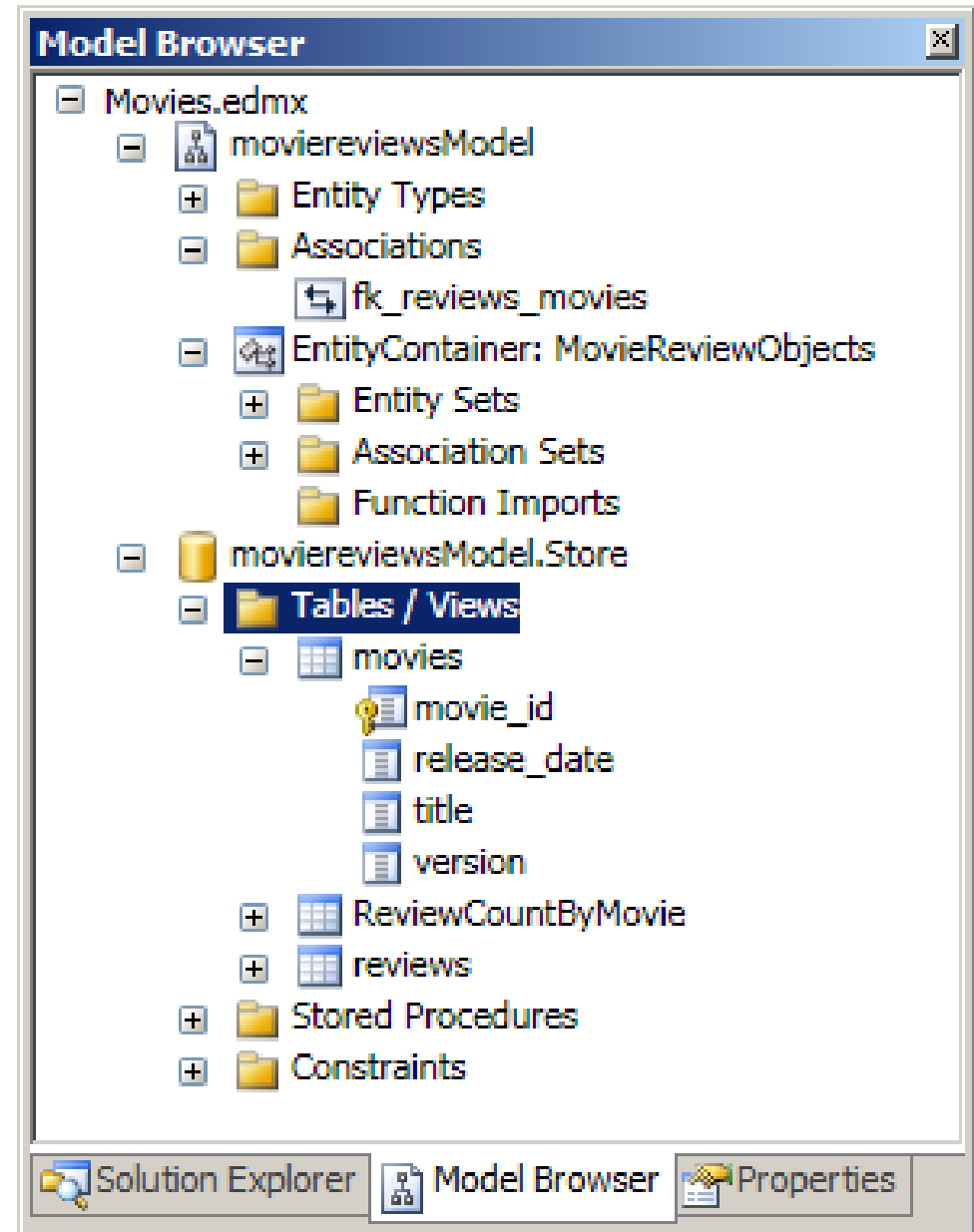
# Entity Designer

- Entity'leri ve relationshipsleri oluşturur.
- Key'leri ,tipleri , ve null olabilecek propertyleri tanımlar.



