

ADO.NET Data Services

Pushing Data Over the Web

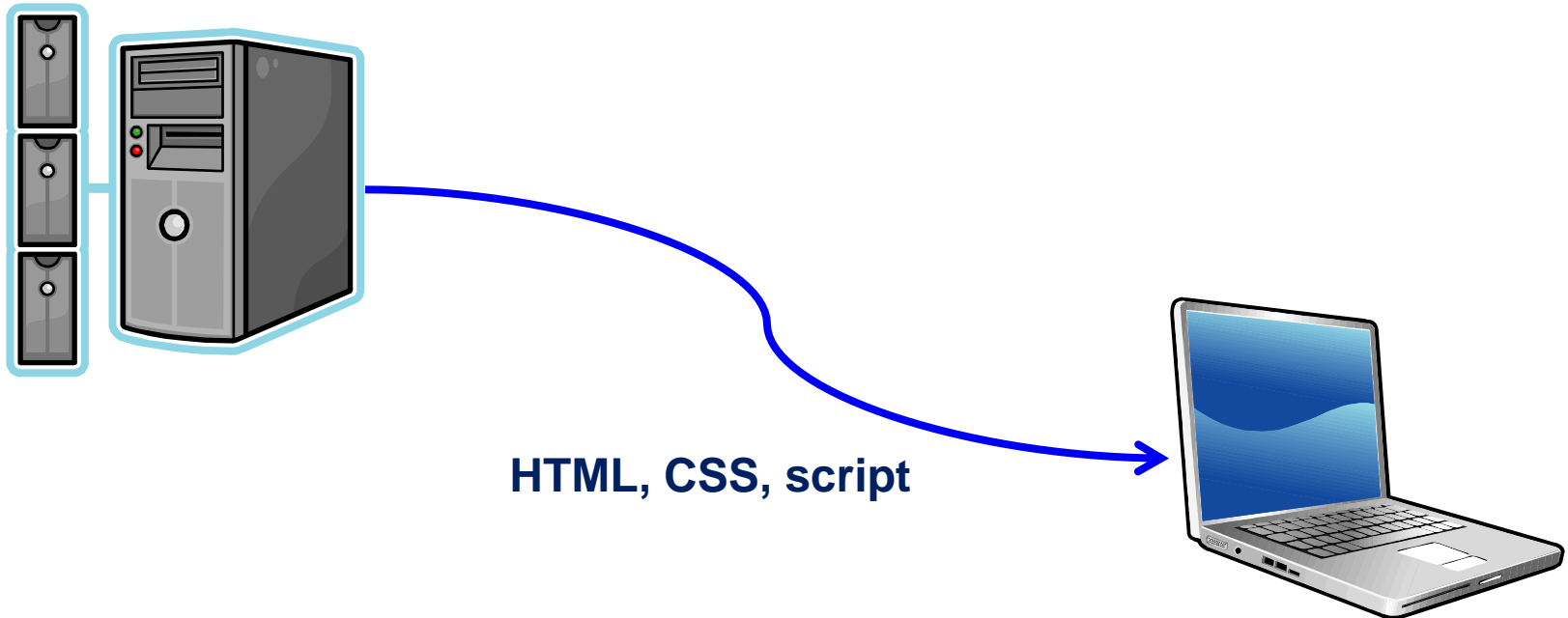


Overview

- **Why Data Services?**
- **ADO.NET Data Services and REST**
- **Building and configuring a Data Service**
- **Consuming Data Services**
- **Query Interceptors**
- **Service Operations**

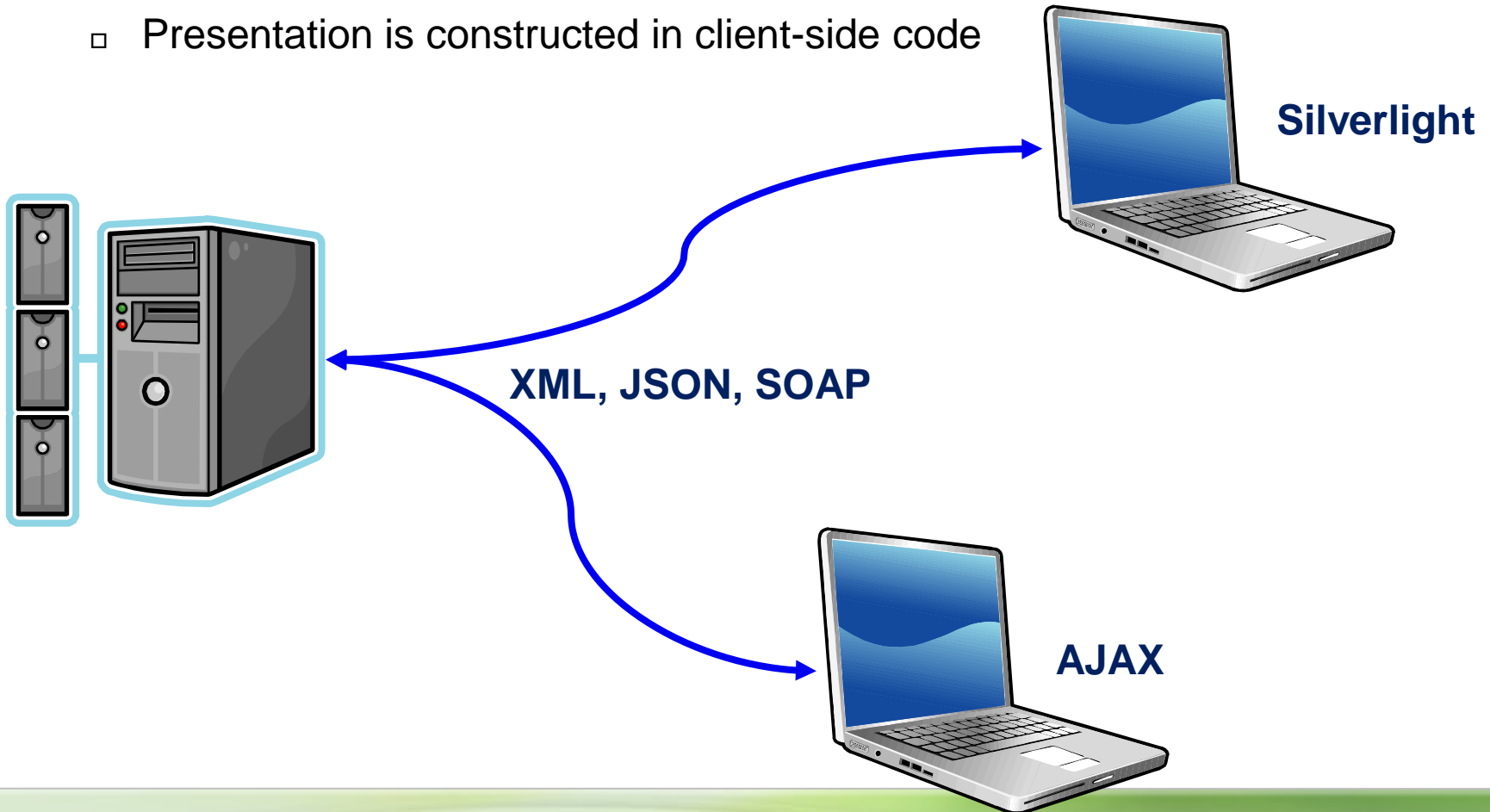
Motivation

- **Traditional web applications consume data on the server**
 - Send only HTML, CSS, and some script to the client



Enter the Rich Internet Application

- **Client wants to consume raw data**
 - Presentation is constructed in client-side code



Challenges

- **Need to expose raw data to the web**
- **Traditional SOAP based web services have some drawbacks**
 - Have an obsession with HTTP POST
 - Focus on operations (verbs) - not data (nouns)
 - Rely on XML, WSDL, WS-*, and SOAP tooling

ADO.NET Data Services Are RESTful

- **REpresentational State Transfer**

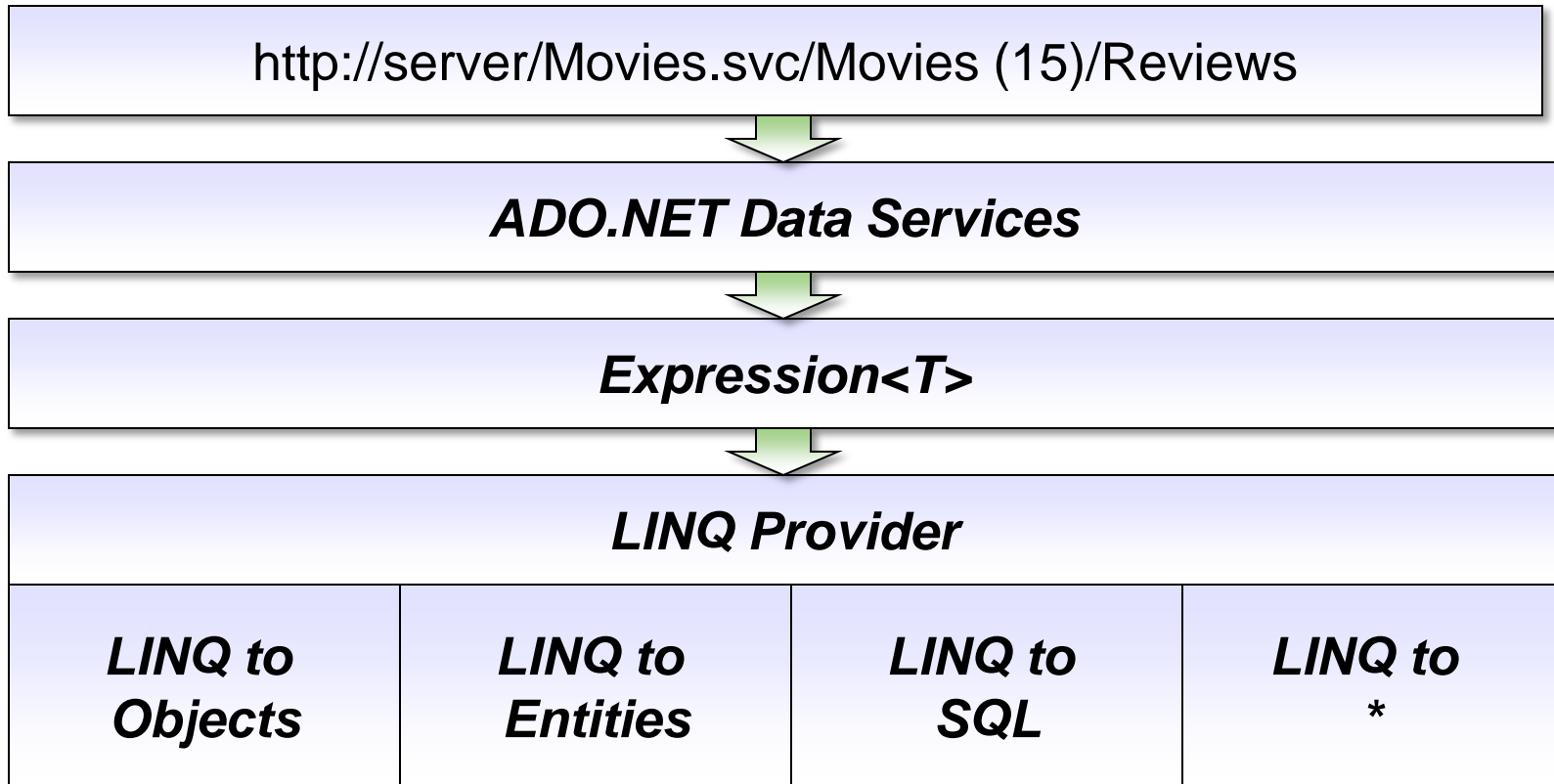
- Uses only HTTP and HTTPS
- Defines 4 operations with the HTTP verbs GET, POST, PUT, DELETE
- Treat entities as resources – entities are addressable by URL

