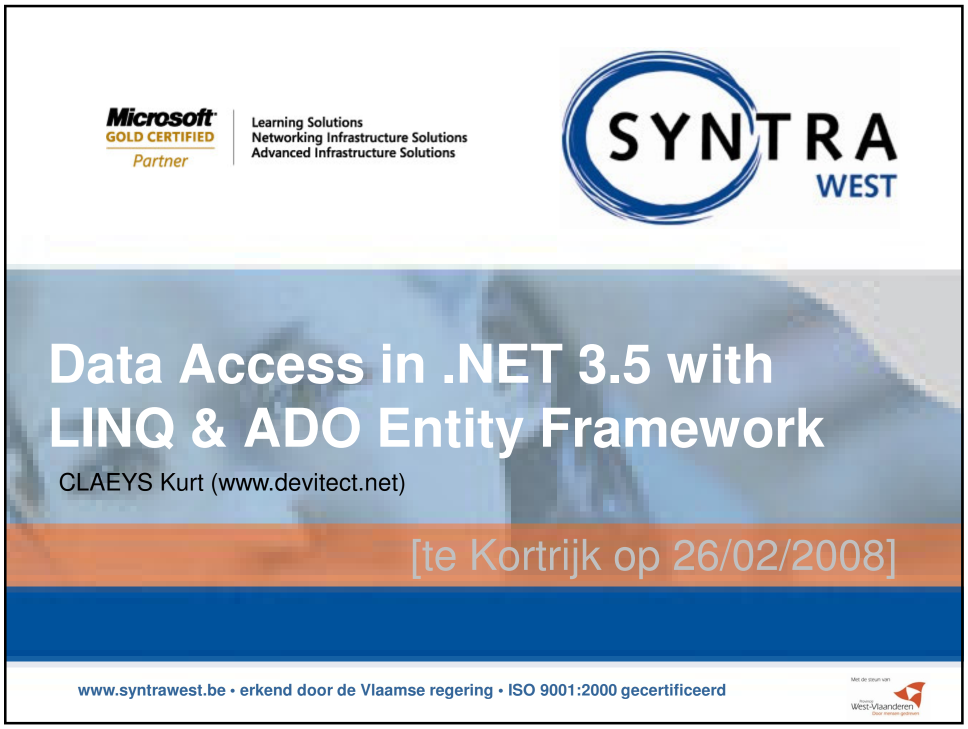


Microsoft GOLD CERTIFIED Partner | Learning Solutions, Networking Infrastructure Solutions, Advanced Infrastructure Solutions

SYNTRA WEST

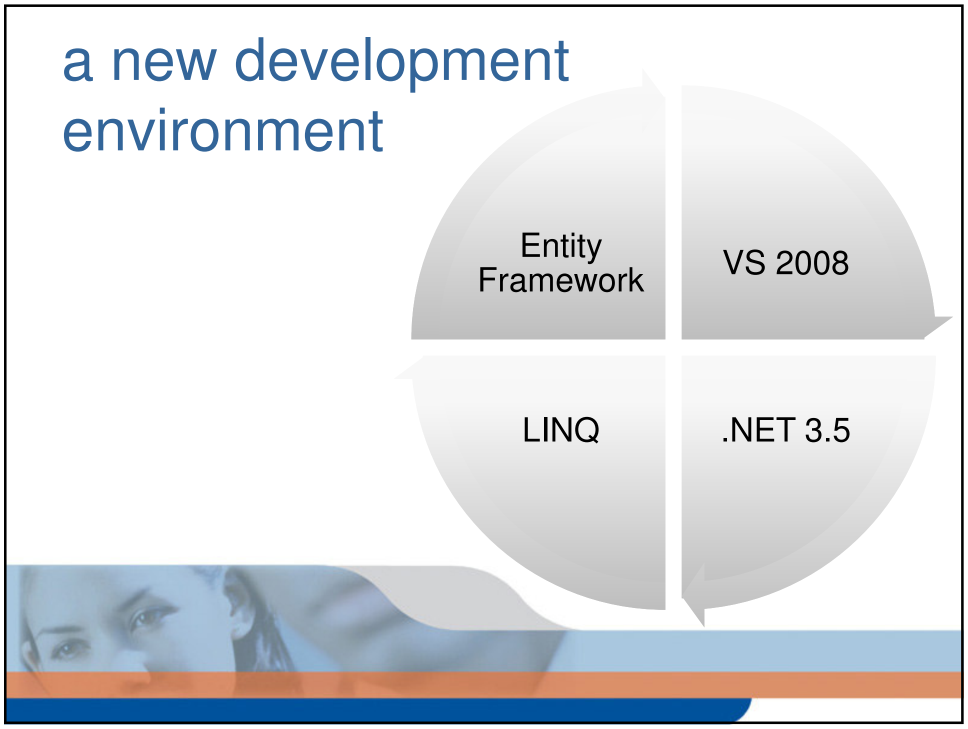
Data Access in .NET 3.5 with LINQ & ADO Entity Framework

CLAEYS Kurt (www.devitect.net)

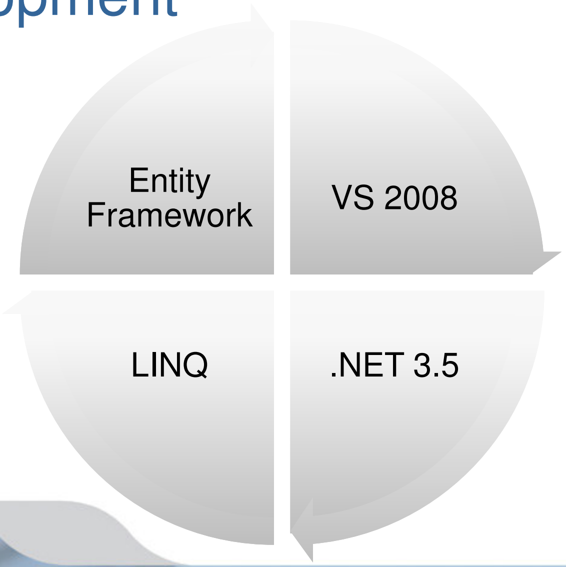
[te Kortrijk op 26/02/2008]

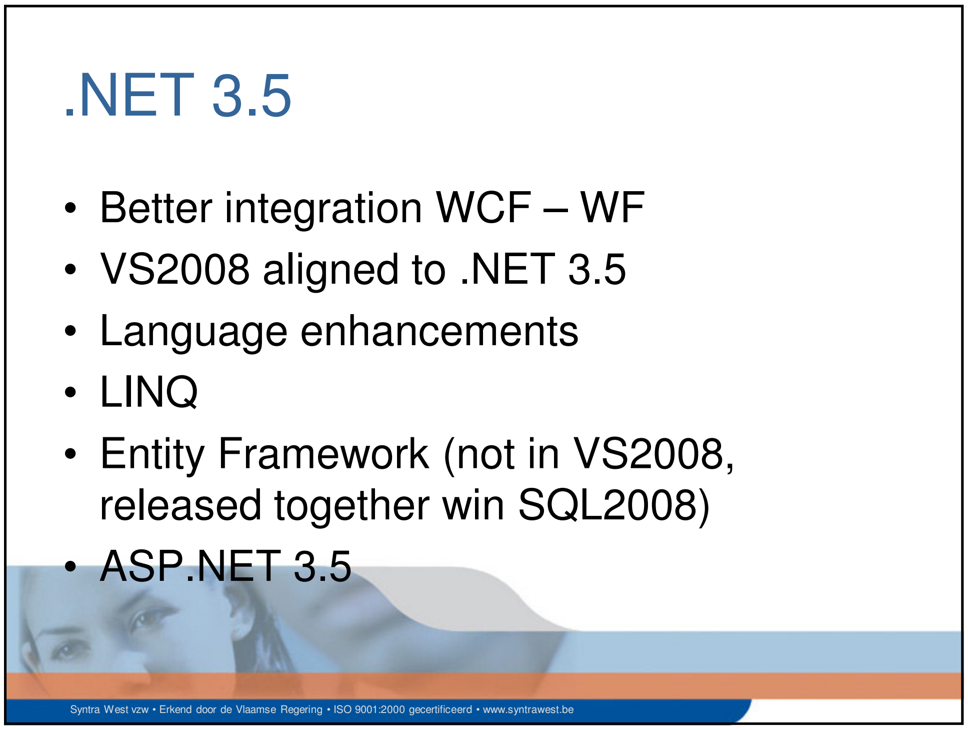
www.syntrawest.be • erkend door de Vlaamse regering • ISO 9001:2000 gecertificeerd