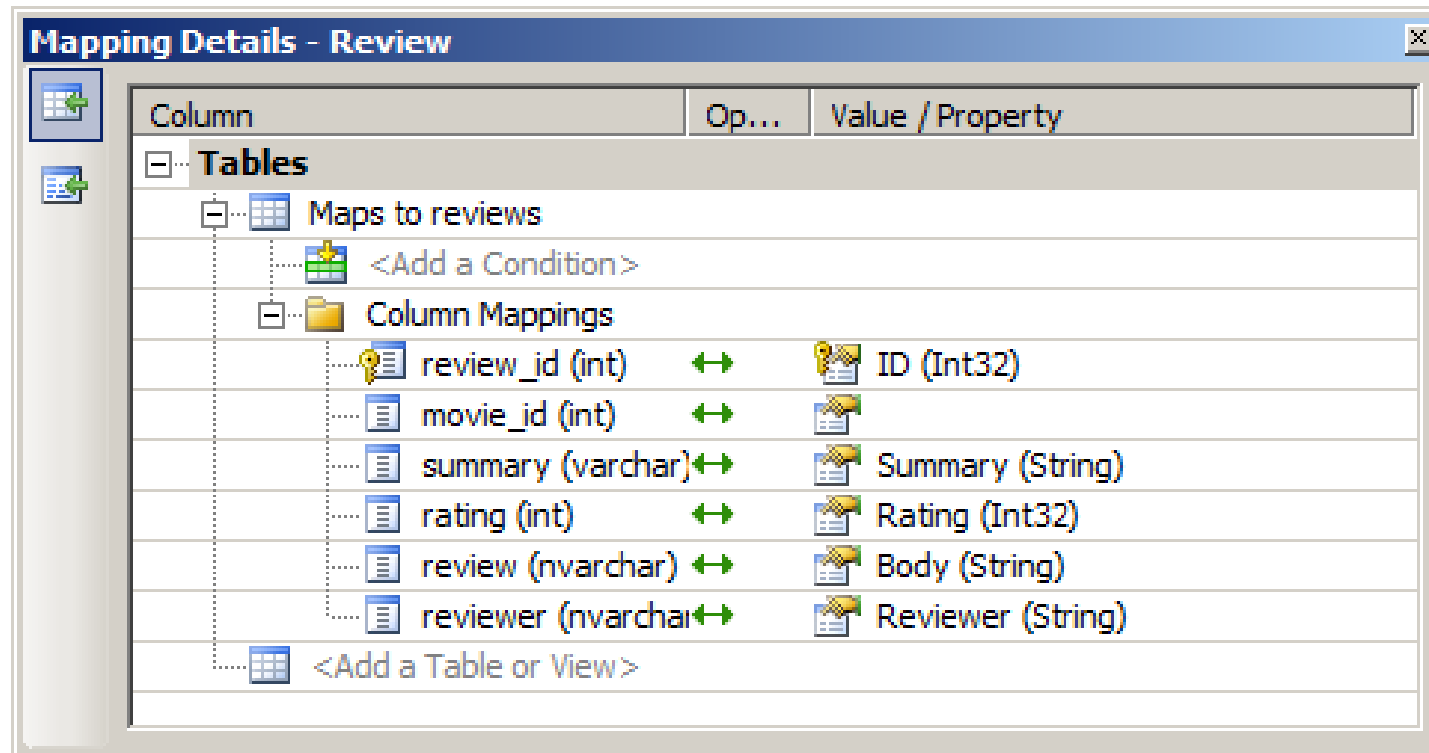
# Model Browser

- Çok fazla entity olduğunda GUI üzerinde navigasyon zor bir hal alabilir. Bu işlemi "model browser" kullanarak daha hızlı bir şekilde gerçekleştirebiliriz.
- Model geçerli kılınabilir. (Validation)
- Model, veritabanından update edilebilir.

