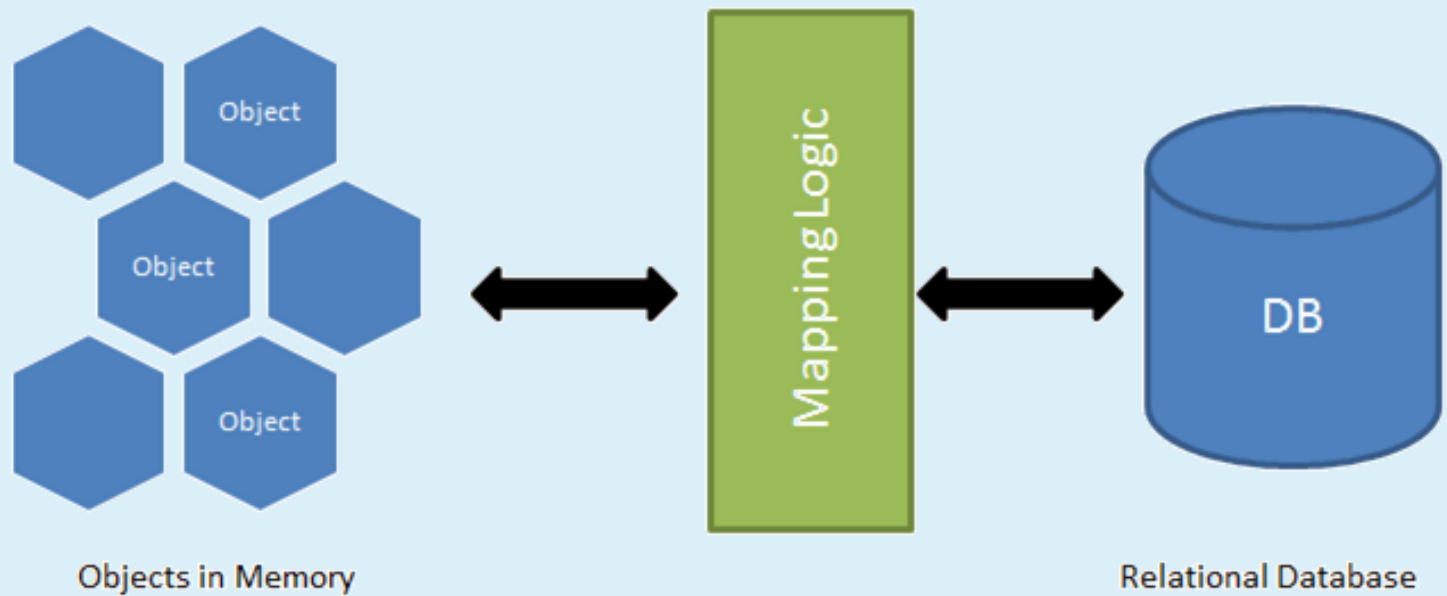


## O/R Mapping



## Object Relational Mapping (ORM)

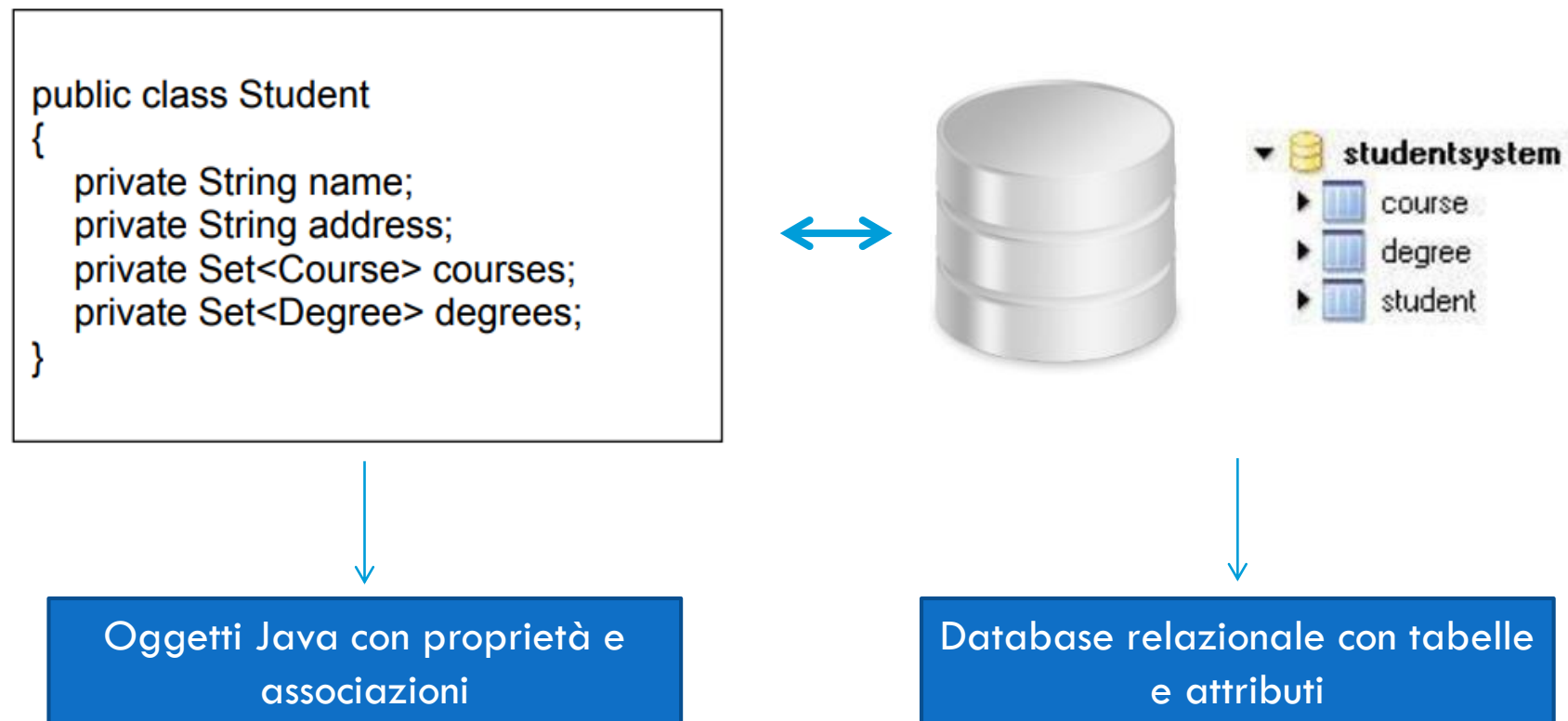
Annalisa Franco, Dario Maio  
Università di Bologna

# Gestione della persistenza dei dati

- La maggior parte delle applicazioni richiede qualche forma di **persistenza** dei dati.
- Possibili soluzioni:
  - Usare meccanismi di **serializzazione** del linguaggio a oggetti
    - Limitazioni sull'accesso a singoli oggetti (un oggetto serializzato può essere acceduto solo nella sua interezza)
  - **Database object-oriented**
    - Query language ancora incompleto
    - RDBMS ancora predominanti
      - Maggiormente affidabili (proprietà ACID)
      - Linguaggio SQL standard
      - Ampio supporto

# Object vs. Relational

- Quando si lavora in sistemi software object-oriented ci si trova di fronte a un mismatch tra il *modello a oggetti* e il *modello relazionale* adottato dal database.



# Object Relational Mapping

- **ORM** è una tecnica che favorisce l'integrazione di sistemi software aderenti al **paradigma di programmazione a oggetti** con sistemi **RDBMS**.



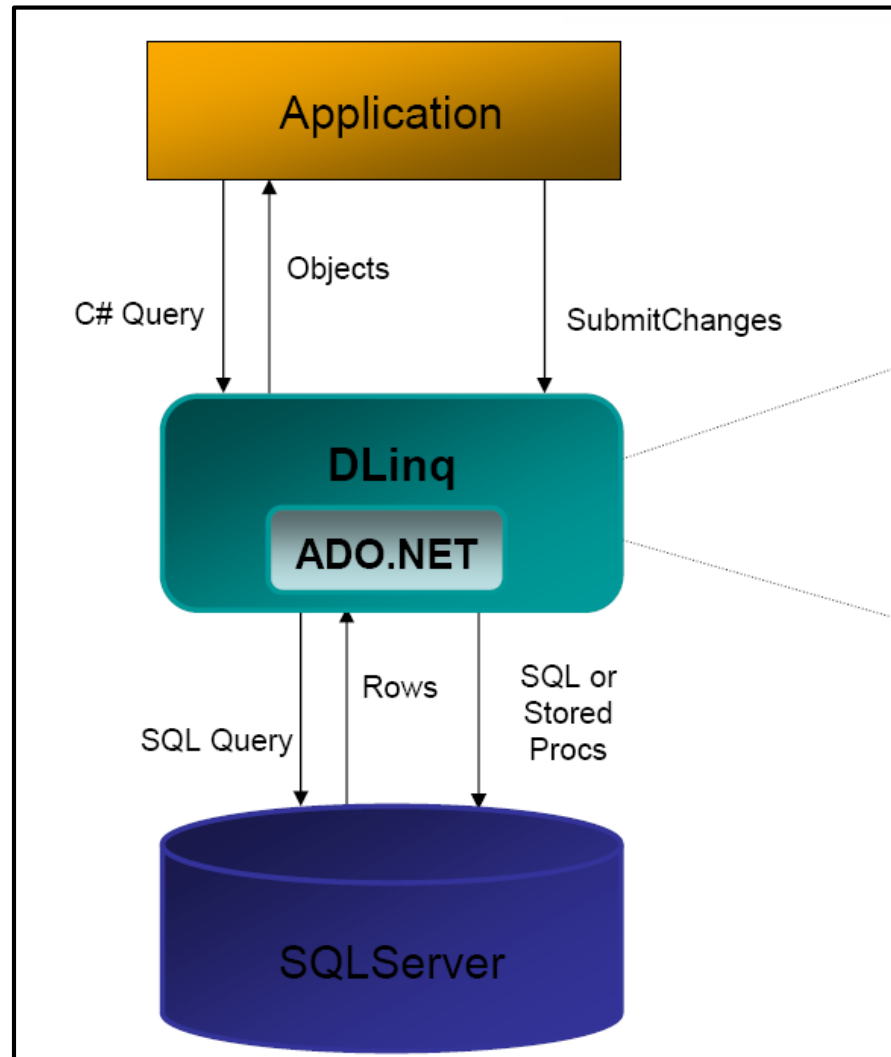
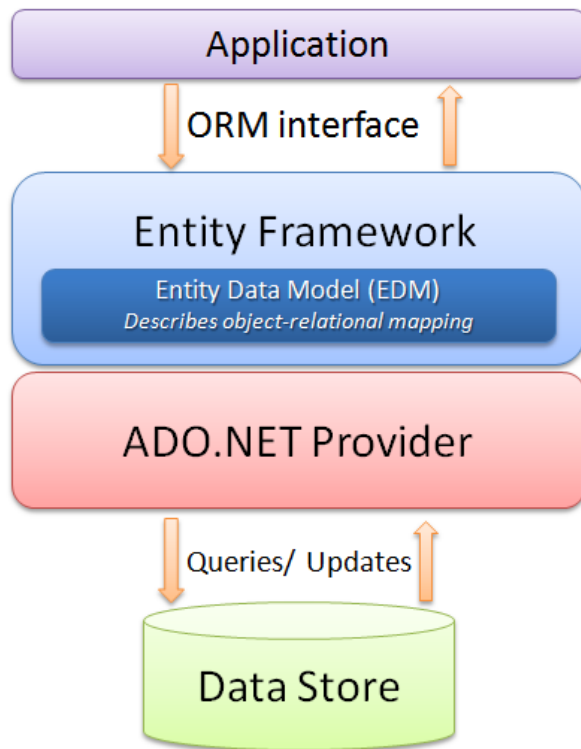
# ORM: Object Relational Mapping

- Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, i servizi per la **persistenza dei dati**, astruendo al contempo dalle caratteristiche implementative dello specifico RDBMS utilizzato.
  
