Entity Framework Part II

Identities, Entities, Patterns

Overview

- Identity – objects versus rows
- Entity lifecycle and the unit of work
- Change tracking
- Updating associations
- Attach and Detach
- Concurrency Management

ORMs and Entity Identity

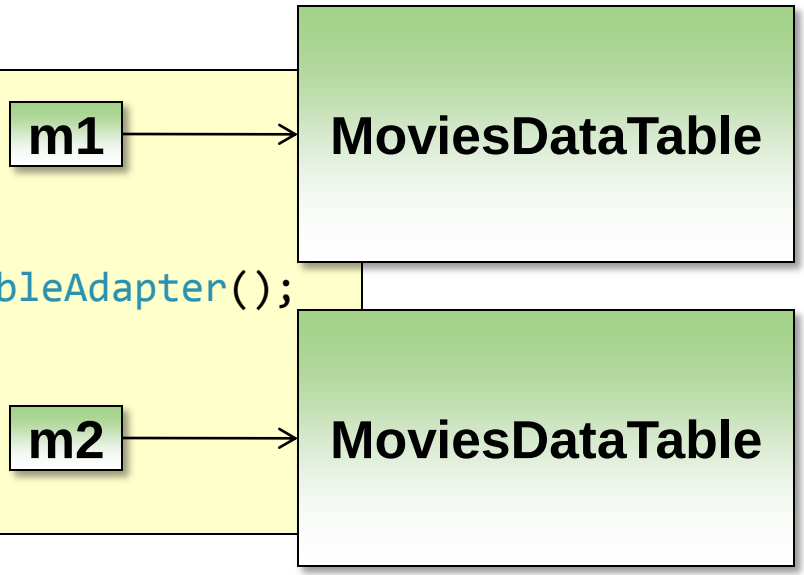
- **ORM tools want entities to behave with some database semantics**
  - Database enforces identity with primary key values
- **In ADO.NET – two objects can represent the same row**
  - The result of two successive invocations of a SQL command
  - This doesn't make sense from an object viewpoint …

Row Identity

- **How do objects relate to rows in the database?**
  - Rows in a database table have a unique primary key
- **What happens if you query for the same movie twice?**
  - Think about the ADO.NET DataSet / SqlReader scenario

```
MoviesDataTable m1;
MoviesDataTable m2;

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

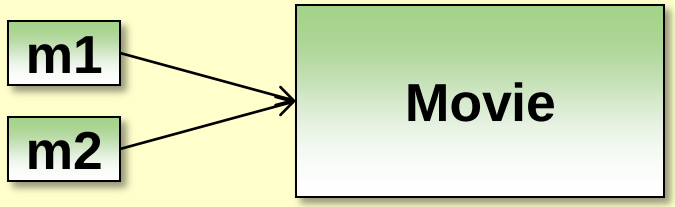


Object Identity

- **What happens if you query for the same movie twice?**
  - As CLR programmers we expect see the same object reference, not two unique objects with the same values.
  - Think about asking a Dictionary<K,T> for an object by unique key

```
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));
}
```



Identity Map Pattern

- **An Identity Map keeps a record of all objects that have been read from the database in a single business transaction.**
  – Fowler
- **EF implements an identity map**
  - Called the “object cache”
  - Retrieved entities are tracked by key value.
  - Asking for a previously retrieved entity will return the previous object instance
  - The type of query used to retrieve the entity is not important
- **Each ObjectContext instance maintains it’s own object cache**
  - Query for the same movie in two different ObjectContexts will return two different objects.
  - Object cache is part of the ObjectStateManager

Consequences of the Identity Map

- **Any changes from "outside" are not visible to our current ObjectContext (if we've already retrieved an entity)**
  - We want consistency and integrity inside our working context
  - The only changes we see are local changes
  - We will revisit concurrency later
- **Entity Framework cannot update a table with no primary key**
  - No way to ensure uniqueness and integrity of retrieved objects

Unit of work Pattern

- ***Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.* – Fowler**
- **ObjectContext is designed to be used in a unit of work**
  - For web apps, a unit of work may represent the processing of a single request
  - For smart client, a unit of work may be the life of a form
  - Unit of work may be encapsulated inside a single method
- **ObjectContext is inexpensive to create**
  - Create as needed
  - Don’t cache or create a singleton
  - Not thread safe

Entity Lifecycle

- **Object becomes an entity when ObjectContext becomes aware of the object**
  - Beginning of the lifecycle
  - Can happen when object is retrieved from database
  - Can also insert new objects and attach existing objects
- **Lifecycle ends when ObjectContext no longer needed**
  - Context and object eligible for garbage collection

Change Tracking

- **ObjectContext uses an internal change tracking service**
  - Service records changes to all known entities
  - ObjectContext uses list of changes to generate SQL command
- **How does the ObjectContext know what changed?**
  - IEntityWithChangeTracker
  - IEntityChangeTracker

Updating Associations

- **Changing an object's relationship requires some work**
  - Object needs to change it's parent reference
  - Object needs to be removed from the original parent's collection
  - Object needs to be added to it's new parent's collection
- **All this work is managed by the framework**
  - Just move the entities, or reassign the parent properties