Met de steun van West-Vlaanderen



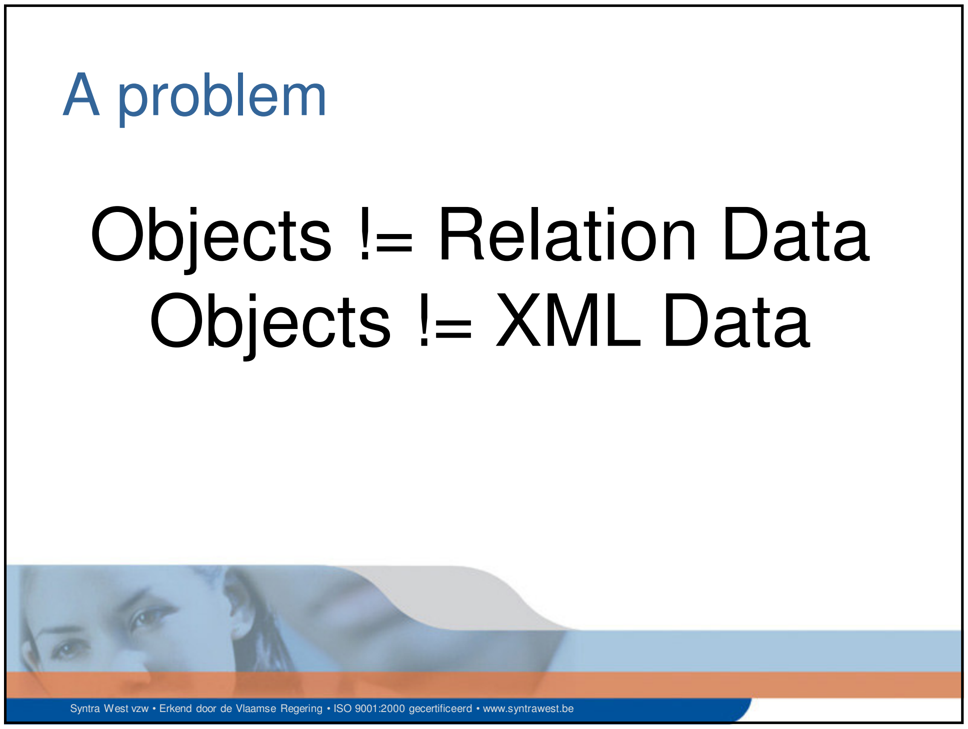
a new development environment





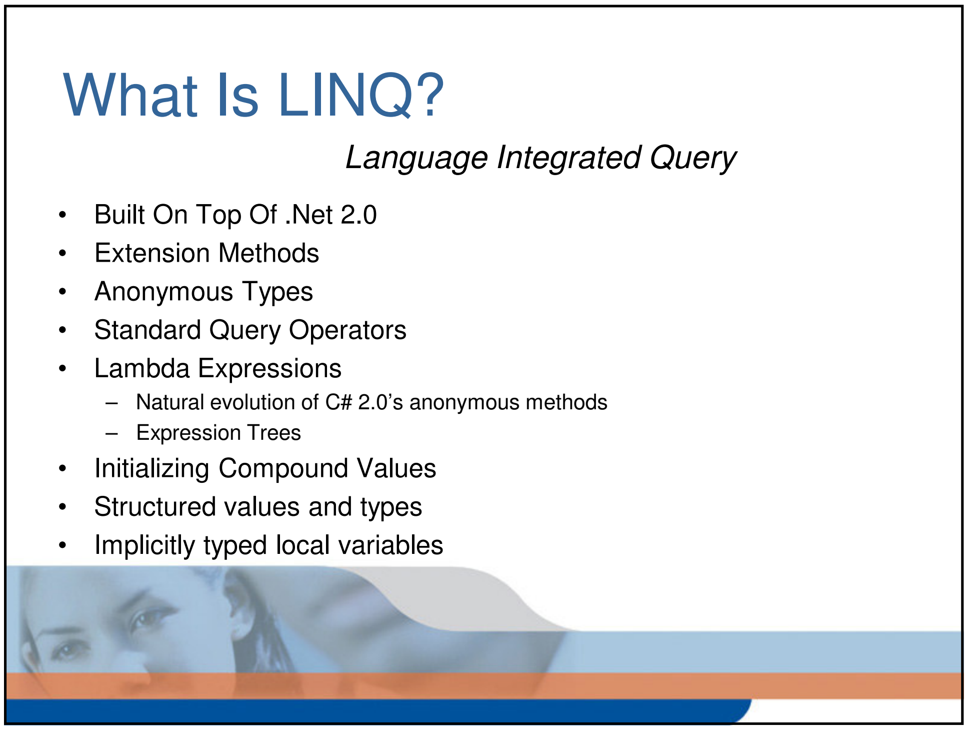
.NET 3.5

- Better integration WCF – WF
- VS2008 aligned to .NET 3.5
- Language enhancements
- LINQ
- Entity Framework (not in VS2008, released together win SQL2008)
- ASP.NET 3.5



A problem

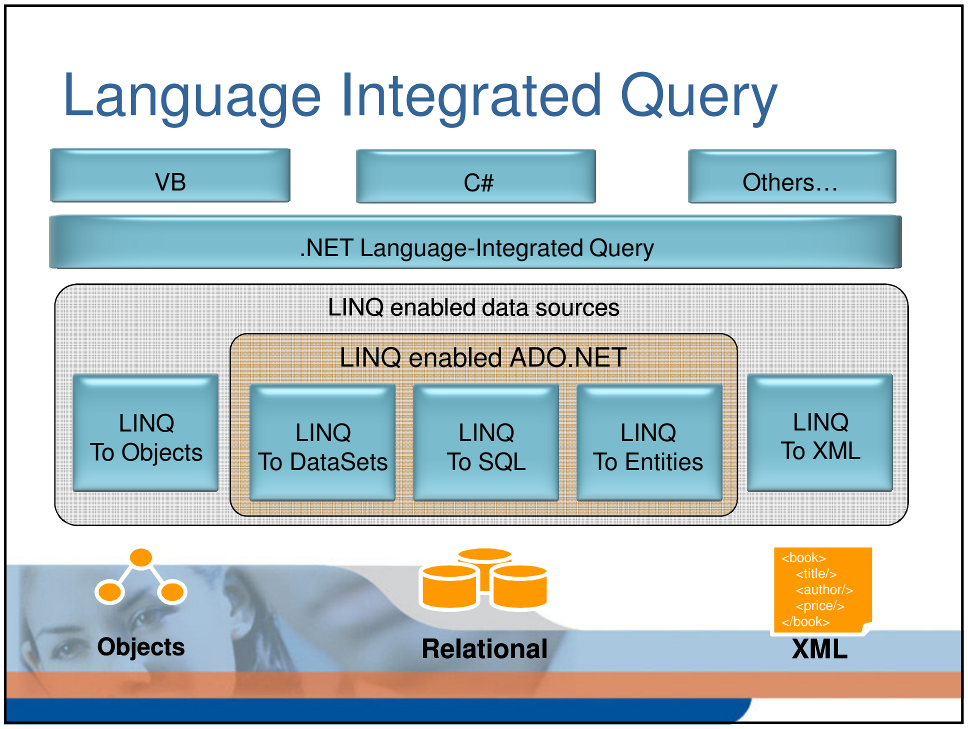
Objects != Relation Data
Objects != XML Data



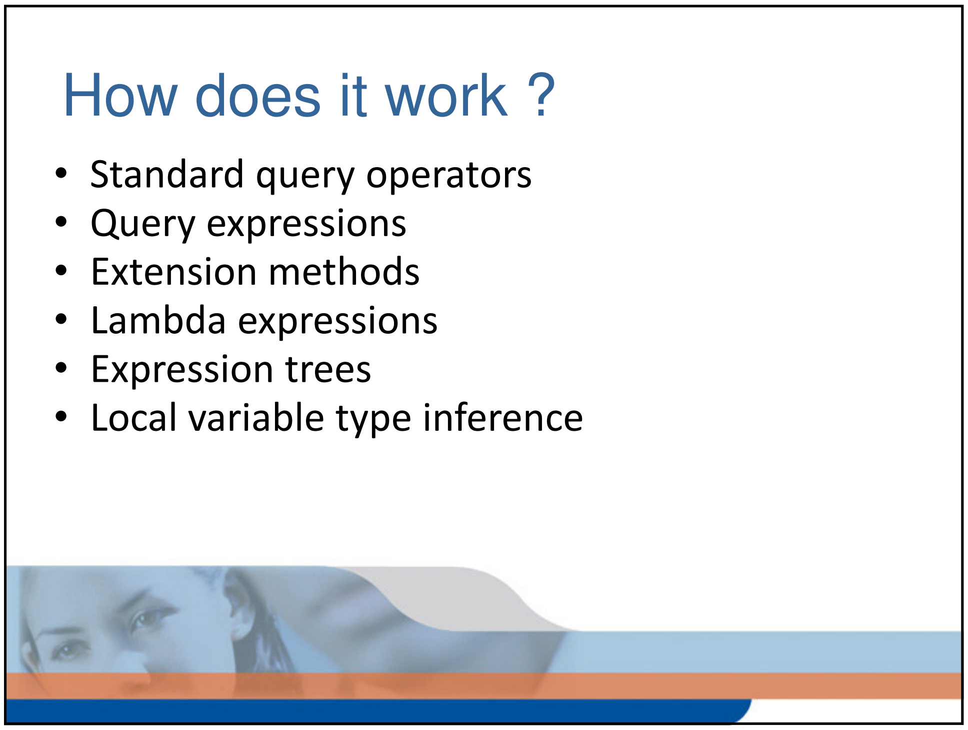
What Is LINQ?

Language Integrated Query

- Built On Top Of .Net 2.0
- Extension Methods
- Anonymous Types
- Standard Query Operators
- Lambda Expressions
  - Natural evolution of C# 2.0's anonymous methods
  - Expression Trees
- Initializing Compound Values
- Structured values and types
- Implicitly typed local variables

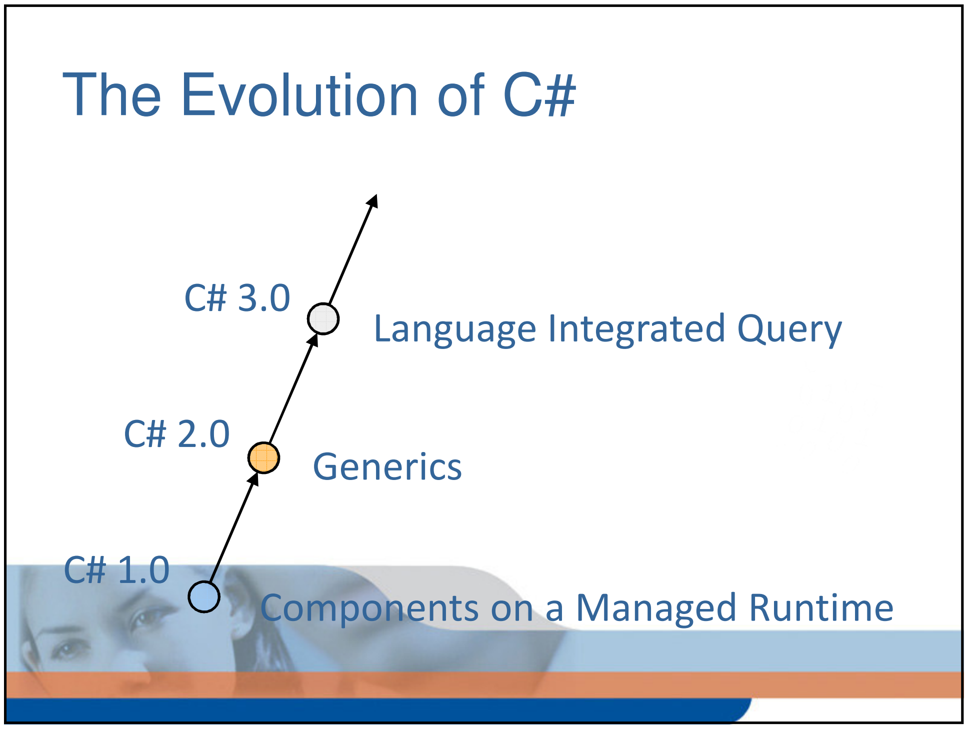


Language Integrated Query



How does it work ?

