Entity Framework @ 20,000 Feet

Rob Vettor
robvettor@hotmail.com

Goals

- Take 20,000 ft pass over EF
- What it is
- What it does
- Its value proposition
- Kick the tires and test drive some examples
- Point out learning resources for self study

Who is Rob?

- Architect/Team Lead → jack henry & ASSOCIATES INC.
- Founder of the Dallas .NET New Technology Group
- Member of Microsoft Developer Guidance Council
- Co-Founder of Presenter-Mentor
- For better or worse, *New Microsoft technology* is my *career*, *passion* and, lately, a *spiritual pursuit*

THE UNIVERSITY OF TEXAS MD ANDERSON CANCER CENTER Making Cancer History®

Raytheon Customer Success Is Our Mission

avanade

Hilton

Insight

DAVACO RETAIL SERVICES

verizon wireless

AMERICAN EXPRESS

CBRE CB RICHARD ELLIS

COMPUWARE

Spot Survey

- How many use EF in production app?
- How many have looked at EF?
- How many have looked at LINQ?
- How many use Visual Studio 2008?
- How many use Visual Studio 2008 and not LINQ?

In a Nutshell…

ADO.NET Entity Framework is a *core technology* in Microsoft's *evolving* data platform that helps *bridge the gap* between data structures and objects in your applications.

LINQ Perspective

C# 3.0

VB 9.0

Others…

.NET Language Integrated Query

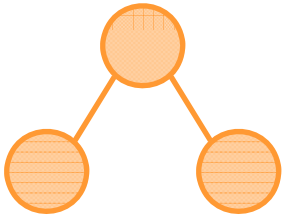
LINQ to Objects

LINQ to Datasets

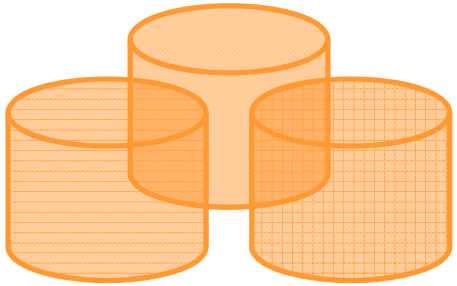
LINQ to SQL

LINQ to Entities

LINQ to XML



Objects



Relational

XML

Agenda

- ***The Problem***
- The EF and EDM
- Mapping a Conceptual Model
- Programming a Conceptual Model
- Key Concepts
- EF vs. L2S
- Advanced Mapping
- How to get started

The Problem

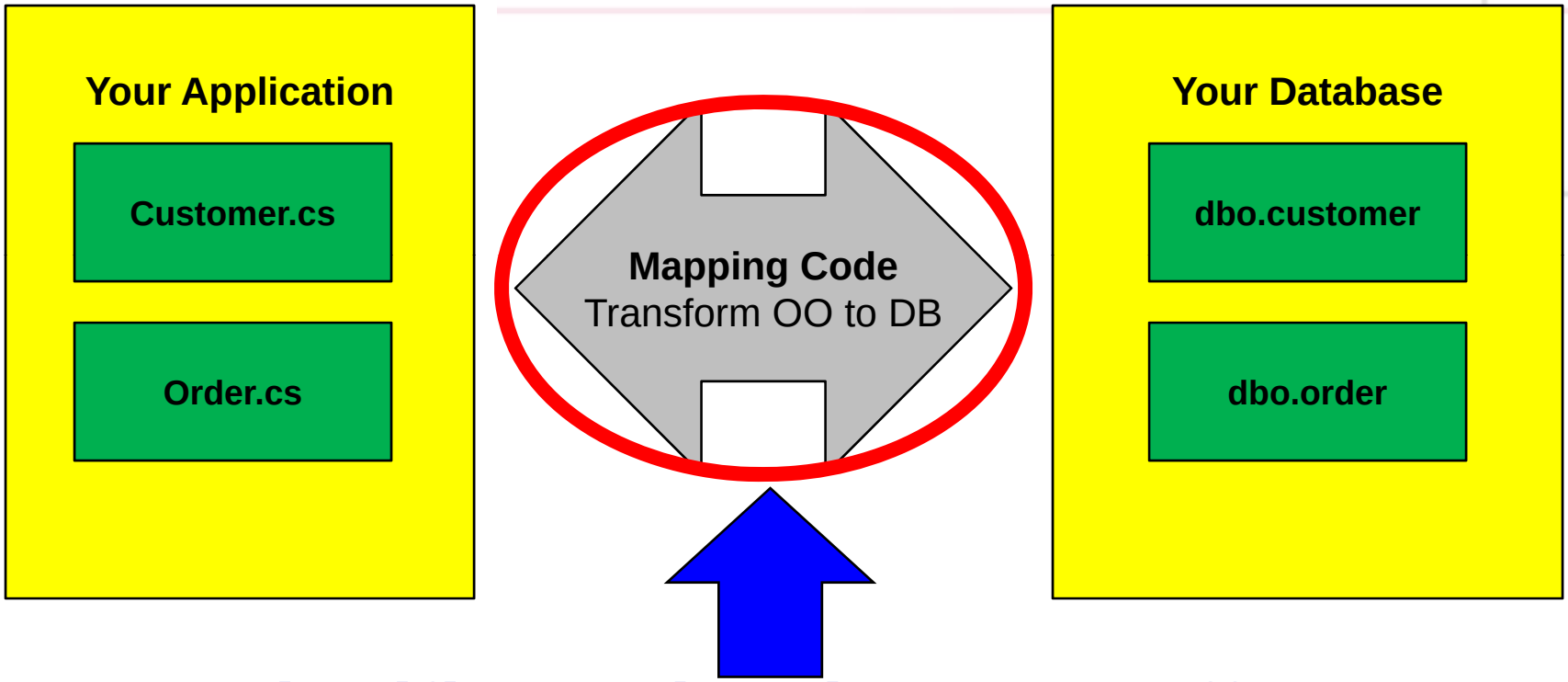
Objects != Relational Data

- OO Programming been around for decades
- Relational databases even longer
- Bridging the gap between these two has been time consuming and expensive:
  - Legacy ADO.NET
  - Hand-built DALs
  - Wide assortment of 3rd party solutions
- But, the core problem remains!
- Relational data and objects aren't the same

This is Your Life

- You build applications for a living
- Each app talks to a relational Database
- Learn (*at least*) 2 different languages (C# and SQL)
  - Different syntax
  - Different type systems
  - Different tools
  - Different paradigms: Object vs. Procedural
- On top of that, must learn the *API* that *binds* these worlds together: ADO.NET
  - Powerful, but fragile and time-consuming

Your World Now…



Significant development effort

invested *building plumbing* to move data back
And forth from data store to domain objects.

