LINQ To SQL Part I

Putting LINQ to Work On Relational Data

pluralsight
see what you can learn

Overview

- Object Relational Mapping
- The Impedance Mismatch
- Mapping Entities
- Object Associations
- Projections, compiled queries, and stored procedures

Data Access With the FCL

- **The FCL (pre .NET 3.5) offered two mechanisms for data access**
  - Data readers – fire hose cursors
  - DataSet - a disconnected model
- **Code generation offers a third technique**
  - Typed DataSet

Problems

- **Lack of Intellisense and compile time checks**
  - Field names as strings
- **Rich domain model requires custom mapping code**
- **SQL statements often embedded in code**
- **Data centric view of an application**
  - We have objects, and we have data
- **Object relational impedance mismatch**

The Infamous Impedance Mismatch

Objects	Databases
Built using OOP principles	Built using relational algebra
Use inheritance and aggregation	Requires data normalization
Link with references	Link with foreign keys
Identified by memory location	Identified by primary key
Use data types defined by runtime	Use datatypes defined by database
Can hold data in lists and trees	Can hold data in tuples
Not transactional (today)	Heavily transactional

Solutions For The Impedance Mismatch

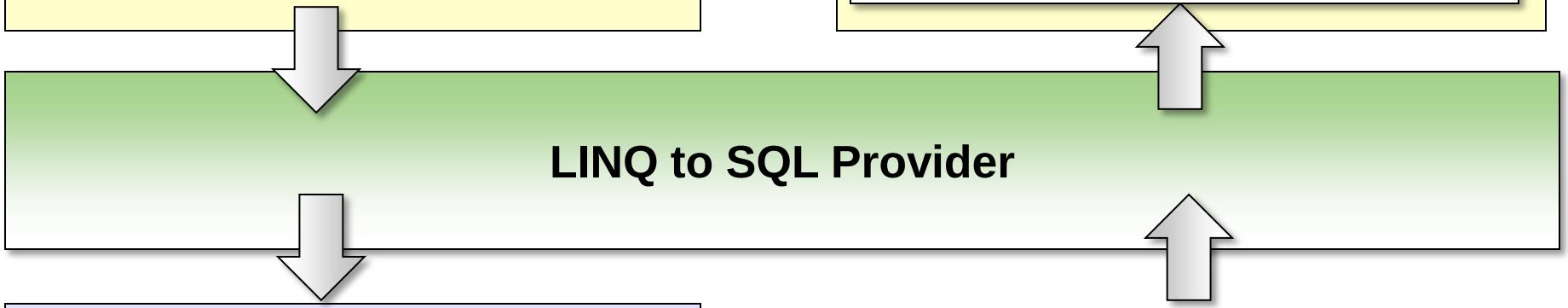
- **A brief list of third party software**
  - NHibernate (http://www.hibernate.org/343.html)
  - CLSA (http://www.lhotka.net/cslanet/)
  - LLBLGen (http://www.llblgen.com/)
  - SubSonic (http://subsonicproject.com/)
  - iBatis (http://ibatis.apache.org/)
  - WilsonORMapper (http://www.ormapper.net/)
- **There is a demand for object-relational mapping**

LINQ to SQL

- **Introduces an ORM into the .NET framework**
  - System.Data.Linq.dll
- **Currently only supports Microsoft SQL Server**
  - 2000, 2005, Mobile Edition
- **Command line tools and Visual Studio designer**
- **Standard LINQ query operators still apply!**
  - Filtering, grouping, sorting, joining (when needed…)
- **Generates parameterized SQL to execute on the database server**
  - Remember IQueryable<T> and Expression<T>?
  - Query operators must be translatable to SQL

LINQ to SQL – The ORM

```
var movies =
 (from m in context.Movies
  select m).Take(3).ToList();
```




```
SELECT TOP (3)
 [t0].[movie_id],
 [t0].[title],
 [t0].[release_date]
FROM [dbo].[movies] AS [t0]
```

movie_id	title	releasedate
1	Casablanca	1942
2	Star Wars	1977
3	Goodfellas	1990

Mapping Entities

- **Mapping tells LINQ to SQL how classes relate to tables and columns**
  - Mapping with plain old CLR objects (POCOs)
  - Mapping with attributes
  - Generating code with sqlmetal.exe
  - Generating code with Visual Studio

Mapping – The POCO Approach

```
class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
}
```

```
<Database Name="moviereviews"
          xmlns="http://schemas.microsoft.com/linqtosql/mapping/2007">
  <Table Name="dbo.movies" Member="Movies">
    <Type Name="Poco.Movie">
      <Column Name="movie_id" Member="ID" />
      <Column Name="title" Member="Title" />
      <Column Name="release_date“ Member="ReleaseDate" />
    </Type>
  </Table>
</Database>
```