- Standard query operators
- Query expressions
- Extension methods
- Lambda expressions
- Expression trees
- Local variable type inference



The Evolution of C#

C# 3.0 — Language Integrated Query

C# 2.0 — Generics

C# 1.0 — Components on a Managed Runtime

Query Expressions

- Query expressions

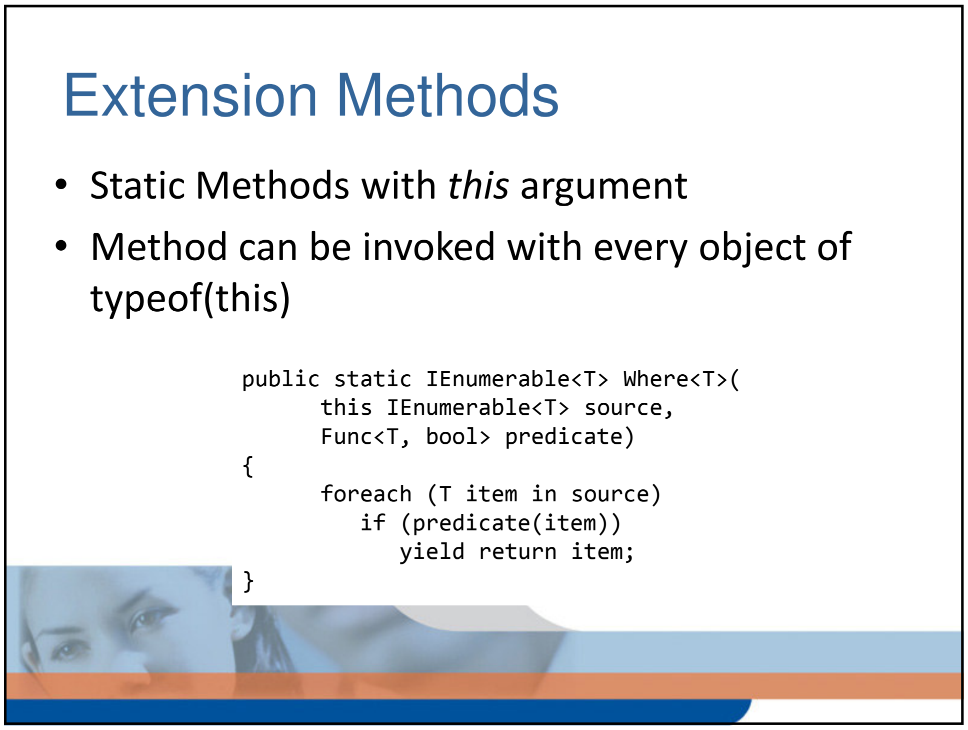
```
var winners = from r in racers
              where r.Wins > 3
              orderby r.Wins descending
              select r;
```

- Extension methods

```
var winners = racers.
                Where(r => r.Wins > 3).
                OrderByDescending(r => r.Wins).
                Select(r => r);
```

Standard Query Operators

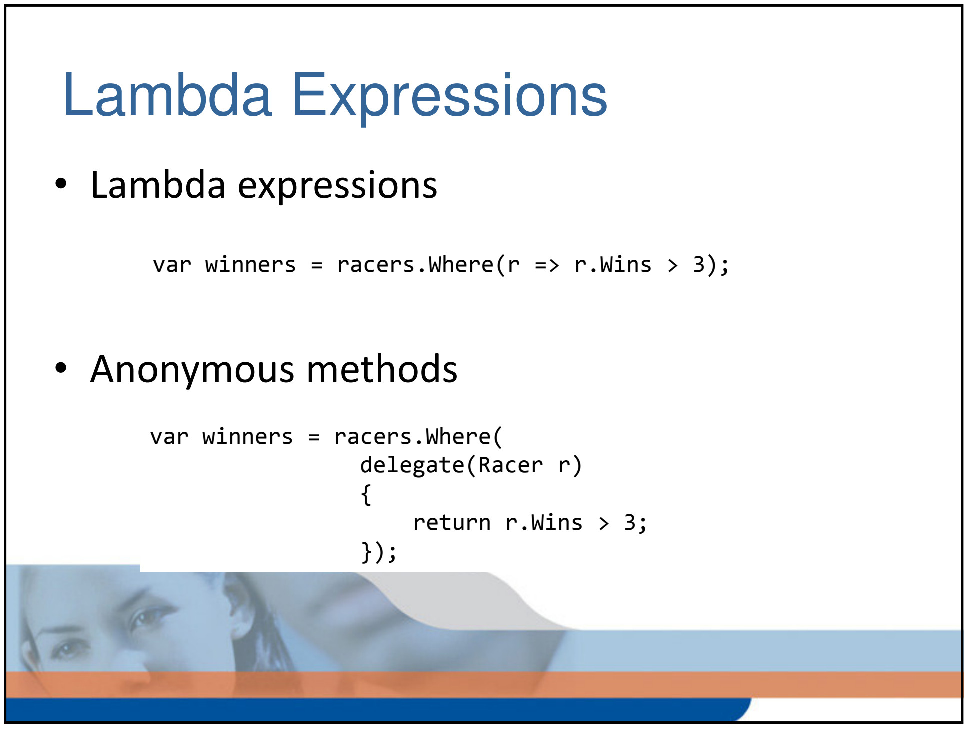
Restrict	s.Where(…)
Project	s.Select(…), s.SelectMany(…)
Order	s.OrderBy(…).ThenBy(…) …
Group	s.GroupBy(…)
Quantify	s.Any(…), s.All(…)
Partition	s.TakeFirst(…), s.SkipFirst(…)
Set	s.Distinct(), s.Union(…), s.Intersect(…), s.Except(…)
Singleton	s.Element(…), s.ElementAt(…)
Aggregate	s.Count(), s.Sum(), s.Min(), s.Max(), s.Average(), s.Aggregate()
Convert	s.ToArray(), s.ToList()
Cast	s.OfType<T>(), s.Cast<T>



Extension Methods

- Static Methods with *this* argument
- Method can be invoked with every object of typeof(this)

```
public static IEnumerable<T> Where<T>(
      this IEnumerable<T> source,
      Func<T, bool> predicate)
{
      foreach (T item in source)
         if (predicate(item))
            yield return item;
}
```



