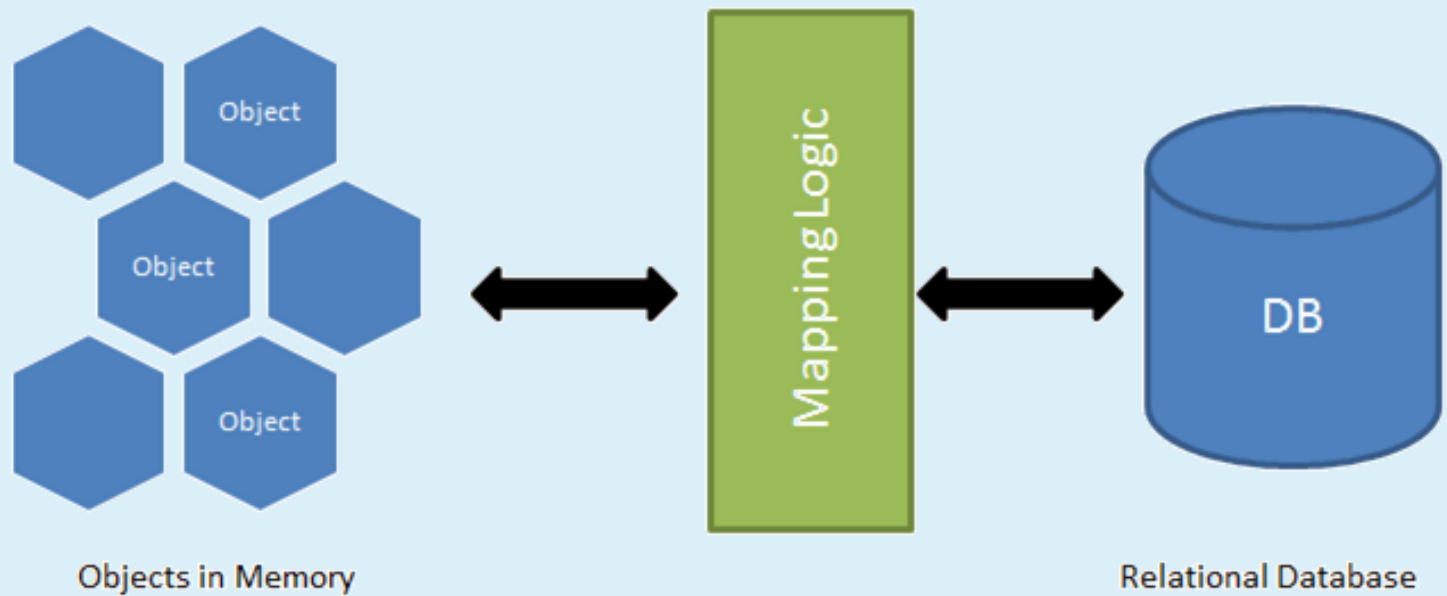


O/R Mapping



Object Relational Mapping (ORM)