- **Vantaggi:**
  - superamento dell'incompatibilità tra progettazione orientata agli oggetti e modello relazionale per la rappresentazione dei dati;
  - elevata portabilità rispetto alla tecnologia DBMS utilizzata;
  - sensibile riduzione dei tempi di sviluppo del codice;
  - approccio stratificato, isolando in un solo livello la logica di persistenza dei dati, a vantaggio della modularità complessiva del sistema.

# Framework ORM

- Microsoft .NET
  - ▣ LINQ
  - ▣ Entity Framework
- Java
  - ▣ Hibernate
  - ▣ JPA
- Ruby on Rails
  - ▣ ActiveRecord
- PHP
  - ▣ CodeIgniter
  - ▣ CakePHP
- iOS
  - ▣ CoreData

# ORM Microsoft .NET



## Services:

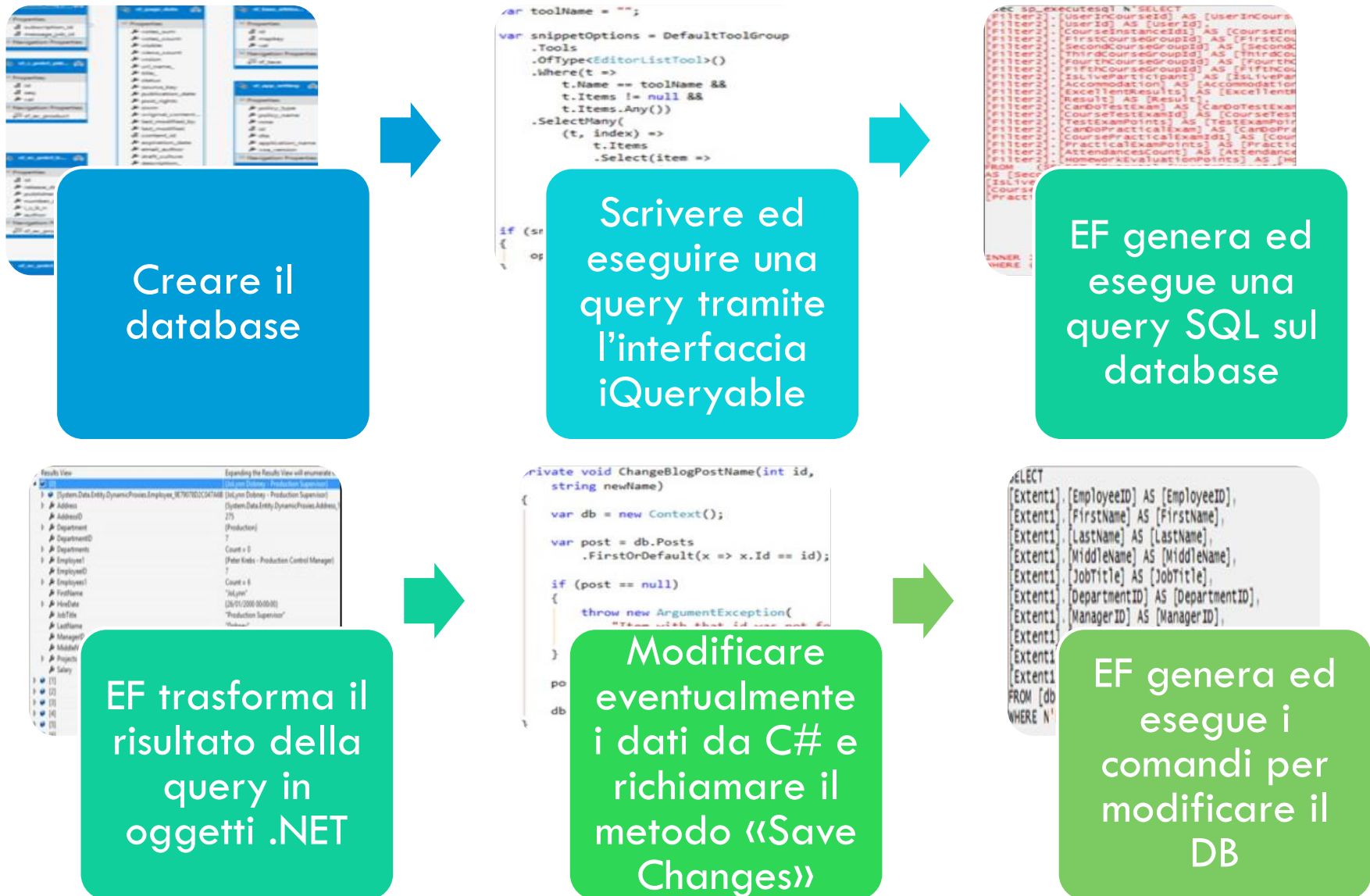
- Object identity
- Change tracking
- Concurrency ctrl

# Entity Framework

- Object-Relational Persistence Framework per .NET
  - ▣ Mappa database relazionali in oggetti C#
  - ▣ Mette a disposizione API per la **manipolazione** dei dati negli schemi mappati
  - ▣ Implementa **operazioni CRUD** e interrogazioni complesse con linguaggio **LINQ**
- Possibili approcci:
  - ▣ **database first**: database → classi C#
  - ▣ **code first**: classi C# → database
- Il Visual Studio genera in automatico i modelli EF
  - ▣ Il mapping si realizza per mezzo di classi C#, attributi e XML



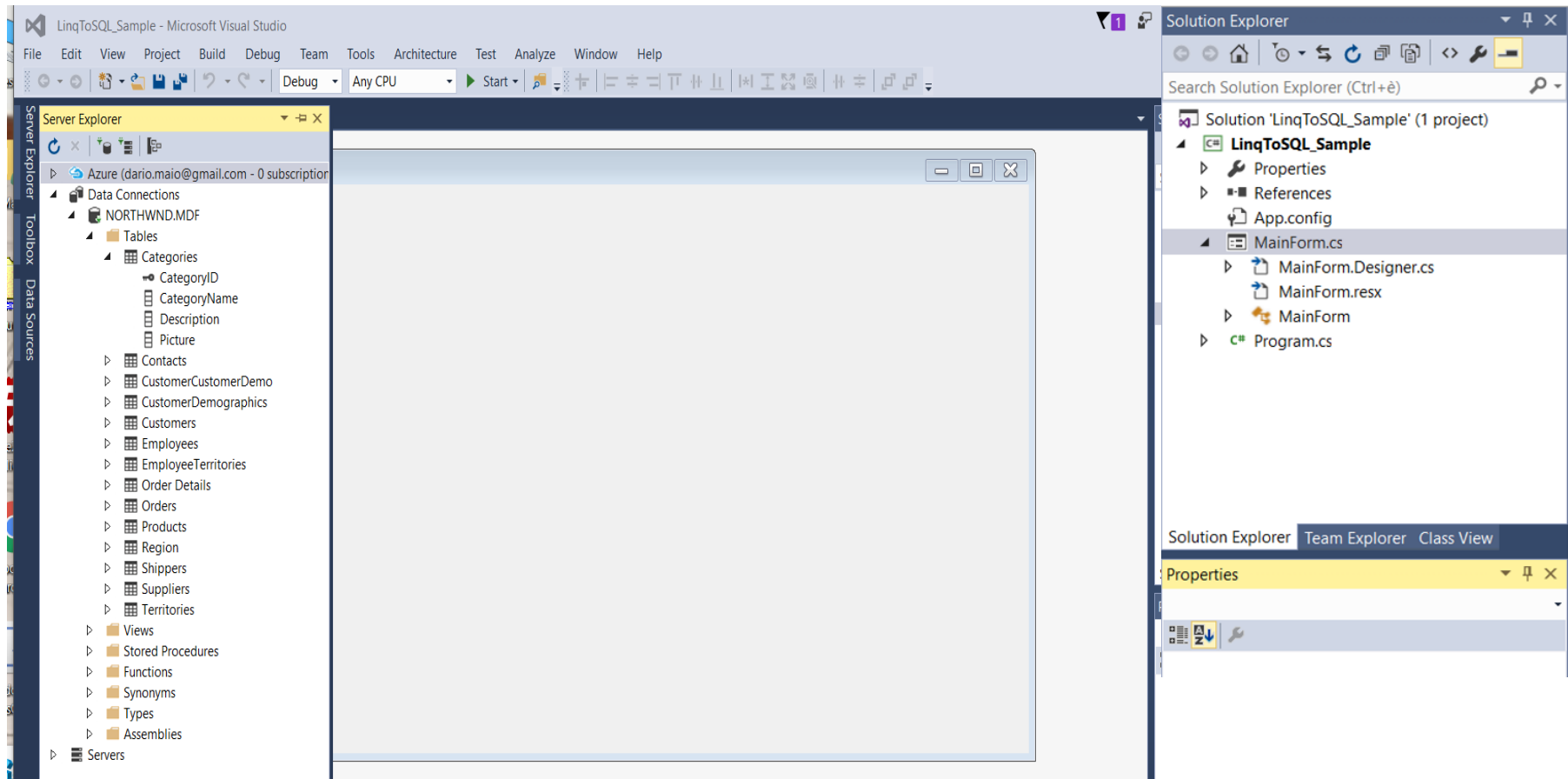
## Approccio DB first: workflow



# Componenti EntityFramework

- Classe **DbContext**
  - ▣ Gestisce la **connessione** al DB
  - ▣ Mappa le relazioni in «**entity classes**»
  - ▣ Fornisce accesso ai dati attraverso il linguaggio **LINQ** (interfaccia IQueryable)
  - ▣ Fornisce API per le operazioni **CRUD**
- **Entity classes:**
  - ▣ Rappresentato **entità**, ovvero oggetti con attributi e relazioni con altri oggetti
  - ▣ Ogni **tabella** del database è tipicamente mappata in una Entity class.