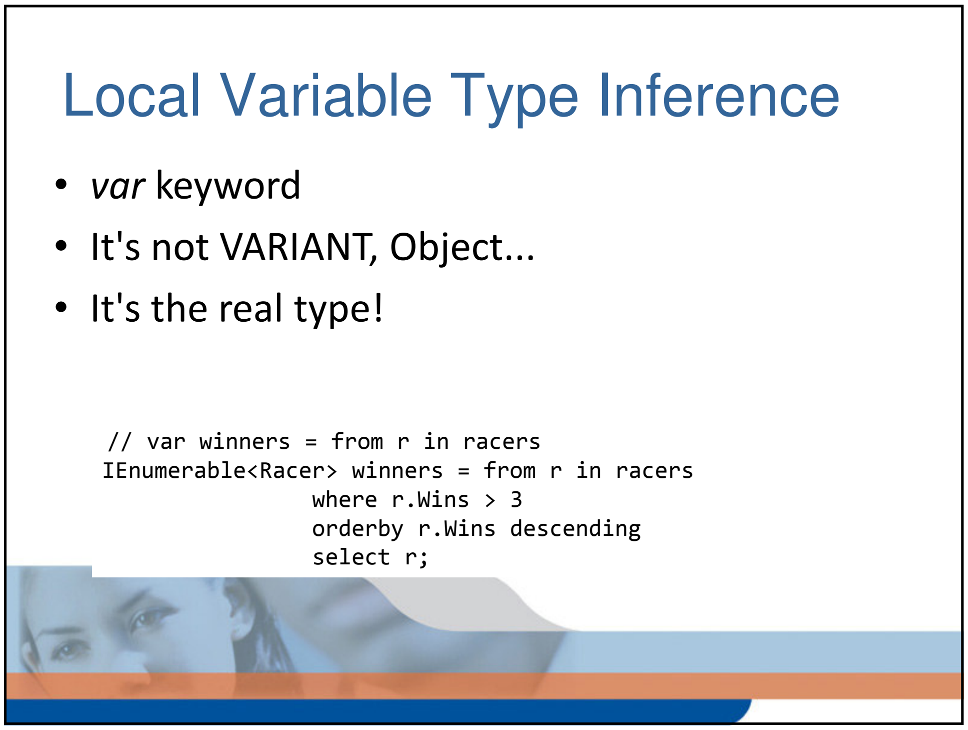
Lambda Expressions

- Lambda expressions

```
var winners = racers.Where(r => r.Wins > 3);
```

- Anonymous methods

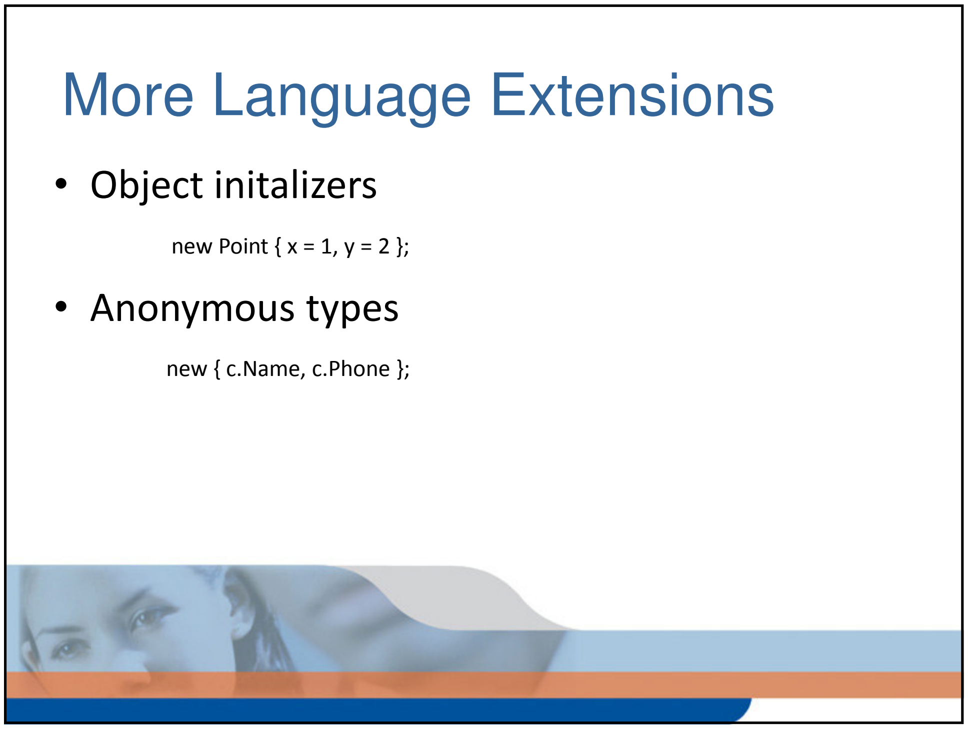
```
var winners = racers.Where(
                delegate(Racer r)
                {
                    return r.Wins > 3;
                });
```



Local Variable Type Inference

- *var* keyword
- It's not VARIANT, Object...
- It's the real type!

```
 // var winners = from r in racers
IEnumerable<Racer> winners = from r in racers
                 where r.Wins > 3
                 orderby r.Wins descending
                 select r;
```



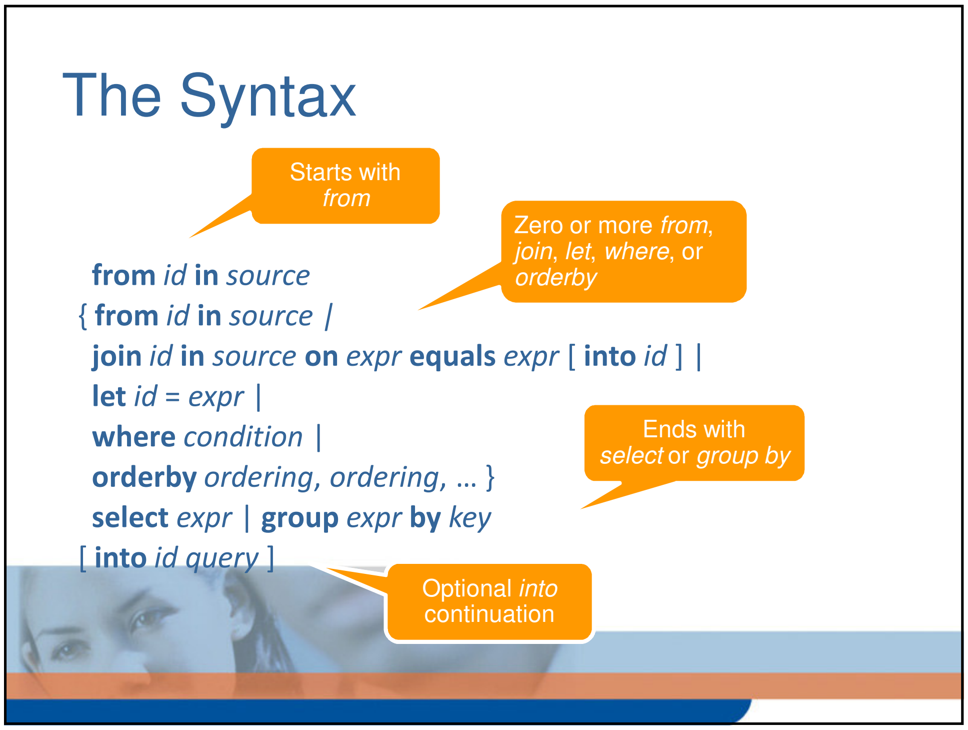
More Language Extensions

- Object initalizers

  new Point { x = 1, y = 2 };

- Anonymous types

  new { c.Name, c.Phone };



The Syntax

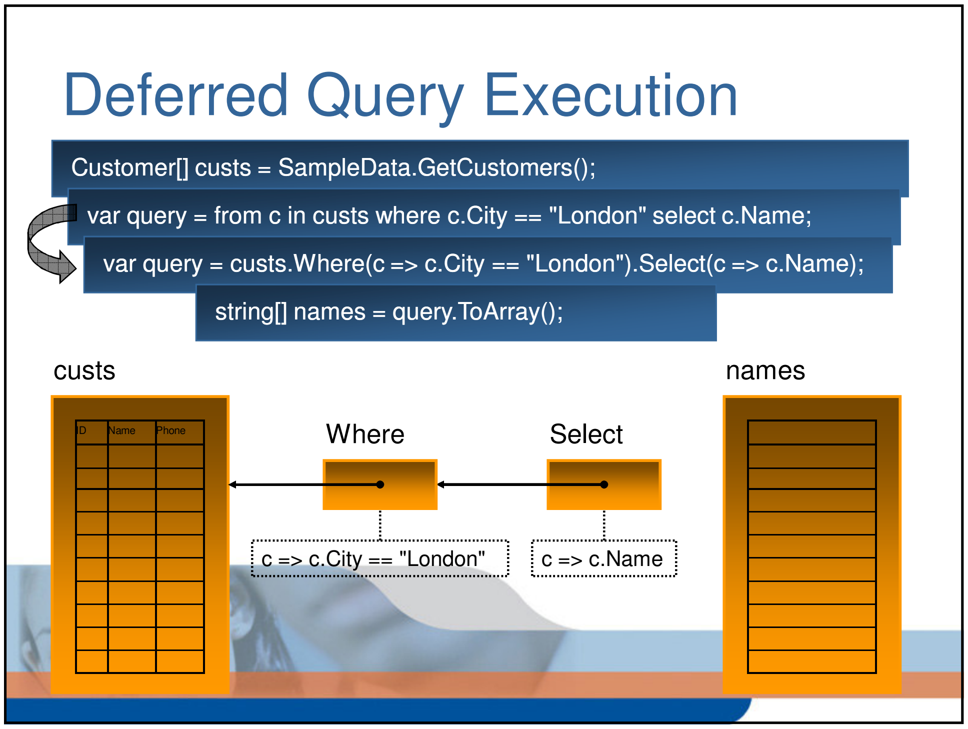
Starts with *from*

Zero or more *from*, *join*, *let*, *where*, or *orderby*

from *id* **in** *source*
{ **from** *id* **in** *source* |
join *id* **in** *source* **on** *expr* **equals** *expr* [**into** *id*] |
let *id* = *expr* |
where *condition* |
orderby *ordering*, *ordering*, … }
select *expr* | **group** *expr* **by** *key*
[**into** *id query*]