Using POCOs

```
string connectionString = "...";
using (DataContext ctx = new DataContext(
          connectionString,
          XmlMappingSource.FromUrl("Poco\\moviereviews.xml")))
{
    var movies =
         from m in ctx.GetTable<Movie>()
         select m;

    foreach (Movie m in movies)
    {
        Console.WriteLine(m.Title);
    }
}
```

Mapping with Attributes

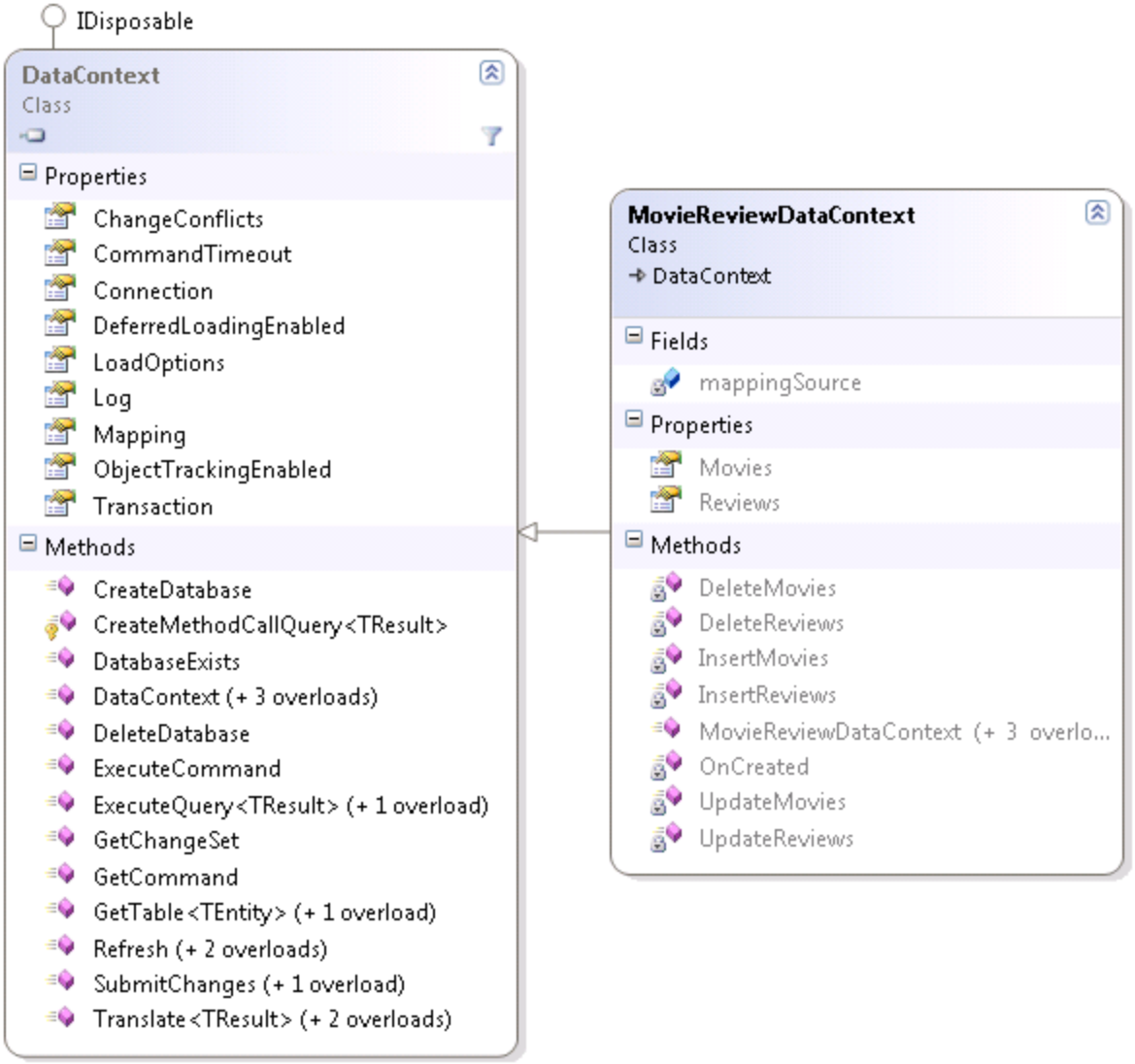
```
[Table(Name="movies")]
class Movie
{
    [Column(Name="movie_id")]
    public int ID { get; set; }

    [Column(Name="title")]
    public string Title { get; set; }

    [Column(Name="release_date")]
    public DateTime ReleaseDate { get; set; }
}
```