# Esempio in Visual Studio



NorthWND.MDF è il DB collegato

# Visual Designer (1)

## Aggiungere una nuova classe di tipo LINQ to SQL Classes

Add New Item - LinqToSQL\_Sample

Installed

- Visual C# Items
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - WPF
  - Cross-Platform
  - iOS
  - SQL Server
  - Workflow
  - XNA Game Studio 4.0
- Graphics
- Online

Sort by: Default

Custom Control	Visual C# Items
DataSet	Visual C# Items
Debugger Visualizer	Visual C# Items
EF 5.x DbContext Generator	Visual C# Items
EF 6.x DbContext Generator	Visual C# Items
HTML Page	Visual C# Items
Icon File	Visual C# Items
Installer Class	Visual C# Items
JavaScript File	Visual C# Items
<b>LINQ to SQL Classes</b>	Visual C# Items
MDI Parent Form	Visual C# Items
OWIN Startup class	Visual C# Items
Resources File	Visual C# Items

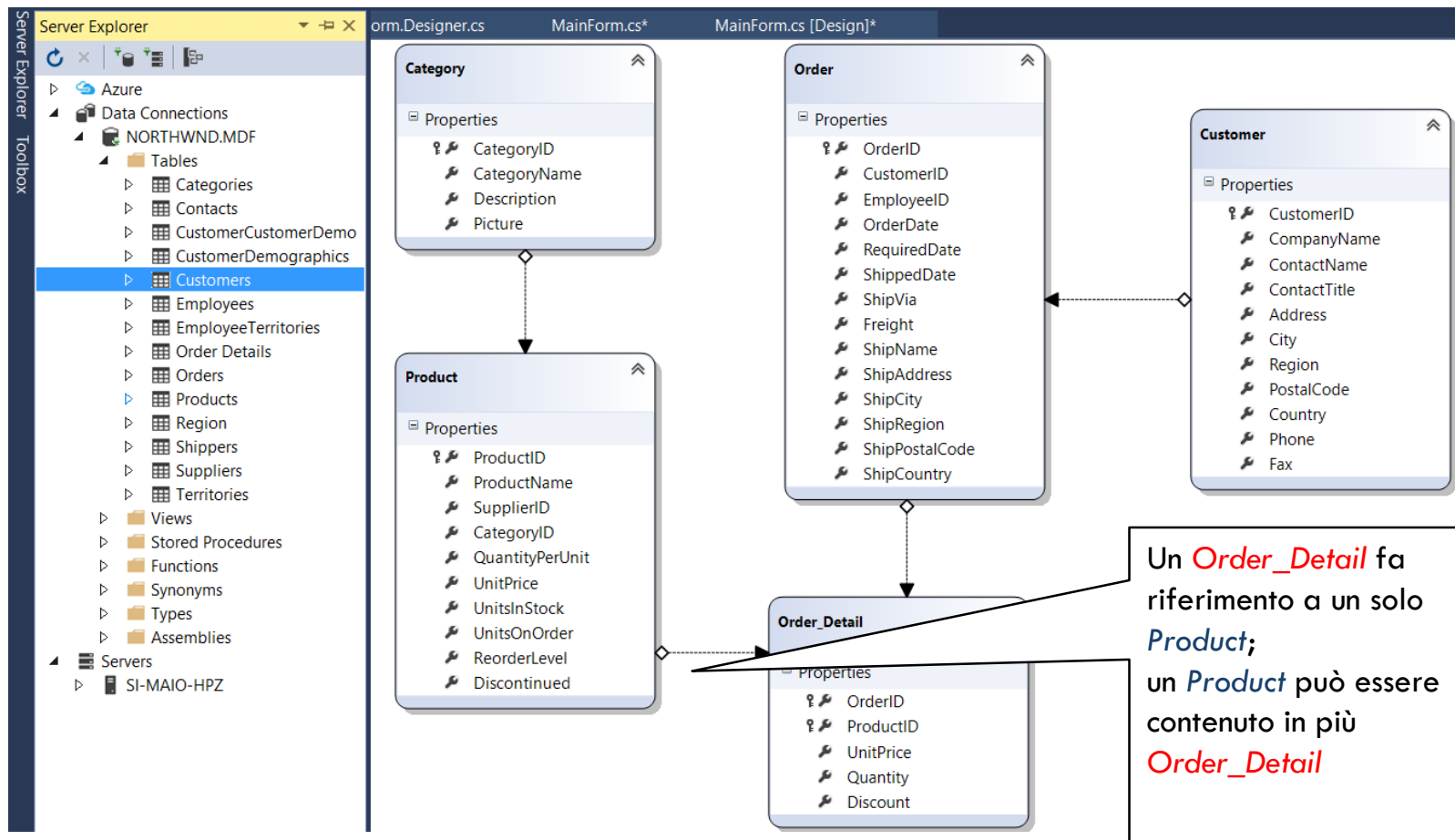
Nome della classe

Name: NorthWindDataClasses.dbml

[Click here to go online and find templates.](#)

# Visual Designer (2)

## Drag and drop delle tabelle d'interesse



# Visual Designer (3)

Solution Explorer

Search Solution Explorer (Ctrl+è)

Solution 'LinqToSQL\_Sample' (1 project)

- LinqToSQL\_Sample
  - Properties
  - References
  - App.config
  - MainForm.cs
    - MainForm.Designer.cs
    - MainForm.resx
    - MainForm
  - NorthWindDataClasses.dbml
    - NorthWindDataClasses.cs
    - NorthWindDataClasses.dbml.layout
    - NorthWindDataClasses.designer.cs
    - NorthWindDataClassesDataContext
    - Category
    - Product
    - Order

Solution Explorer Team Explorer Class View

LinqToSQL\_Sample - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help

Debug Any CPU Start

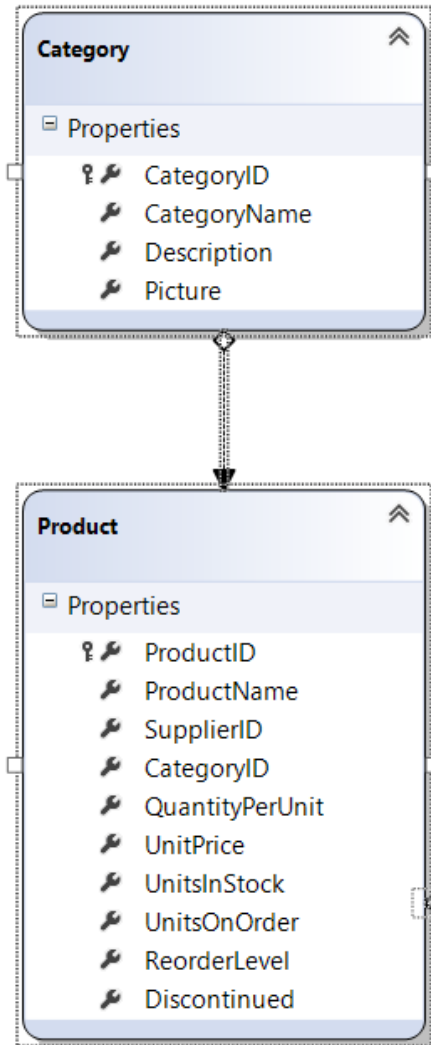
NorthWindDataClasses.designer.cs\* MainForm.cs\* NorthWindDataClasses.dbml\* MainForm.cs [Design]\*

LinqToSQL\_Sample LinqToSQL\_Sample.NorthWindDataClassesDataContext mappingSource

```
13 {
14     using System.Data.Linq;
15     using System.Data.Linq.Mapping;
16     using System.Data;
17     using System.Collections.Generic;
18     using System.Reflection;
19     using System.Linq;
20     using System.Linq.Expressions;
21     using System.ComponentModel;
22     using System;
23
24
25     [global::System.Data.Linq.Mapping.DatabaseAttribute(Name="NORTHWND")]
26     public partial class NorthWindDataClassesDataContext : System.Data.Linq.DataContext
27     {
28
29         private static System.Data.Linq.Mapping.MappingSource mappingSource = new Attri
30
31     Extensibility Method Definitions
32
33
34
35
36 }
```

Classe generata  
automaticamente che  
consente di  
interfacciarsi al DB

# Le classi per il mapping O-R (1)



```
[global::System.Data.Linq.Mapping.TableAttribute(Name="dbo.Categories")]
10 references
public partial class Category : INotifyPropertyChanging, INotifyPropertyChanged
{
    private static PropertyChangingEventArgs emptyChangingEventArgs =
        new PropertyChangingEventArgs(String.Empty);

    private int _CategoryID;

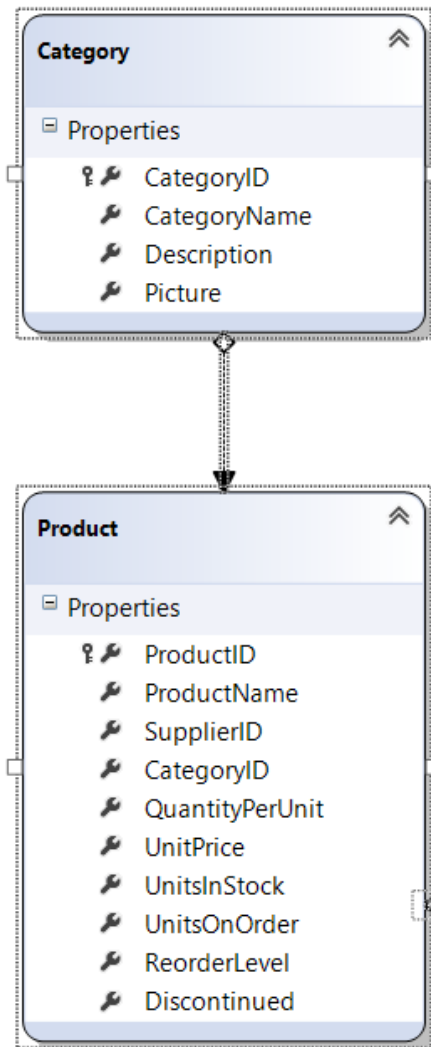
    private string _CategoryName;

    private string _Description;

    private System.Data.Linq.Binary _Picture;

    private EntitySet<Product> _Products;
```

# Le classi per il mapping O-R (2)



```
[global::System.Data.Linq.Mapping.TableAttribute(Name="dbo.Products")]
17 references
public partial class Product : INotifyPropertyChanging, INotifyPropertyChanged
{
    private static PropertyChangingEventArgs emptyChangingEventArgs = new PropertyChangingEventArgs(String.Empty);

    private int _ProductID;

    private string _ProductName;

    private System.Nullable<int> _SupplierID;

    private System.Nullable<int> _CategoryID;

    private string _QuantityPerUnit;

    private System.Nullable<decimal> _UnitPrice;

    private System.Nullable<short> _UnitsInStock;

    private System.Nullable<short> _UnitsOnOrder;

    private System.Nullable<short> _ReorderLevel;

    private bool _Discontinued;

    private EntitySet<Order_Detail> _Order_Details;

    private EntityRef<Category> _Category;
```

