Entity Framework 6

Microsoft's Object Relational Mapper

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Agenda

- What is an ORM
- Mapping relational data into objects
- LINQ for queries
- Change tracking and persistence
- Loading strategies
- Stored Procedures

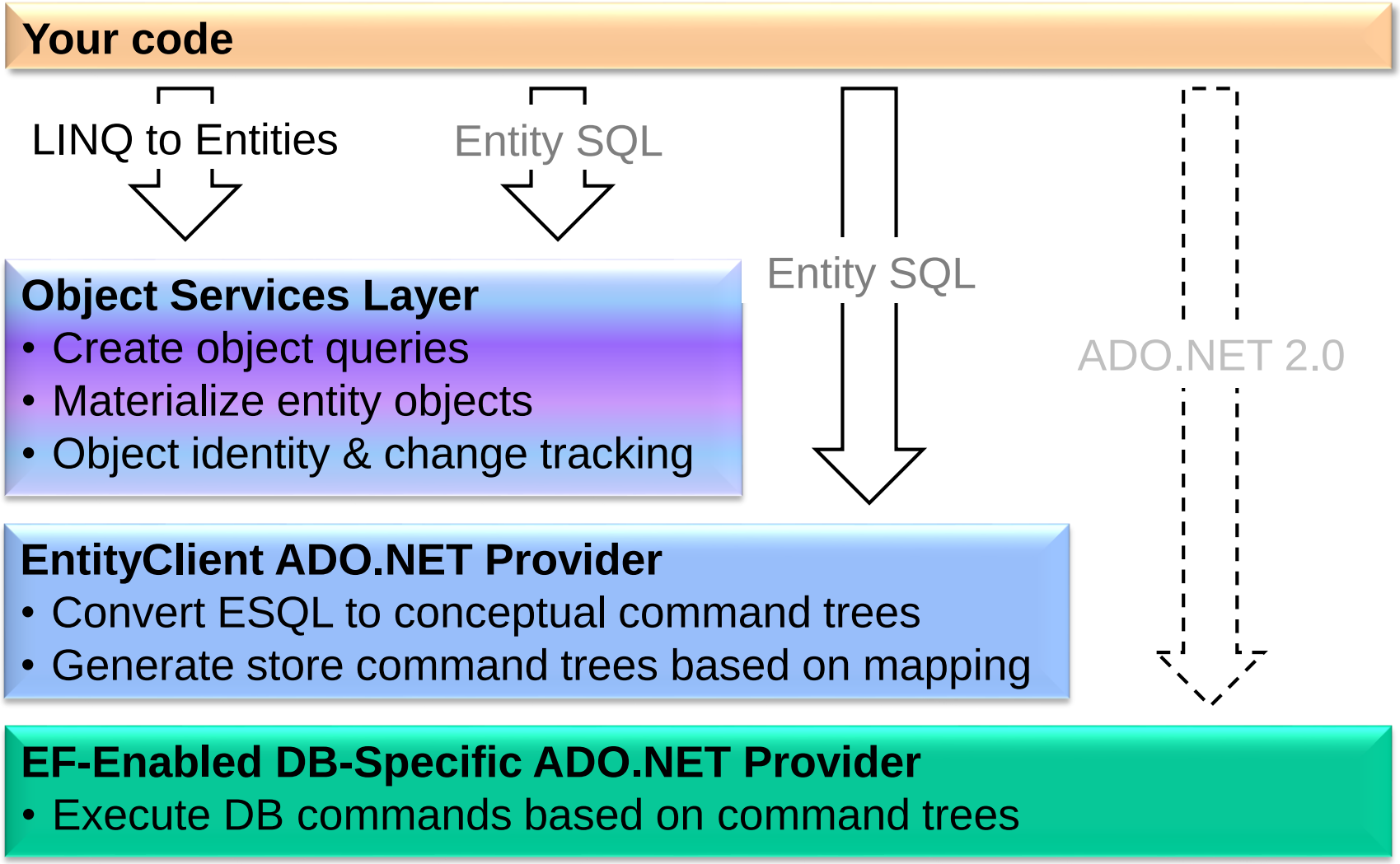
Object Relational Mapper (ORM)

- **The age old problem**
  - Developers want to consume objects
  - Data resides in non object like storage (tables , XML documents)
  - How to move information between table form and object form
- **Writing data access layers by hand is tedious**
- **ORM's automate the process**
  - Database schema and Object schema defined independently
  - Mapping defined between the two worlds
- **ORM's on the .NET Platform**
  - Entity Framework
  - Nhibernate
  - Many more…