The DataContext

- **Gateway to the database**
- **Retrieve, add, update, delete objects**
  - Translate LINQ queries into SQL
  - Assemble objects from the SQL command result

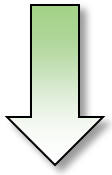


Generating Code with sqlmetal.exe

- **Point sqlmetal.exe to an existing database**
  - Can generate code + mapping file
  - Can generate code with attributes
  - Can also generate DBML file
- **Database support**
  - SQL Server 2000, 2005, 2008
  - SQL Server Express
  - SQL Server Compact Edition
- **Can target multiple languages**
- **No auto-syncing support**

Strongly Typed DataContext

```
$>sqlmetal /server:. /database:moviereviews
          /dbml:MovieReviews.dbml /context:MovieReviewDataContext
```



```
using (MovieReviewDataContext ctx = new MovieReviewDataContext())
{
    var movies =
        from m in ctx.Movies
        select m;

    // ...
}
```

Relationships

- **In the database, records are associated with key values**
  - Foreign key references a primary key
  - Requires JOIN operations to navigate relationships
- **Objects use references to point to associated objects**
  - Associated objects can be navigated with a dot (.)
  - Collections can reference multiple associated objects
- **LINQ to SQL turns relational associations into object references**

Defining Relationships (One to Many)

```
[Table(Name="movies")]
public class Movie
{
    [Column(Name="movie_id", IsPrimaryKey=true, IsDbGenerated=true)]
    public int ID { get; set; }

    [Column(Name="title")]
    public string Title { get; set; }

    [Column(Name="release_date")]
    public DateTime ReleaseDate { get; set; }

    [Association(OtherKey="MovieID")]
    public EntitySet<Review> Reviews { get; set; }
}
```

Defining Relationships (One to One)

```
[Table(Name="reviews")]
public class Review
{
    [Column(Name="review_id", IsPrimaryKey=true, IsDbGenerated=true)]
    public int ID { get; set; }

    [Column(Name="movie_id")]
    public int MovieID { get; set; }

    [Association(Storage="_reviewedMovie")]
    public Movie ReviewedMovie
    {
        get { return _reviewedMovie.Entity; }
        set { _reviewedMovie.Entity = value; }
    }
    EntityRef<Movie> _reviewedMovie;
}
```

Navigating Relationships

- **LINQ to SQL manages joins and correlated sub queries when associations are defined**

```
var topMovies =
        ctx.Movies
           .Where(m => m.Reviews.Count > 3)
           .OrderByDescending(m => m.Reviews.Average(r => r.Rating))
           .Take(10);

foreach (Movie m in topMovies)
{
    Console.WriteLine(m.Title);
    foreach (Review r in m.Reviews)
    {
        Console.WriteLine("\t{0}:{1}", r.Reviewer, r.Rating);
    }
}
```

LINQ to SQL is Lazy

- **LINQ to SQL queries also defers execution**
  - This is important for composing queries
- **For associations, LINQ to SQL also defers loading**
  - This behavior is an important performance consideration

```
var allMovies =
   from movie in ctx.Movies
   orderby movie.Reviews.Average(m => m.Rating)
   select movie;

foreach (var movie in allMovies)
{
   Console.WriteLine(movie.Title);
   foreach (var review in movie.Reviews)
   {
       Console.WriteLine("\t{0}:{1}",
                         review.Reviewer,
                         review.Rating);
   }
}
```

```
SELECT [t0].[movie_id] AS [Movie_id],
       [t0].[title] AS [Title],
       [t0].[release_date] AS [Release_date]
FROM [dbo].[movies] AS [t0]
ORDER BY (
    SELECT AVG([t1].[rating])
    FROM [dbo].[reviews] AS [t1]
    WHERE [t1].[movie_id] = [t0].[movie_id]
    )
```

```
SELECT [t0].[review_id] AS [Review_id],
       [t0].[movie_id] AS [Movie_id],
       [t0].[summary] AS [Summary],
       [t0].[rating] AS [Rating],
       [t0].[review] AS [ReviewText],
       [t0].[reviewer] AS [Reviewer]
FROM [dbo].[reviews] AS [t0]
WHERE [t0].[movie_id] = @p0
```