http://server/Movies.svc/Movies

http://server/Movies.svc/Movies (15)/Reviews

LINQ's Role in ADO.NET Data Services

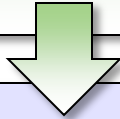
- LINQ's server side role is to provide storage independence



Response Formats

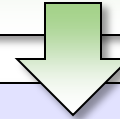
- **Negotiated via the HTTP Accept header**
- **Can choose from Atom (default) and JSON**

```
GET /movieservice.svc/Movies(1)  
Accept: application/atom+xml
```



```
<?xml version="1.0" ?>  
...  
  <d:movie_id>1</d:movie_id>  
  <d:title>  
    Where the Wild Things Are  
  </d:title>  
...
```

```
GET /movieservice.svc/Movies(1)  
Accept: application/json
```



```
{ "d" : {  
  ...  
  "movie_id": 1,  
  "title": "Where the Wild Things Are",  
  ...  
} }
```


URL \$ Options

Option	Description	Example
\$value	Retrieve a value without any surrounding metadata	/Movies(2)/Title/\$value
\$expand	Eager loading of specified elements	/Movies(1)?\$expand=Reviews
\$filter		/Movies?\$filter=Title eq 'Star Wars'
\$orderby	Sort the target resources	/Movies?\$orderby=Title desc
\$top	Return only the top n resources	/Movies?\$top=10 /Movies?\$orderby=Title&\$top=5
\$skip	Skip the first n resources	/Movies?\$skip=100&\$top=10

Basic Ingredients For a Data Service

- **One assembly reference**
 - System.Data.Services
- **One WCF service endpoint**
 - Use Factory=DataServiceHostFactory in the @ Service attribute
 - Service class derives from DataService<T>
- **One data source**
 - Entity Framework EDM Model (ObjectContext derived class)
 - Any CLR type with one or more public IQueryable<T> properties

Setting Up A CLR Model Data Source

- **Resources must have a primary key**
 - ID property, or a [DataServiceKey] decoration
- **Implement IUpdateable if CUD support is required**
 - Entity Framework provides it's own IUpdateable implementation

```
public class Movie
{
    public int ID {get; set;}
    public string Title {get; set;}
    public List<Review> Reviews
        {get; set;}
}
```

```
public class MovieDataSource
{
    ...
    public IQueryable<Movie> Movies
    {
        get { return _movies.AsQueryable(); }
    }

    List<Movie> _movies;
}
```

Configuration

■ Entity Access Rules

- Everything is off by default
- SetEntityAccessRule uses property names to specify access rights
- Wildcard (*) allows access to all public, queryable properties

```
public class InMemoryMovies :  
    DataService<MovieDataSource>  
{  
    public static void InitializeService(  
        IDataServiceConfiguration config)  
    {  
        config.SetEntitySetAccessRule("Movies",  
                                       EntitySetRights.All);  
    }  
}
```

A Basic .NET Client

- Use any class that can send an HTTP request
 - System.Net.WebRequest
 - System.Linq.Xml.XDocument

```
string url = "http://server/InMemoryMovies.svc/Movies/";
string orderby = "$orderby=Title desc";

XDocument result = XDocument.Load(String.Format("{0}?{1}", url, orderby));

XNamespace atom = "http://www.w3.org/2005/Atom";
XNamespace ds = "http://schemas.microsoft.com/ado/2007/08/dataservices";
var movies = from m in result.Descendants(atom + "content")
              select new {
                  ID = m.Descendants(ds + "ID").First().Value,
                  Title = m.Descendants(ds + "Title").First().Value
              };
};
```

Using the Data Services Client

- **DataServiceContext class** – designed to work with Data Services
 - Lives in the System.Data.Services.Client assembly
 - Converts LINQ queries to REST requests
 - Converts response to types defined in client assembly via reflection

```
Uri serviceRoot = new Uri("http://server/InMemoryMovies.svc");
DataServiceContext ctx = new DataServiceContext(serviceRoot);

var result = from m in ctx.CreateQuery<Movie>("Movies")
              orderby m.Title descending
              select m;
```



```
GET /MovieSite/InMemoryMovies.svc/Movies()?$orderby=Title%20desc
HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
Accept: application/atom+xml,application/xml
Connection: Keep-Alive
```

Strongly Typed Client

- **Command line code generation - Datasvcutil.exe**

- Generate codes for a DataServiceContext derived class
- Generates code client side resource representations
- Similar to sqlmetal.exe and Entity FrameworkObjectContext derivations

```
>datasvcutil /uri:http://server/moviesite/movies.svc /out:Movies.cs
```

```
Uri uri = new Uri("http://localhost:8080/InMemoryMovies.svc/");  
  
MovieDataSource ctx = new MovieDataSource(uri);  
var result = from m in ctx.Movies  
              orderby m.Title  
              select m;
```

Creating an Entity

- **Pass the entity set name and the new object to AddObject**
 - Server will respond with fresh representation
 - DataService context will update client side object
 - No data sent to server until you invoke SaveChanges

```
MovieDataSource ctx = new MovieDataSource(uri);
Movie movie = new Movie {
    Title = "No Country For Young Men"
};

ctx.AddToMovies(movie);
ctx.SaveChanges();
```

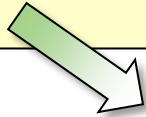


```
POST /MovieSite/InMemoryMovies.svc/Movies HTTP/1.1
...
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
...
<content type="application/xml">
  <m:properties>
    <d:Title>No Country For Young Men</d:Title>
  </m:properties>
</content>
```


Updating An Entity

- Query for the object, make changes, then invoke UpdateObject

```
MovieDataSource ctx = new MovieDataSource(uri);  
Movie movie = (from m in ctx.Movies  
               where m.ID == 2  
               select m).First();  
  
movie.Title = "Monsters, Inc.";  
ctx.UpdateObject(movie);  
ctx.SaveChanges();
```

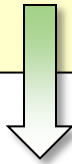


```
PUT /MovieSite/InMemoryMovies.svc/Movies(2) HTTP/1.1  
...  
<?xml version="1.0" encoding="utf-8" standalone="yes"?>  
...  
  <content type="application/xml">  
    <m:properties>  
      <d:ID>2</d:ID>  
      <d:Title>Monsters, Inc.</d:Title>  
    </m:properties>  
  </content>
```

Delete Entity

- Query for the entity, then pass the entity to Delete object

```
MovieDataSource ctx = new MovieDataSource(uri);  
Movie movie = (from m in ctx.Movies  
               where m.ID == 2  
               select m).First();  
ctx.DeleteObject(movie);  
ctx.SaveChanges();
```



```
DELETE /MovieSite/InMemoryMovies.svc/Movies(2) HTTP/1.1  
Content-Length: 0
```

Silverlight Client

- Can “Add Service Reference” for a strongly typed client
 - All network requests in Silverlight are asynch

```
var ctx = new MovieReviews.MovieReviewEntities(  
    new Uri("MovieReviewService.svc",  
        UriKind.Relative));  
  
DataServiceQuery<Movie> query =  
    ctx.Movies.OrderBy(m => m.Title)  
        .Take(100) as DataServiceQuery<Movie>;  
query.BeginExecute((result) =>  
    _grid.ItemsSource = query.EndExecute(result).ToList(),  
    null  
);
```

AJAX Clients

■ ADO.NET Data Service AJAX Client Library

- <http://www.codeplex.com/aspnet/Release/ProjectReleases.aspx?ReleaseId=13357>
- Provides query, insert, update, and delete methods
- Parses results into columns and rows.

```
var service =  
    new Sys.Data.DataService("http://localhost:8080/InMemoryMovies.svc/");  
service.query("Movies?$orderby=Title desc",  
    onQueryComplete, onQueryFailed);
```

```
function onQueryComplete(result) {  
    var sb = new Sys.StringBuilder();  
    for (row in result) {  
        sb.append(  
            String.format("ID = {0} Title = {1}<br />",  
                result[row].ID, result[row].Title));  
    }  
    $get("resultsDiv").innerHTML = sb.toString();  
}
```

Query Interceptors

- **Interceptors are fired on a GET request for a particular resource**
 - Inject custom logic into processing pipeline on a per request basis
 - Uses: custom authorization, custom validation

```
public class InMemoryMovies : DataService<MovieDataSource>
{
    [QueryInterceptor("Movies")]
    public Expression<Func<Movie, bool>> OnQueryMovies()
    {
        return movie => movie.Title.StartsWith("Star");
    }

    // ...
}
```