Entity Framework

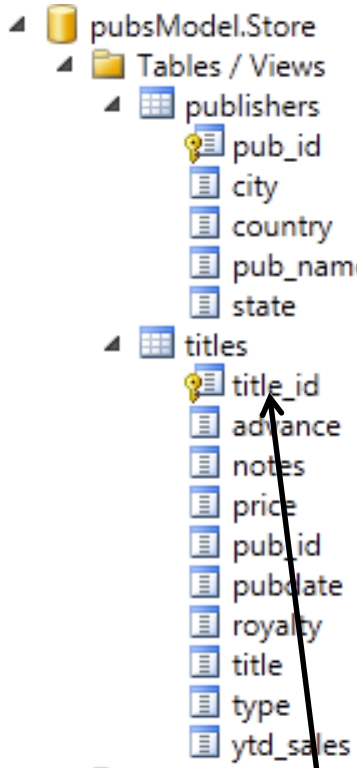
- **Conceptual model (CSDL)**
  - Defines Entity Sets
    - Entities will become .NET classes
- **Storage model (SSDL)**
  - Defines tables, views , sprocs in database
- **Mapping defined between two worlds.**
- **Application classes are not coupled to database structure.**
- **Database schema can change without breaking conceptual model**

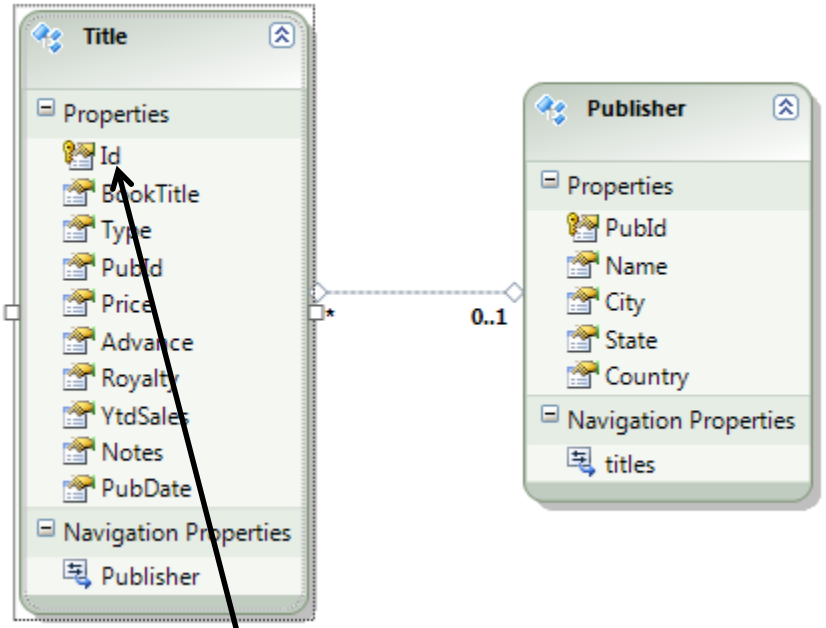
Architectural overview

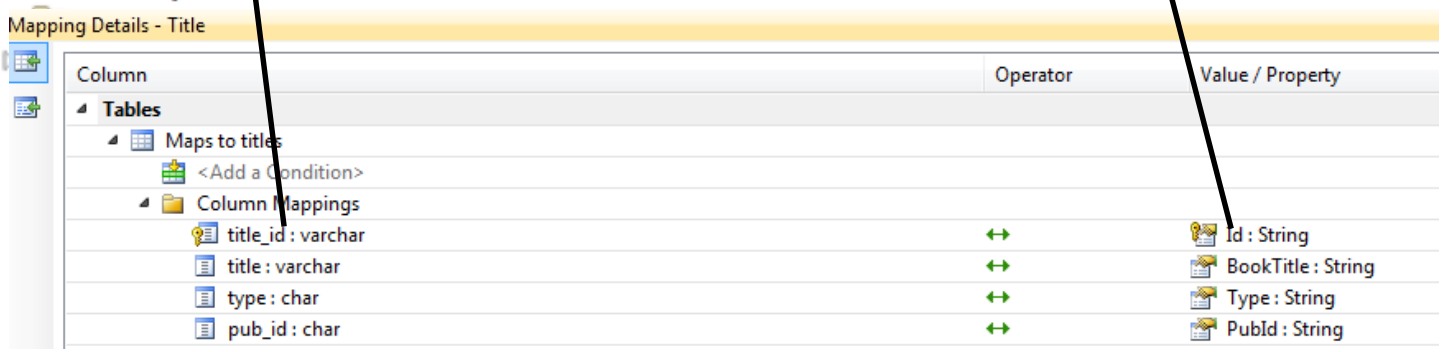


EF Designer

Conceptual Model







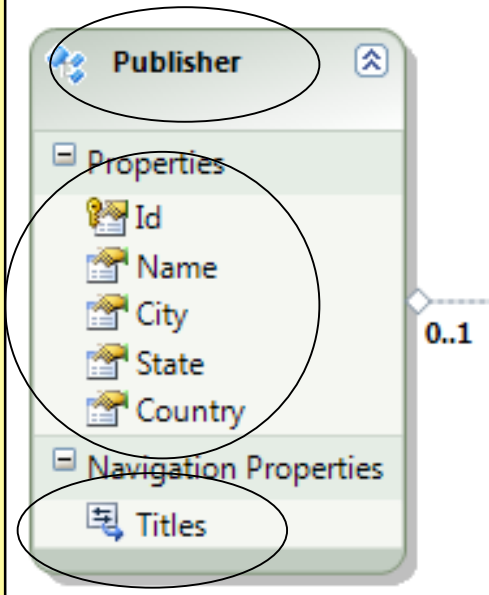
Conceptual model to code

- **Each entity type has an associated class**
  - Built in code generator, translates conceptual model into Entity classes
    - Classes derive from EntityObject
    - Application classes tightly coupled to Entity Framework.