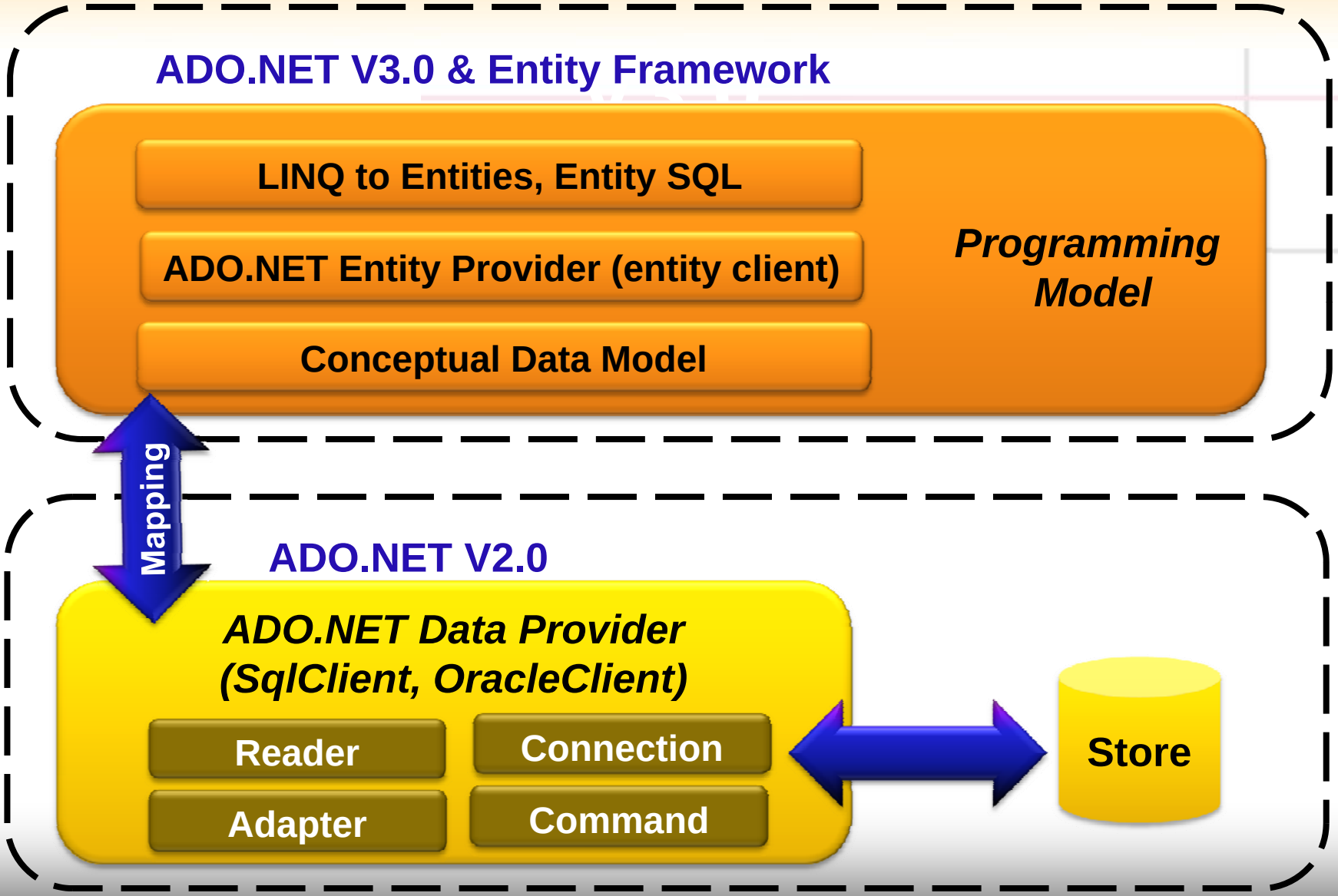
Agenda

- The Problem
- *The EF and EDM*
- Mapping a Conceptual Model
- Programming a Conceptual Model
- Key Concepts
- EF vs. L2S
- Advanced Mapping
- How to get started

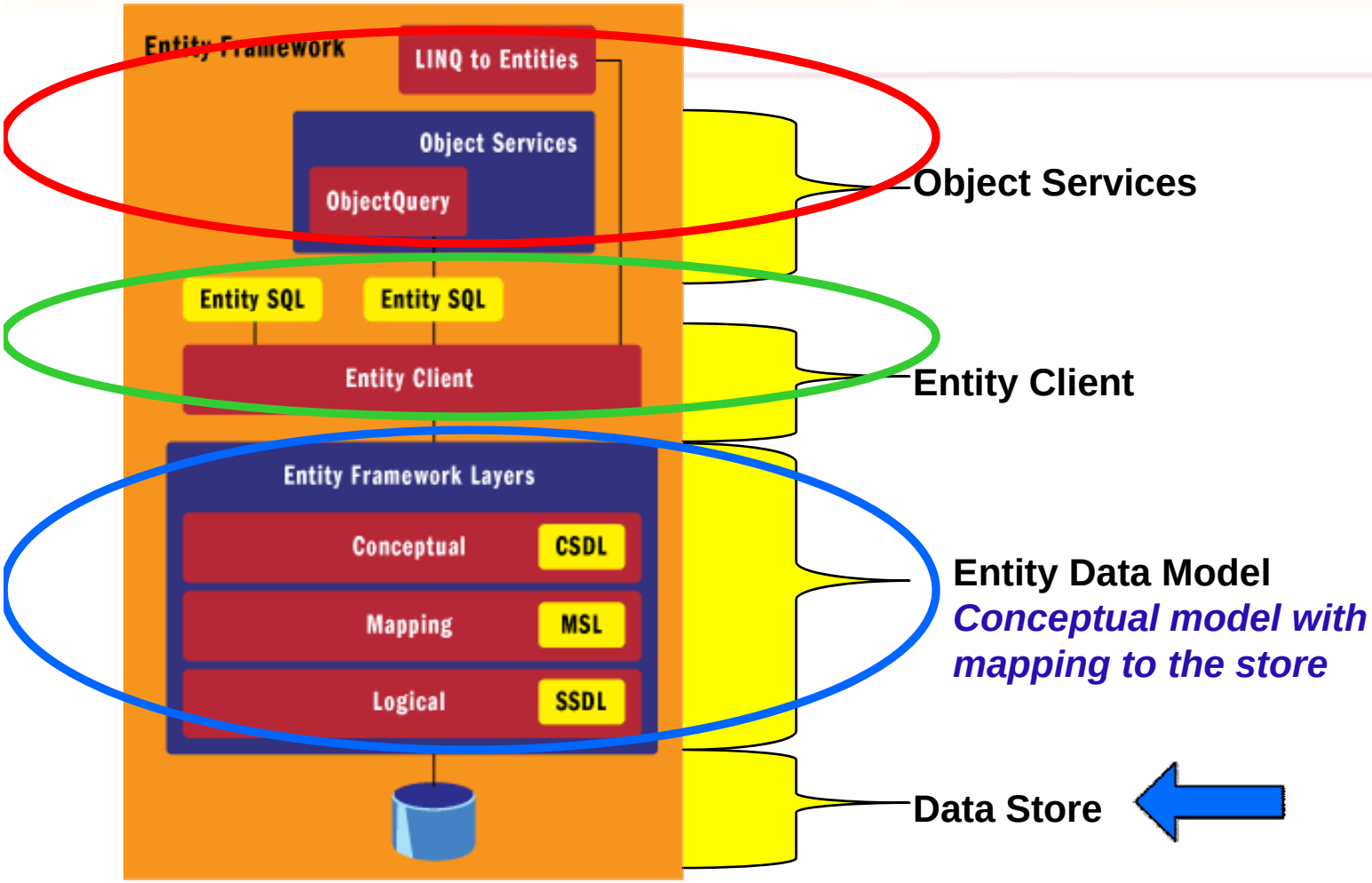
What is the Entity Framework (EF)?