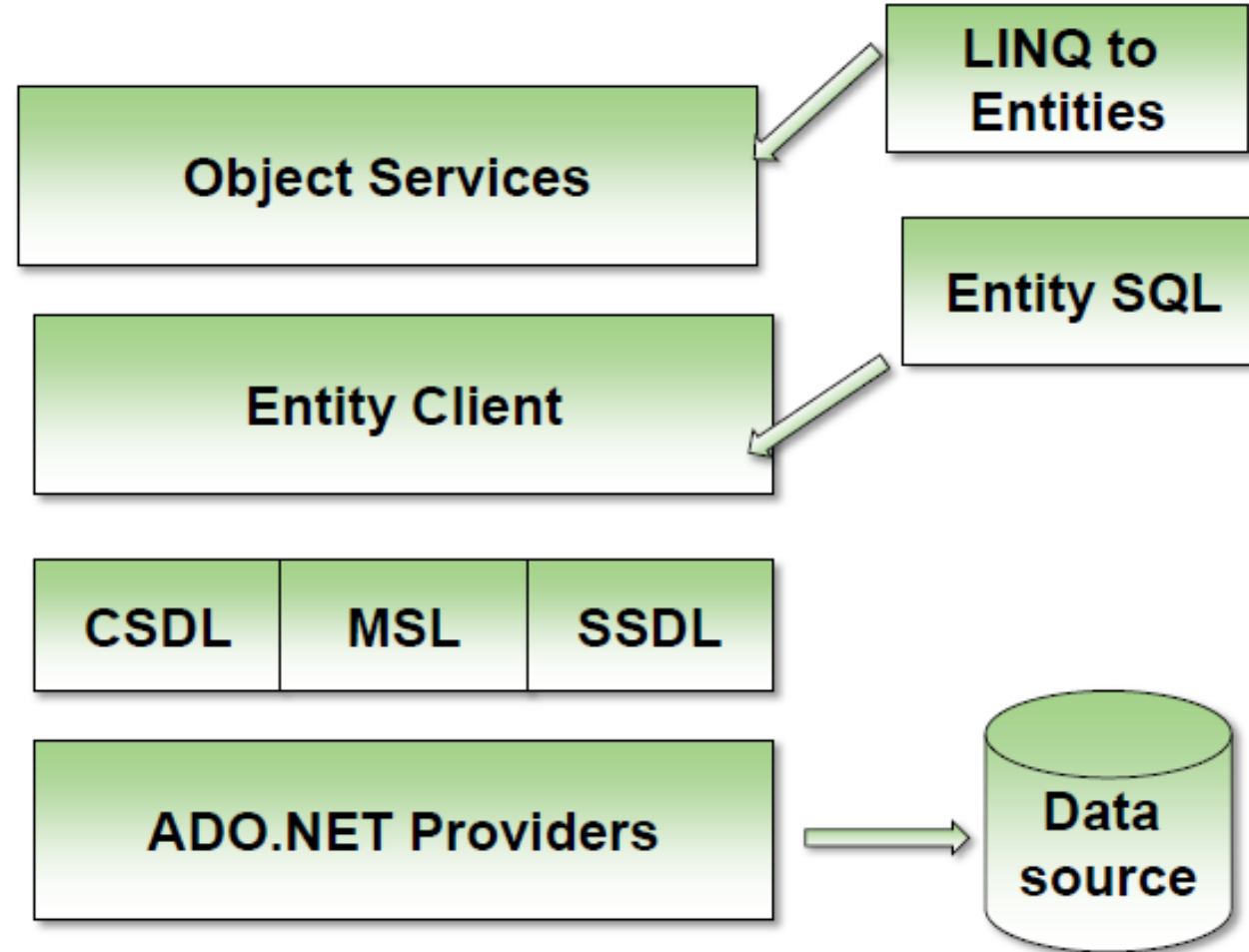


# Mapping Details

- Entity'ler bir veya daha fazla tablo arasında haritalanabilir.
- Detayları görmek ve editlemek için dizayn ekranında bir entity'nin üzerinde sağ tıklanarak ulaşılabilir.