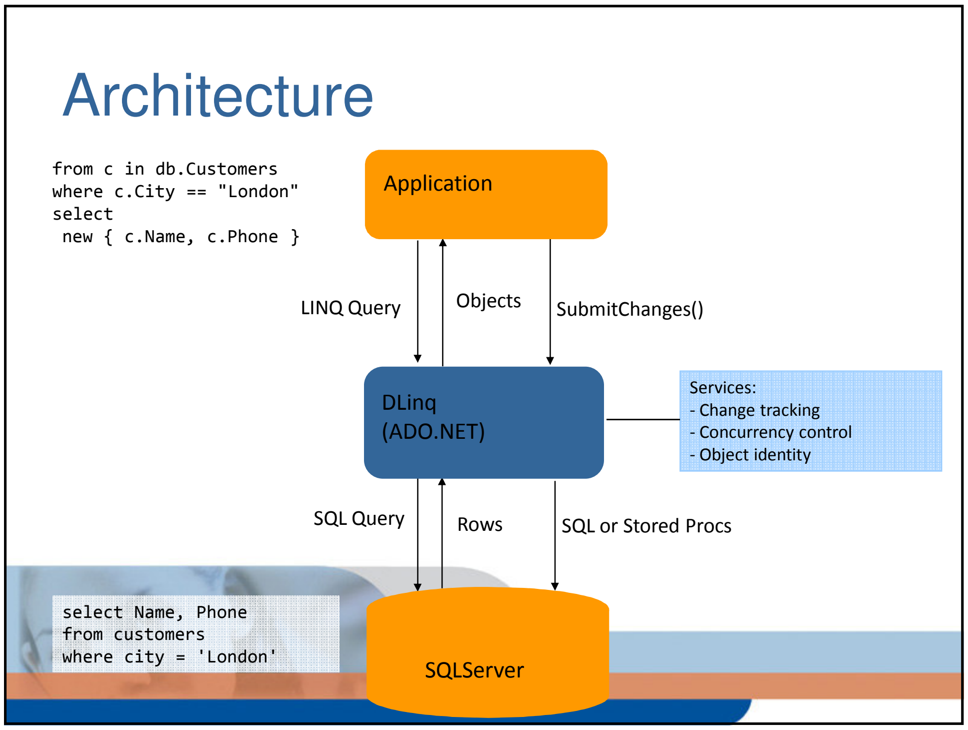
Ends with *select* or *group by*

Optional *into* continuation



Deferred Query Execution

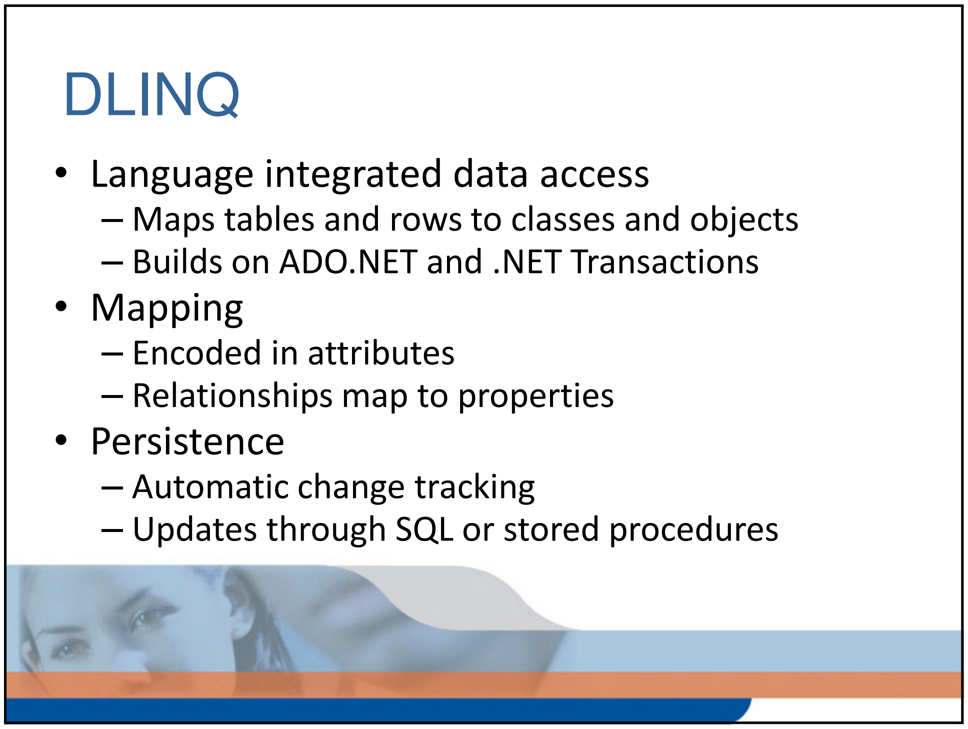
```
Customer[] custs = SampleData.GetCustomers();
var query = from c in custs where c.City == "London" select c.Name;
var query = custs.Where(c => c.City == "London").Select(c => c.Name);
string[] names = query.ToArray();
```



Architecture

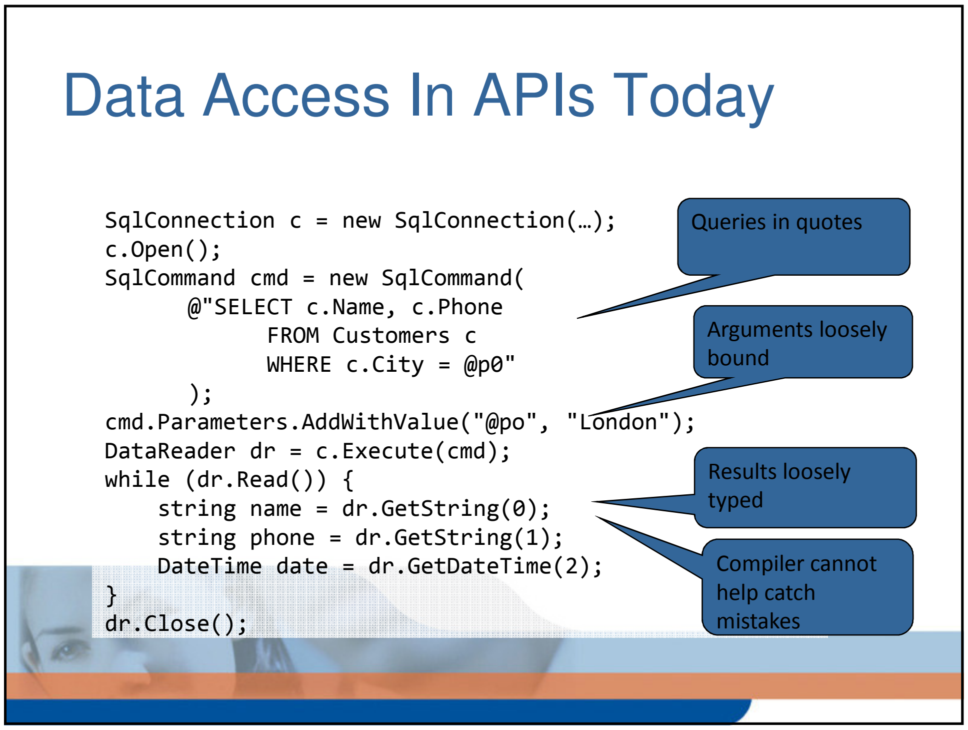
```
from c in db.Customers
where c.City == "London"
select
 new { c.Name, c.Phone }
```

```
select Name, Phone
from customers
where city = 'London'
```



DLINQ

- Language integrated data access
  – Maps tables and rows to classes and objects
  – Builds on ADO.NET and .NET Transactions
- Mapping
  – Encoded in attributes
  – Relationships map to properties
- Persistence
  – Automatic change tracking
  – Updates through SQL or stored procedures



Data Access In APIs Today

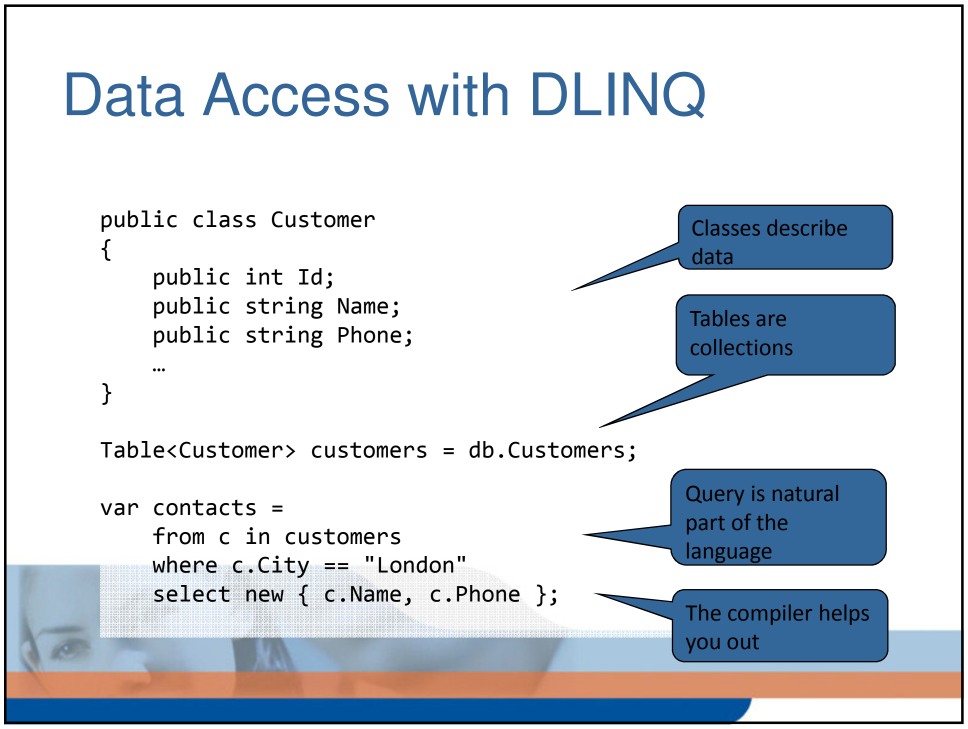
```
SqlConnection c = new SqlConnection(…);
c.Open();
SqlCommand cmd = new SqlCommand(
      @"SELECT c.Name, c.Phone
            FROM Customers c
            WHERE c.City = @p0"
      );
cmd.Parameters.AddWithValue("@po", "London");
DataReader dr = c.Execute(cmd);
while (dr.Read()) {
    string name = dr.GetString(0);
    string phone = dr.GetString(1);
    DateTime date = dr.GetDateTime(2);
}
dr.Close();
```

Queries in quotes

Arguments loosely bound

Results loosely typed

Compiler cannot help catch mistakes



Data Access with DLINQ

```
public class Customer
{
    public int Id;
    public string Name;
    public string Phone;
    …
}

Table<Customer> customers = db.Customers;

var contacts =
    from c in customers
    where c.City == "London"
    select new { c.Name, c.Phone };
```

Classes describe data

Tables are collections

Query is natural part of the language

The compiler helps you out

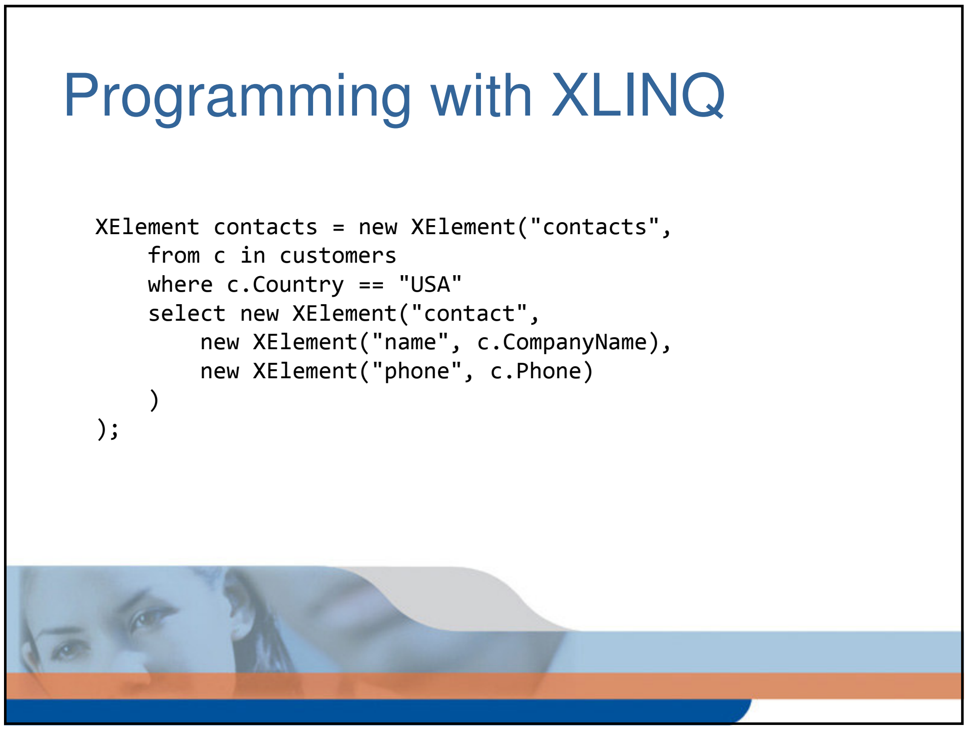
XLINQ

- Language integrated query for XML
- Leverages experience with DOM
- Standard Query Operators
- XML Specific Query Operators (Axes)