- New *data access framework* from Microsoft
  - Released in 7/08 in .NET 3.5 SP 1
  - Microsoft eagerly investing $$$ and resources
- Develop against *conceptual view, not* the data store
  - Less normalized, more business friendly domain model
  - Generates strongly-typed entity objects
  - Generates mapping/plumbing code
  - Translates LINQ queries to database queries
  - Materializes objects from data store calls
  - Enables customized mapping scenarios, beyond one-to-one
  - Visual Modeling tools
  - Automatic change tracking

EF High-Level Architecture



Components of Entity Framework



What Is The Entity Data Model (EDM)?

- *Bridge* between application and data store
- *Consists of three meta data layers*:

→Business domain objects

→ *Maps* conceptual to relational

→ Database schema

- *Abstracts* developer from a *model* pleasing to a DBA (normalized, maintainable, efficient, secure), but complicated to program against.

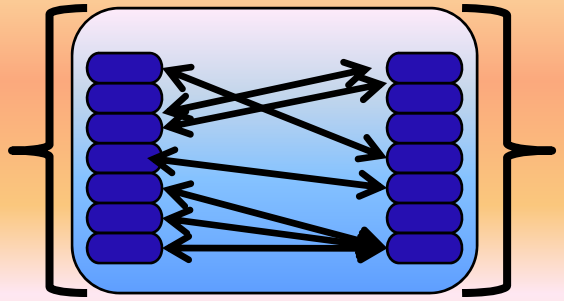
*.SSDL

Parts of the EDM

Design Time → Three Sections of *.edmx file:

Database Schema

Storage Model



Mapping

Domain Objects

Conceptual Model

Database

Shreds

UI

OO

Services

Run Time → Shredded into three files - embedded into assembly

***.SSDL**

Store Schema Definition Layer

***.MSL**

Mapping Schema Layer

***.CSDL**

Conceptual Schema Definition Layer

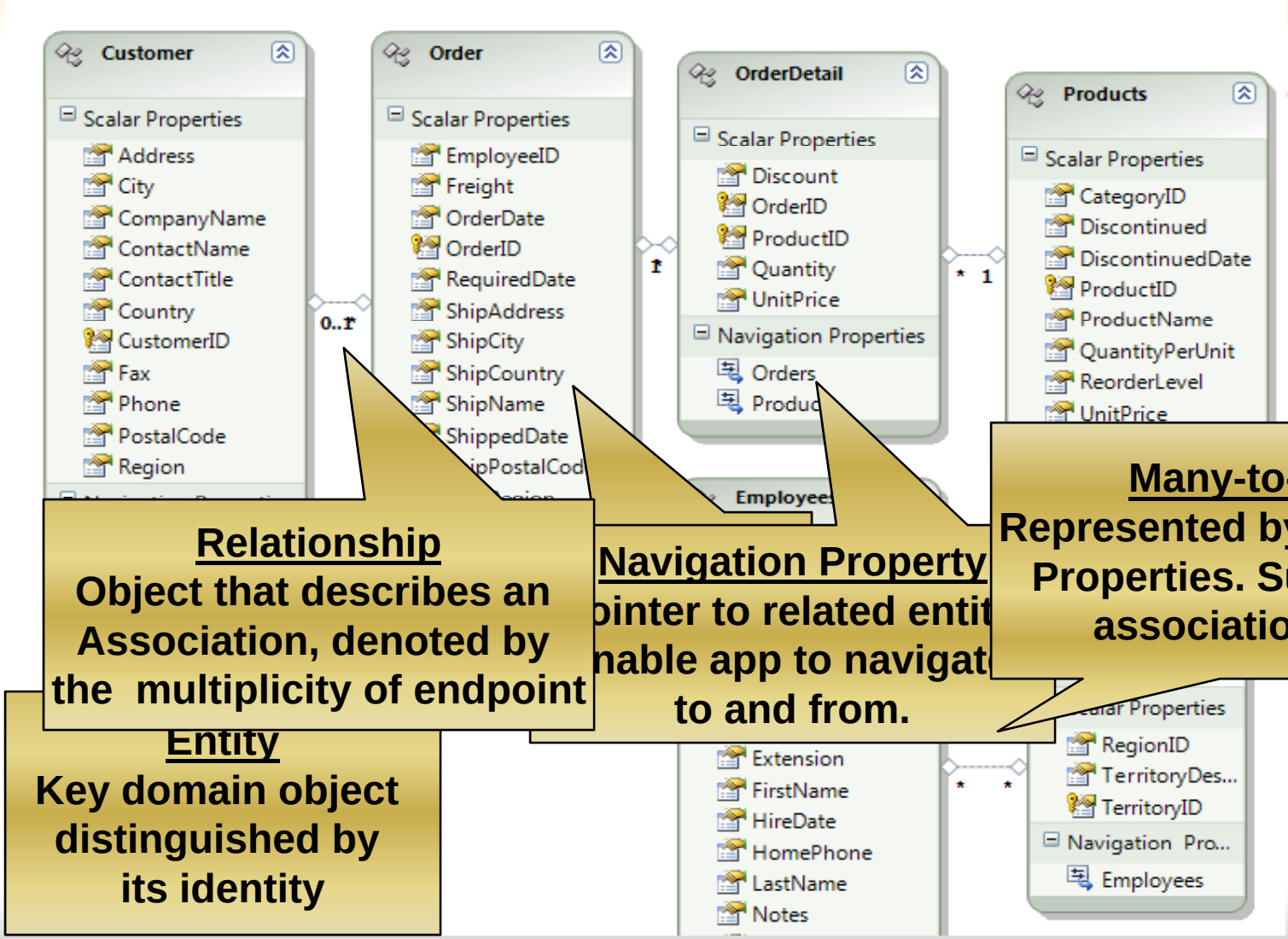
Agenda

- The Problem
- The EF and EDM
- *Mapping a Conceptual Model*
- Programming a Conceptual Model
- Some Intermediate Topics
- EF vs. L2S
- Advanced Mapping
- How to get started

Mapping Demo

- Create a new ADO.NET Entity Data Model
- Demonstrate mapping wizard tool
- Demonstrate default (simple 1-to1) mapping
- Look at the EDMX Designer
- Examine entity, scalar and navigation properties
- Look at the Model Browser
- Look at the Mapping Details
- Look at the EDMX file
- Look at generated entities

Mapping Recap



Relationship
Object that describes an Association, denoted by the multiplicity of endpoint

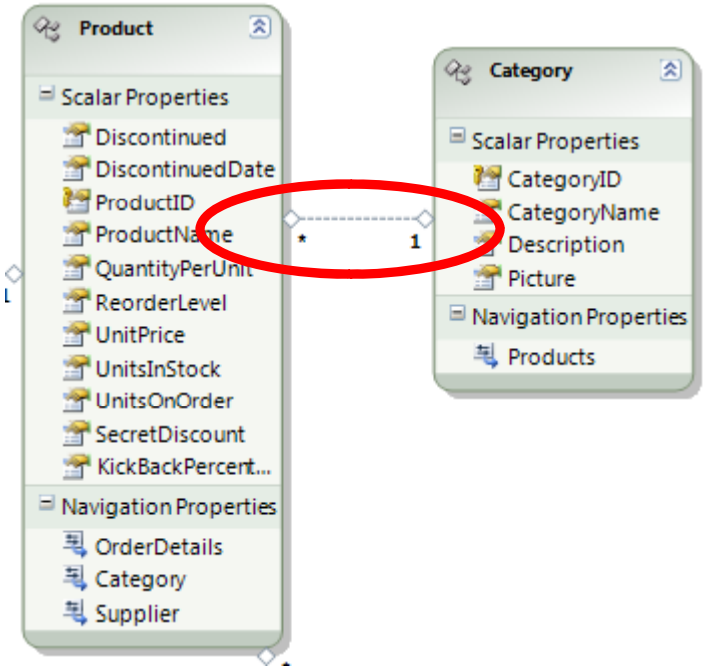
Navigation Property
pointer to related entit
nable app to navigat
to and from.