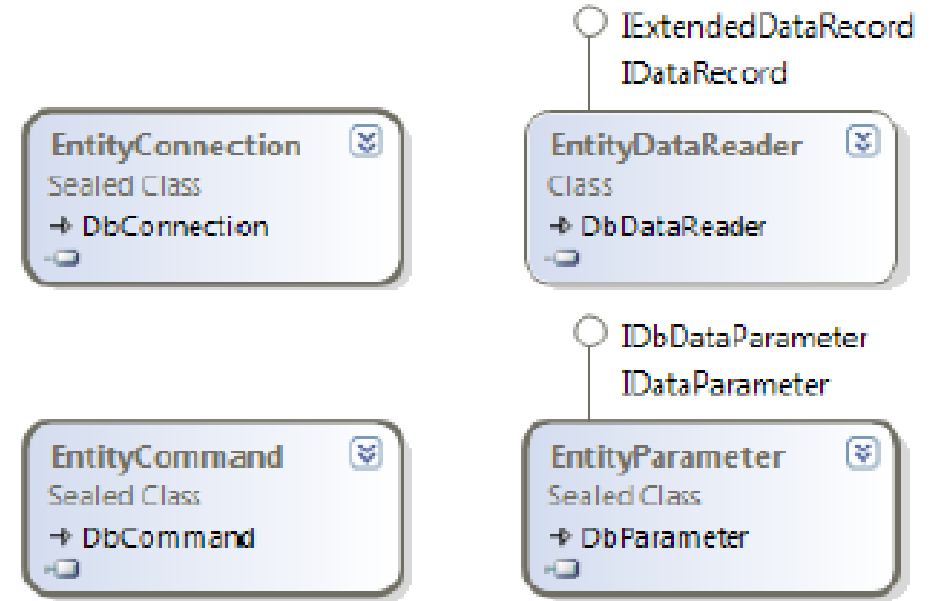


# Entity Framework Servisleri



# Entity Client

- Bu katmanda altta var olan veritabanından bağımsız olarak sorgular oluşturulur.
  - Sorgular istencilere eSQL (Entity SQL ) şeklinde gönderilirler.
- Entity client, veritabanı ile iletişimini database provider ile sorunsuz bir şekilde gerçekleştirir.
- Sonuçlar DbDataReader ile kullanılır.



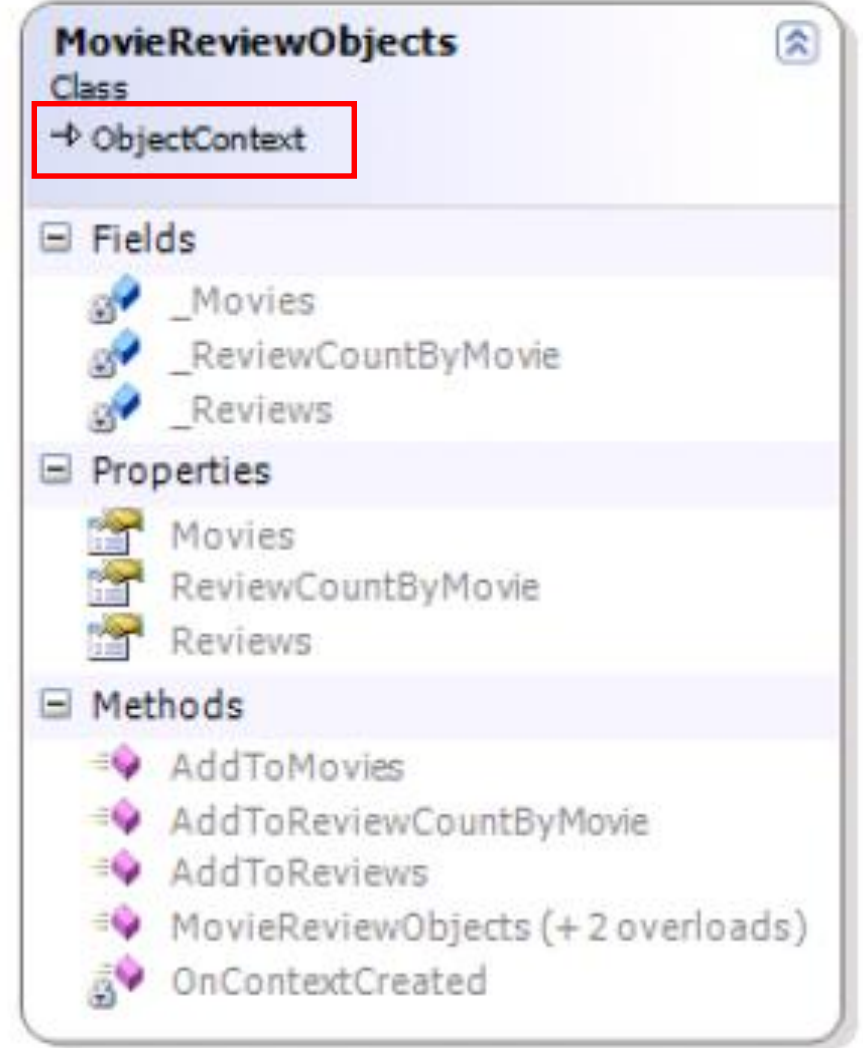
# Entity SQL

- Entity Data Model için var olan SQL'dir.
- Provider neutral'dır.

```
using(MovieReviewObjects ctx =  
    new MovieReviewObjects(connectionString))  
{  
    string command = "SELECT VALUE m FROM Movies as m";  
    var movies = new ObjectQuery<Movie>(command, ctx);  
  
    foreach (Movie m in movies)  
    {  
        Console.WriteLine(m.Title);  
    }  
}
```