  - Write classes by hand POCO (**P**lain **O**ld **C**LR **O**bject)
    - Classes just need to be same shape as Conceptual model
    - Not coupled to Entity Framework
    - Build your own or leverage code generation via T4 templates
    - This is considered best practice, and the default in EF5
- **Each conceptual model property maps to a class property**

POCO, Publisher

- POCO class needs to look like conceptual model.
- Can be auto generated via T4 templates

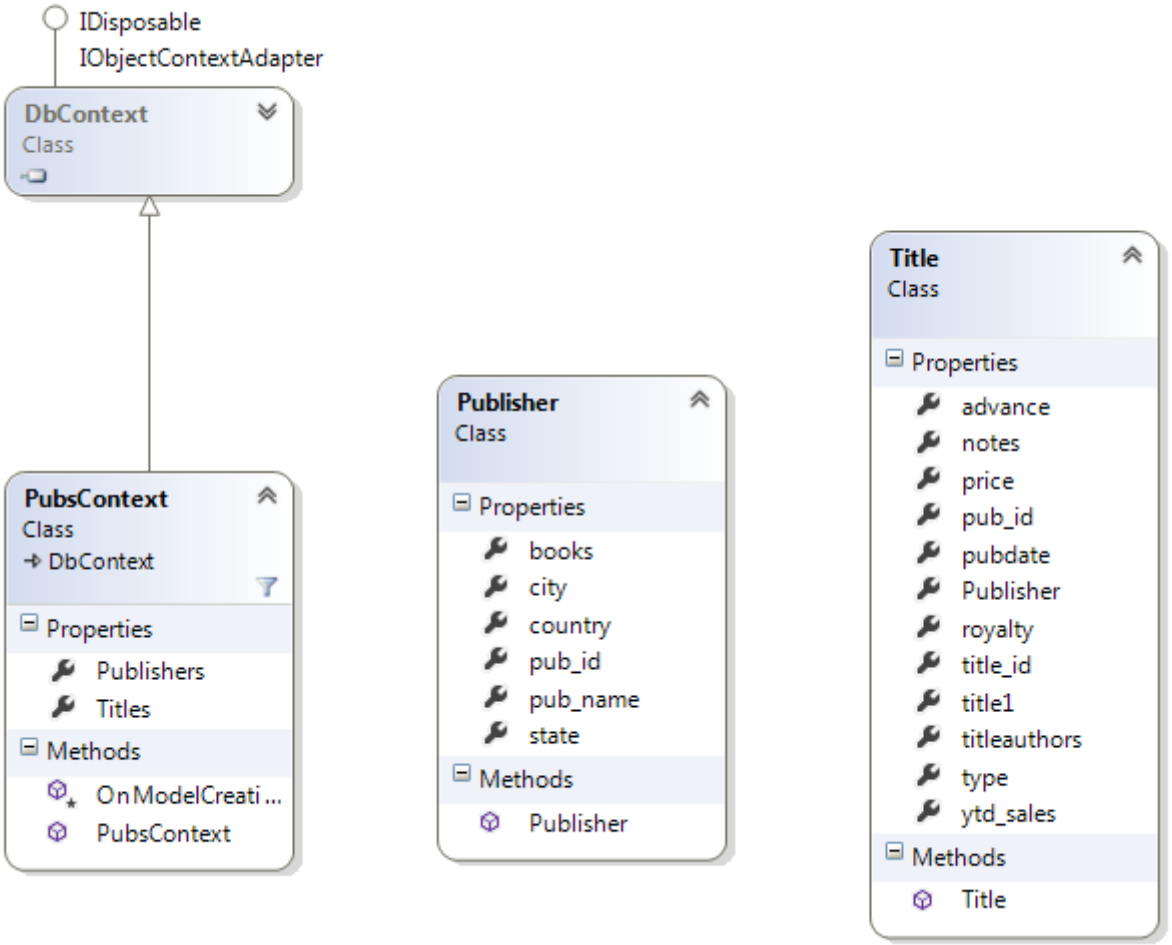
```
 public class Publisher
{
  public Publisher()
  {
    Titles = new List<Title>();
  }

  public string Id { get; private set; }
  public string Name { get; set; }
  public string City { get; set; }
  public string State { get; set; }
  public string Country { get; set; }


  public ICollection<Title> Titles {get;private set;}
}
```