Many-to-Many
Represented by navigation Properties. Suppresses association entity

Entity
Key domain object distinguished by its identity

Associations

- Represents relationships between entities:
  - Multiplicity
  - Non-Exclusiveness
  - Direction
  - First-class object in .NET



Connection String

- Examine Connection String from web.config
  - connectionString="metadata=res://*/EDMX.Test.csdl|res://*/EDMX.Test.ssdl|res://*/EDMX.Test.msl; *provider*=System.Data.SqlClient; provider connection string="Data Source=ROB64\SQL10;Initial Catalog=NorthwindEF;Integrated Security=True;MultipleActiveResultSets=True"" *providerName*="System.Data.EntityClient"
- Use relector to examine the csdl, ssdl, and msl files in the assembly (look under resources)

Agenda

- The Problem
- The EF and EDM
- Mapping a Conceptual Model
- *Programming a Conceptual Model*
- Key Concepts
- EF vs. L2S
- Advanced Mapping
- How to get started

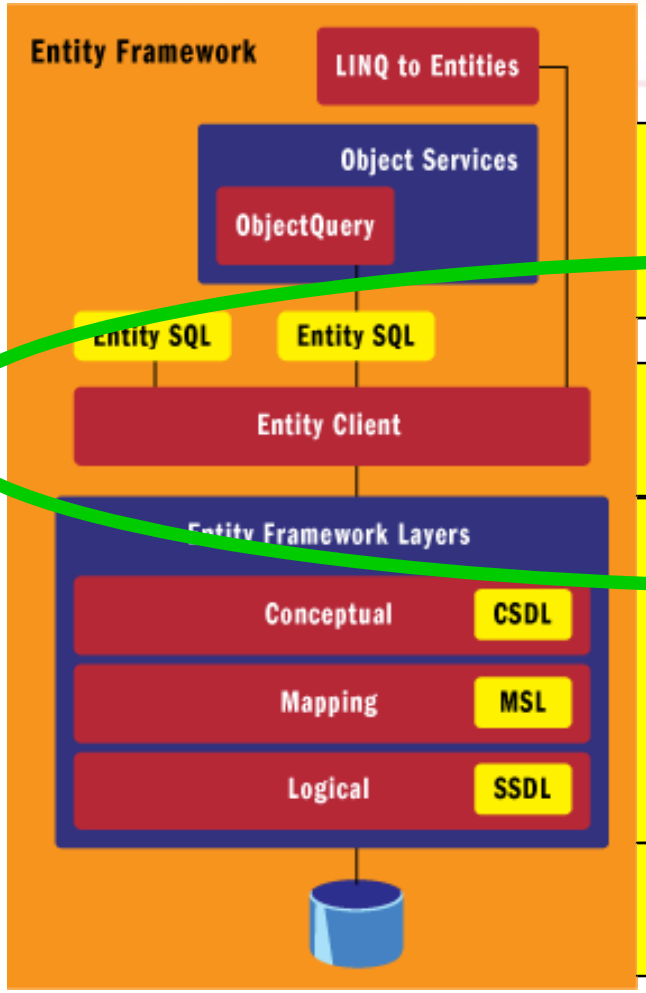
Programming EF

Top APIs available for coding against EDM

Entity Client

Object Services

Programming with Entity Client



Object Services
-- Strongly-typed object abstraction

Entity Client
Lower-level .NET data provider

Entity Data Model

Data Store

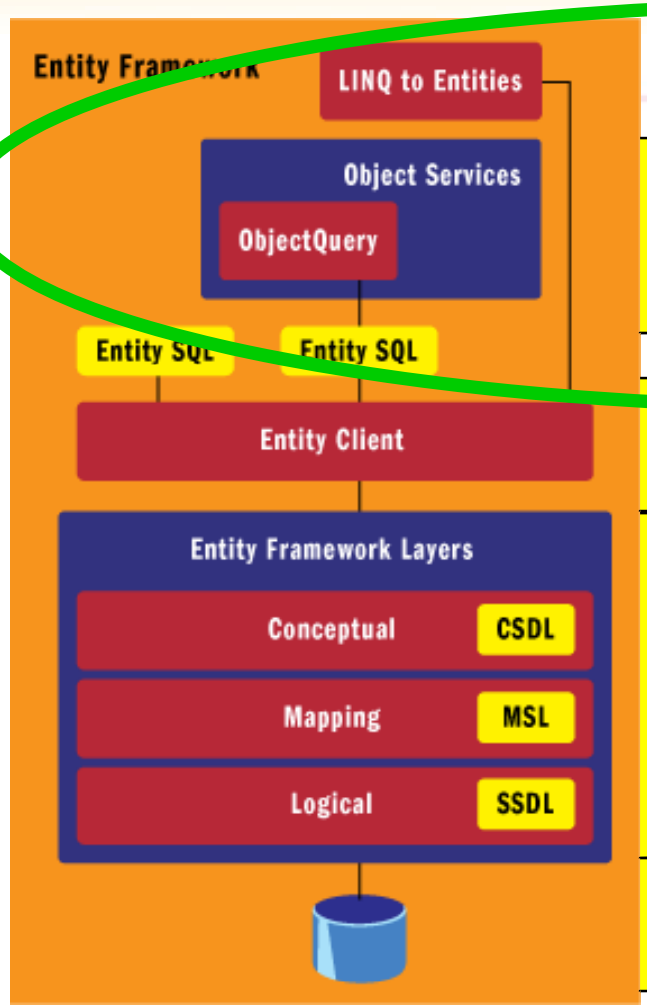
Entity Client

- New ADO.NET data provider
- Queries directly against conceptual model
- Implements new SQL dialect → ***eSQL***
  - Delivers high performance - does not materialize objects from its result set.
  - Accepts connections, commands, returns data readers
  - Returns hierarchical result sets (vs. tabular-shaped)
  - Use when do not need “materialized” objects
  - ***SELECT VALUE p FROM NorthwindEFEntities.Products AS p***

Returns single object, not a row

Query Entity Collection

Programming with Object Services



Object Services

Strongly-typed object abstraction

--

Entity Client

Lower-level .NET data provider

Entity Data Model

Data Store

Object Services

- Query against conceptual model using LINQ
- Benefits:
  - Done within managed code (C# vs. T-SQL)
  - Compile time type checking
  - Intellisense
- Query results are *strongly-typed objects*
  - Entity Types
  - Anonymous Types
  - Scalar values