Programming XML Today

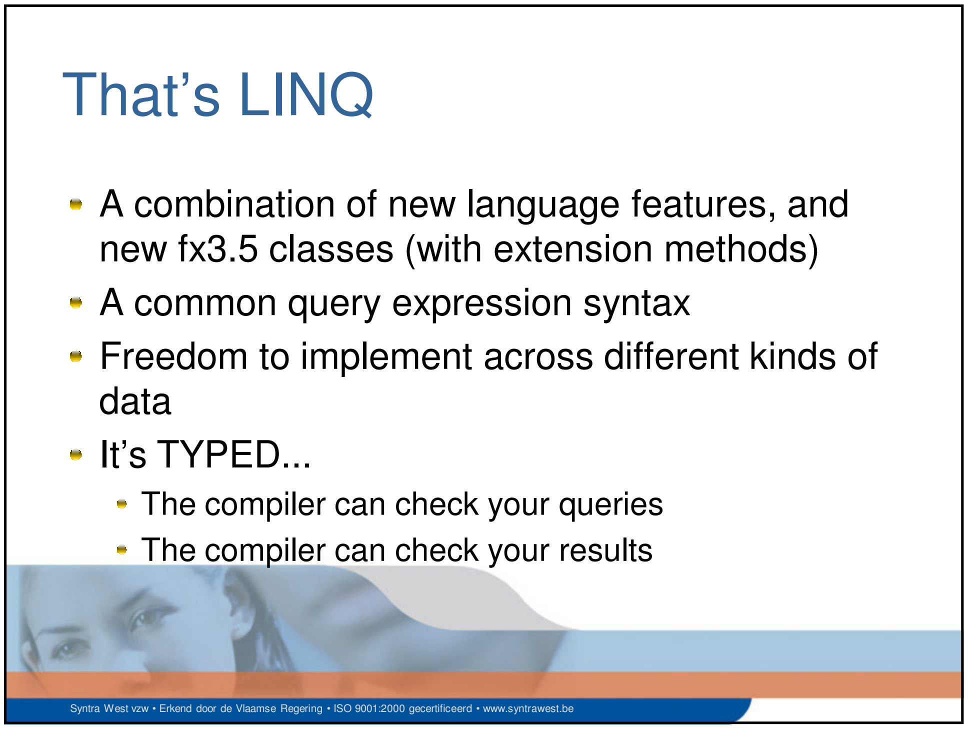
```
XmlDocument doc = new XmlDocument();
XmlElement contacts = doc.CreateElement("contacts");
foreach (Customer c in customers)
    if (c.Country == "USA") {
        XmlElement e = doc.CreateElement("contact");
        XmlElement name = doc.CreateElement("name");
        name.InnerText = c.CompanyName;
        e.AppendChild(name);
        XmlElement phone = doc.CreateElement("phone");
        phone.InnerText = c.Phone;
        e.AppendChild(phone);
        contacts.AppendChild(e);
    }
doc.AppendChild(contacts);
```

```
<contacts>
   <contact>
      <name>Great Lakes Food</name>
      <phone>(503) 555-7123</phone>
   </contact>
   …
</contacts>
```



Programming with XLINQ

```
XElement contacts = new XElement("contacts",
    from c in customers
    where c.Country == "USA"
    select new XElement("contact",
        new XElement("name", c.CompanyName),
        new XElement("phone", c.Phone)
    )
);
```



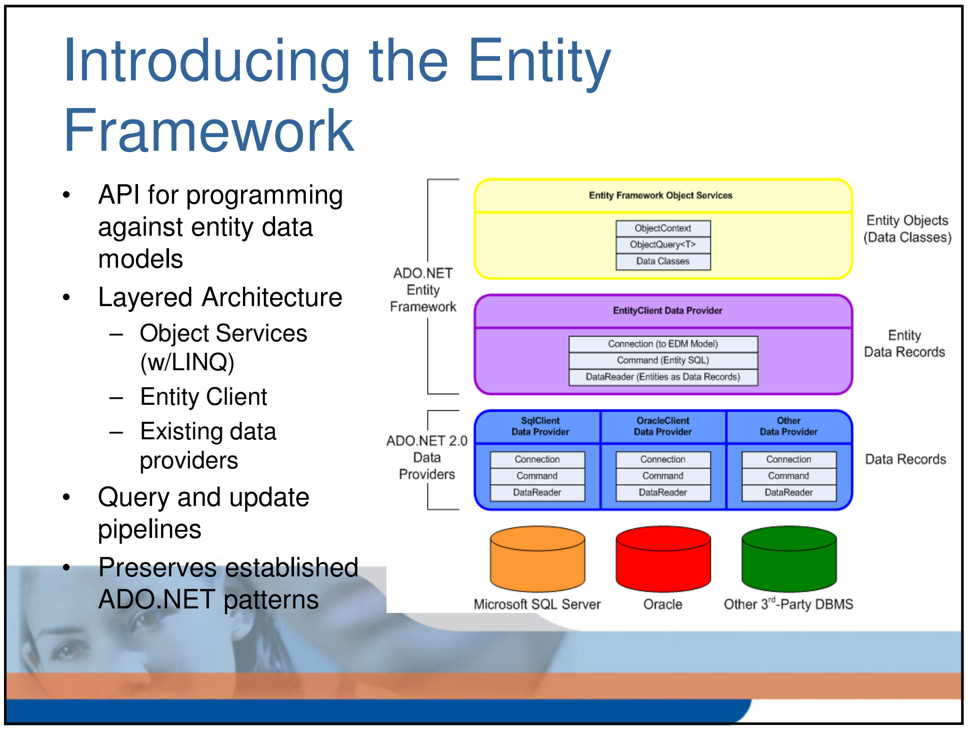
That’s LINQ

- A combination of new language features, and new fx3.5 classes (with extension methods)
- A common query expression syntax
- Freedom to implement across different kinds of data
- It’s TYPED...
  - The compiler can check your queries
  - The compiler can check your results

OR Mapping

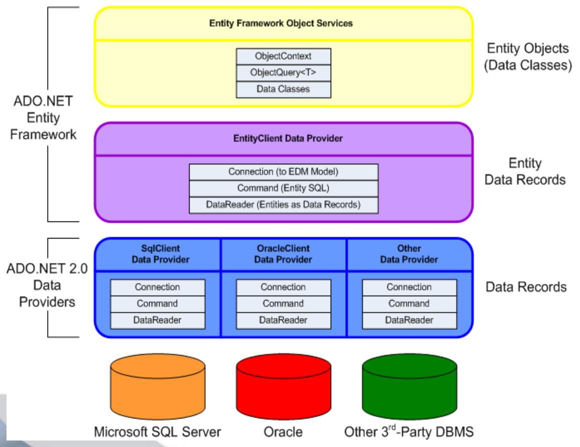
Data Access Evolution

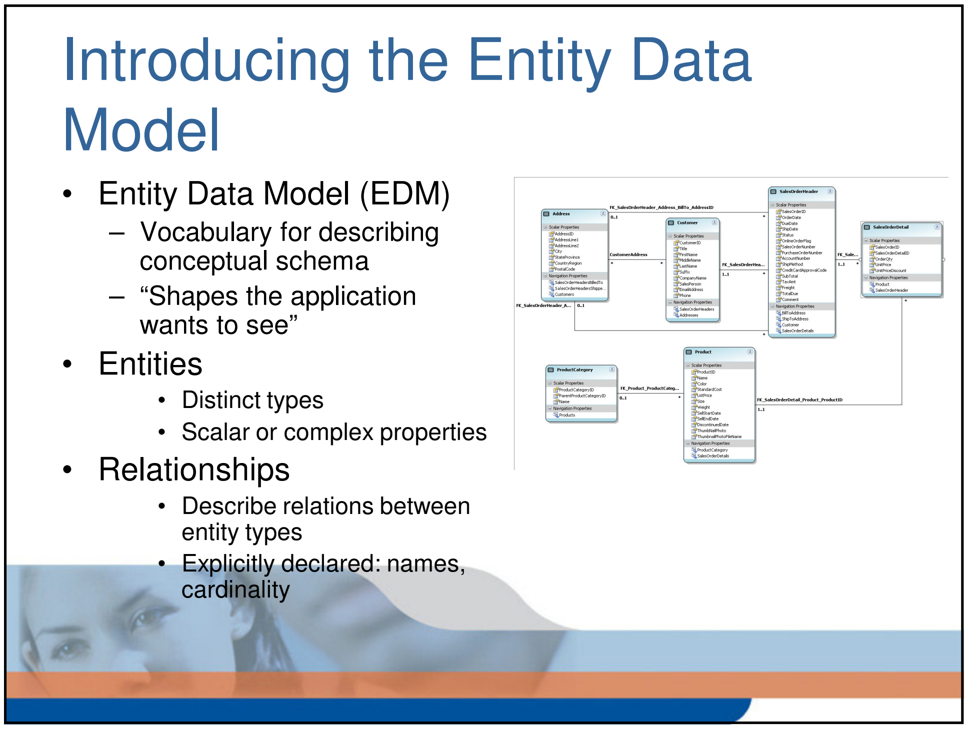
	Level of Abstraction	Query Language	Query Results
ADO.NET 2.0	Low-level: database schema	Query in quotes	Loosely-typed (brittle, error-prone)
ADO.NET Entity Framework	Higher-level: conceptual schema	Language-Integrated Query (LINQ)	Strongly-typed