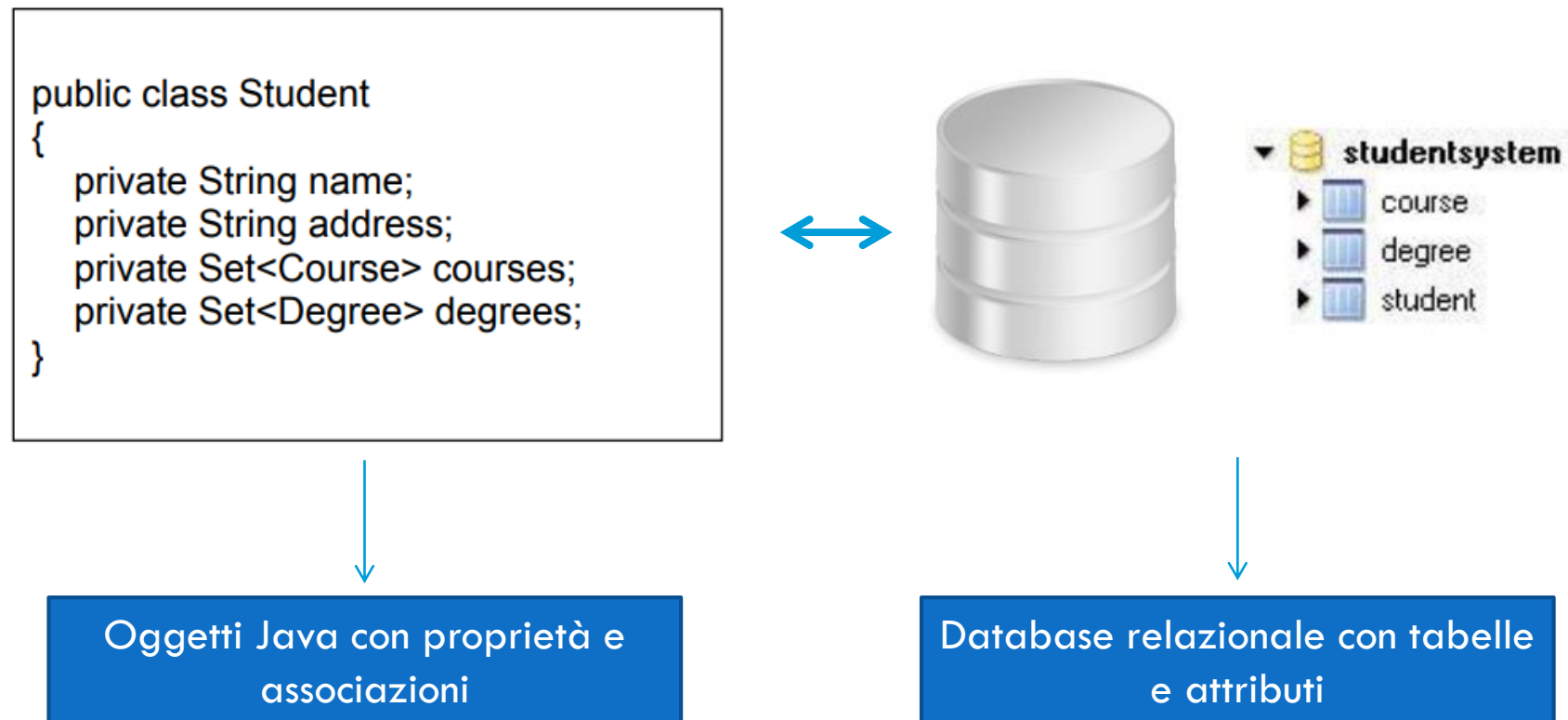
Annalisa Franco, Dario Maio
Università di Bologna

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

Entity Framework Core

- ❑ **Entity Framework (EF) Core** è una versione semplice, estendibile, **open source** e **multiplatforma** della tecnologia Entity Framework.
- ❑ Con Entity Framework Core, l'accesso ai dati viene eseguito tramite un **modello**. Un modello è costituito da **classi di entità** e da un **contesto** dell'oggetto che rappresenta una **sessione con il database** che consente di eseguire query e salvare i dati.
- ❑ EF Core consente agli sviluppatori .NET di operare su un database usando oggetti .NET, eliminando la necessità della maggior parte del codice di accesso ai dati che in genere deve essere scritto.
- ❑ EF Core supporta molti **motori di database**
<https://learn.microsoft.com/it-it/ef/core/providers/?tabs=dotnet-core-cli>
- ❑ Implementa **operazioni CRUD** e interrogazioni con linguaggio **LINQ**
- ❑ Link utili:
 - ❑ <https://learn.microsoft.com/en-us/ef/core/>
 - ❑ <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>