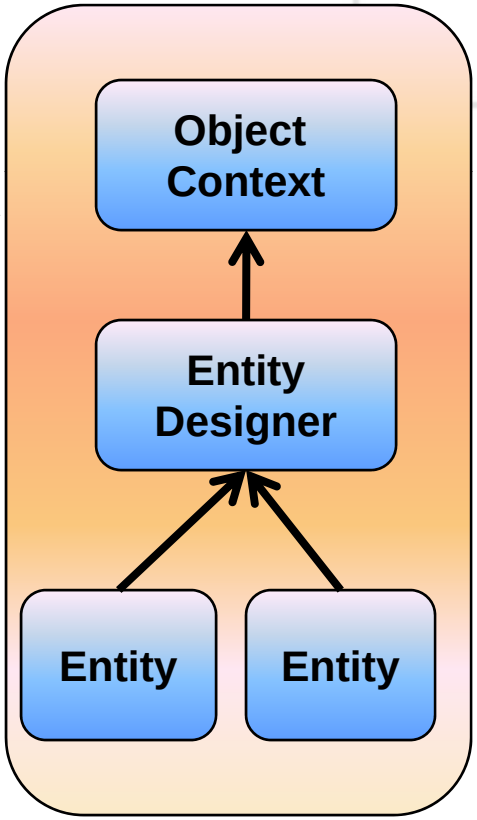
Simple LINQ Query

- First Demo…
  - Simple web form with GridView
  - Simple “customers” entity
  - Query all customers in Oregon
- We are *querying conceptual model*, not store
- Demonstrate “Query Syntax” vs “Method Syntax”
- ***DEMO***

Object Context

```
using (NorthwindEFEntities ctx = new NorthwindEFEntities())
```

- *Workhouse* of EF
  - Manages the connection to data store
  - Composes and executes queries directly against store
  - Marshals data across boundaries and materializes entities
  - Acts as caching container in memory
  - Maintains full change tracking and concurrency management



ObjectContext and Productivity

- Did I have to manage a connection object?
  - *It Manages* connection
- Did I have to manually write entity classes and mappings?
  - *It Maps* relation tables to the entity classes
- Did I have to write SQL statements?
  - *It Generates*(parameterized) SQL
- Did I have to write ADO.NET code?
  - *It Marshalls* data from database to entity objects and back
- Can you see the productivity gain?

Agenda

- The Problem
- The EF and EDM
- Mapping a Conceptual Model
- Programming a Conceptual Model
- *Key Concepts*
- EF vs. L2S
- Advanced Mapping
- How to get started

Anonymous Types

- New language feature in C# 3.0 and VB 9 essential for LINQ projections
- Project *strongly-typed* result set *without* explicitly declaring type
- Enables compiler to return types defined inline without a formal type definition
- *Saves time*, as relieves you from having to define a class for every type returned from a LINQ query

Type Inference and Anonymous Types

- Anonymous Type…
  - *select new { … }* *(new, no type name followed by curly braces)*
  - Products a *projection*
- Type Inference…
  - `var myName = "Hey";` → `string myName = "Hey";`
  - `var myAge = 25;` → `int myAge = 25`
  - *Infer type on the fly*
  - *var* keyword tells compiler to *infer strong type* based on right-side expression
  - *enforces strong typing* (not a VB6 variant)
- ***DEMO***

Inferred Relationships

- When EDMX designer *detects* <PK><FK> relationship, it *automatically creates* "navigation property" that *associates* related entities and *traverses* the relationship
  - From *category*, use *navigation property* to find related product entities

Category
CategoryID <pk>

Category contains collection of product objects (EntitySet Type)

```
from c in Category
where c.products.unitPrice > 20
select c;
```

From *products* object, use *navigation property* to find category entities

Product
ProductID <pk>
CategoryID <fk>

Product contains reference to category parent object (EntityRef Type)

```
from p in Product
where p.category.categoryID == 1
select p;
```

- *DEMO*

Lazy Loading

- LINQ is *built* on concept of "*deferred execution*"
  - Most query operators don't execute when declared
    - //-- First, define query
    - var query = from c in ctx.Customers
      where c.Region == "OR"
      orderby c.CompanyName
      select c;
    - //-- Then, execute query
    - gvQuery.DataSource = query;
    - gvQuery.DataBind();
  - Query executed when *referenced* or *enumerated*
- *(No Demo Yet)*

Store query in variable

Execute query when absolutely necessary (when enumerated or referenced)

Immediate (Eager) Loading

- Query is executed *immediately* at declaration
- Specific query operators support immediate execution:
  - Operators that return a *singleton*:
    - Aggregate Methods
    - Element Methods
    - Retrieve specific element from sequence
    - First(), Last(), Singe()
  - Collection Conversion Operators
    - ToArray(), ToList(), ToDictionary(), ToLookup()
- *Include()* method supports immediate execution:
  - Retrieves parent and child entities *immediately*, not when referenced
- *(No Demo Yet)*

Loading Single Objects

- Basic Example
  - Run SQL Profiler and step thru code
  - Demonstrate Lazy loading – do not execute until absolutely needed
  - Explore how and when deferred execution takes place
  - See how each enumeration of same query generates a separate call to DB
- ***DEMO NOW!***

Loading Object Graphs

- Object Graph: Set of individual, but related objects, that together form a logical whole unit

- Child objects load *one at a time* (separate data store call for each) as needed – default behavior
- For object graphs, use *Include* keyword to *predefine* the *object graph*, forcing immediate loading of entire graph by a single query
- Child data retrieved *immediately*, not when referenced
- ***DEMO NOW!***

Deferred vs. Immediate

- Deferred Execution
  - Benefits:
    - Fetches data only when needed, *not all at once*
    - Minimize network traffic, memory consumption and database load.
    - Great for when *we don't need to display all children* of parent objects, but fetch them only as user requests
  - Drawbacks:
    - *Chatty Interface: Each* request for *child* record *invokes explicit database query*

Deferred vs. Immediate

- Immediate Execution
  - Benefits:
    - Singe query returns all data
    - Great if you know that you will need all data returned
  - Drawbacks:
    - *Large amount* of data returned, whether used or not.