Conceptual model to code

- **DbContext Class**
  - Provides initial access to entities (entity set)
  - Provides support for change tracking
    - Unit of Work pattern.
  - Generated derived class exposing strongly typed entity sets as properties.
- **DbSets**
  - Exposed by the DbContext, represents all instances of a given entity type in the physical storage.
  - Implements IQueryable<T> interface to allow the construction of LINQ queries.
  - Provides Add and Delete methods to remove entities from the set.

Class Diagram



Fetching entities

DbContext

DbSet

```
 using (PubsEntities ctx = new PubsEntities())
{
   foreach (Title title in ctx.Titles)
   {
    Console.WriteLine("{0} Costing {1:C}" , title.BookTitle ,
                title.Price);
   }
}
```

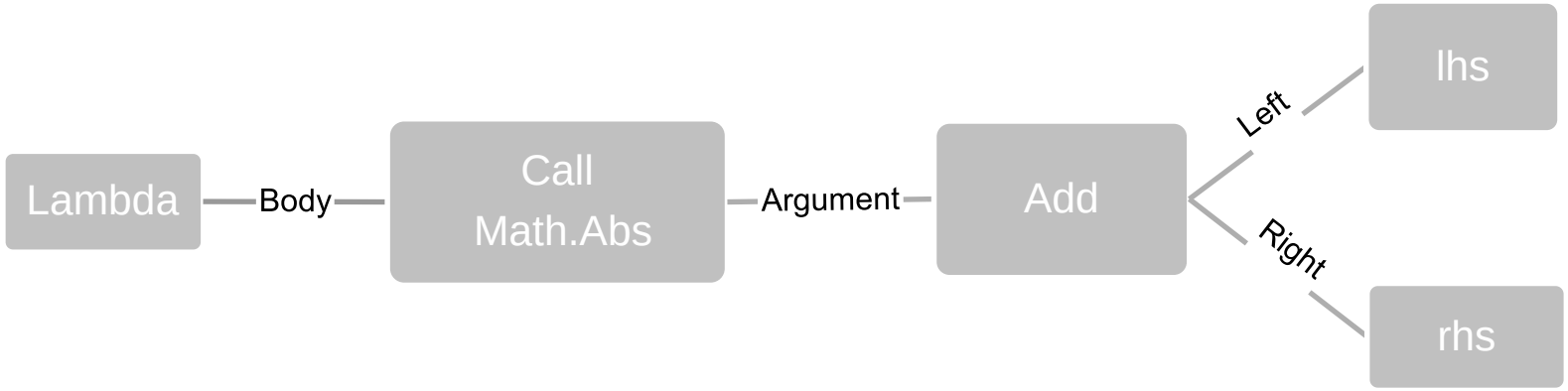
```
using (PubsEntities ctx = new PubsEntities())
{
   var expensiveBooks = (from book in ctx.Titles
                         where book.Price > 20.0m
                         select new { book.BookTitle,
                                      book.Price
                                    }
                        ).ToList();
 }
```

Expressions

- **C# allows Lambda expressions to be compiled into**
  - IL
  - Or Expression Trees
- **Expression trees**
  - Capture programmer intent
  - Can be built from lambda expressions or dynamically
  - Can be compiled into
    - IL
    - . . .