Introducing the Entity Framework

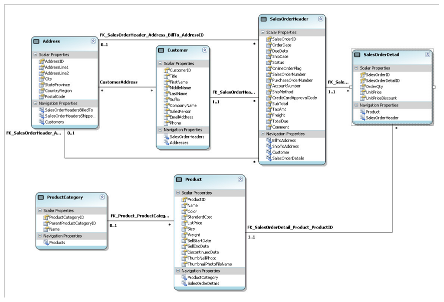
- API for programming against entity data models
- Layered Architecture
  - Object Services (w/LINQ)
  - Entity Client
  - Existing data providers
- Query and update pipelines
- Preserves established ADO.NET patterns

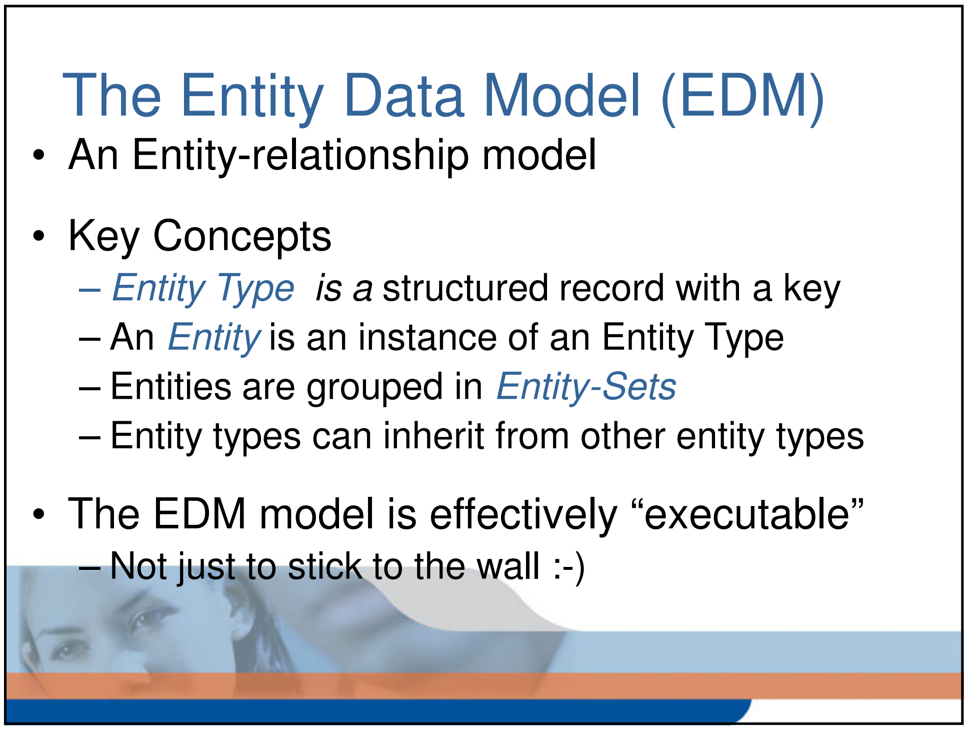




Introducing the Entity Data Model

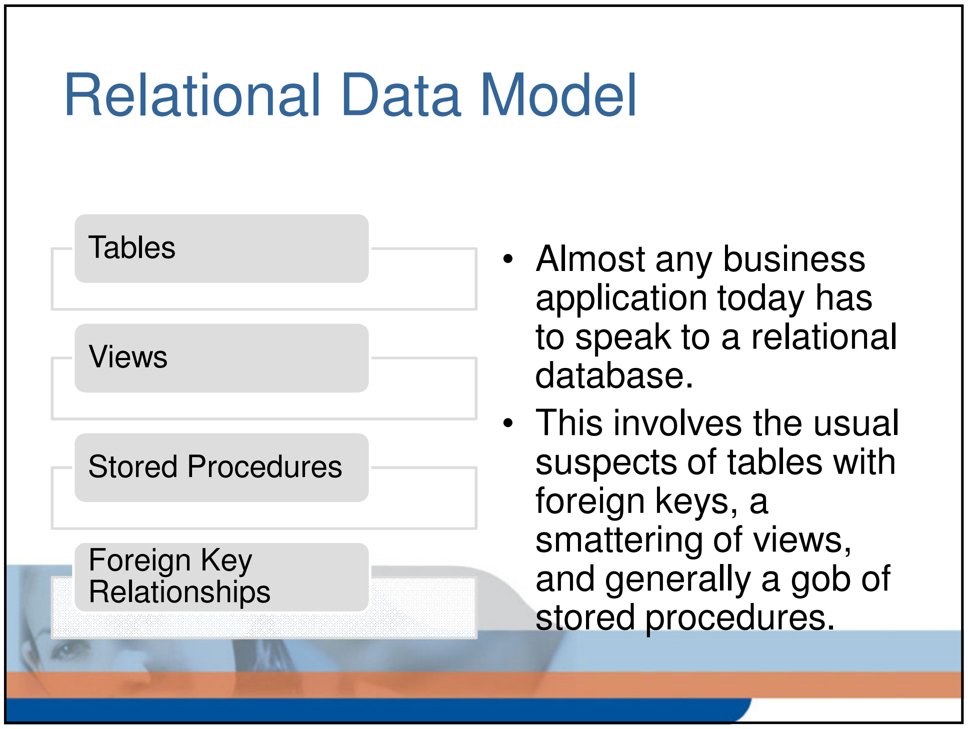
- Entity Data Model (EDM)
  - Vocabulary for describing conceptual schema
  - “Shapes the application wants to see”
- Entities
  - Distinct types
  - Scalar or complex properties
- Relationships
  - Describe relations between entity types
  - Explicitly declared: names, cardinality





The Entity Data Model (EDM)

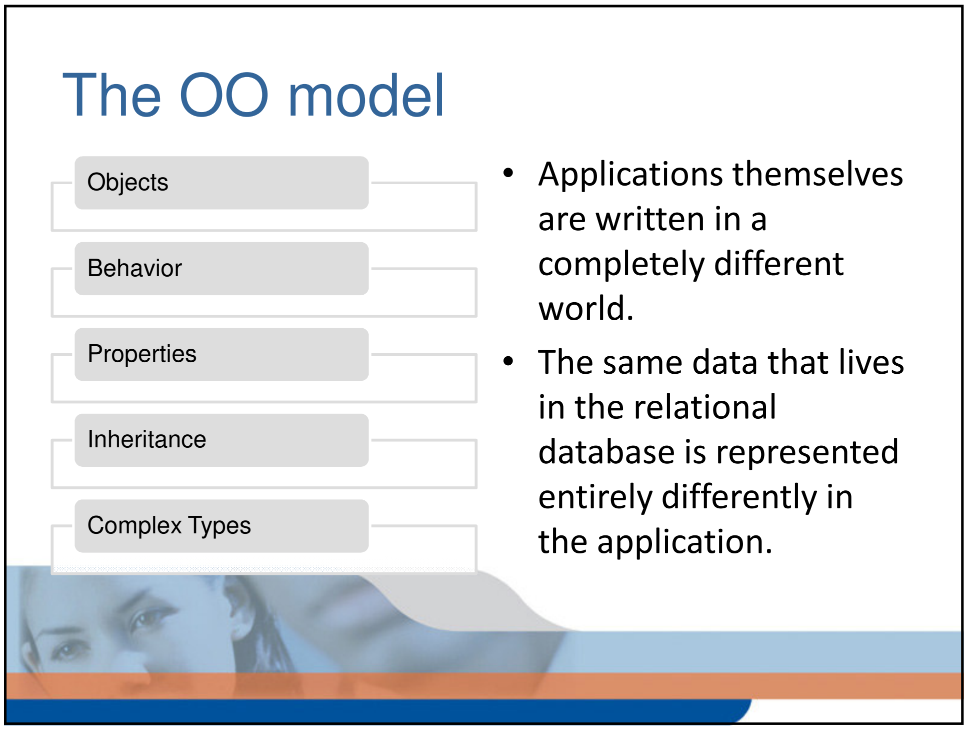
- An Entity-relationship model
- Key Concepts
  - *Entity Type* *is a* structured record with a key
  - An *Entity* is an instance of an Entity Type
  - Entities are grouped in *Entity-Sets*
  - Entity types can inherit from other entity types
- The EDM model is effectively "executable"
  - Not just to stick to the wall :-)



Relational Data Model

- Tables
- Views
- Stored Procedures
- Foreign Key Relationships

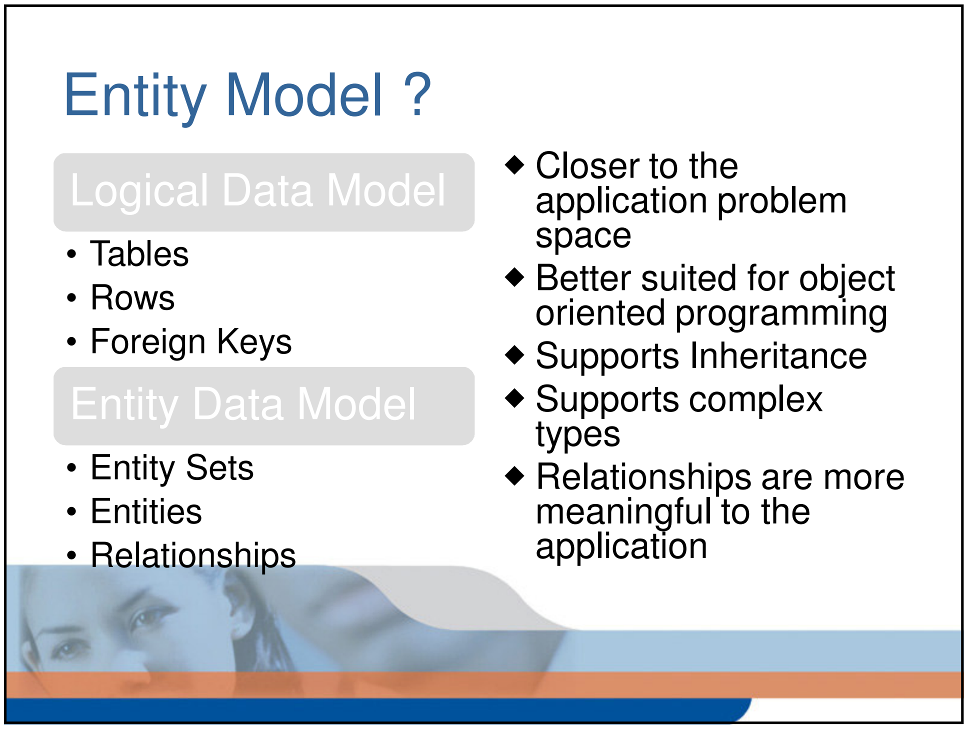
- Almost any business application today has to speak to a relational database.
- This involves the usual suspects of tables with foreign keys, a smattering of views, and generally a gob of stored procedures.