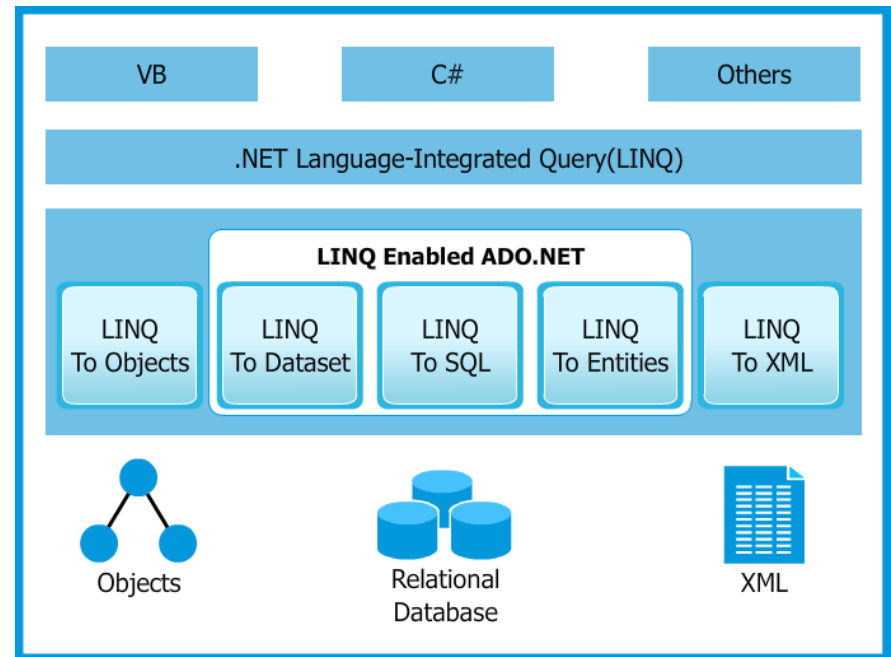


# Il progetto Microsoft LINQ

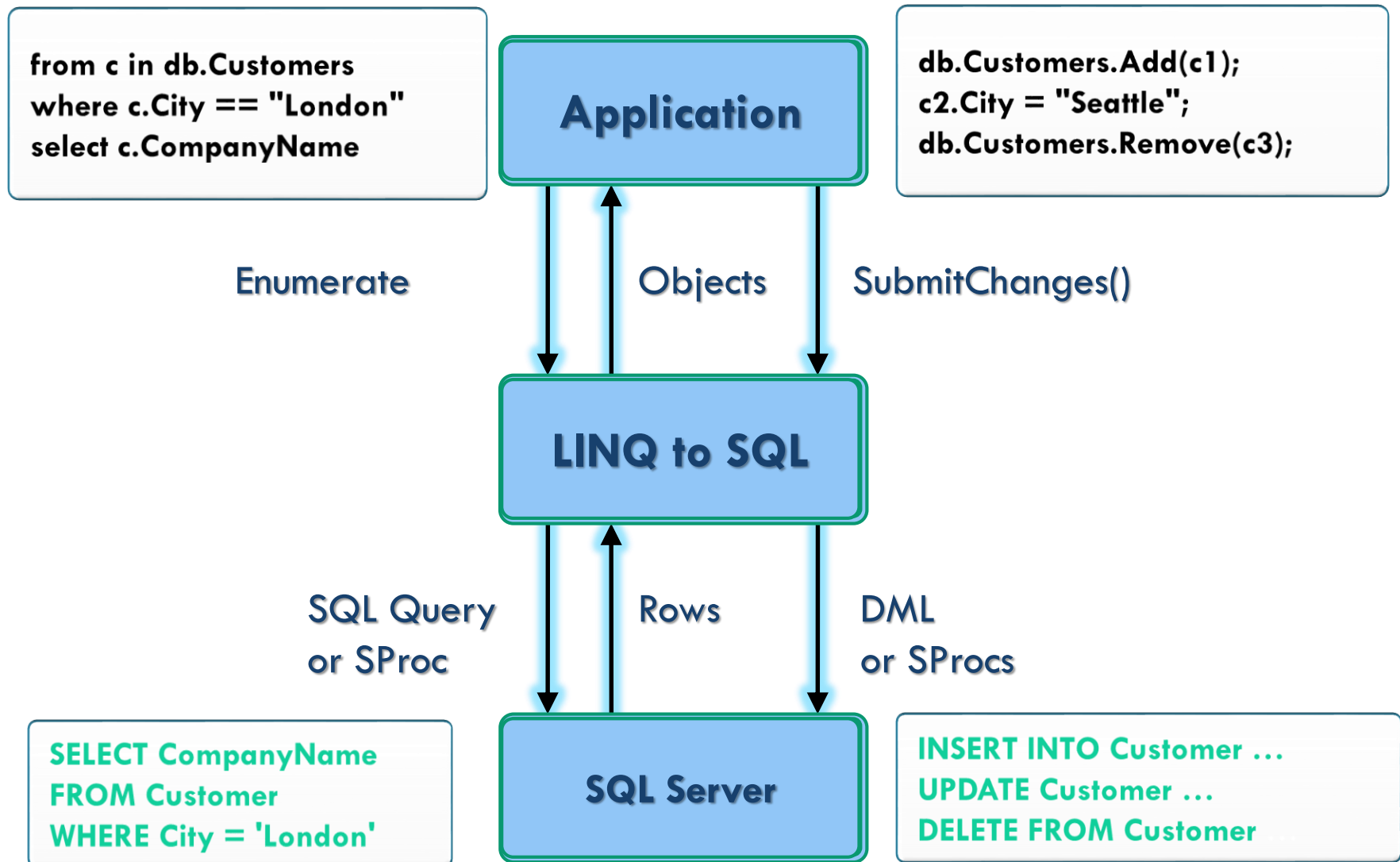
- **LINQ** è un set di estensioni per .NET Framework che comprende operazioni di query, d'impostazione e di trasformazione. Estende C# e Visual Basic con sintassi del linguaggio nativa per le query e offre librerie di classi che consentono di sfruttare al meglio tali funzionalità.
- **Language Enhancements**
  - Local Variable Type Inference
  - Object Initializers
  - Anonymous Types
  - Lambda Expressions
  - Extension Methods

+ Query Expressions

= LINQ ☺




# Architettura LINQ to SQL




# Local variable type inference

```
int i = 2010;  
string s = "Ciao";  
double d = 3.14;  
int[ ] numbers = new int[ ] {10, 20, 30};  
Dictionary<int,Order> orders = new Dictionary<int,Order>();
```



```
var i = 2010;  
var s = "Ciao";  
var d = 3.14;  
var numbers = new int[ ] {10, 20, 30};  
var orders = new Dictionary<int,Order>();
```



“Il tipo è dedotto dalla  
parte a destra della  
dichiarazione”

# Object initializers

```
public class Point
{
    private int x, y;

    public int X { get { return x; } set { x = value; } }
    public int Y { get { return y; } set { y = value; } }
}
```

Field or property assignments

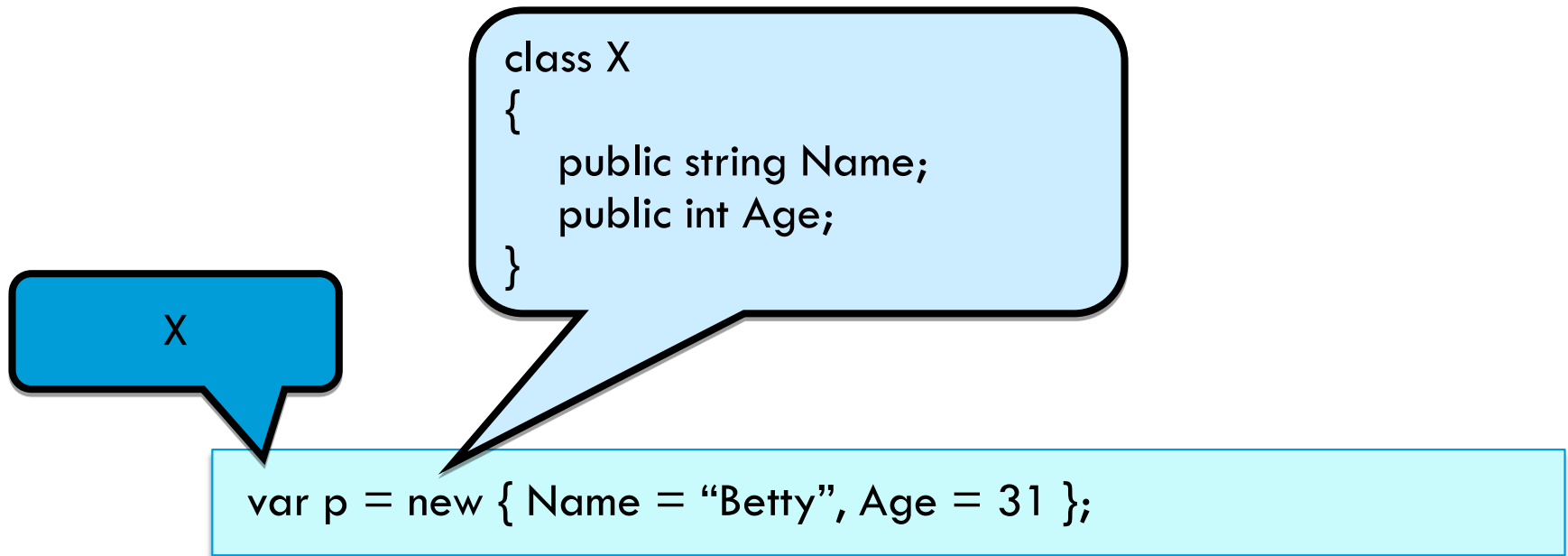
```
Point myPoint = new Point { X = 0, Y = 1 };
```

Autoimplemented properties

```
Point myPoint = new Point();
myPoint.X = 0;
myPoint.Y = 1;
```

```
public class Point
{
    public int X {get; set;}
    public int Y {get; set;}
}
```

# Anonymous Types



# Lambda Expressions

- ❑ Una **lambda expression** è una funzione anonima che può contenere espressioni e istruzioni e che può essere utilizzata per creare delegati. Tutte le espressioni lambda utilizzano l'operatore lambda **=>**, che è letto come "goes to".
- ❑ Il lato sinistro dell'operatore lambda specifica i parametri di input, se presenti, e il lato destro contiene l'espressione o il blocco di istruzioni.
- ❑ L'espressione lambda **x => x\*x** viene letta "x goes to x times x". Questa espressione può essere assegnata a un tipo delegato:

```
delegate int del(int i);  
static void Main(string[] args)  
{  
    del myDelegate = x => x * x;  
    int k = myDelegate(6);    // k = 36  
}
```