Deferred Loading

- **Deferred loading creates the illusion of loading an entire tree of objects**
  - In reality, LINQ to SQL only fetches the primary object you asked for
- **Override behavior with DataLoadOptions**
  - Set for life of the DataContext

```
DataLoadOptions loadOptions = new DataLoadOptions();
loadOptions.LoadWith<Movie>(m => m.Reviews);
context.LoadOptions = loadOptions;

var allMovies =
      from movie in context.Movies
      orderby movie.Reviews.Average(m
      select movie;
```

```
SELECT [t0].[movie_id] AS [Movie_id],
        ...
       [t1].[reviewer] AS [Reviewer]
FROM [dbo].[movies] AS [t0]
LEFT OUTER JOIN [dbo].[reviews] AS [t1]
     ON [t1].[movie_id] = [t0].[movie_id]
...
```

Filtering Relationships

- **AssociateWith can filter related objects**
  - Useful when you want to restrict what related objects are loaded
- **AssociateWith is not an eager load**
  - Need to combine AssociateWith and LoadWith
  - Beware – some combinations lead to deferred loading!

```
DataLoadOptions loadOptions = new DataLoadOptions();

loadOptions.LoadWith<Movie>(m => m.Reviews);

loadOptions.AssociateWith<Movie>(
                m => m.Reviews.Where(r => r.Reviewer == "Orson Buggy"));
ctx.LoadOptions = loadOptions;
```

Projections with LINQ to SQL

- **Project a named or anonymous type**
  - Note: you are not loading a proper entity (no updates or deletes)
  - Useful for reporting, mapping into DTOs

```
var movieSummaries =
    from movie in ctx.Movies
    orderby movie.Reviews.Average(r => r.Rating)
    select new
            {
                Title = movie.Title,
                ReviewCount = movie.Reviews.Count,
                AverageRating = movie.Reviews.Average(r => (float)r.Rating)
            };

foreach (var summary in movieSummaries)
{
    Console.WriteLine("{0,-40}\n\t {1,2}:{2}",
        summary.Title, summary.ReviewCount, summary.AverageRating);
}
```

Inheritance

- **LINQ to SQL permits "filtered mapping" to model inheritance**
  - Requires all types to be stored in same table
  - Underlying table needs columns for all possible properties
  - Discriminator column used to map to type

```
[Table]
[InheritanceMapping(Code="D", Type=typeof(Dog))]
[InheritanceMapping(Code="C", Type=typeof(Cat), IsDefault=true)]
public abstract class Animal
{
    [Column]
    public string Name { get; set; }
    // ...
}
```

```
public class Dog : Animal
{
    [Column]
     public bool KennelClubMemeber { get; set; }
    // ...
}
```

Compiled Queries

- **There is some overheard in the SQL translation**
  - How much overhead depends on the types of queries you need
- **CompiledQuery can cache a translated LINQ query**

```
var findMovieByIDQuery =
    CompiledQuery.Compile(
        (MovieReviewDataContext dc, int movieID) =>
            (from movie in dc.Movies
             where movie.ID == movieID
             select movie).FirstOrDefault()
    );

using (MovieReviewDataContext ctx = new
MovieReviewDataContext(connectionString))
{
    Movie movie = findMovieByIDQuery(ctx, 1);
}
```

Executing SQL

- **You may need to execute an arbitrary SQL command**
  - Use ExecuteQuery to process a resultset
    - Column names mapped to properties
  - Use ExecuteCommand to retrieve a scalar value

```
var moreMovies =
    ctx.ExecuteQuery<Movie>(
         "SELECT * FROM movies WHERE movie_id < {0}", 10);

foreach (var movie in moreMovies)
{
    Console.WriteLine(movie.Title);
}
```

Other Database Objects

- **Views can be accessed via the DataContext**
  - Uses the same [Table] mapping as a real table
- **Stored procedure and Function support**
  - Can map intro strongly typed members of the DataContext

```
[Function(Name="dbo.GetMoviesSinceDate")]
public ISingleResult<GetMoviesSinceDateResult>
       GetMoviesSinceDate([Parameter(Name="StartDate“)] DateTime? startDate)
{
  IExecuteResult result = this.ExecuteMethodCall(
        this, ((MethodInfo)(MethodInfo.GetCurrentMethod())), startDate);
  return ((ISingleResult<GetMoviesSinceDateResult>)(result.ReturnValue));
}
```

Summary

- LINQ to SQL is an ORM
  - Reduces the effects of the object relational impedance mismatch
- Translates Expression trees into SQL statements
- Entity classes are mapped to tables
  - XML mapping,
  - Attribute mapping
- LINQ to SQL Manages object associations

References

- .LINQ to SQL Performance http://blogs.msdn.com/ricom/archive/2007/06/25/dlinq-linq-to-sql-performance-part-2.aspx