Change Interceptors

- Change Interceptors fired on PUT, POST, DELETE operations

```
[ChangeInterceptor("Movies")]  
public void OnChangeMovies(Movie m, UpdateOperations operations)  
{  
    if ((operations & UpdateOperations.Delete) == UpdateOperations.Delete  
        && Thread.CurrentPrincipal.Identity == null)  
    {  
        throw new DataServiceException(400, "YOU cannot delete movies");  
    }  
}
```

Service Operations

- **Expose a method of the data service class as a URI**
 - Method could include custom business logic or complicated query logic
- **Restrictions**
 - Method can only accept in parameters
 - Must return void, 'IQueryable<T>', or IEnumerable<T>
 - Must decorate with [WebGet] or [WebInvoke]

```
[WebGet]
public IQueryable<Movie> TopRatedMovie(int minReviews)
{
    var result = from m in CurrentDataSource.Movies
                  where m.Reviews.Count > minReviews
                  orderby m.Reviews.Average(r => r.Rating) ascending
                  select m;

    return result;
}
```

Summary

- **ADO.NET Data Services provides a RESTful interface to data**
 - A great option to expose data to the web
- **GET, PUT, POST, DELETE are the four basic operations**
- **Data Service response comes in ATOM or JSON**
- **DataServiceContext class enables LINQ to Data Services**

The ADO.NET Entity Framework

Part I

Beyond Object Relational Mapping



Overview

- **Background**
- **Models, Mapping, and Metadata**
- **Entity SQL and LINQ to Entities**
- **Object Services**
- **Compare LINQ to Entities with LINQ to SQL**

Impedance Mismatch Redux

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

Entity Framework

- **The new ADO.NET**

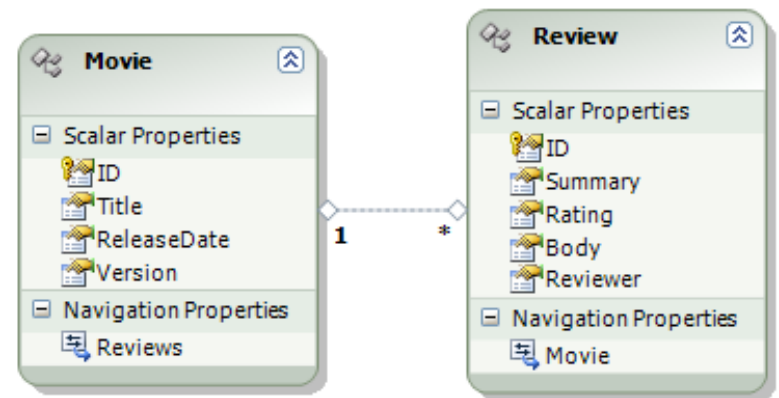
- Higher level of abstraction than ADO.NET
- Introduces the concept of an Entity Data Model
- Vision goes beyond traditional ORM tools to provide “data services”

- **Features**

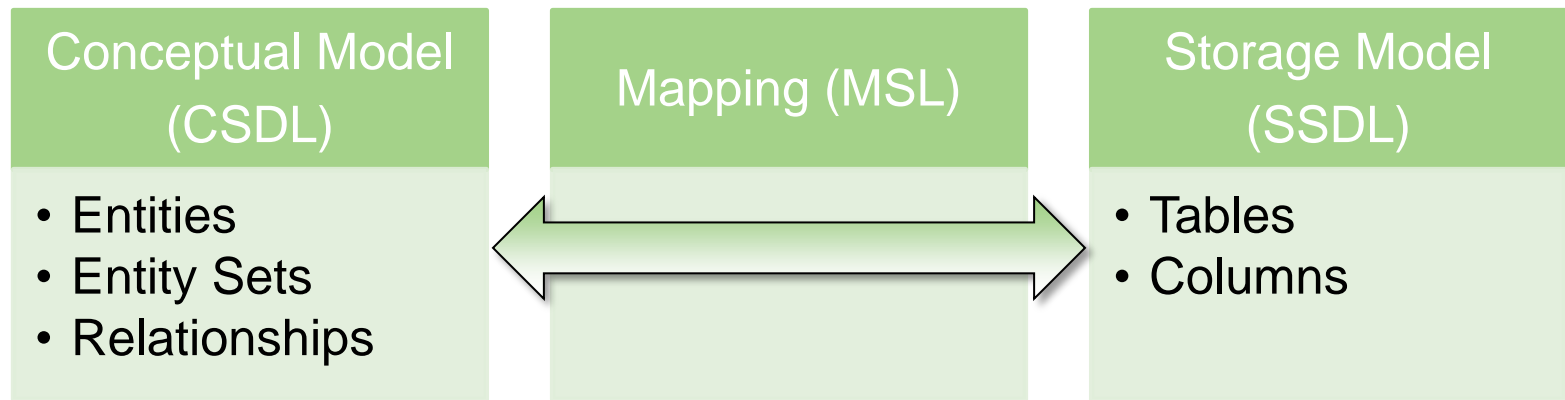
- LINQ Provider
- Visual Studio designer support
- Flexible mapping
- Data provider model (to support Oracle, DB2, etc)

Entity Focus

- **An Entity is**
 - An object we can persist
- **An Entity has**
 - An entity key that makes the entity uniquely identifiable
 - One or more scalar or complex properties
 - One or more relationships to other entities