Expressions to IL

```
Expression<Func<int, int, int>>
   mathProcExpr =  (lhs, rhs) => Math.Abs(lhs + rhs);
```




```
Func<int,int,int> mathProc = mathProcExpr.Compile();
int result = mathProc(3, -5);
```

Query on the wire

- **DbSet implements IQueryable**
  - Enumerable extension methods take compiled lambdas
  - Queryable extension methods take Expressions
- **LINQ query is turned into an Expression tree**
- **Query Provider responsible for executing the query**
  - Compiles Expression tree into SQL commands when query is evaluated.

```
SELECT
1 AS [C1],
[Extent1].[title] AS [title],
[Extent1].[price] AS [price]
FROM [dbo].[titles] AS [Extent1]
WHERE [Extent1].[price] > cast(20 as
decimal(18))
```

Linq to Entities Gotcha

- **The Query Provider has to compile expression tree into SQL**
- **Query Provider doesn't have a mapping for IsExpensiveBook**
- **Does have mapping for some framework methods**
  - E.g. String.Contains, Queryable.Sum , Queryable.Min
  - See documentation on "Entity Framework Canonical Functions" for full list

```
var expensiveBooks = (from book in ctx.Titles
                          where IsExpensiveBook(book)
                          select new { book.BookTitle,
                                       book.Price
                                     }
                      ).ToList();

 public bool IsExpensiveBook( Title book)
  { return book.Price > 20m }
```

Updating entities

- **Entity Objects are self tracking**
  - Report any changes to the DbContext.
- **Changes are persisted to database**
  - On call to SaveChanges is made.
- **Each update results in a round trip.**

```
using (PubsEntities ctx = new PubsEntities())
{
 foreach (Title book in ctx.Titles)
 {
   book.Price *= 1.10m;
 }
 ctx.SaveChanges();
}
```

POCO Change tracking

- **Two Ways**
  - DbContext snapshots fetched objects
    - Diffs current objects against snapshot
  - DbSet serves up proxy not real object
    - Proxy tracks changes
- **Both have pros and cons**
  - Proxy
    - More expensive to create
    - Real time update of references
    - POCO with rules, virtual members
  - Non Proxy
    - 100% POCO
    - Pay cost at save time
    - Creation quick
    - Updates references when calling EF APIs
      - SaveChanges,Add,Remove,

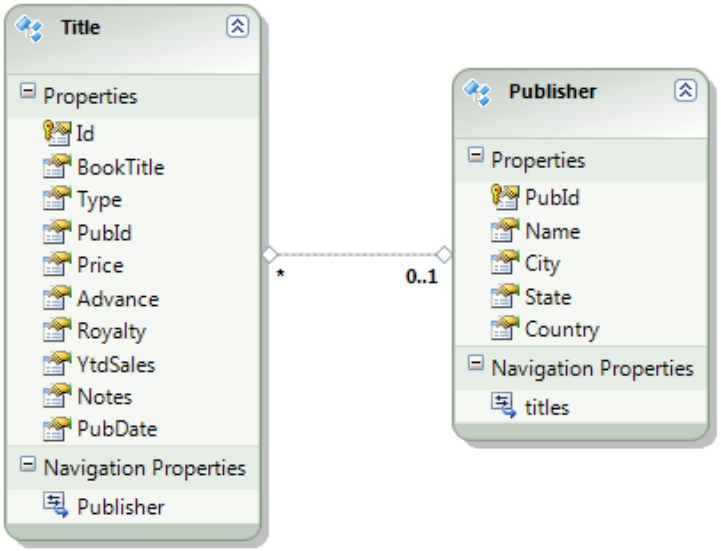
POCO Proxy Change tracking

- **Rules**
  - A custom class
    - must be declared with public access
    - must not be sealed
    - must not be abstract
    - must have a public or protected constructor, with no parameters
    - properties must be marked virtual
- Consider writing unit test to confirm type maintains change tracking behaviour

```
bool isUpdateProxy = ctx.Set<Title>().Create() is
                                        IEntityWithChangeTracker;
```