Updates and Change Tracking

- ObjectStateManager tracks changes
  - Caches both *original* and *changed* data values
  - From the cache, dynamically constructs SQL statements
    - If not updating, *disable Change Tracking*
      - *ctx.Products.MergeOption = MergeOption.NoTracking;*
- In next demo…
  - Turn on SQL Profiler
  - Change postal code freight charge for an order
  - See how ObjectStateManager *tracks **both** changes* and *original* values
  - See how generate parameterized SQL *update* statement
  - Explicitly wrap in *transaction*
- *DEMO*

Inserts

- Inserting with EF
  - inserting is a *multi-step* process:
    - Create new object and populate
    - Attach new record to existing entitySet
    - SaveChanges() to commit to database
- In next demo…
  - (1) show how ObjectContext *automatically* re-queries db after insert and *automatically* returns new identity key and *automatically* shoves it into your new entity object
  - (2) see how ObjectContext leverages *inferred relationship*
    - simultaneously inserts both category and product records
    - inserts records in correct order
    - Maintains foreign key relationship
    - Automatically manages the identity values
- Demo

Agenda

- The Problem
- The EF and EDM
- Mapping a Conceptual Model
- Programming a Conceptual Model
- Key Concepts
- *EF vs. L2S*
- Advanced Mapping
- How to get started

EF vs. LTS

- General consensus:
  - LTS
    - Strongly typed LINQ access for rapid development against SQL Server
    - Support direct (1-to-1) mapping with some functionality limitations
    - Limited out-of-the box support for complex scenarios
  - EF
    - Designed for larger enterprise applications
    - Enables complex mapping complex scenarios
    - More robust tools and designer
    - Supports various types of inheritance, many-to-many relationships and composite types
    - Supports provider independence

EF vs. LTS

Category	LINQ to SQL	Entity Framework
Model	domain model	conceptual data model
Databases Supported	SQL server only	Many
Complexity/Learning Curve	simple to use	complex to use
Development Time	rapid development	slower development but more capabilities
Mapping	Direct 1-to-1	Custom mapping
Inheritance	hard to apply	simple to apply
Support	?	$$$, resources, innovation

Agenda

- The Problem
- The EF and EDM
- Mapping a Conceptual Model
- Programming a Conceptual Model
- Some Intermediate Topics
- EF vs. L2S
- *Advanced Mapping*
- How to get started

Advanced EDM Mapping

- EDM provides tremendous flexibility customizing conceptual model
  - Table Per Hierarchy Inheritance mapping
  - Table Per Type Inheritance mapping
  - Entity (vertical) splitting
- Designer supports most, but not all, customization

Mapping Between Layers

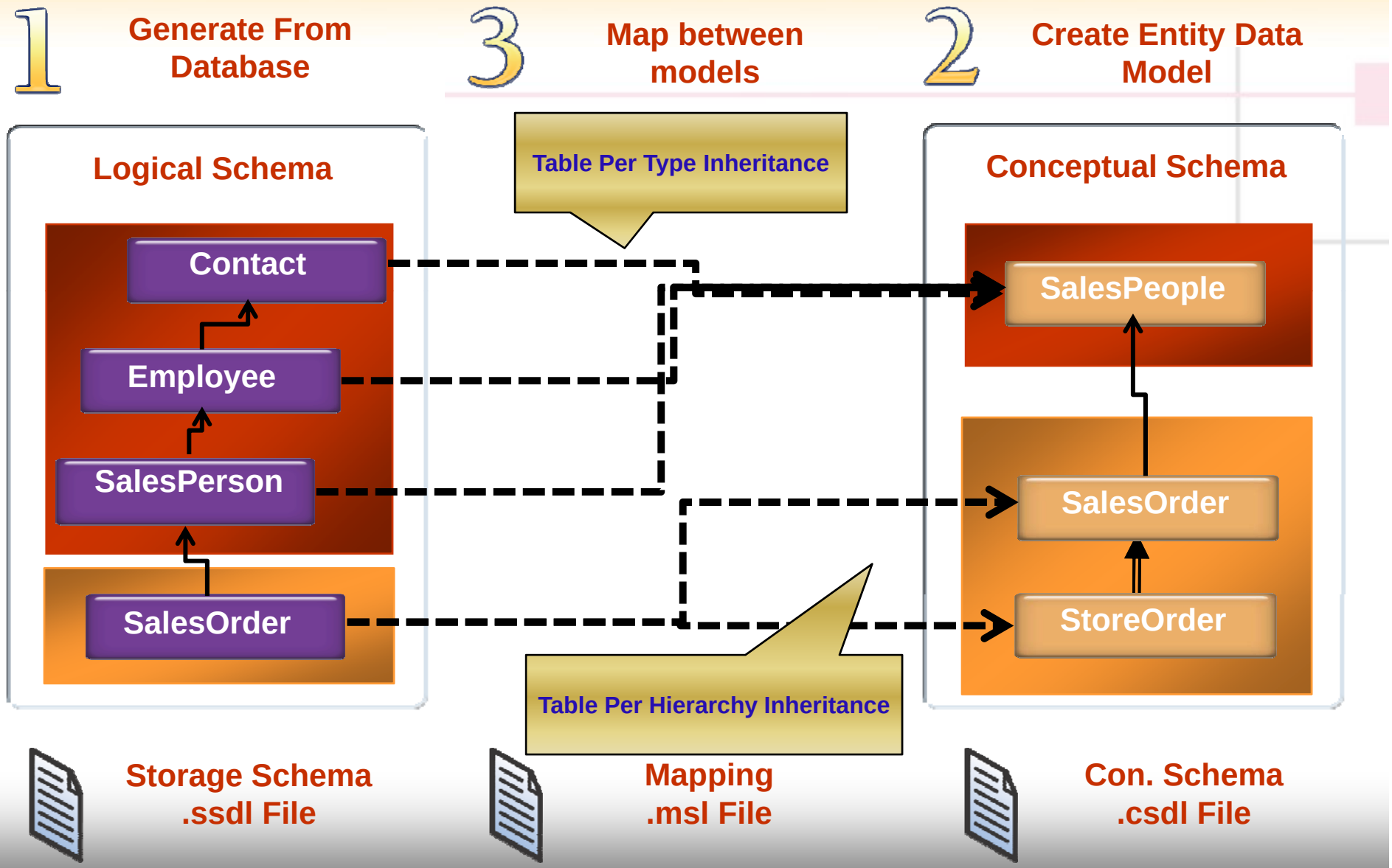
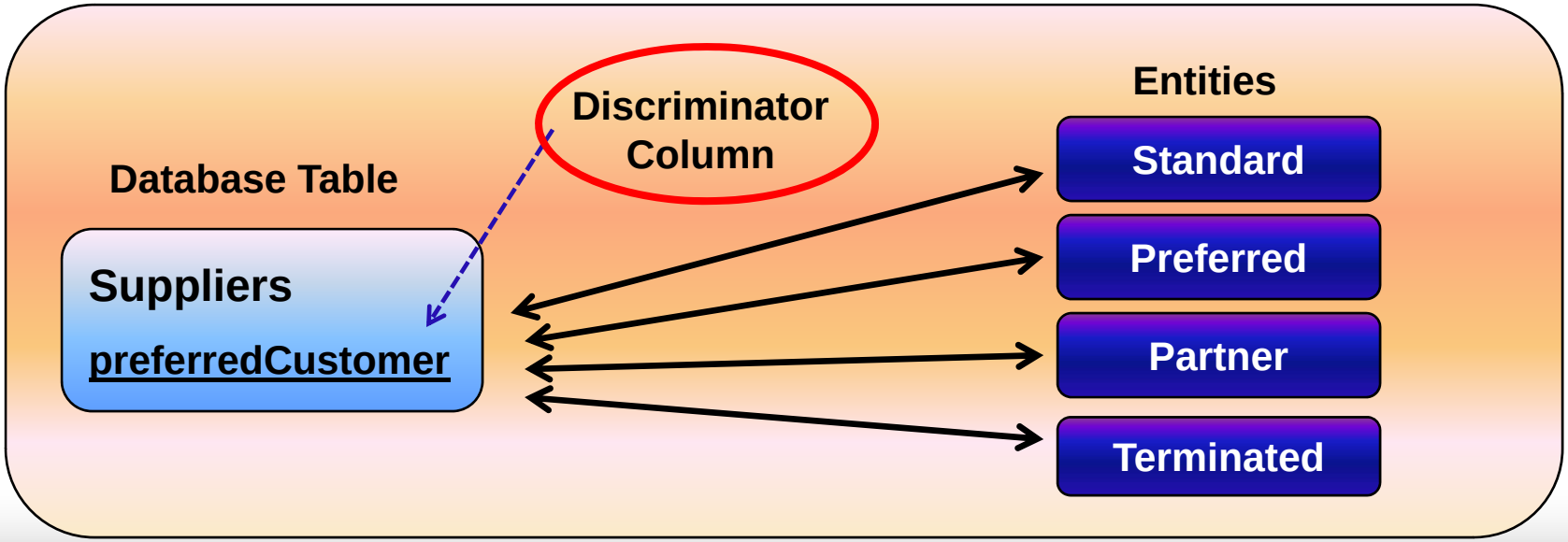