# ObjectContext (DbContext)

- Tüm entity'ler için gateway görevi görür.
  - Mapping ve object metadata üzerine kuruludur.
- Entity'ler Entity Set'lerin içerisinde varlıklarını sürdürürler.
- Geriye döndüş için DataReader yerine objelerin kendisini kullanır.






# LINQ to Entities

```
using (MovieReviewObjects context =  
    new MovieReviewObjects(connectionString)) {  
    var movies = from m in context.Movies  
                  where m.Reviews.Count > 1  
                  select m;  
  
    foreach (var m in movies) {  
        Console.WriteLine(m.Title);  
        m.Reviews.Load();  
        foreach (var r in m.Reviews) {  
            Console.WriteLine("\t" + r.Summary);  
        }  
    }  
}
```

# Deferred Loading

- EF, varlıklar arasındaki ilişkiler için "lazy loading" kavramını kullanmaktadır.
- İlişkili varlıklar (entity'ler) açıkça belirtilip "Load" edilmelidir.
- VeyaObjectContext üzerinde "Include" metodu da kullanılabilir.

```
foreach (var m in movies) {  
    Console.WriteLine(m.Title);  
    m.Reviews.Load();  
    foreach (var r in m.Reviews) {  
        Console.WriteLine("\t" + r.Summary);  
    }  
}
```



# Insertion

- Herhangi bir entity eklemek için `AddObject()` metodu kullanılır.
- Tip güvenli olan `ObjectContext` her bir entity için `Add` metotları barındırır.

```
using (MovieReviewObjects ctx =  
    new MovieReviewObjects(connectionString))  
{  
    Movie movie = new Movie()  
    {  
        ReleaseDate = new DateTime(2008, 1, 1),  
        Title = "Revenge of Riverdance"  
    };  
    ctx.AddToMovies(movie);  
    ctx.SaveChanges();  
}
```

# Updates

- "Change tracking service" herhangi bir entity'de meydana gelen değişiklikleri kayıt altına almaktadır.
- `SaveChanges()` , değişen entity'leri otomatik olarak update edecektir.

```
using (MovieReviewObjects ctx =  
    new MovieReviewObjects(connectionString))  
{  
    var movie = (from m in ctx.Movies  
                  where m.Title == "Revenge of Riverdance"  
                  select m).First();  
  
    movie.ReleaseDate = movie.ReleaseDate.AddDays(1);  
    ctx.SaveChanges();  
}
```