# Extension Methods

```
namespace StringExtensions
{
    public static class StringExtensionsClass
    {
        public static string RemoveNonNumeric(this string s)
        {
            MatchCollection col = Regex.Matches(s, "[0-9]");
            StringBuilder sb = new StringBuilder();
            foreach (Match m in col) sb.Append(m.Value);
            return sb.ToString();
        }
    }
}
```

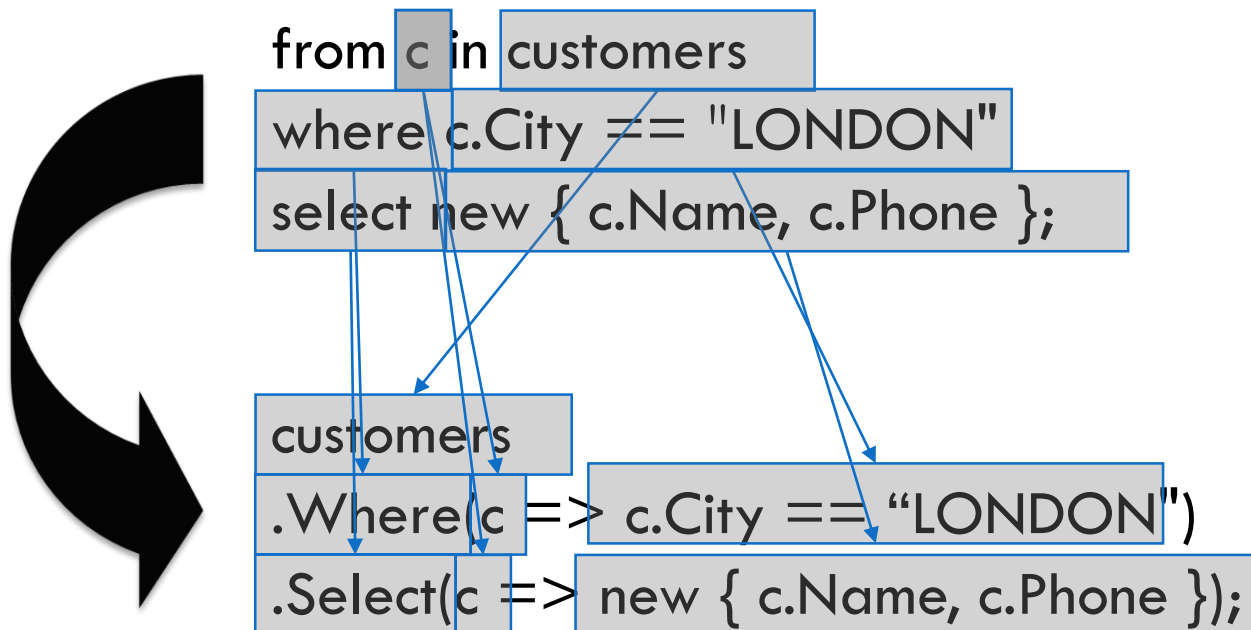
Si estende la classe  
String con il metodo  
RemoveNonNumeric

```
using StringExtensions;
....
.....
string phone = "123-123-1234";
string newPhone = phone.RemoveNonNumeric();
```

Esempio di uso

# Query Expressions → LINQ

- Query che invocano metodi
  - ▣ Where, Join, OrderBy, Select, GroupBy, ...

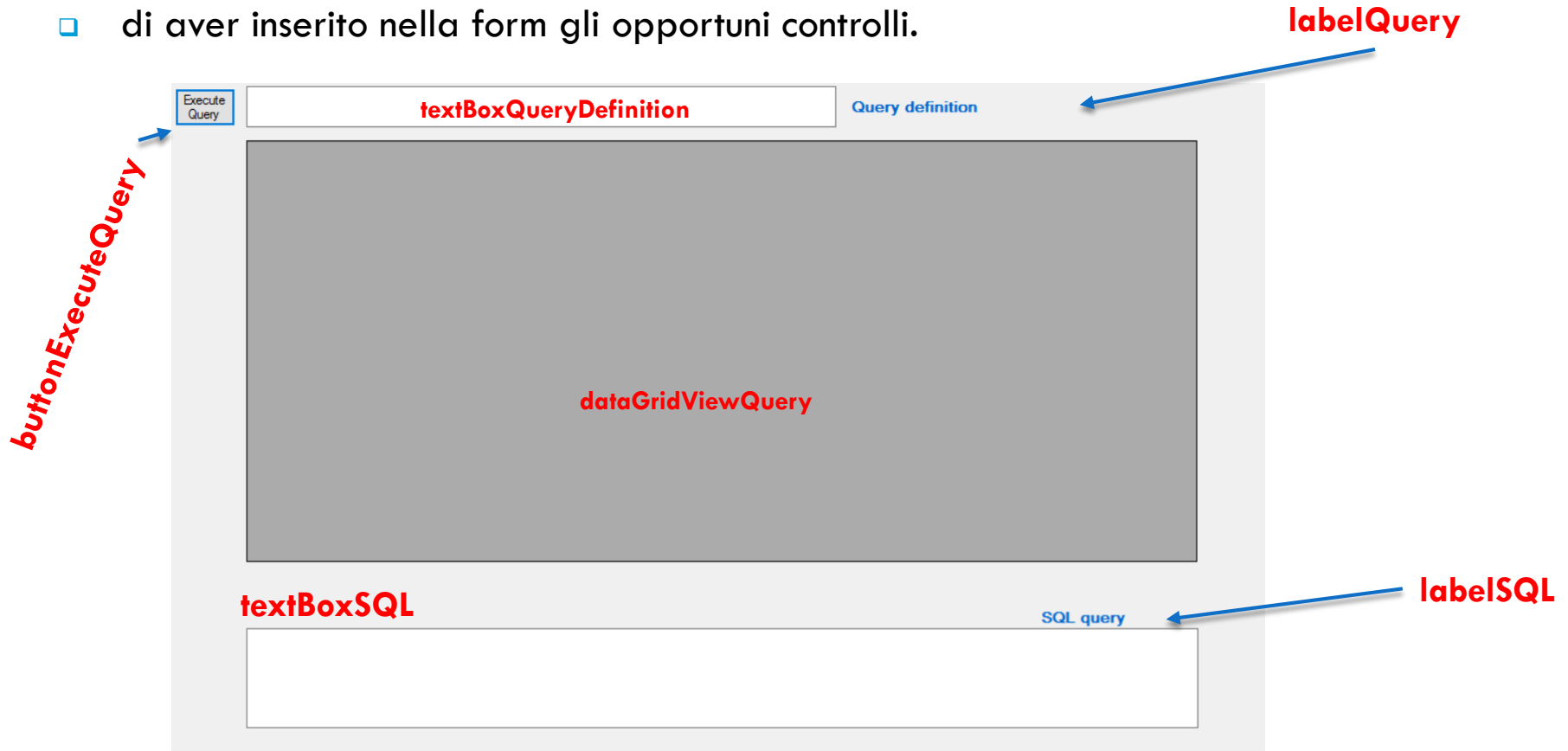




# Main Form

Si suppone:

- ❑ di aver dichiarato `NorthWindDataClassesDataContext db;`
- ❑ di aver istanziato il contesto `db = new NorthWindDataClassesDataContext();`
- ❑ di aver inserito nella form gli opportuni controlli.




# Esempi di query in LINQ

Si suppone inoltre di disporre di un metodo **showResults** per visualizzare il risultato di una query.

```
/// <summary>
/// mostra il risultato della query e la sua traduzione in SQL operata dal compilatore
/// </summary>
/// <param name="queryResult"> risultato della query</param>
/// <param name="queryDefinition"> definizione a parole della query</param>
/// <param name="querySQL">traduzione in SQL</param>
/// <param name="removeLastColumn"> se true si rimuove l'ultima colonna della tabella</param>
/// <param name="message"> se non vuoto rappresenta un messaggio da visualizzare prima di queryDefinition</param>
2 references
private void showResults(System.Linq.IQueryable queryResult, string queryDefinition, string querySQL, bool removeLastColumn, string message)
{
    // binding della query con il dataGridViewQuery della form
    // se necessario si cancella l'ultima colonna che rappresenta il link a category
    dataGridViewQuery.Visible = true;
    dataGridViewQuery.DataSource = queryResult;
    if (removeLastColumn)
        dataGridViewQuery.Columns.RemoveAt(dataGridViewQuery.Columns.Count - 1);
    textBoxQueryDefinition.Clear();
    textBoxSQL.Clear();
    if (message != null)
        queryDefinition = message + " : " + queryDefinition;
    textBoxQueryDefinition.AppendText(queryDefinition);
    textBoxSQL.AppendText(querySQL);
}
```

# Query – 1 (definizione)

**Tutti i prodotti per i quali il nome della categoria inizia con una lettera maggiore della lettera "M".**

```
private void ExecuteQuery_1()  
{  
    // query 1:  
    // dammi tutti i prodotti per cui il nome della categoria  
    // inizia con una lettera maggiore della lettera M  
  
    var query = from p in db.Products  
                  where String.Compare(p.Category.CategoryName, "M", true) > 0  
                 select p;  
  
    string queryDefinition =  
        "products whose category name is greater than or equal to letter M";  
  
    showResults(query, queryDefinition, query.ToString(), true, "query 1");  
}
```

Implicitly Typed Local Variable

# Query - 1 (risultato)

Execute Query

query 1 : products whose category name is greater than letter M

Query definition

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel
▶	9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97,0000	29	0	0
	17	Alice Mutton	7	6	20 - 1 kg tins	39,0000	0	0	0
	29	Thüringer Rostbr...	12	6	50 bags x 30 sau...	123,7900	0	0	0
	53	Perth Pasties	24	6	48 pieces	32,8000	0	0	0
	54	Tourtière	25	6	16 pies	7,4500	21	0	10
	55	Pâté chinois	25	6	24 boxes x 2 pies	24,0000	115	0	20
	7	Uncle Bob's Orga...	3	7	12 - 1 lb pkgs.	30,0000	15	0	10
	14	Tofu	6	7	40 - 100 g pkgs.	23,2500	35	0	0
	28	Rössle Sauerkraut	12	7	25 - 825 g cans	45,6000	26	0	0
	51	Manjimup Dried A...	24	7	50 - 300 g pkgs.	53,0000	20	0	10
	74	Longlife Tofu	4	7	5 kg pkg.	10,0000	4	20	5
	10	Ikura	4	8	12 - 200 ml jars	31,0000	31	0	0
	13	Konbu	6	8	2 kg box	6,0000	24	0	5
	18	Camaron Tigers	7	8	16 kg pkg.	62,5000	42	0	0
<	20	Nord-Ost Matinab...	12	8	10 - 200 g pkgs...	25,8000	10	0	15

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
WHERE [t1].[CategoryName] > @p0
```

SQL generato  
ed eseguito  
tramite  
ADO.NET

# Query – 2 (definizione)

**I primi 5 prodotti per cui il nome della categoria inizia con la lettera "D"**

```
private void ExecuteQuery_1()
{
    // query 2:
    // dammi tutti i prodotti per cui il nome della categoria
    // inizia con la lettera D, e prendi solo i primi 5
    var query = (from p in db.Products
                  where p.Category.CategoryName.StartsWith("D")
                  select p).Take(5);

    string queryDefinition =
        "first 5 products whose category name starts with letter D";

    showResults(query, queryDefinition, query.ToString(), true, "query 2");
}
```

# Query – 2 (risultato)

Execute  
Query

query 2 : first 5 products whose category name starts with letter D

Query definition

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	Reord
▶	11	Queso Cabrales	5	4	1 kg pkg.	21,0000	22	30	30
	12	Queso Mancheg...	5	4	10 - 500 g pkgs.	38,0000	86	0	0
	31	Gorgonzola Telino	14	4	12 - 100 g pkgs	12,5000	0	70	20
	32	Mascarpone Fabioli	14	4	24 - 200 g pkgs.	32,0000	9	40	25
	33	Geitost	15	4	500 g	2,5000	112	0	20
*									

SQL query

```
SELECT TOP (5) [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock],  
[t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]  
FROM [dbo].[Products] AS [t0]  
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]  
WHERE [t1].[CategoryName] LIKE @p0
```

# Query – 3 (definizione)

Per le categorie che contengono più di un prodotto visualizzare per ciascun prodotto la quantità totale ordinata e il relativo ricavo.