Table-Per-Hierarchy Mapping

- Simple inheritance that maps multiple entities to a single database table
- Requires discriminator column to differentiate types
- Quick to implement, but lacks scalability – Derived from single database table

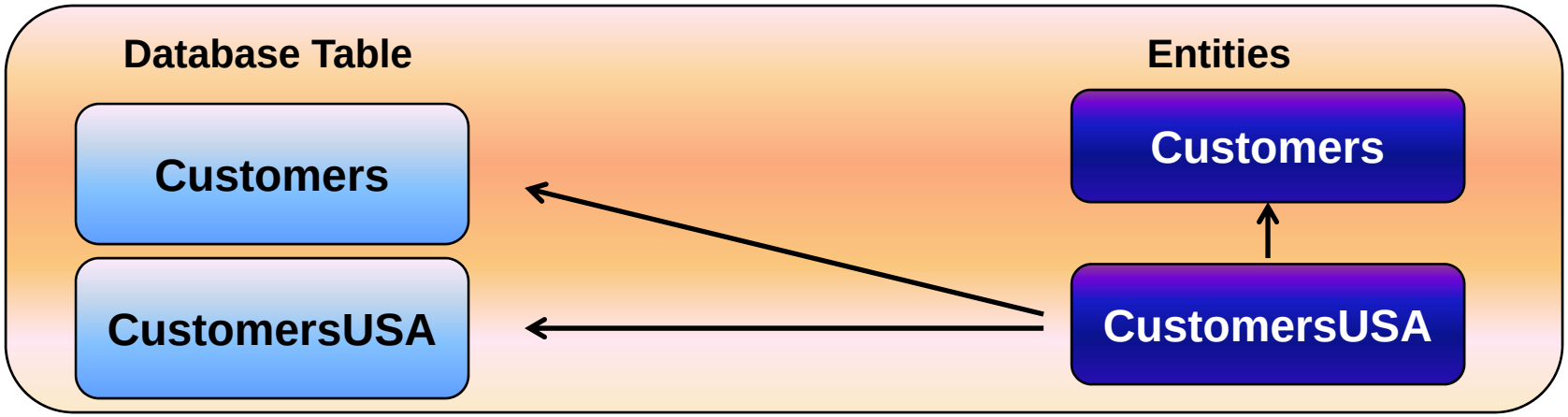


Using TPH

- Discriminator column identifies whether row belongs to base or derived type
- Can include additional columns for derived type whose value is NULL for base type
- Can query on derived types (i.e., preferred vendors) without a where clause by using OfType() method
  - from c in ctx.Customers.OfType<PreferredVendors>()
- ***Show in Designer***
- ***Demo***
- Limitations
  - Difficult to update discriminator column value via EF – must use database workaround (trigger/stored proc)

Table-Per-Type Mapping

- Inheritance which spans related database tables.
- Exposes attributes from both parent table and child table into single derived entity
- Defined in database with separate tables where one table (the child) describes a new type based on another table (the parent)

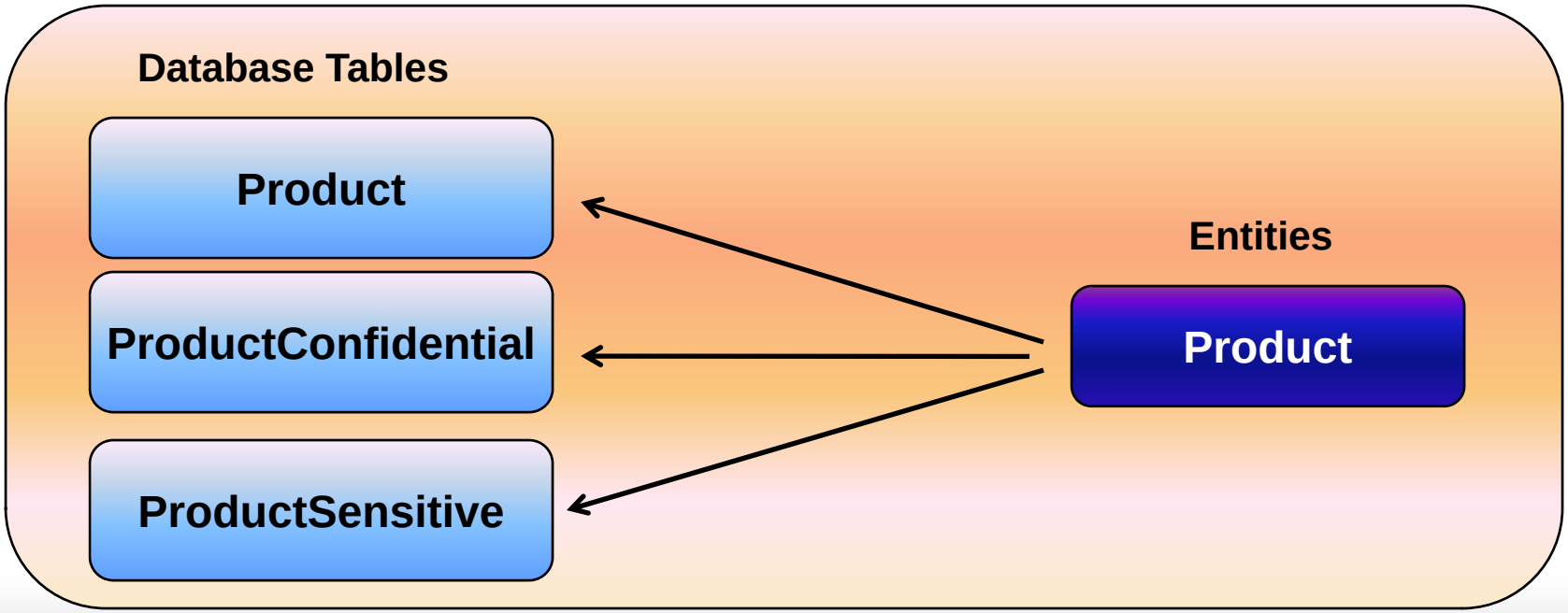


Using TPT

- Eliminates need to traverse navigation properties to move from child (CustUSA) to parent (Cust) to get information
- When filtering or projecting on derived type, use OfType() method
  - from c in ctx.Customers.OfType<CustomersUSA>()
- ***Show in Designer***
- ***Demo***
- Limitations:
  - Derived type is *completely bound* to parent type
  - Cannot delete child without deleting parent
  - Cannot change existing customer to CustomerUSA