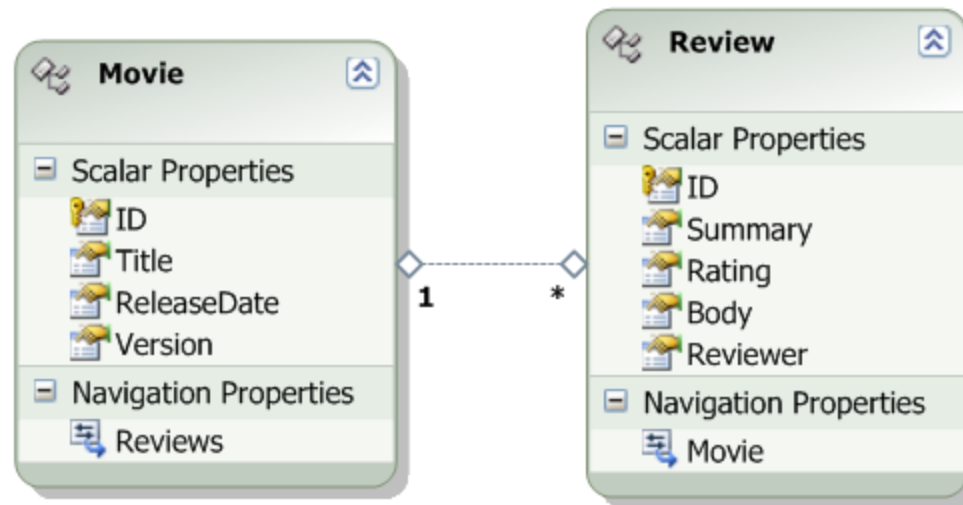


The Entity Data Model



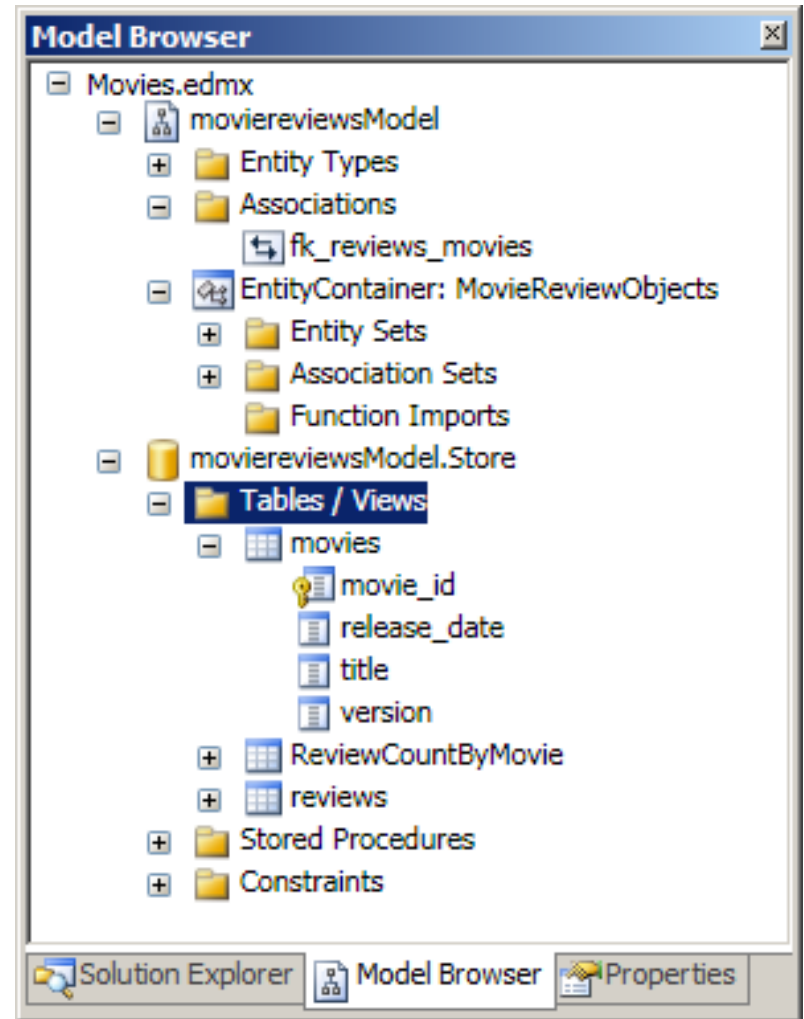
Entity Designer

- Create entities and relationships
- Define keys, types, nullability



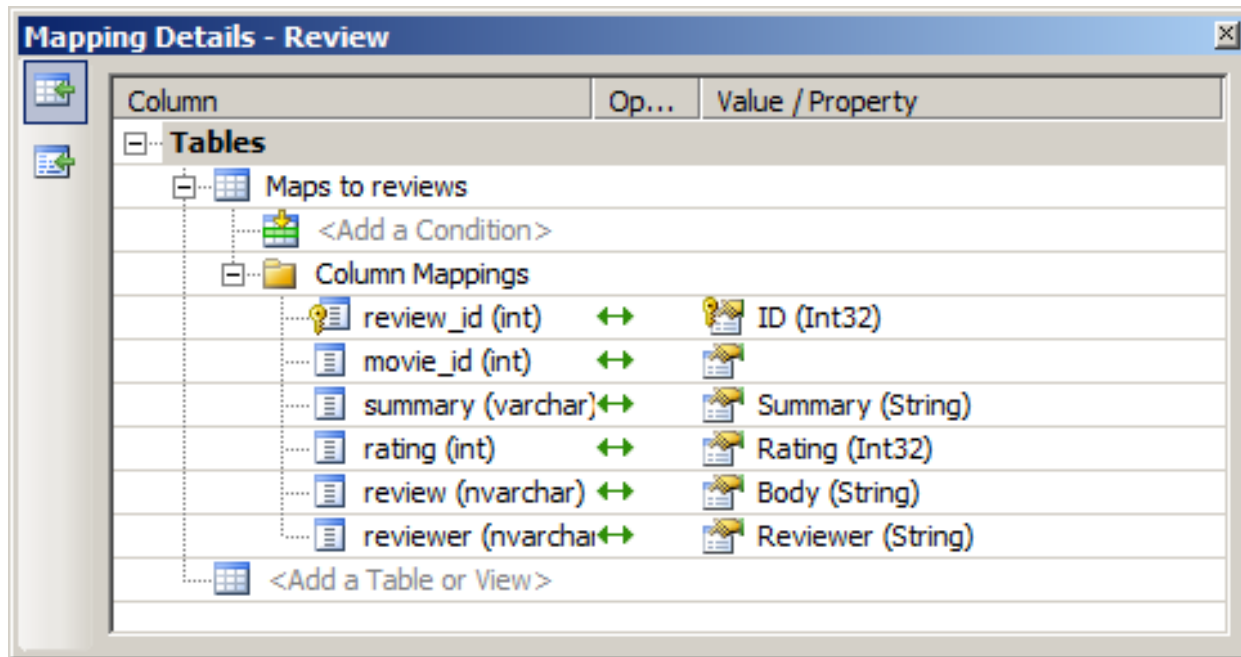
Model Browser

- **Browse model**
 - GUI can be difficult to navigate
- **Validate model**
- **Update model from database**

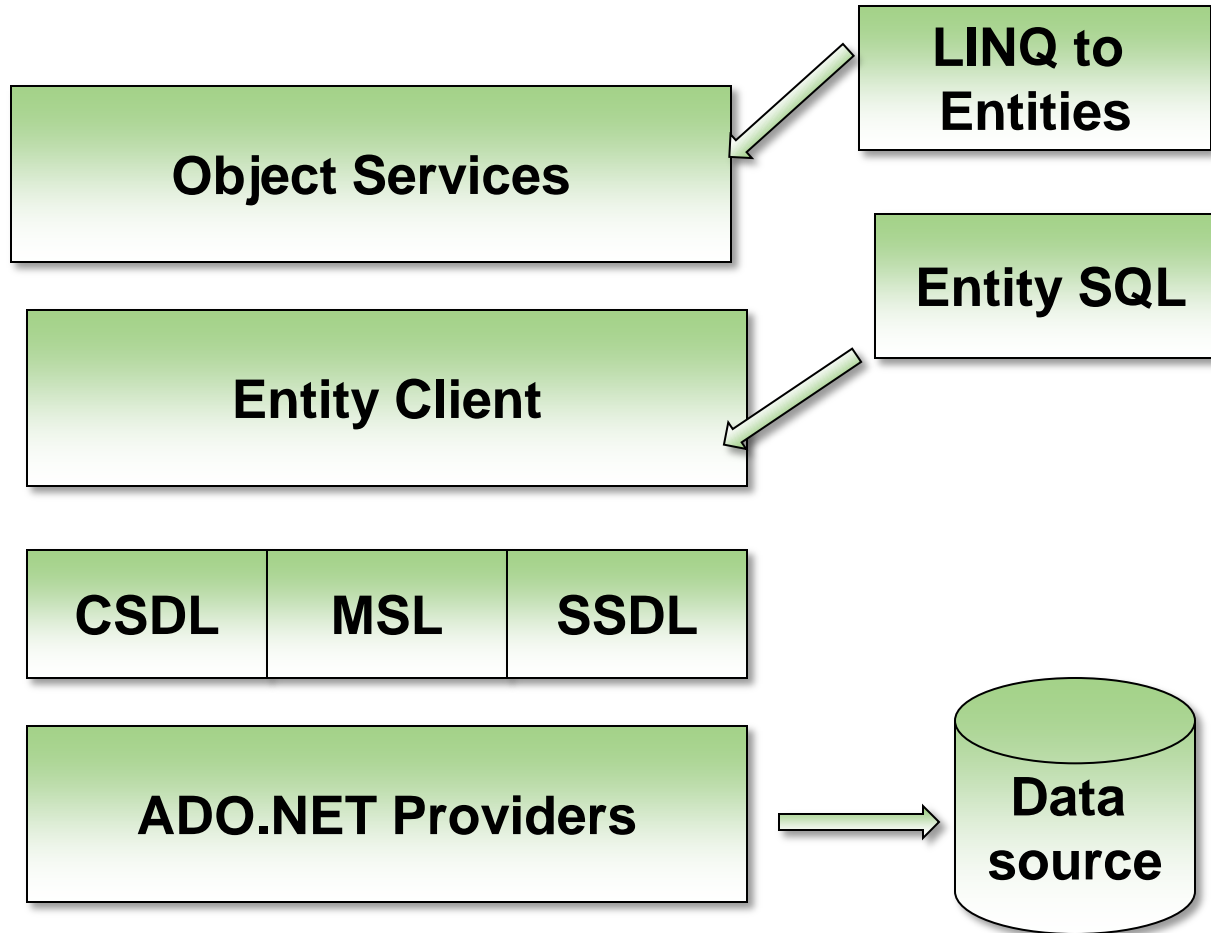


Mapping Details

- Map entities across one or more tables
- Right-click on entity in design or browser and select “table mapping”

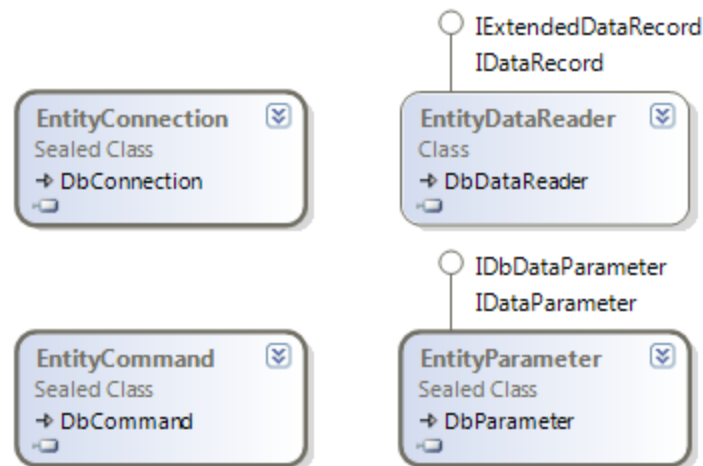


Entity Framework Services



Entity Client

- **No more database specific constructs in ADO.NET code**
 - Queries sent to client as eSQL (Entity SQL)
- **Queries run against entity model, not the underlying storage model**
 - Entity client communicates with a database specific provider
- **Results can be consumed through a DbDataReader**



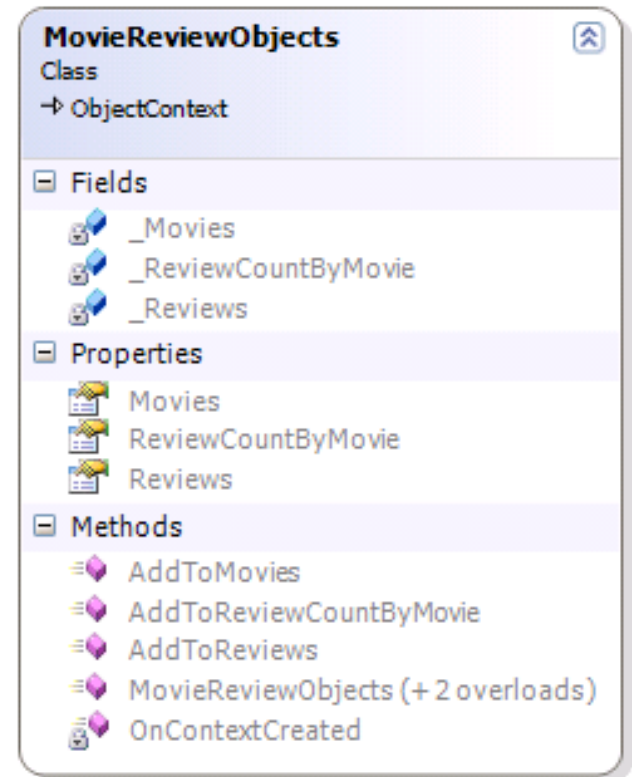
Entity SQL

- Structured Query Language for the entity data model
- Provider neutral

```
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

- **Gateway to all entities**
 - Relies on mapping and object metadata
- **Entities live inside Entity Sets**
 - Exposed as `ObjectQuery<T>` properties on the `ObjectContext`
 - `ObjectQuery<T>` implements `IQueryable<T>`
- **Materializes objects instead of returning a data reader**



LINQ to Entities


- Same standard operators and query syntax

```
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

- Entity Framework does use “lazy loading” for relationships
- Related entities must be explicitly loading using Load
- Can also eager load using an Include method on theObjectContext

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



Inserting Data

- Use `AddObject` to add any type of entity
- Strongly typed `ObjectContext` includes `Add` methods for each entity.

```
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 will record any changes to materialized entities
- **SaveChanges** will atomically update all changed entities