```
var m1 = ctx.Movies.Include("Reviews“).Where(m => m.Reviews.Count > 1)
                   .First();
var m2 = ctx.Movies.Include("Reviews“).First();

var reviews = m1.Reviews.ToList();
foreach (var review in reviews)
{
     m2.Reviews.Add(review);
}
ctx.SaveChanges();
```

Concurrency Management

- **Concurrency checks are OFF by default**
  - Control in mapping setting concurrency property to “Fixed” for each property

```
UPDATE [movies]
SET [release_date] = @p0
WHERE ([movie_id] = @p1)
```

```
UPDATE [movies]
SET [release_date] = @p3
WHERE ([movie_id] = @p0) AND
      ([title] = @p1) AND
      ([release_date] = @p2)
```

Concurrency Violations

- **SaveChanges can throw an exception**
  - OptimisticConcurrencyException
  - StateEntries property will hold conflicted entities
- **SubmitChanges is atomic - all changes roll back**
- **ObjectContext left unchanged**
  - Changes can be resubmitted
  - Entities can be refreshed from database
  - Use MergeOptions on query

Transactions

- **Use the promotable TransactionScope from System.Transactions**

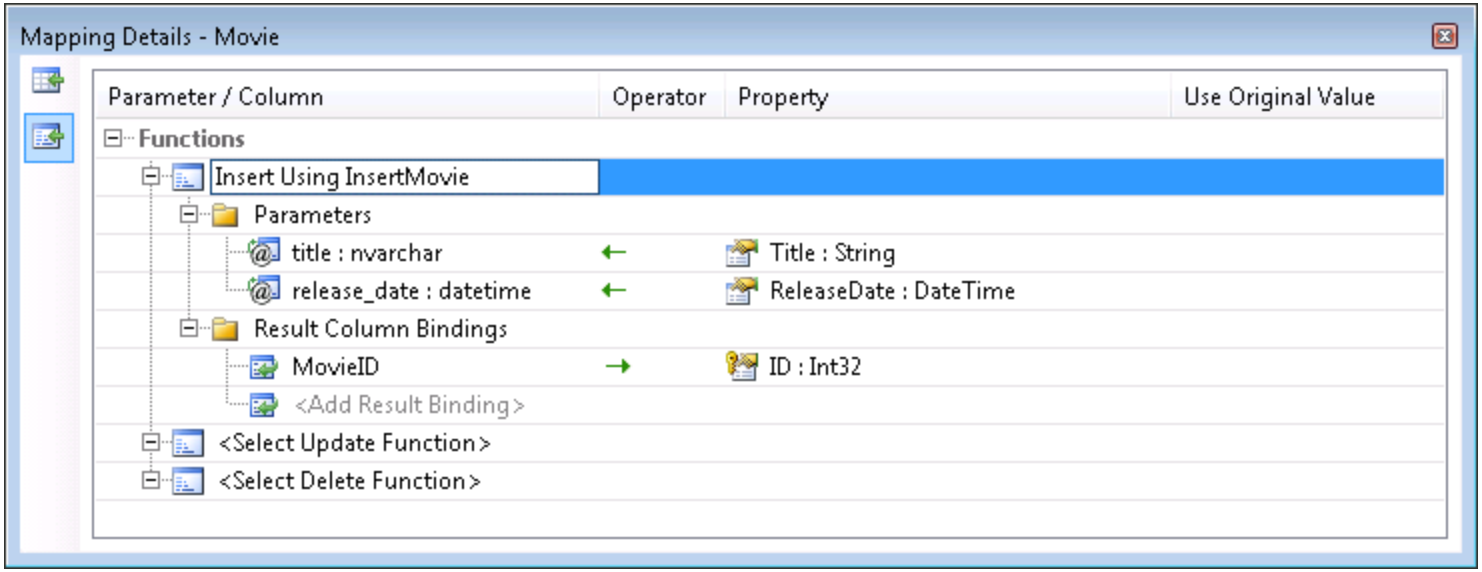
```
using (var txn = new TransactionScope())
using (var ctx = new MovieReviewEntities())
{
    var movie = ctx.Movies.First();

    // do work ...

    ctx.SaveChanges();
}
```

Stored Procedures

- **Can map Insert, Update, Delete operations to stored procedures**
  - Must map all three operations for model to validate



Detached Entities

- **Detached entities are entities that “leave” their ObjectContext**
  - Sent over the wire in a web service call
  - Sent to a client browser
- **Later the entity can be re-attached**
  - But you have to describe how the entity has changed
  - No remote change tracking

Summary

- ObjectContext is the unit of work for Entity Framework
  - Maintains a change tracking service
  - Maintains an identity map
- EF uses optimistic concurrecy
  - Concurrency checks off by default
- Relationships managed by framework
- Think of objects, not database operations