Navigation

- **Entities often have relationship to other entities**
- **Related entities can be reached via navigation properties.**
- **Related entities can be loaded**
  - Eagar, at the same time that as root entity
  - Lazy, when the first navigation takes place
  - Explicit, manually load the related entity



Lazy Loading

- Lazy loading flag set to true for the context.
- It is typically the default value, set via the Designer.

```
 using (PubsEntities ctx = new PubsEntities())
{
   ctx.Configuration.LazyLoadingEnabled = true;
   foreach (Title book in ctx.Titles)
   {
     Console.WriteLine(book.Publisher.Name);
   }
}
```

Explicit Loading

- Take complete control of when data is loaded.
- Can help to focus developers on the true cost of the navigation.
- Use Collection, as opposed to Reference for child collections

```
using (PubsEntities ctx = new PubsEntities())
{
  ctx.Configuration.LazyLoadingEnabled = false;
  foreach (Title book in ctx.Titles)
  {
    ctx.Entry(book)
        .Reference( b => b.Publisher)
        .Load();

    Console.WriteLine(book.Publisher.Name);
   }
}
```

Eager loading

- **Load all required data using a SQL single roundtrip.**
- **Decorate the Query with Include**
  - Takes a string denoting the additional entity to load
  - Use “dot” notation to dive deeper.
    - `ctx.Orders.Include(“OrderItems.Product”)`

```
 using (PubsEntities ctx = new PubsEntities())
 {
  foreach (Title book in ctx.Titles.Include("Publisher") )
  {
    Console.WriteLine(book.Publisher.Name);
  }
}
```

Strongly typed include

- **String based include cumbersome**
  - EF 5 introduces lambda based includes, via extension methods in System.Data.Entity.DbExtensions

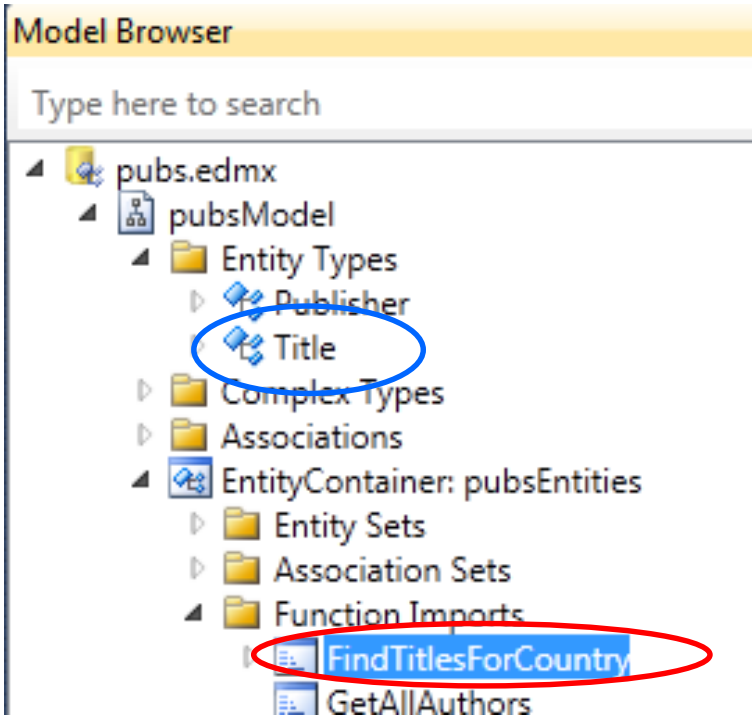
```
using System.Data.Entity;
using (PubsEntities ctx = new PubsEntities())
 {
  foreach (Title book in ctx.Titles.Include(t=>t.Publisher)
  {
    Console.WriteLine(book.Publisher.Name);
  }
}
```