```
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

- Use DeleteObject on the object context.
- SaveChanges will create one DELETE for each object

```
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();  
}
```

Compare and Contrast

	LINQ to SQL	Entity Framework
Advanced Mapping	No	Yes
POCO support	Yes	No
Lazy loading	Implicit	Explicit
Other database support	SQL only	Planned
Full query language	No	eSQL
Enhanced in .NET 4.0	?	Yes

Summary

- **Entity Data Model is the centerpiece**
 - **Broken into three layers**
- **Object Services**
 - **Change tracking**
 - **LINQ to Entities**

Entity Framework Part II

Identities, Entities, Patterns



Overview

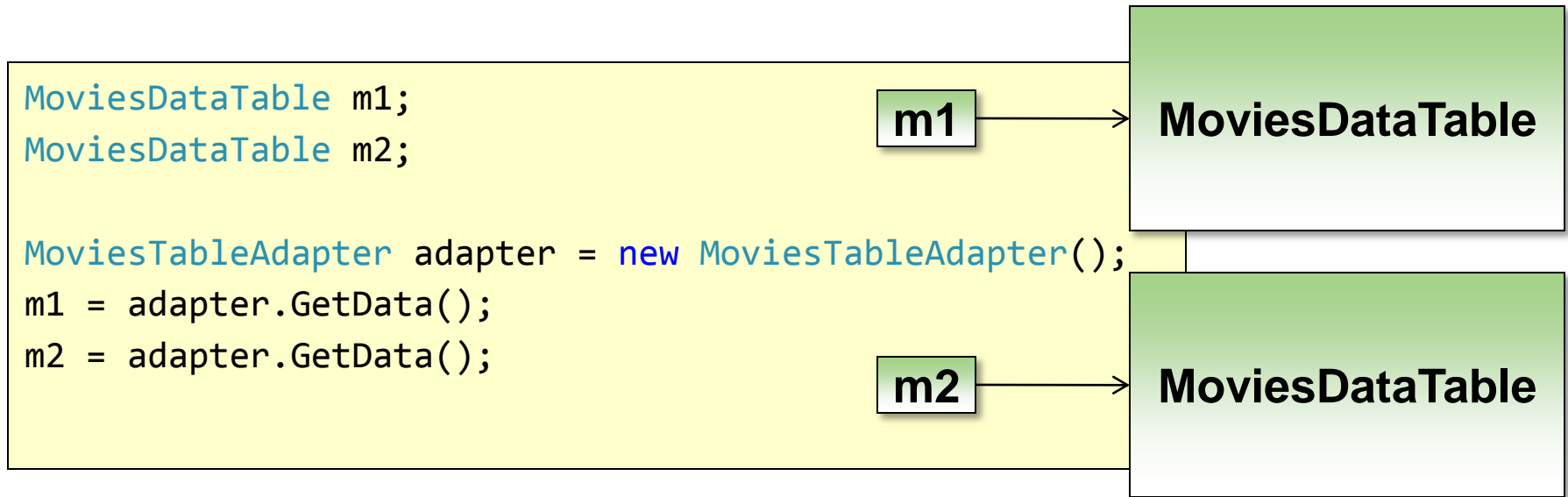
- **Identity – objects versus rows**
- **Entity lifecycle and the unit of work**
- **Change tracking**
- **Updating associations**
- **Attach and Detach**
- **Concurrency Management**

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

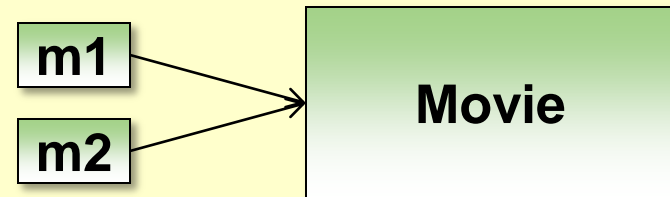
- **How do objects relate to rows in the database?**
 - Rows in a database table have a unique primary key
- **What happens if you query for the same movie twice?**
 - Think about the ADO.NET DataSet / SqlDataReader scenario



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
- **EachObjectContext 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

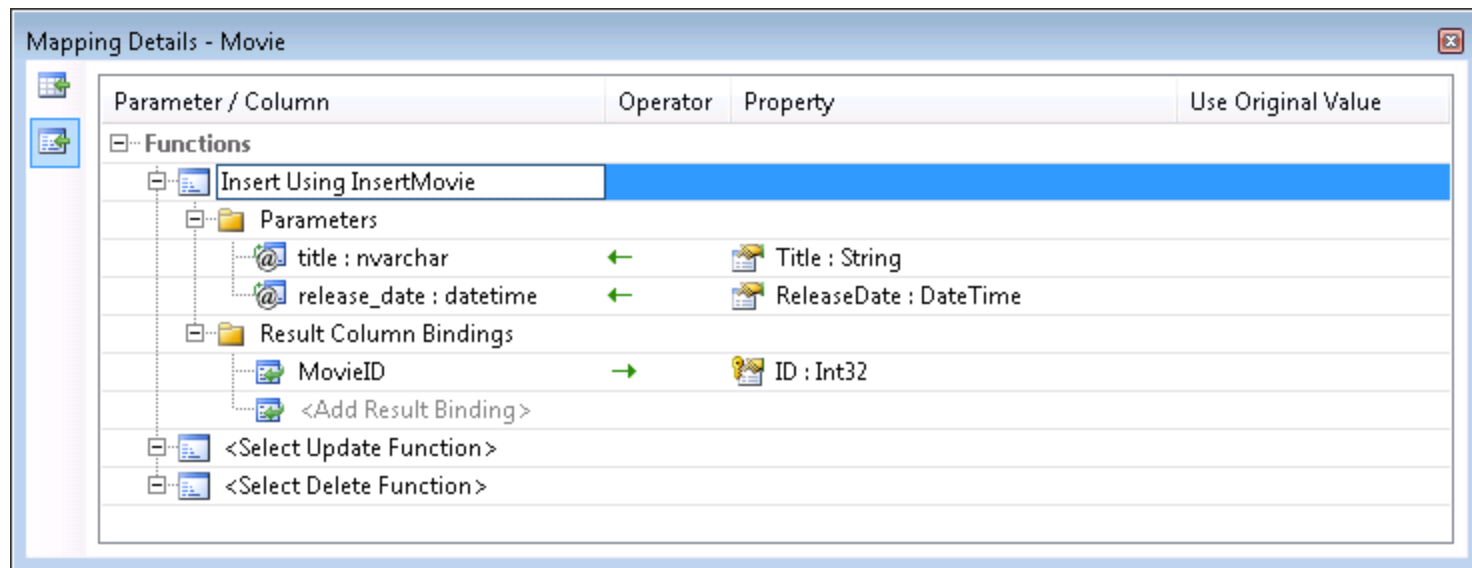
Transactions

- 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” theirObjectContext**
 - 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 concurrency**
 - **Concurrency checks off by default**
- **Relationships managed by framework**
- **Think of objects, not database operations**

LINQ To SQL Part I

Putting LINQ to Work On Relational Data



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();
```

Movie

Movie

Movie

LINQ to SQL Provider

```
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

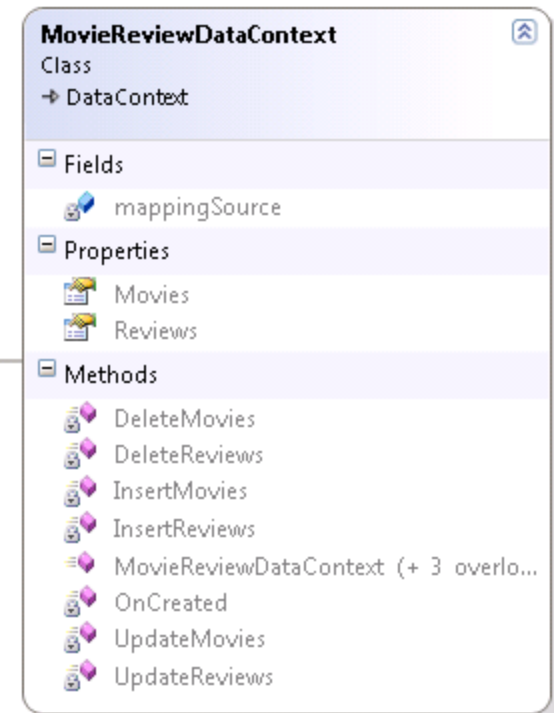
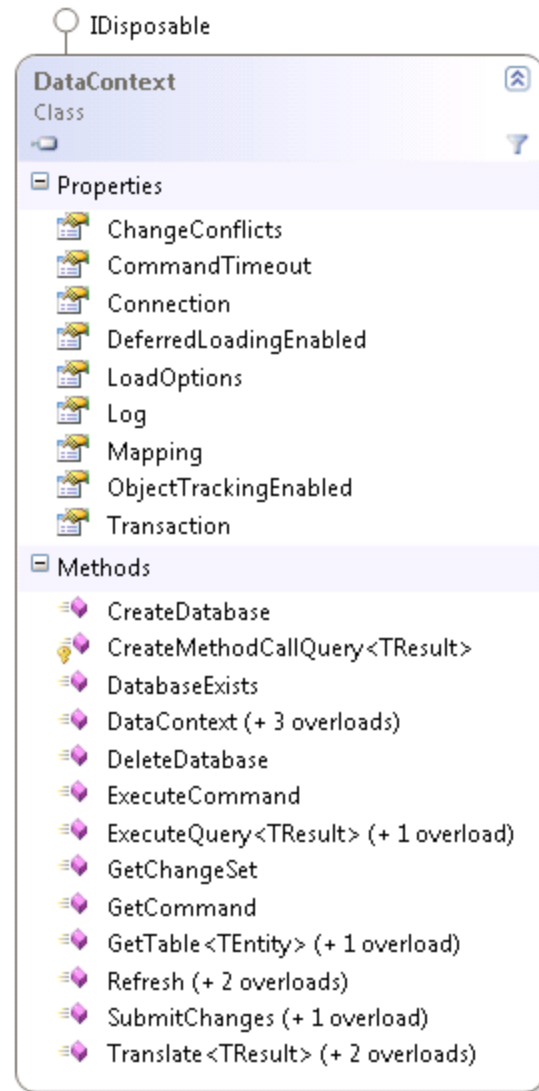
```
[Table(Name="movies")]
class Movie
{
    [Column(Name="movie_id")]
    public int ID { get; set; }

    [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

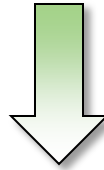


Generating Code with sqlmetal.exe

- **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

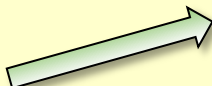
- LINQ to SQL manages joins and correlated sub queries when associations are defined