Entity (Vertical) Splitting

- Map single entity to multiple tables
- Can implement when tables share common key



Advanced Mapping

- Vertical Entity Partitioning
  - Practice of decomposing large table into number of smaller tables
  - In EDM, map single entity to multiple underlying tables
    - Conceptual model contains single entity
    - Store (SSDL) contains multiple DB tables
    - MSL metadata maps single entity to multiple tables

Show in Designer

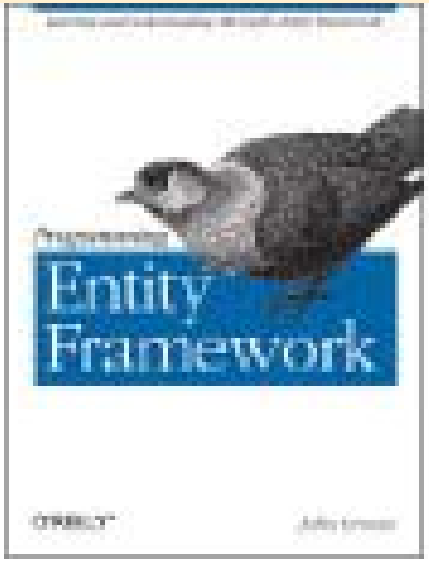
Demo

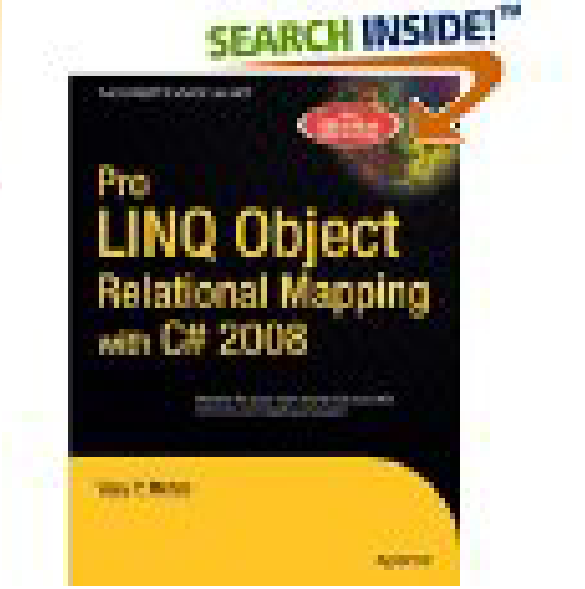
- SSDL contains three product tables
- Conceptual model has single product Entity
- Demonstrate retrieving and updating data

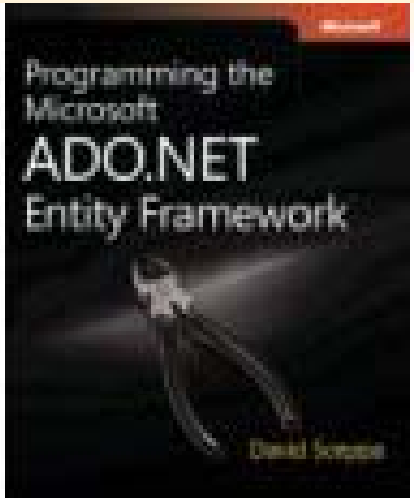
Agenda

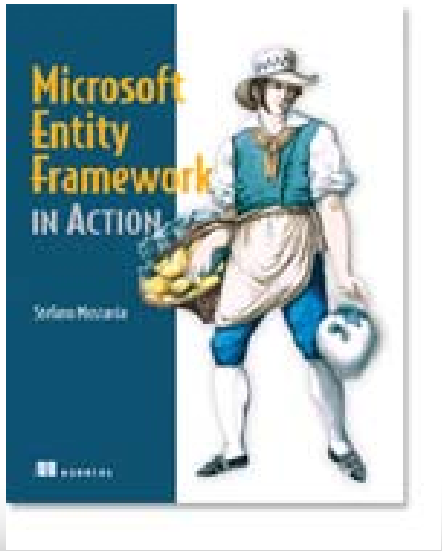
- The Problem
- The EF and EDM
- Mapping a Conceptual Model
- Programming a Conceptual Model
- Key Concepts
- EF vs. L2S
- Advanced Mapping
- *How to get started*

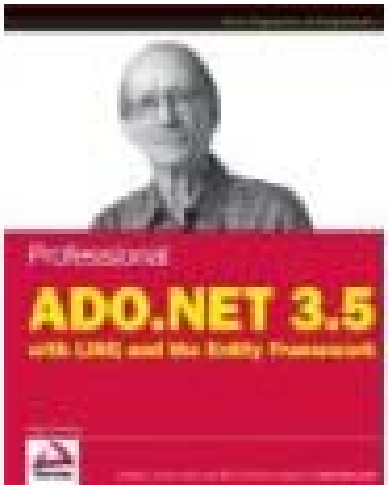
EF Books


Entity
Framework


SEARCH INSIDE!™
Pro
LINQ Object
Relational Mapping
with C# 2008

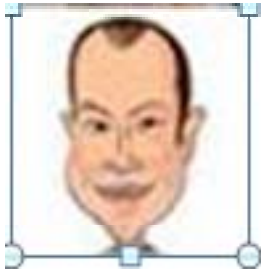

Programming the
Microsoft
ADO.NET
Entity Framework


Microsoft
Entity
Framework
IN ACTION


Professional
ADO.NET 3.5

EF Webcasts

- eBook → http://weblogs.asp.net/zeeshanhirani
- Mike Taulty
  - 150 Screen Casts



Questions?
Comments?
Suggestions?

Extra Slides

Querying Syntax

- *"Friendly"* and *"less-friendly"* way to write LINQ queries
  - "Query Expressions" → are *Friendly*
    - declarative → Easy-to-use SQL-like language
  - "Method Expressions" → is *Less-Friendly*
    - c# query methods with lambda expressions arguments, chained together with "." notation
    - more powerful and includes more query operators
  - At Compile Time…
    - CLR doesn't understand *"friendly"* query expressions
    - translates *"friendly"* query expressions into *"less-friendly"* dot notation syntax
- *DEMO, including Reflector*

Generated SQL

- Parameterized Queries vs. Stored Procedures
  - LINQ *fully supports both approaches*
  - Parameterized queries
    - CQT – Page 148 of Pro Book
    - Good for CRUD operations
    - Executions plans are cached and reused
    - Automatically generated by LINQ framework
    - *Do require* server permissions at the table level
  - Stored Procedures
    - More complex business logic (beyond CRUD)
    - Security, performance, auditing

The Future

Challenges

Compiled Queries

- Parameterized Queries vs. Stored Procedures
  - Page 173 of Pro Book

Advanced Mapping

- Horizontal Entity Partitioning
  - Create table with fewer columns with additional tables to store remaining columns.
  - In EDM, map single entity to multiple underlying tables
    - Conceptual model contains single entity
    - Store (SSDL) contains multiple DB tables
    - MSL metadata maps single entity to multiple tables
  - ***Demo***
    - SSDL contains Orders and Priority Orders tables
    - Conceptual model has single Order Entity
    - Discriminate with boolean value
    - Demonstrate retrieving and updating data