# Deletes

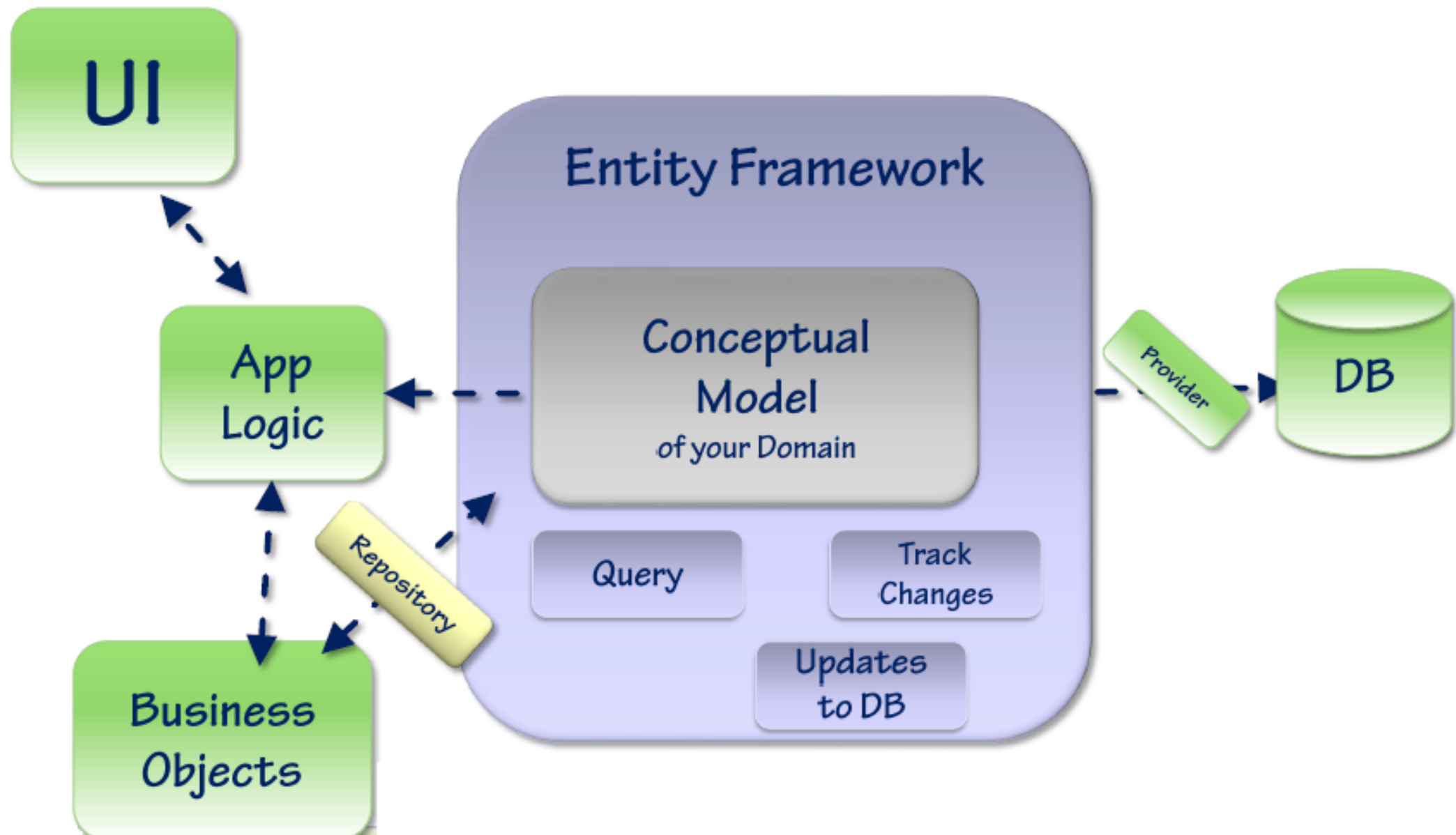
- Silme için DeleteObject() metodu kullanılır.
- SaveChanges() her bir obje için bir tek DELETE yaratacaktır.
- SQL'den farklı olarak silme işleminden önce silinecek olan kaydın sorgulanarak getirilmesi gerekmektedir.



```
using (MovieReviewObjects ctx =  
    new MovieReviewObjects(connectionString))  
{  
    var movies = from m in ctx.Movies  
                  where m.Title == "Revenge of Riverdance"  
                  select m;  
  
    foreach (var m in movies)  
    {  
        ctx.DeleteObject(m);  
    }  
    ctx.SaveChanges();  
}
```