```
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);  
    }  
}
```

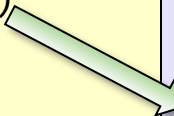

LINQ to SQL is Lazy

- **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 overhead 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 into 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>

LINQ To SQL Part II

Inside the DataContext and Modifying Data



Overview

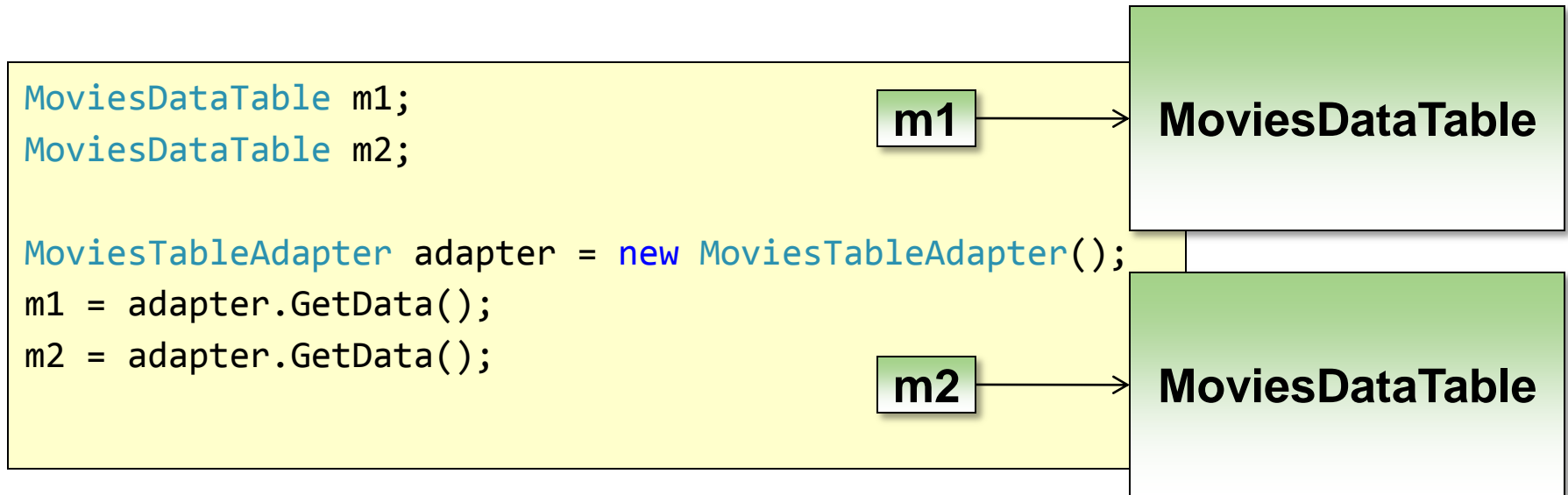
- **Identity – objects versus rows**
- **Entity lifecycle and the unit of work**
- **Change tracking**
- **Updating associations**
- **Attach and Detach**
- **Concurrency Management**

Modifying Data

- **Object Relational Mappers want you to think about objects!**
- **CUD operations with ADO.NET typically not about objects.**
 - Insert records by passing parameters
 - Update records by passing parameters
 - Delete records by passing a primary key value
 - All three are data centric approaches
- **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

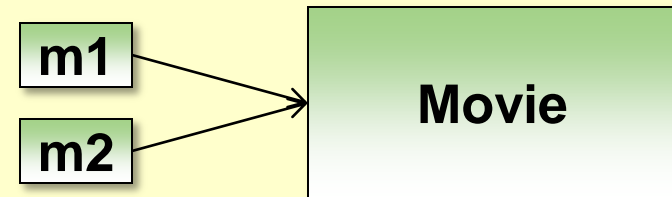
- **How do objects relate to rows in the database?**
 - Rows in a database table have a unique primary key
- **What happens if you query for the same movie twice?**
 - Think about the ADO.NET DataSet / SqlDataReader scenario



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 (MoviesDataContext dc =  
    new MoviesDataContext(connectionString))  
{  
    m1 = dc.Movies.Where(movie => movie.ID == 1).First();  
    m2 = dc.Movies.Where(movie => movie.ID == 1).First();  
}
```



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**
- **LINQ to SQL implements an identity map**
 - Retrieved rows are tracked by primary key value.
 - Asking for a previously retrieved row will return the previous object instance
 - The type of query used to retrieve the row is not important
- **Each DataContext instance maintains it's own Identity Map**
 - Query for the same movie in two different DataContexts will return two different objects.
 - We will talk about a “unit of work” with the DataContext soon ...

Consequences of the Identity Map

- **Any changes from “outside” are not visible to our current DataContext (if we’ve already retrieved a row)**
 - We want consistency and integrity inside our working context
 - The only changes we see are local changes
 - We will revisit concurrency later
- **LINQ to SQL 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*
- **LINQ to SQL DataContext 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
- **DataContext is inexpensive to create**
 - Create as needed
 - Don't cache or create a singleton DataContext
 - DataContext is not thread safe

Entity Lifecycle

- **Object becomes an entity when DataContext 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 DataContext no longer needed**
 - DataContext and object eligible for garbage collection

Updates

- Retrieve an entity from the DataContext
- Update the entity as you would any object instance
- Use **SubmitChanges** to conclude the current unit of work
 - SubmitChanges will “flush” *all* changes to the database

```
using (MoviesDataContext context =  
        new MoviesDataContext(connectionString))  
{  
    Movie movie = context.Movies.Where(m => m.ID == 1).First();  
    movie.ReleaseDate = movie.ReleaseDate.AddDays(1);  
    context.SubmitChanges();  
}
```

Change Tracking

- **DataContext tracks changes for you**
 - DataContext uses list of changes to generate SQL commands
- **How does the DataContext know what changed?**

Change Tracking with POCOs

- **For strict POCOs, LINQ to SQL will take a snapshot of the object when it begins life as an entity.**
 - All original values are copied
 - During SubmitChanges, LINQ to SQL must compare existing values to original values
 - Some expense incurred
 - Turn off this feature by with the DataContext's ObjectTrackingEnabled property

INotifyPropertyChanging

- **INotifyPropertyChanging is an optimization for LINQ to SQL**
 - Does not need a snapshot until a PropertyChanging event fires
 - Implementing this interface and you don't pay for change tracking unless you need it

```
[Column(Name="movie_id", Storage = "_movie_id")]
public int ID
{
    get { return this._movie_id; }
    set {
        if ((this._movie_id != value)) {
            this.SendPropertyChanging();
            this._movie_id = value;
        }
    }
}
private int _movie_id;
```

Updating Associations

- **Changing an object's relationship to other objects in a graph 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

Updating Associations with POCOs

- **Never update a foreign key field manually.**
 - LINQ to SQL will figure this out
- **LINQ can will figure out the updates, inserts, deletes**
 - But its not always obvious how to get there...

```
Movie m1 = context.Movies.Where(m => m.ID == 1).First();
Movie m2 = context.Movies.Where(m => m.ID == 2).First();

m2.Reviews.AddRange(m1.Reviews);
m1.Reviews.Clear();
context.SubmitChanges(); // nothing happens
```

```
Review[] reviews = m1.Reviews.ToArray();
foreach(Review r in reviews) {
    m1.Reviews.Remove(r);
    m2.Reviews.Add(r);
    r.Movie = m2; // must change the parent
}
context.SubmitChanges(); // this works!!
```

Using EntitySet<T>

- **EntitySet<T>** helps manage associations
 - As does generated code ...

```
public Movie() {  
    Action<Review> onAdd = r => r.Movie = this;  
    Action<Review> onRemove = r => r.Movie = null;  
    _reviews = new EntitySet<Review>(onAdd, onRemove);  
}  
  
[Association(ThisKey="ID", OtherKey = "MovieID", Storage="_reviews")]  
public EntitySet<Review> Reviews  
{  
    get { return _reviews; }  
    set { _reviews.Assign(_reviews);}  
}  
  
private EntitySet<Review> _reviews;  
  
// ...
```

```
m2.Reviews.AddRange(m1.Reviews);  
context.SubmitChanges();    // this works!!
```

Inserts

- LINQ to SQL will compute an INSERT statement for all new objects in the graph
- LINQ to SQL can retrieve autogenerated IDs

```
Movie movie = new Movie {  
    Title = "Hairspray",  
    ReleaseDate = new DateTime(2007, 6, 1)  
};  
  
Review myReview = new Review {  
    Rating = 10, Reviewer = "scott",  
    ReviewText = "I want to see it again and again!",  
    Summary = "Fantastic!"  
};  
  
movie.Reviews.Add(myReview);  
context.Movies.InsertOnSubmit(movie);  
context.SubmitChanges();
```

Deletes

- **LINQ to SQL will calculate ordering of command to avoid key violations**
- **Associated entities are not deleted**
 - This behavior is configurable

```
Movie movie = context.Movies.Where(m => m.ID == 1).First();  
  
context.Movies.DeleteOnSubmit(movie);  
context.Reviews.DeleteAllOnSubmit(movie.Reviews);  
context.SubmitChanges();
```

Concurrency Management

- **Optimistic concurrency checks by default**
 - Control in mapping with UpdateCheck: Always, Never, WhenChanged
- **Optimization: use a version column**
 - In mapping: IsVersion = true
 - In SQL: use timestamp or rowversion type