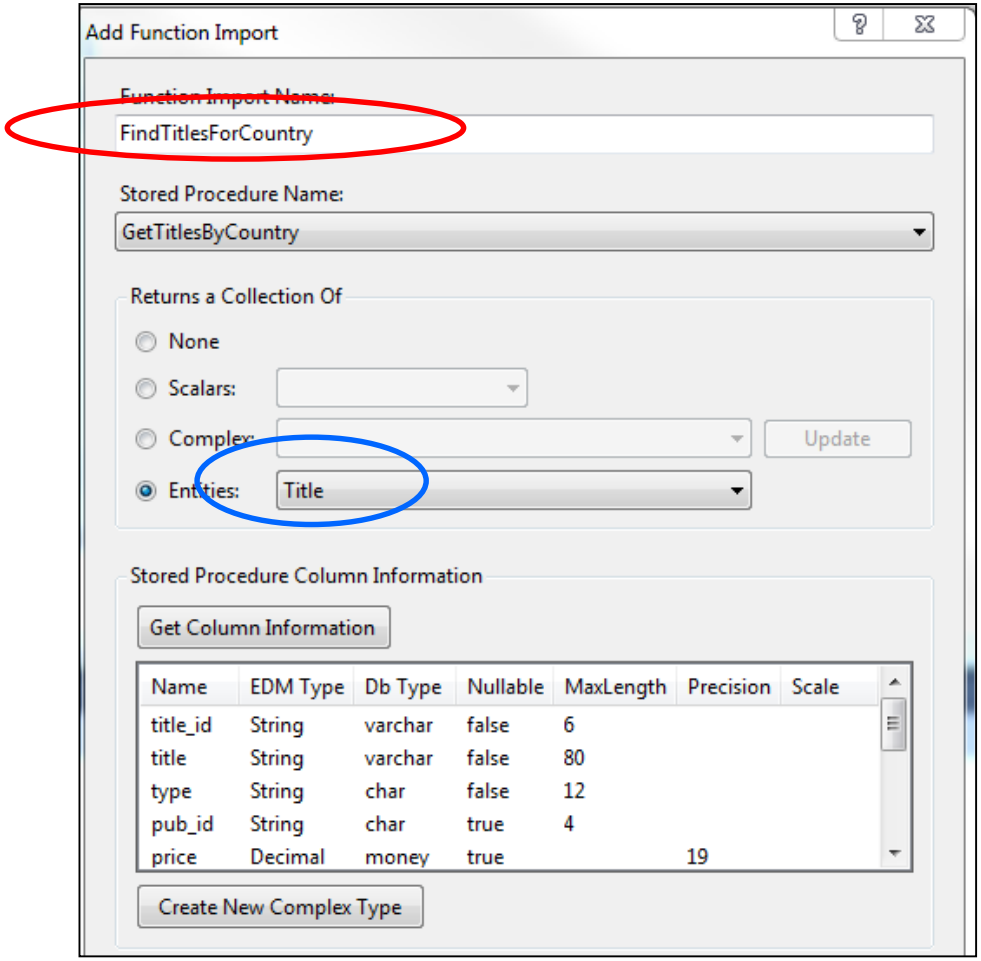
```
using (PubsContext ctx = new PubsContext()){
   var publishers = ctx.Publishers
                     .Include(p => p.titles)
                     .Include(p => p.titles
                               .Select(t => t.titleauthors
                                .Select(ta => ta.author)));
}
```

Stored Procedures

- **Some operations best supported inside the database**
  - Increase all prices by 10%
- **Direct table access restricted**
- **Entity framework allows**
  - Invocation of SPROCS
  - Table mappings to use SPROCS for CUD operations
- **Steps**
  - Import Stored Procedure into Storage Model
    - Update Model from Database
  - Add Mapping to Stored Procedure in Conceptual Model
    - Add Function Import
- **Execute stored procedure via DbContext**

Procedures returning entities





Executing stored procedure

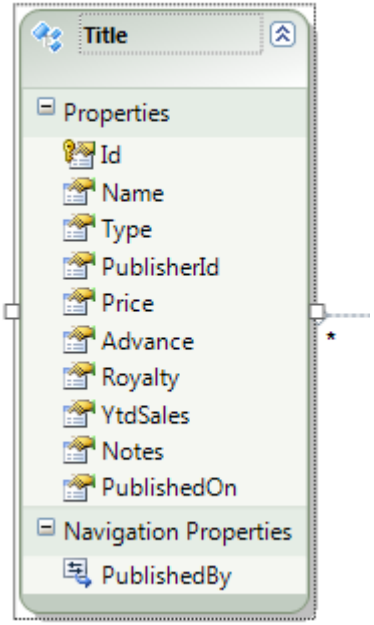
- **Function binding via string**
  - Code gen creates strongly typed method on DbContext