# ADO.NET Entity Framework

*Yaklaşımlar*



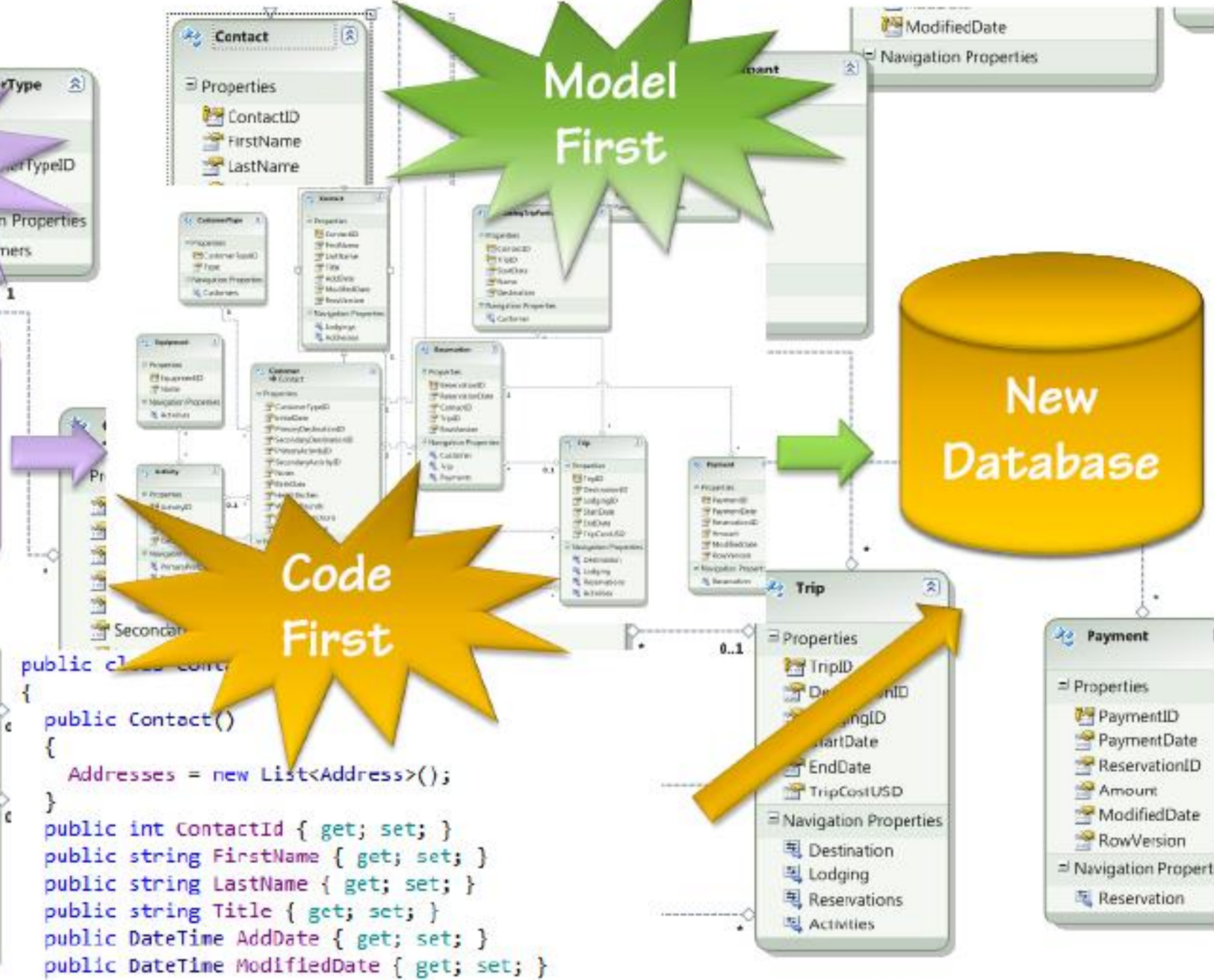
# Ele Alınacak Başlıklar

- Yaklaşımlar (\*)
- Identity (\*\*)
- Varlıkların yaşam döngüsü ve iş ünitesi (Entity Lifecycle and Unit of Work) (\*\*\*)
- Değişikliklerin takibi (Change Tracking)
- İlişkilerin güncellenmesi (Updating Associations)
- Eşzamanlığın yönetimi (Concurrency Management)

## \* Yaklaşımlar

EF'ü kullanmaya başlamadan önce izlenmesi gereken 3 temel yaklaşım bulunmaktadır:

- Database First
- Model First
- Code First



# ORM'ler ve Entity Identity

- ORM araçları entity'lerden database semantikleri çerçevesinde davranmasını ister.
  - Veritabanı ise identity'leri primary keyler olarak ele alır.
- ADO.NET'de iki nesne veritabanındaki aynı satırı temsil edebilir. Böyle bir yaklaşım ise nesne yönelik bakış açısından mantıklı değildir.