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

```
[Column(Name="version", IsVersion=true,
        IsDbGenerated=true)]
public Binary Version { get; set; }
```

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

Concurrency Violations

- **SubmitChanges will throw an exception**
 - ChangeConflictException
 - DataContext includes ChangeConflict details (original value, submitted value, database value)
- **SubmitChanges is atomic - all changes roll back**
- **DataContext left unchanged**
 - Changes can be resubmitted

Transactions

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

```
using(TransactionScope txn = new TransactionScope())
using (MoviesDataContext context = new MoviesDataContext(...))
{
    Movie movie = context.Movies.Where(m => m.ID == 1).First();
    movie.ReleaseDate = movie.ReleaseDate.AddDays(1);
    context.SubmitChanges();

    txn.Complete();
}
```

Detached Entities

- **Detached entities are entities that “leave” their DataContext**
 - Sent over the wire in a web service call
 - Sent to a client browser for editing
- **Later the entity can be re-attached**
 - But you have to describe how the entity has changed
 - One approach is to query for the current entity in the database then apply changes
 - Entities cannot move between DataContext instances easily

LINQ to SQL Limitations

- **Mapping limitations**

- Inheritance mapping with discriminators only
- No mapping for value types (a domain driven design concept)

- **Platform limitations**

- Currently no support beyond SQL Server

- **Design limitations**

- No bulk inserts or massive database updates (slow)

- **Other issues to know about**

- Will not use default values in database
- Change tracking for detached entities

Summary

- **DataContext is the unit of work for LINQ to SQL**
 - **Maintains a change tracking service**
 - **Maintains an identity map**
- **LINQ to SQL uses optimistic concurrency**
- **DataContext will work with System.Transactions**
- **Think of objects, not database operations**

LINQ To XML

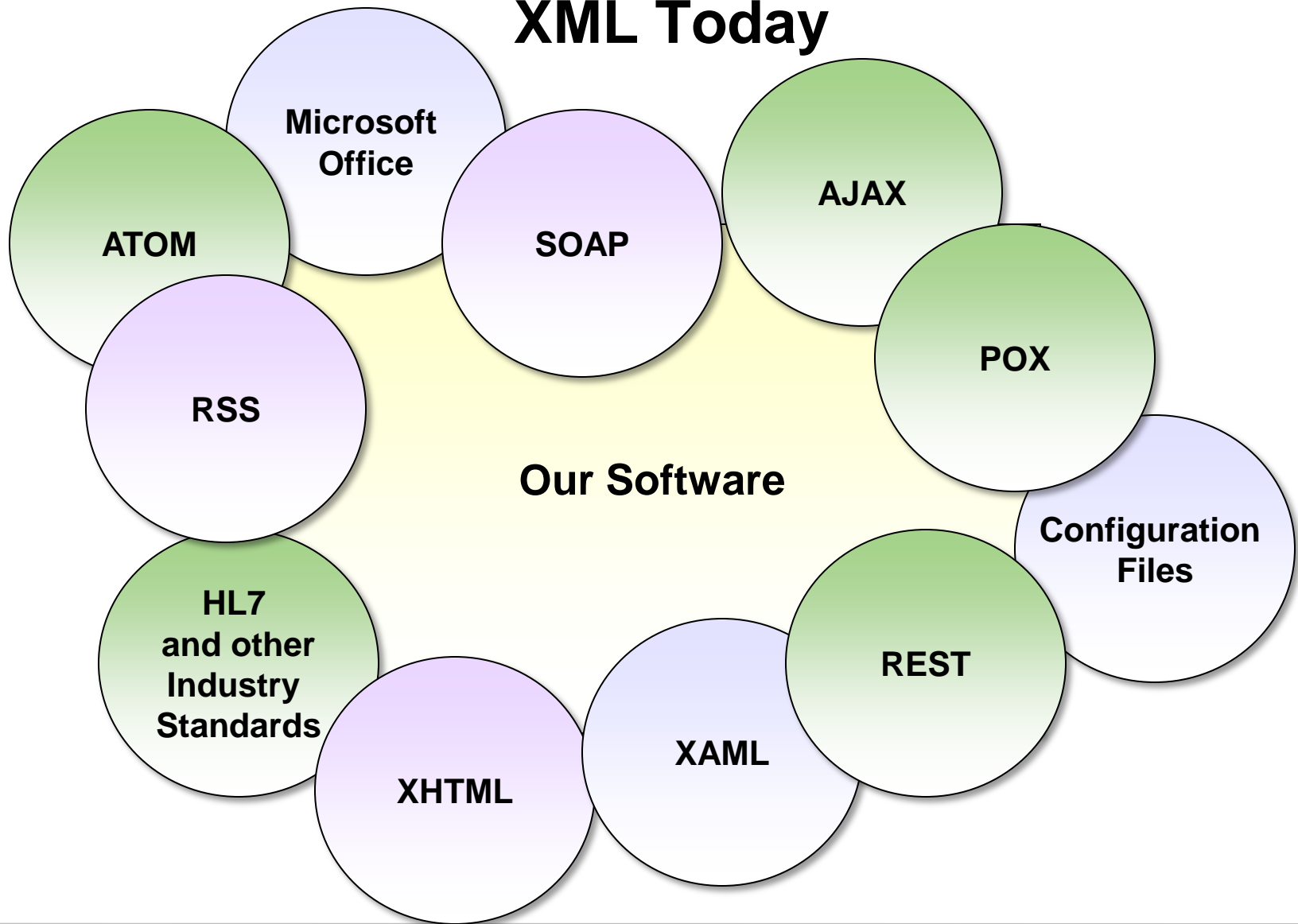
New Paradigms for Angled Brackets



Overview

- **Why another XML API?**
- **Programming XML with LINQ to XML**
 - Loading, creating and updating
- **Querying XML**
 - Query expressions, operators, XPath and XSLT

XML Today



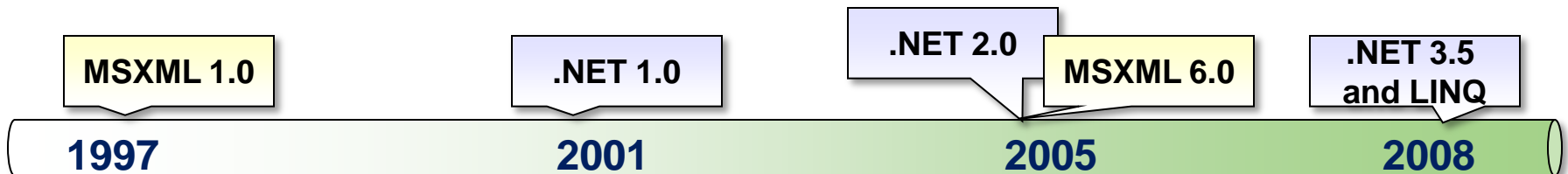
Microsoft XML Support

- **MSXML**

- 40+ releases over 8 years
- XML DOM and streaming SAX APIs
- XSLT, XPath, and XSD and XDR support

- **System.Xml**

- 2 major releases over 5 years
- XML DOM and XmlReader (pull) APIs
- XPathDocument and cursor APIs



We Need Another API?

- **System.Xml is showing some signs of age**
 - Verbose, and lacks new language features
- **Individual technologies require time to learn**
 - XPath for querying, XSLT for transformations

```
XmlDocument document = new XmlDocument();  
XmlElement employees = document.CreateElement("Employees");  
document.AppendChild(employees);
```

```
XmlElement employee = document.CreateElement("Employee");  
employee.InnerText = "Matt";  
employees.AppendChild(employee);
```

```
employee = document.CreateElement("Employee");  
employee.InnerText = "Dan";  
employees.AppendChild(employee);
```

```
<?xml version="1.0"?>  
<Employees>  
  <Employee>Matt</Employee>  
  <Employee>Dan</Employee>  
</Employees>
```

New XML API Goals

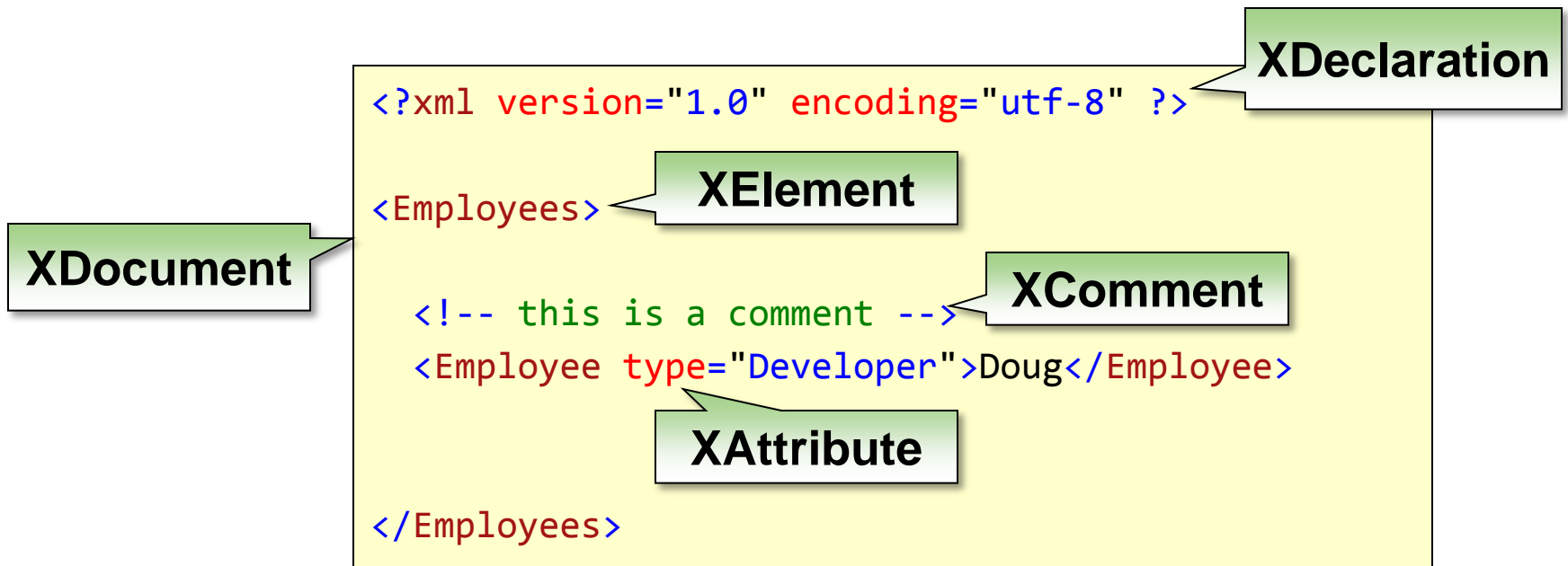
- **Work with Language Integrated Query**
 - Use the standard query operators
 - Add some new operators for XML
- **Provide a modern programming API**
 - Cleaner, faster, lighter
 - Take advantage of generics and nullable types
 - Innovate beyond the DOM APIs

```
XDocument document =  
    new XDocument(  
        new XElement("Employees",  
            new XElement("Employee", "Matt"),  
            new XElement("Employee", "Dan")  
        )  
    );
```

```
<?xml version="1.0"?>  
<Employees>  
    <Employee>Matt</Employee>  
    <Employee>Dan</Employee>  
</Employees>
```