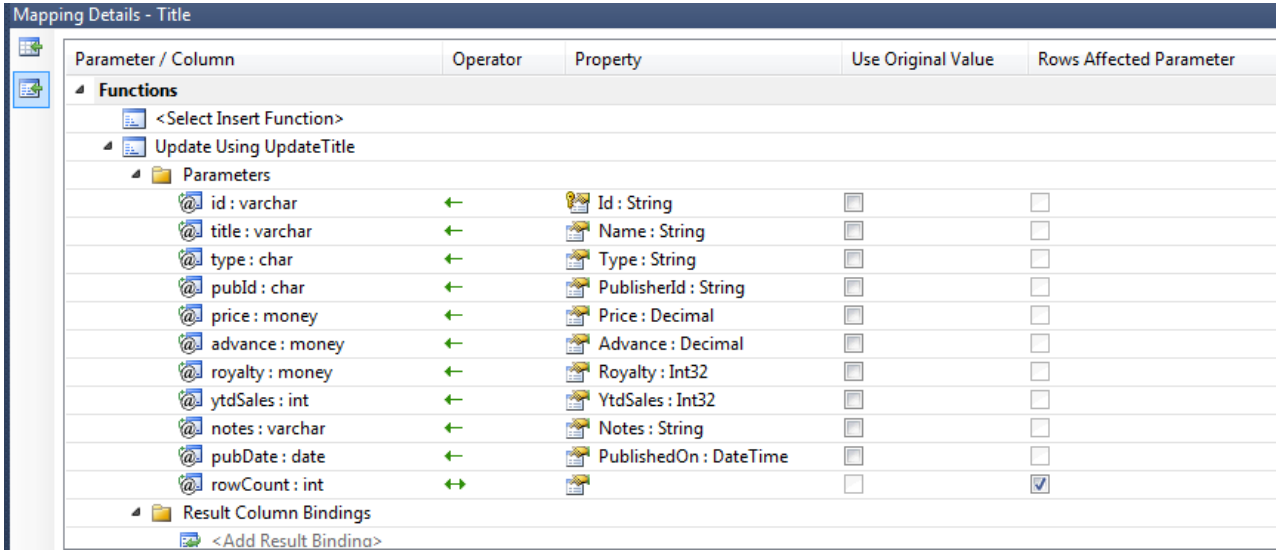
```
public ObjectResult<Title> FindTitlesForCountry( string country )
{
  ExecuteFunction<Title>(“FindTitlesForCountry",
                         new ObjectParameter("country", country));
}
```

```
CREATE PROCEDURE dbo.GetTitlesByCountry
(@country varchar(30)) AS

select * from titles
join publishers on publishers.pub_id = titles.pub_id
where publishers.Country = @country
and titles.pub_id = publishers.pub_id

RETURN
```

Create,Update,Delete via Stored Procs

- **Import Stored Procedures into storage model**
- **Use Stored Procedure mappings instead of Table Mappings**
- **Map stored procedure parameters to properties**





Updated Stored Proc

- **Update method returns row count used to determine if update succeeded.**

```
CREATE PROCEDURE dbo.UpdateTitle
(
  @id tid,@title varchar(80),@type char(12),@pubId char(4),@price money,
  @advance money,@royalty money,@ytdSales int,@notes varchar(200),
  @pubDate date,
 @rowCount int output
)AS

UPDATE titles
SET title = @title,
    price = @price
    . . .
where title_id  = @id

set @rowCount = @@ROWCOUNT
```

Changes from 4.x to 5

- **Prefer DbContext,DbSet to ObjectContext,ObjectSet**
- **Improved POCO support**
  - Auto fixup of navigational properties
- **Enumeration support**
- **Local key based queries (Find)**
- **Table value function support**
- **Spatial data types**
  - DbGeography and DbGeometry
- **Multiple diagrams**
- **Batch import of sprocs**
  - Creates complex return types

Changes from 5 to 6

- **Async query (More on this later)**
  - ForEachAsync
  - ToListAsync,ToDictionaryAsync,ToArrayAsync
  - Many more …
- **Connection Resiliency**
  - Will retry commands if database connection is lost
- **Interception and custom Logging**
  - DbContext.Database.Log
  - IDbCommandInterceptor
- **Performance improvements**
  - Enumerable.Contains, Add/Remove Range

Summary

- **Don’t write data access code**
  - Its tedious, and you have more interesting things to do ;-)
- **Entity Framework 5**
  - Allows Developers to work with objects not tables
  - Isolates developers from changes to schema
- **Can use stored procedures**
  - But don’t leak business logic into sprocs, keep it in one place.
- **Use a SQL profiler to validate the SQL that’s produced**