```
private void ExecuteQuery_3()
{
    // query 3:
    // Per le categorie che contengono più di un prodotto visualizzare
    // per ciascun prodotto la quantità totale ordinata e il relativo ricavo.

    var query = from p in db.Products
                where p.Category.Products.Count > 1
                select new
                {
                    PID = p.ProductID,
                    PN = p.ProductName,
                    TotalUnits = p.Order_Details.Sum(o => o.Quantity),
                    Revenue = p.Order_Details.Sum(o => o.UnitPrice * o.Quantity)
                };

    string queryDefinition = "referring to categories containing more than one product"+
                             " for each product show total ordered quantity"+
                             " and related revenue";

    showResults(query, queryDefinition, query.ToString(), false, "query 3");
}
```

Implicitly Typed  
Local Object

# Query - 3 (risultato)

Execute Query

query 3 : referring to categories containing more than one product for each product  
show total ordered quantity and related revenue

Query definition

	PID	PN	TotalUnits	Revenue
▶	1	Chai	828	14277,600000
	2	Chang	1057	18559,200000
	24	Guaraná Fantásti...	1125	4782,600000
	34	Sasquatch Ale	506	6678,000000
	35	Steeleye Stout	883	14536,800000
	38	Côte de Blaye	623	149984,200000
	39	Chartreuse verte	793	13150,800000
	43	Ipoh Coffee	580	25079,200000
	70	Outback Lager	817	11472,000000
	67	Laughing Lumber...	184	2562,000000
	75	Rhönbräu Kloster...	1155	8650,550000
	76	Lakkaikööri	981	16794,000000
	77	Original Frankfurt...	791	9685,000000
	61	Sirop d'érable	603	16438,800000
	63	Vegie-spread	445	17696,300000

SQL query

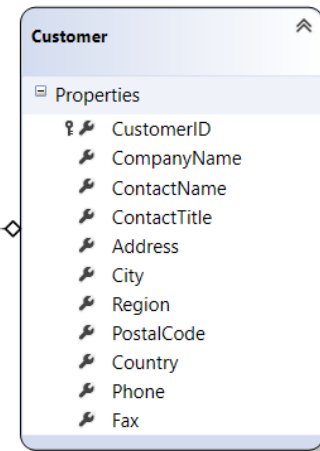
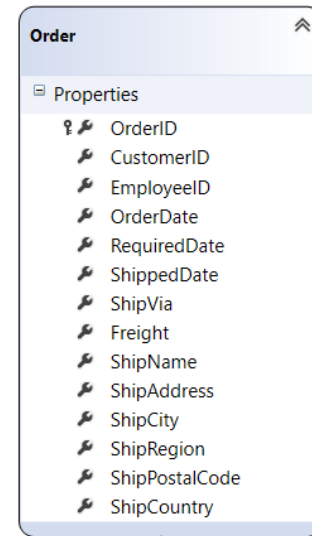
```
SELECT [t0].[ProductID] AS [PID], [t0].[ProductName] AS [PN], (  
  SELECT SUM(CONVERT(Int,[t3].[Quantity]))  
  FROM [dbo].[Order Details] AS [t3]  
  WHERE [t3].[ProductID] = [t0].[ProductID]  
) AS [TotalUnits], (  
  SELECT SUM([t5].[value])  
  FROM (
```



# Query - 4

Per i clienti **che hanno effettuato più di 20 ordini** visualizzare, **ordinati per codice cliente**, **il codice cliente**, **il nome della persona da contattare**, **il quantitativo di ordini** e **le spese totali di trasporto**.

```
int orderCutoff = 20;
var query = from c in db.Customers
    where c.Orders.Count() > orderCutoff
    orderby c.CustomerID
    select new
    {
        c.CustomerID,
        Name = c.ContactName,
        OrderCount = c.Orders.Count(),
        SumFreight = c.Orders.Sum(o => o.Freight)
    };
```



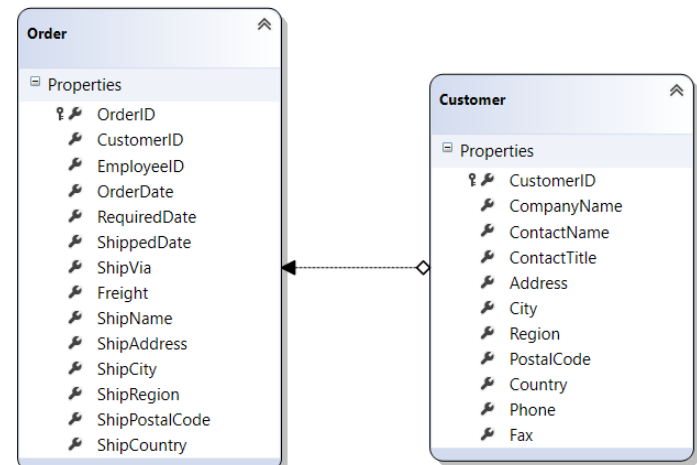
	CustomerID	Name	OrderCount	SumFreight
►	ERNSH	Roland Mendel	30	6205,3900
	QUICK	Horst Kloss	28	5605,6300
	SAVEA	Jose Pavarotti	31	6683,7000

# Query - 5

Per i clienti **con sede a Washington** visualizzare, **ordinati in senso decrescente per data ordine**, **il nome della persona da contattare e la data**.

```
var country = "USA";  
var region = "WA";  
var query = from c in db.Customers  
             where c.Country == country && c.Region == region  
             join o in db.Orders on c.CustomerID equals o.CustomerID  
             orderby o.OrderDate descending  
             select new { c.ContactName, o.OrderDate };
```

ContactName	OrderDate
Karl Jablonski	01/05/1998
Karl Jablonski	17/04/1998
Karl Jablonski	24/02/1998
Karl Jablonski	30/01/1998
Helvetius Nagy	08/01/1998
Karl Jablonski	13/11/1997
Karl Jablonski	30/10/1997
Karl Jablonski	08/10/1997
Karl Jablonski	06/10/1997
Karl Jablonski	11/07/1997
Helvetius Nagy	23/06/1997
Helvetius Nagy	19/06/1997



# Query - 6

```
var orderList = from o in db.Orders
                join c in db.Customers on o.CustomerID equals c.CustomerID
                orderby c.ContactName
                select new
                (
                    Name = c.ContactName,
                    c.Country,
                    c.City,
                    Revenue = o.Order_Details.Sum(p => p.UnitPrice * p.Quantity)
                );

var customerRevenue = from o in orderList
                      group o by o.Name into g
                      select new
                      (
                          CustomerName = g.Key,
                          Country = g.Select(a => a.Country).First(),
                          City = g.Select(b => b.City).First(),
                          TotalRevenue = g.Sum(o => o.Revenue)
                      );

dataGridViewQuery.DataSource = customerRevenue;
```

Per i clienti **che hanno effettuato ordini**, **ordinati per nome**, visualizzare **l'importo totale ordinato**.

	CustomerName	Country	City	TotalRevenue
►	Alejandra Camino	Spain	Madrid	1467,290000
	Alexander Feuer	Germany	Leipzig	5042,200000
	Ana Trujillo	Mexico	México D.F.	1402,950000
	Anabela Doming...	Brazil	Sao Paulo	7310,620000
	André Fonseca	Brazil	Campinas	8702,230000
	Ann Devon	UK	London	15033,660000
	Annette Roulet	France	Toulouse	10272,350000
	Antonio Moreno	Mexico	México D.F.	7515,350000
	Aria Cruz	Brazil	Sao Paulo	4438,900000
	Art Braunschweiler	USA	Lander	12489,700000
	Bernardo Batista	Brazil	Rio de Janeiro	6973,630000
	Carine Schmitt	France	Nantes	3172,160000
	Carlos González	Venezuela	Barquisimeto	17825,060000
	Carlos Hernández	Venezuela	San Cristóbal	23611,580000
	Catherine Dewey	Belgium	Bruxelles	10430,580000
	Christina Berglund	Sweden	Luleå	26968,150000
	Daniel Tonini	France	Versailles	1992,050000

# Query - 7

	ContactName	OrderDate
►	Ann Devon	26/11/1996
	Ann Devon	01/01/1997
	Ann Devon	09/05/1997
	Ann Devon	03/11/1997
	Ann Devon	31/03/1998
	Ann Devon	15/04/1998
	Ann Devon	24/04/1998
	Ann Devon	28/04/1998
	Elizabeth Brown	23/01/1998
	Elizabeth Brown	03/03/1997
	Elizabeth Brown	04/02/1997
	Hari Kumar	21/11/1996
	Hari Kumar	19/12/1996
	Hari Kumar	09/12/1996
	Hari Kumar	12/03/1997

```
var query = from c in db.Customers
             from o in c.Orders
             where c.City == "London"
             orderby c.ContactName
             select new { c.ContactName, o.OrderDate };
```

**Visualizzare contatto e data ordine dei clienti di Londra che hanno effettuato ordini ordinando il risultato per contatto cliente.**

# Altri esempi di query

**Visualizzare i clienti che fanno tutti gli ordini con consegna nella città dove risiedono.**

```
var query = from c in db.Customers
             where c.Orders.All(o => o.ShipCity == c.City)
             select c;
```

**Visualizzare, per ciascuna categoria a cui appartengono più di 10 prodotti, la categoria e il prezzo medio dei prodotti che vi appartengono.**