# Row Identity

- Bellekteki nesneler, veritabanındaki satırlar ile nasıl ilişkilidir?
  - VT'deki satırlar, birbirlerinden primary key ile birbirlerinden ayırt edilirler.
- ADO.NET'de var olan DataSet ve SqlDataReader kullanılarak, veritabanındaki bir satır, aynı Select sorgusu iki kez çalıştırılarak elde edildiğinde bellekteki görünümü nasıl olacaktır?



```
MoviesDataTable m1;  
MoviesDataTable m2;
```

```
MoviesTableAdapter adapter = new MoviesTableAdapter();  
m1 = adapter.GetData();  
m2 = adapter.GetData();
```

m1

**MoviesDataTable**

m2

**MoviesDataTable**

# Object Identity

- LINQ to SQL kullanılarak bir satır üzerinde aynı sorguyu iki kez işletildiğinde bellekte farklı konumlarda olan iki farklı nesne oluşturulmaz. (Key unique olmak üzere Dictionary <Key, Value> şeklinde düşünülebilir. )

ObjectContext

```
Movie m1;  
Movie m2;  
  
using (var ctx = new MovieReviewEntities())  
{  
    m1 = ctx.Movies.Where(movie => movie.ID == 100).First();  
    m2 = ctx.Movies.Where(movie => movie.ID == 100).First();  
    Debug.Assert(Object.ReferenceEquals(m1, m2));  
}
```

m1

m2

Movie

## \*\* Identity Map Pattern

- Bu kalıp, veritabanından okunan tüm objelerin kaydını tek bir business transaction içerisinde saklamaktadır.
- Eentity Framework bu kalıbı kullanmaktadır.
- Her birObjectContext nesnesi kendi Identity Map'i ile ilişki içerisindedir.
  - Aynı kayıt için iki farklıObjectContext üzerinden yapılan sorgulamalar iki farklı nesne döndürürler.

# Identity Map Pattern(...)

- Veriye dışarıdan yapılan sonradan müdahaleler ObjectContext tarafından görüntülenmez. Bunun nedeni;
  - Üzerinde işlem yapılan ObjectContext'de "consistency" ve "integrity" nin sağlanmak istenmesidir.
  - ObjectContext üzerinde yapılan değişiklikler, yerelde (localde) yapılan değişikliklerdir.
- Entity Framework primary key' e sahip olmayan bir tabloyu update edemez.

## **\*\*Unit of Work Pattern (İş Ünitesi Deseni)**

- Business transaction tarafından etkilenen nesnelerin bir listesini ele alır ve eğer objeler üzerinde bir değişiklik olmuşsa bunu VT'ye yansıtır.
- ObjectContext nesnesi bu prensibe göre çalışır.
  - Bir web uygulaması için iş ünitesi deseni tek bir requestin işlenmesi olabilir.
  - Akıllı bir istemci için ise, bir formun yaşam döngüsü olabilir.
- Unutulmamalıdır ki ObjectContext' in oluşturulmasının maliyeti çok düşüktür.

# Varlıkların Yaşam Döngüsü (Entity Lifecycle)

- Bir nesne ObjectContext'in, kendisinden haberdar olmasından sonra bir entity olarak işlem görür.
  - Böylelikle yaşam döngüsü başlamış olur.
  - Diğer bir değişle, obje veritabanından çekildikten sonra Entity olarak işlem görür.
  - Eğer istenirse, yeni objeler Entity'e eklenebilir.
- Yaşam döngüsü ObjectContext'in GC tarafından bellekten toplanması ile son bulur.

# Değişiklerin Takibi (Change Tracking) \*\*\*

- ObjectContext değişiklikleri bizim yerimize takip edebilmektedir.
- ObjectContext hangi nesnenin değişikliğe uğradığını nereden bilmektedir?

- EF, bir obje yaşam döngüsüne başladığında objenin ilk görünümünü (snapshot) almaktadır.
- Tüm değerler kopyalanmaktadır.
- `SaveChanges()` metodu işletileceği zaman, önceden alınan snapshot ile objenin o anki değerleri karşılaştırılmaktadır.
- Bu durum ilave iş yükü getirmektedir.
- Bu özellik `ObjectContext`'in `"ObjectTrackingEnabled"` özelliği kapatılarak iptal edilebilir.



# Özet

- ObjectContext, EF'deki en temel çalışma birimidir (unit of work).
- EF, "optimistic concurrency" mekanizmasını kullanmaktadır.
- Relationshipler, framework tarafından yönetilir.
- EF kullanırken, vt öğelerinden ziyade objeler şeklinde düşünülmeye çalışılmalıdır.