The OO model

- Objects
- Behavior
- Properties
- Inheritance
- Complex Types

- Applications themselves are written in a completely different world.
- The same data that lives in the relational database is represented entirely differently in the application.



Entity Model ?

Logical Data Model

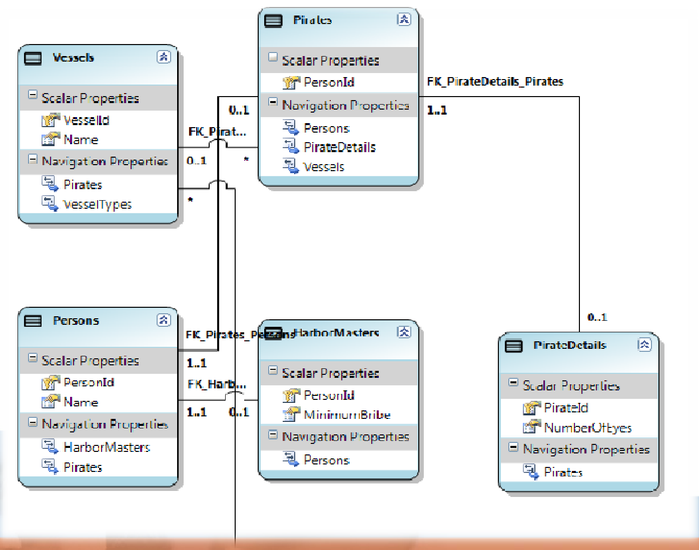
- Tables
- Rows
- Foreign Keys

Entity Data Model

- Entity Sets
- Entities
- Relationships

- Closer to the application problem space
- Better suited for object oriented programming
- Supports Inheritance
- Supports complex types
- Relationships are more meaningful to the application

EDM Designer in Visual Studio



Logical Model

Object Model
(O – Space)

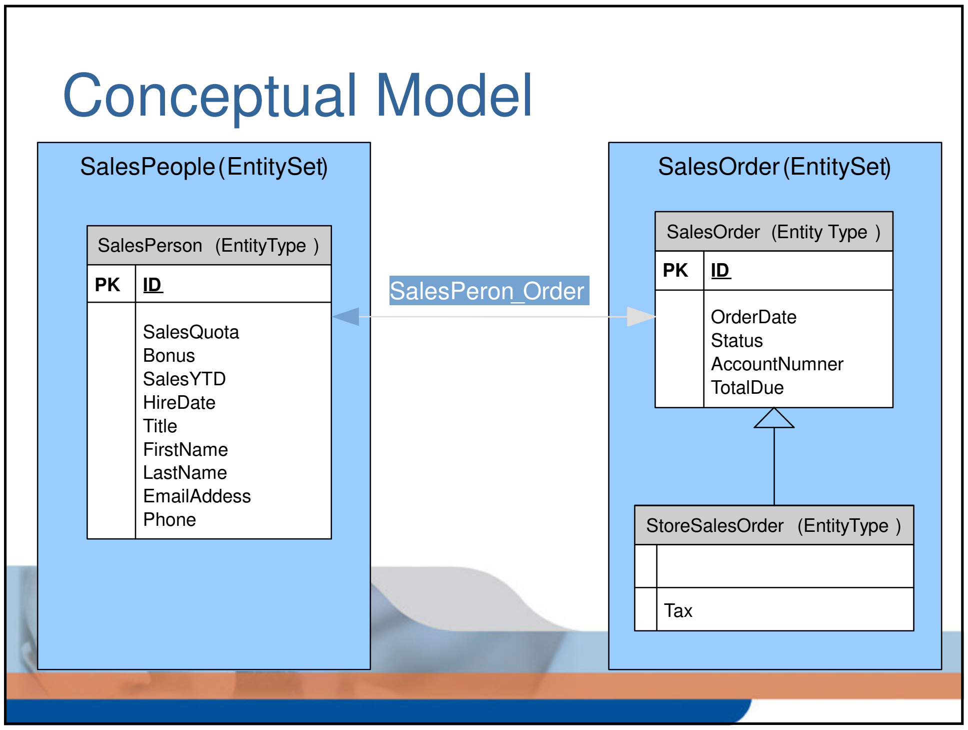
Linq

Conceptual Model
(C - Space)

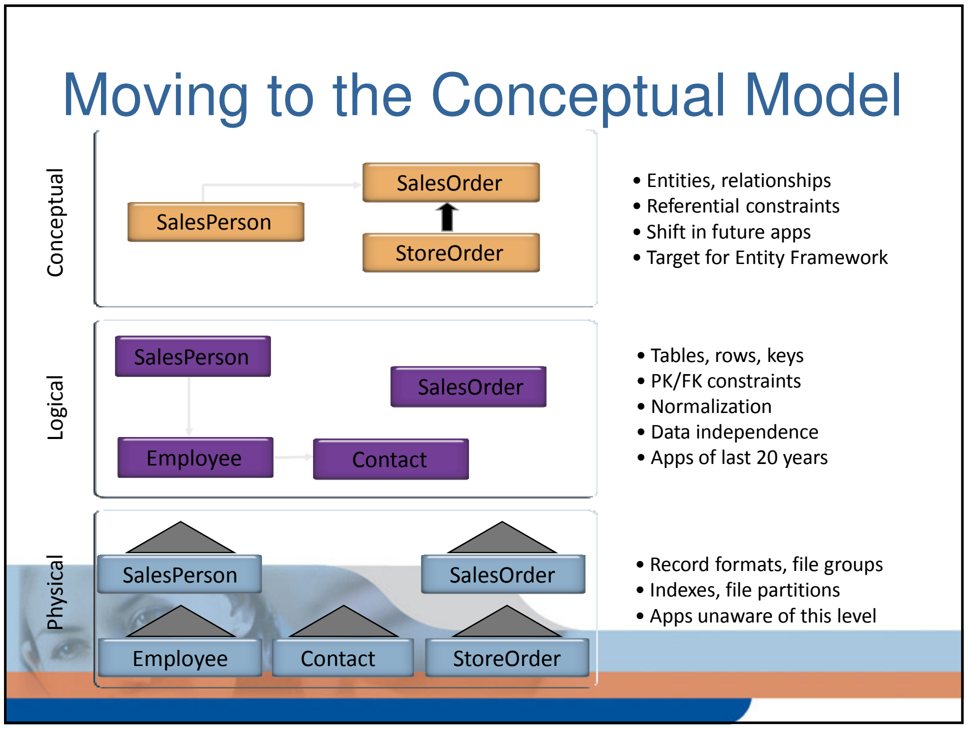
ADO. Net Provider
e.g.EntityClient

Store Model
(S-Space)

ADO. Net Provider
e.g. SqlClient



Conceptual Model



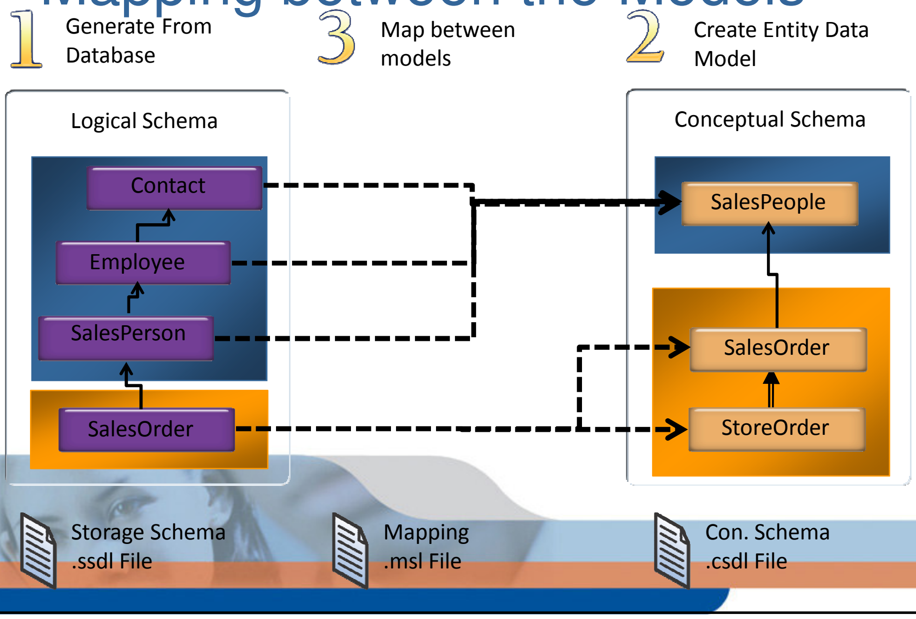
Moving to the Conceptual Model

- Entities, relationships
- Referential constraints
- Shift in future apps
- Target for Entity Framework

- Tables, rows, keys
- PK/FK constraints
- Normalization
- Data independence
- Apps of last 20 years

- Record formats, file groups
- Indexes, file partitions
- Apps unaware of this level

Mapping between the Models



Demo time

Q&A