```
var query = from p in db.Products
             where p.Category.Products.Count() > 10
             group p by new
             {
                 p.Category.CategoryID,
                 p.Category.CategoryName
             } into g
             select new
             {
                 g.Key.CategoryID,
                 g.Key.CategoryName,
                 AvgPrice = g.Average(p => p.UnitPrice)
             };
```

# Esempi di aggiornamento del DB

```
var products = from p in db.Products
               where (p.CategoryID == 4)
               select p;

foreach (var prod in products)
{
    prod.UnitPrice += (decimal)0.1 * prod.UnitPrice;
}

db.SubmitChanges();
```

**Aggiorna il prezzo unitario di tutti i prodotti di categoria 4, aumentandolo del 10%.**

**Modifica della persona da contattare**

```
// Query for a specific customer.
var cust =
    (from c in db.Customers
     where c.CustomerID == "ALFKI"
     select c).First();
```

```
// Change the name of the contact.
cust.ContactName = "Jane Andersen";
```

```
// create and add a new order to the orders collection.
Order ord = new Order { OrderDate = DateTime.Now };
cust.Orders.Add(ord);
```

```
// Delete an existing Order.
Order ord6 = cust.Orders[6];
```

**Inserimento di un nuovo ordine**

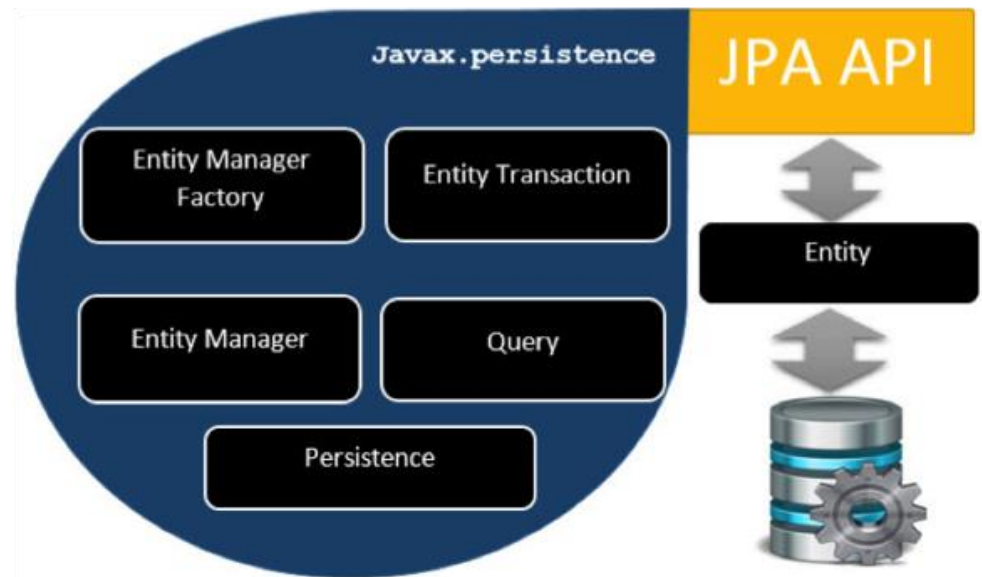
**Cancellazione di un ordine (nell'ipotesi che non siano stati inseriti dettagli d'ordine)**

```
// Removing it from the table also removes it from the Customer's list.
db.Orders.DeleteOnSubmit(ord6);
```

```
// Ask the DataContext to save all the changes.
db.SubmitChanges();
```

# Java Persistence API

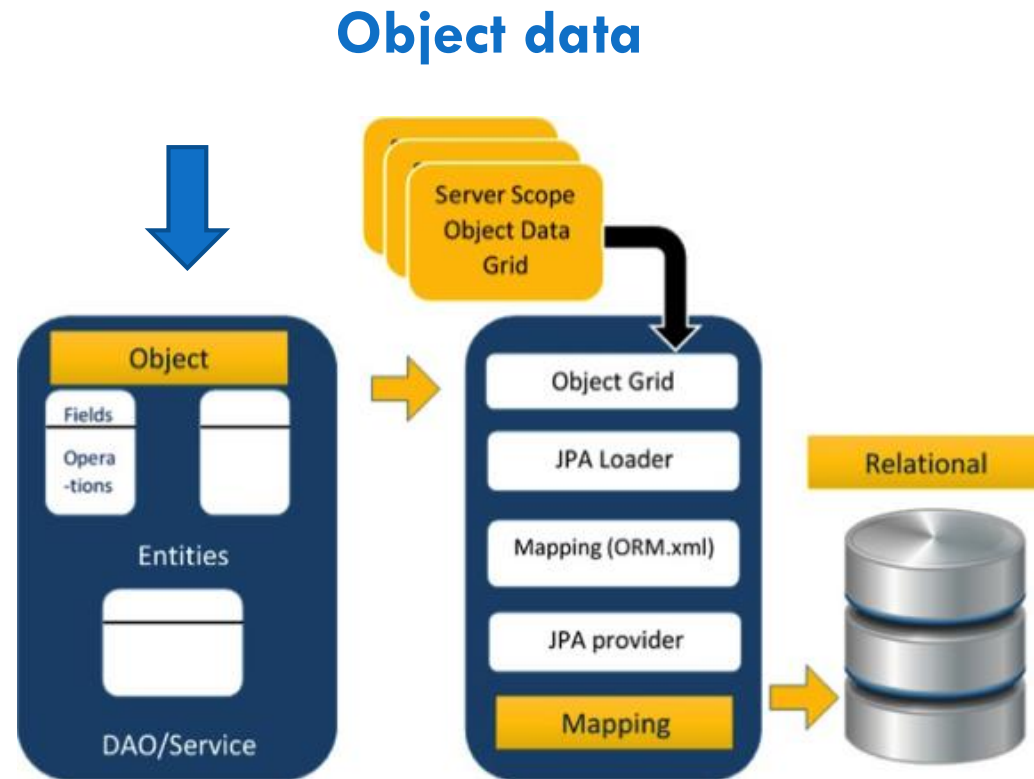
- La Java Persistence API è lo strumento standard per il mapping Object/Relational e la gestione della persistenza per la piattaforma Java EE 5.0.
- JPA è una API open source ed è pertanto supportata dai maggiori produttori quali Oracle, Redhat, Eclipse attraverso specifiche implementazioni:
  - ▣ Hibernate, [Eclipselink](#), Toplink, Spring Data JPA, etc.



[https://www.tutorialspoint.com/jpa/jpa\\_quick\\_guide.htm](https://www.tutorialspoint.com/jpa/jpa_quick_guide.htm)  
<https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>  
<https://www.objectdb.com/java/jpa/getting/started>

# ORM Mapping in JPA – fase 1

- Contiene classi, servizi e interfacce.
- Strato di business logic.
- Esempio (impiegati):
  - La classe Impiegato contiene attribute come ID, nome, stipendio e metodi (set e get degli attribute).
  - Le classi DAO/Service per Impiegato contengono metodi di servizio per la creazione, ricerca e cancellazione degli impiegati.

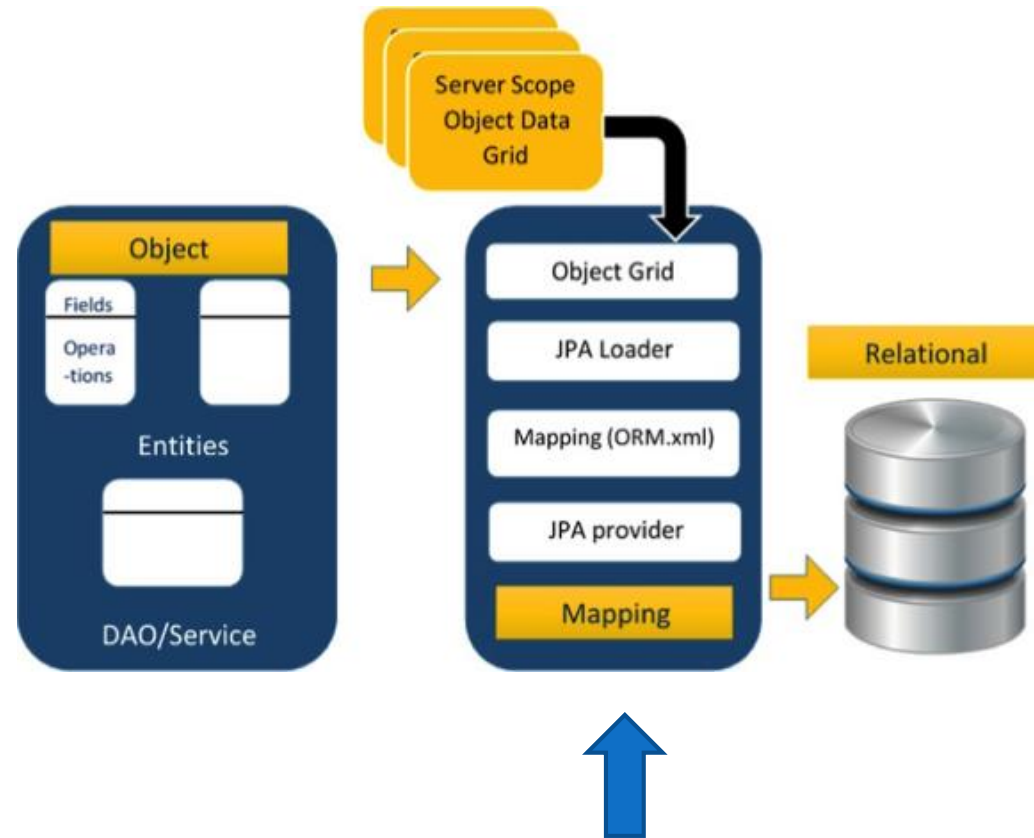




# ORM Mapping in JPA – fase 2

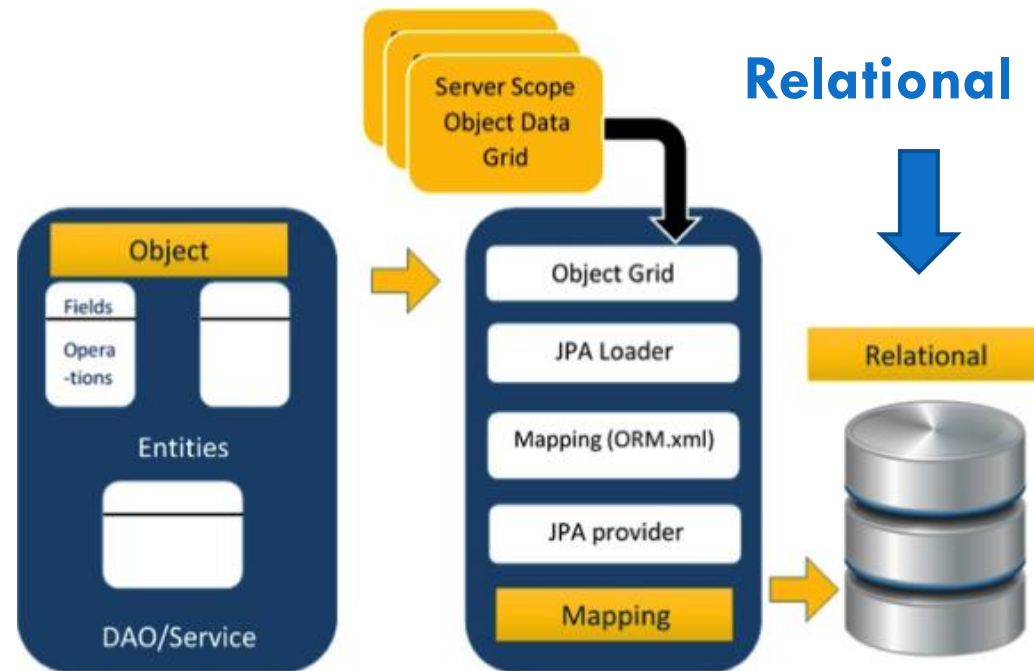
- Contiene:
  - ▣ Il **provider JPA**: software che implementa JPA (es. Hibernate)
  - ▣ Il **file di mapping ORM.xml** (tra classi Java e db relazionale)
  - ▣ Il **loader JPA**: memoria cache per il caricamento dei dati
  - ▣ **Object Grid**: posizione temporanea che può contenere una copia del db relazionale. Ogni query sul db viene prima eseguita sui data dell'object grid e viene eseguita sul db solo dopo l'eventuale commit.

## Mapping / persistence



# ORM Mapping in JPA – fase 3

- Contiene i **dati relazionali** logicamente connessi allo strato precedente.
- Solo quando viene eseguito il commit le operazioni eseguite sui dati vengono memorizzate in modo persistente nel database.
- Fino a quel momento, i dati modificati sono memorizzati nella memoria cache (Object Grid).

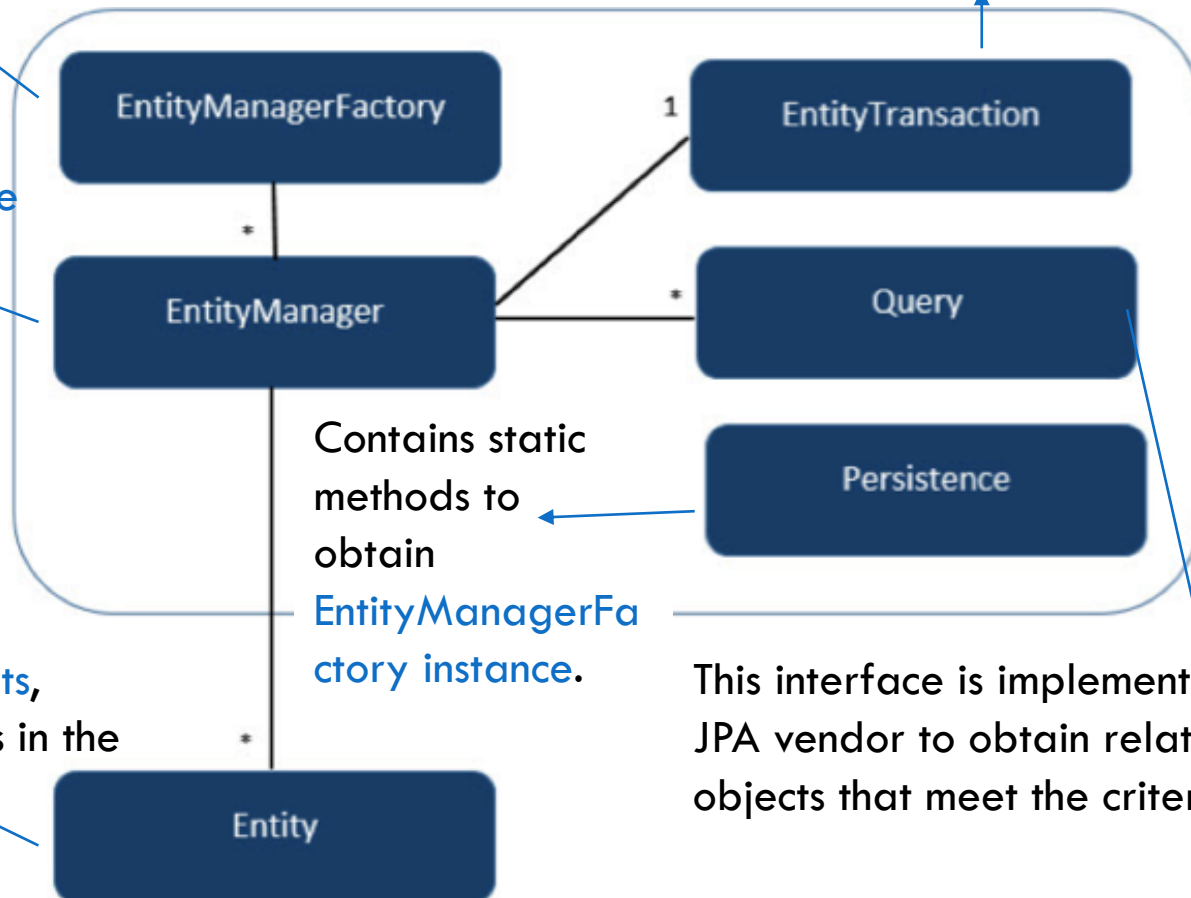


# JPA: architettura delle classi

This is a factory class of EntityManager.

For each EntityManager, operations are maintained by EntityTransaction class.

It is an Interface, it manages the persistence operations on objects.



Entities are the persistence objects, stored as records in the database.

This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

# Connessione al DB

- La connessione al database è rappresentata dall'interfaccia EntityManager.
- La creazione di un'istanza EntityManager richiede due step:
  1. Definizione di una persistence unit in un file XML dedicato, necessaria per la creazione della EntityManagerFactory.
  2. Creazione di un'istanza di EntityManagerFactory