System.Xml.Linq

- The assembly *and* the namespace for LINQ to XML
 - XDocument, XElement, XAttribute, XElement, XObject
 - Extension methods to facilitate queries
- XElement is the heart of the API
 - Provides the ability to load XML from URL, stream, or string



Creating XML

- **Functional construction**

- A params object array used in many constructors
- Conversions tend to “do the right thing”
- Every parameter added to Nodes or Attributes properties

- **Context free**

- No XmlDocument was used in the making of this XML

- **Deep cloning**

```
// functional construction
XElement xml = new XElement("Employees",
    new XComment(" this is a comment "),
    new XElement("Employee", new XAttribute("Type", "Developer"), "Scott"),
    new XElement("Employee", new XAttribute("Type", "Developer"), "Poonam"),
    new XElement("Employee", new XAttribute("Type", "Sales"), "Andy"));
```

Saving XML

- **ToString on any XmlNode yields a string of XML**
 - With line breaks and formatting!
- **Save will write out a declaration by default**
- **TextWriter and XmlWriter support**

```
XElement xml = new XElement("Employees",  
    new XElement("Employee", new XAttribute("type", "Developer"), "Scott"));  
  
xml.Save("employees.xml");  
  
Console.WriteLine(xml.ToString());  
Console.WriteLine(xml.ToString(SaveOptions.DisableFormatting));
```

Loading XML

- Load XML from a file, URL, or XmlReader
- Parse XML from a string
- XmlReader is always behind the scenes

```
XDocument document = XDocument.Load("employees.xml");
XElement element =
    XElement.Load("http://www.pluralsight.com/blogs/rss.aspx");

using (XmlReader reader = document.CreateReader())
{
    if (reader.Read())
        XmlNode node = XmlNode.ReadFrom(reader);
}

XElement inline = XElement.Parse("<Employees/>");
```

Reading XML

- **Explicit conversions perform value extraction**
- **Values stored as text**
 - Parsed on an as-needed basis
- **Support for nullable types**

```
XElement xml = XElement.Parse(  
    "<Employee Type=\"Developer\">Scott</Employee>");  
  
string name = (string)xml; // "Scott"  
string type = (string)xml.Attribute("Type"); // "Developer"  
  
double? salary = (double?)xml.Attribute("Salary"); // yields null  
  
int age = (int)xml.Attribute("Age"); // exception!
```

Namespaces

- **Use XNamespace to encapsulate an XML namespace**
 - Overrides operator + to combine namespace and name
- **Alternative is to place namespace inside { and } delimiters**

```
XNamespace ns = "http://schemas.contonso.com/Employees";

XElement xml =
    new XElement(ns + "Employees",
        new XElement(ns + "Employee", "Aaron"),
        new XElement("{http://schemas.contonso.com/Employees}Employee",
            "Bill"));
```

Prefixes

- **Prefixes in LINQ to XML only significant during output**
 - Can provide prefix hints

```
XNamespace xmlns = "http://schemas.contonso.com/Employees";  
XNamespace ext = "http://schemas.contoso.com/EmployeeExtensions";  
  
XElement xml = new XElement(xmlns + "Employees",  
    new XAttribute(XNamespace.Xmlns + "ns1", ext),  
    new XElement(ext + "Employee", "Aaron"),  
    new XElement(ext + "Employee", "Bill"));
```

```
<Employees  
  xmlns:ns1="http://schemas.contoso.com/EmployeeExtensions"  
  xmlns="http://schemas.contonso.com/Employees">  
  <ns1:Employee>Aaron</ns1:Employee>  
  <ns1:Employee>Bill</ns1:Employee>  
</Employees>
```

Traversal

- **Properties:** NextNode, PreviousNode, Document, and Parent
- **Methods:** Elements, Ancestors, Descendants

```
XDocument document = XDocument.Load("employees.xml");
foreach (XElement element in document.Descendants("Employee"))
{
    Console.WriteLine((string)element);
    foreach (XAttribute attribute in element.Attributes())
    {
        Console.WriteLine("\t{0}:{1}", attribute.Name, (string)attribute);
    }
}
```


Modification

- **Add variations**
 - Add, AddFirst, AddBeforeSelf, AddAfterSelf
- **Remove variations**
 - Remove, RemoveAll, ReplaceAll, RemoveContent
- **For individual nodes there is a Value property**
 - Also a powerful SetElementValue method

```
XDocument document = XDocument.Load("employees.xml");
foreach (XElement element in document.Descendants("Employee"))
{
    if ((string)element == "Scott") {
        element.Value = "K. Scott";
        element.Attribute("Type").Value = "Sales";
        XElement newElement = new XElement("Employee", "Joy",
                                             new XAttribute("Type", "Executive"));
        element.AddAfterSelf(newElement);
    }
}
```

Standard Query Operators

- **Methods like Nodes and Elements return IEnumerable<T>**
- **All the standard query operators are in play**
 - Where, Select, Join, OrderBy, etc.

```
XDocument document =  
XDocument.Load("employees.xml");  
  
var developers =  
    from e in document.Descendants("Employee")  
    where e.Attribute("Type").Value == "Developer"  
    orderby e.Value  
    select e.Value;  
  
foreach (var developer in developers)  
{  
    Console.WriteLine(developer);  
}
```

LINQ to XML Extensions

- **System.Xml.Linq namespace**
 - Query extensions
 - Work for IEnumerable<XNode> and IEnumerable<XElement>
- **System.Xml.Schema namespace**
 - Schema validation extensions
 - For XDocument
- **System.Xml.XPath**
 - XPath processing extensions
 - For any XNode

Query Extensions

Method	Description
Ancestors / AncestorsAndSelf	Return ancestors of every node or element in the source collection
Attributes	Returns attributes from every element in the source collection
Descendants / DescendantsAndSelf	Return descendants of every node or element in the source collection
Elements	Returns collection of child elements
Nodes	Returns child nodes from every node in the source collection
InDocumentOrder	Sort a collection of nodes into document order
Remove	Remove every node in the source collection from its parent

Removing Nodes

- LINQ's deferred execution is a tremendous help

```
XDocument document = XDocument.Load("employeedetail.xml");  
foreach (var phoneElement in document.Descendants("Phone"))  
{  
    if (phoneElement.Attribute("Type") == null)  
    {  
        phoneElement.Remove();  
        // exception will be waiting for you ...  
    }  
}
```



```
(from phone in document.Descendants("Phone")  
 where phone.Attribute("Type") == null  
 select phone).Remove();
```



Constructing XML Redux

- Combine query syntax and standard operators with functional construction

```
XDocument document = new XDocument(  
    new XElement("Processes",  
        from p in Process.GetProcesses()  
        where p.ProcessName == "devenv"  
        select new XElement("Process",  
            new XAttribute("Name", p.ProcessName),  
            new XElement("Modules",  
                from m in p.Modules.Cast<ProcessModule>()  
                select new XElement("Module", m.FileName)))));
```

```
<Processes>  
  <Process Name="devenv">  
    <Modules>  
      <Module>C:\Windows\system32\ntdll.dll</Module>  
      <Module>C:\Windows\system32\kernel32.dll</Module>  
      <!-- ... -->
```

Transformation

- LINQ to XML will generally be faster than using XSLT

```
XDocument document = XDocument.Load("employees.xml");
XDocument transformed = new XDocument(
    new XElement("Employees",
        new XElement("Developers",
            from e in document.Descendants("Employee")
            where e.Attribute("Type").Value == "Developer"
            select new XElement("Employee", e.Value)),
        new XElement("Sales",
            from s in document.Descendants("Employee")
            where s.Attribute("Type").Value == "Sales"
            select new XElement("Employee", s.Value))));
```

Validation

- **Use the System.Xml.Schema namespace**
 - Validate extensions available for XDocument, XElement, XAttribute

```
XDocument employees = XDocument.Load("employees.xml");

XmlSchemaSet schemaSet = new XmlSchemaSet();
schemaSet.Add(null, "employees.xsd");
employees.Validate(schemaSet, (s, e) => Console.WriteLine(e.Message));

employees.Element("Employees").Add(new XElement("Foo"));
employees.Validate(schemaSet, (s, e) => Console.WriteLine(e.Message));
```


XPath Extensions

Method	Description
CreateNavigator	Creates an XPathNavigator for an XmlNode
XPathEvaluate	Evaluates an XPath expression
XPathSelectElement / XPathSelectElements	Select elements using an XPath expression

Summary

- **LINQ to XML offers a new XML API**
 - Create, load, save, modify, and query XML fragments and documents
 - Combine functional construction with LINQ queries
- **But LINQ remains the same**
 - Same standard query operators as LINQ to Objects
 - Same query comprehension syntax

References

- **.NET Language Integrated Query for XML Data**
<http://msdn2.microsoft.com/en-us/library/bb308960.aspx>
- **Paste XML As LINQ**
<http://msdn2.microsoft.com/en-us/library/bb397977.aspx>