```
EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory(  
        persistenceUnitName    );  
emf.close();
```

- A questo punto è possibile ottenere un'istanza di EntityManager

```
EntityManager em = emf.createEntityManager();  
em.close();
```

- Le operazioni di modifica del contenuto del db richiedono l'uso di transazioni

```
em.getTransaction().begin();  
em.getTransaction().commit();
```

I metodi `persist` e `remove` modificano il contenuto del db

# Entity

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
            "Customer[id=%d, firstName='%s', lastName='%s']",
            id, firstName, lastName);
    }

    public Long getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

È necessario importare librerie del modulo `javax.persistence`

Un'entità è preceduta dall'annotazione `@Entity`

È possibile modificare il nome della tabella annotando l'entità con `@javax.persistence.Table`:

`@Entity`

`@Table(name = "ARTICLES")`

**`public class Article { // ... }`**

Un dato membro può anche essere annotato con `@Id` a indicare che tale attributo corrisponde alla chiave primaria degli oggetti istanze dell'entità (assenza valori nulli e unicità).

# Mappatura delle relazioni tra classi

- In JPA la mappatura delle relazioni tra classi in corrispondenze tra tabelle sulla base di vincoli di integrità richiede di specificare tre aspetti:
  - ▣ Cardinalità: corrispondente alla cardinalità del modello relazionale.
    - @OneToOne, @ManyToOne, @OneToMany, @ManyToMany
  - ▣ Navigabilità: individua una classe come owner dell'associazione e stabilisce se la relazione è navigabile in entrambi i sensi oppure solo in senso diretto.
  - ▣ Politiche di gestione: specifica come deve comportarsi il sistema quando carica in memoria centrale un oggetto estratto dal database. Ad esempio:
    - LAZY se gli oggetti correlati devono essere caricati solo quando serve;
    - EAGER se gli oggetti correlati devono essere caricati subito.

# Mappatura relazioni: esempio @OneToMany

```
1  @Entity
2  public class Order {
3
4      @OneToMany
5      private List<OrderItem> items = new ArrayList<OrderItem>();
6
7      ...
8  }
```

```
1  @Entity
2  public class OrderItem {
3
4      @ManyToOne
5      @JoinColumn(name = "fk_order")
6      private Order order;
7
8      ...
9  }
```

# Mappatura relazioni: esempio @ManyToMany

```
1  @Entity
2  public class Store {
3
4      @ManyToMany
5      @JoinTable(name = "store_product",
6                  joinColumns = { @JoinColumn(name = "fk_store") },
7                  inverseJoinColumns = { @JoinColumn(name = "fk_product") })
8      private Set<Product> products = new HashSet<Product>();
9
10     ...
11 }
```

```
1  @Entity
2  public class Product{
3
4      @ManyToMany(mappedBy="products")
5      private Set<Store> stores = new HashSet<Store>();
6
7      ...
8  }
```



# Interrogazioni in JPA (1)

- Il linguaggio di interrogazione e aggiornamento dati offerto da JPA è il **Java Persistence Query Language (JPQL)**:
  - ▣ Le interrogazioni e gli aggiornamenti sono espressi a partire da un **modello a oggetti di alto livello** (entità, relazioni, attributi, ecc.);
  - ▣ Sono disponibili due tipi di interrogazioni a oggetti, Query e TypedQuery (tipo del risultato);

```
Query q1 = em.createQuery("SELECT c FROM Country c");
```

```
TypedQuery<Country> q2 =  
    em.createQuery("SELECT c FROM Country c", Country.class);
```

- ▣ Il linguaggio consente la scrittura di espressioni ed operatori che sfruttano caratteristiche avanzate del modello dei dati, quali la navigazione delle relazioni e l'accesso a collezioni di oggetti.

# Interrogazioni in JPA (2)

- L'interfaccia `Query` (e analogamente `TypedQuery`) definisce due metodi per l'esecuzione di query di selezione:
  - ▣ `Query.getSingleResult`: da utilizzare quando è atteso un singolo oggetto come risultato;
  - ▣ `Query.getResultList`: da utilizzare nel caso generale per gestire risultati multipli
- Per le query di aggiornamento il metodo da richiamare è
  - ▣ `Query.executeUpdate`

<https://www.objectdb.com/java/jpa/query/execute>

# JPQL esempi

## A Basic Select Query

```
SELECT p  
FROM Player p
```

## Using Named Parameters

```
SELECT DISTINCT p  
FROM Player p  
WHERE p.position = :position AND p.name = :name
```

## Navigating to Single-Valued Relationship Fields

Use the `JOIN` clause statement to navigate to a single-valued relationship field:

```
SELECT t  
FROM Team t JOIN t.league l  
WHERE l.sport = 'soccer' OR l.sport = 'football'
```

## Traversing Relationships with an Input Parameter

```
SELECT DISTINCT p  
FROM Player p, IN (p.teams) AS t  
WHERE t.city = :city
```

<https://docs.oracle.com/javaee/6/tutorial/doc/bnbt1.html#bnbtm>

# Domande?

---