EF Core – applicazioni e OS

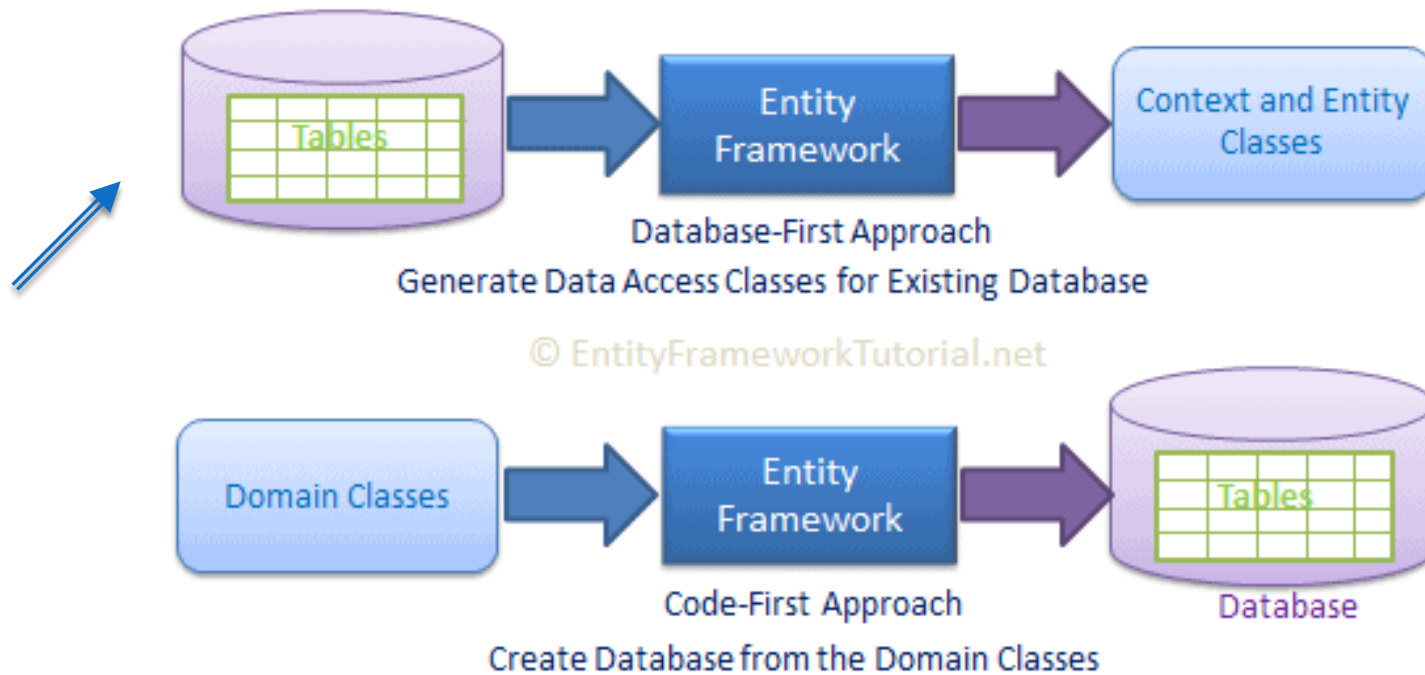
- EF Core è pensato per essere usato in applicazioni .NET Core (ma può essere usato anche in applicazioni .NET 4.5+)

| | | | | |
|-------------------|---|---|--|--|
| Application Types | <u>ASP.NET Core Applications</u> Web, API, Console, etc. | <u>.NET 4.5+ Applications</u> Console, WinForm, WPF, ASP.NET | Devices + IoT, Mobile, PC, Xbox, Surface Hub | <u>Mobile Application</u> Android, iOS, Windows |
| EF Core | EF Core | EF Core | EF Core | EF Core |
| Framework | .NET Core | .NET 4.5+ | UWP | Xamarin |
| OS | Windows, Mac, Linux | Windows | Windows 10 | Mobile |

© EntityFrameworkTutorial.net

<https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>

Entity Framework Core - approcci

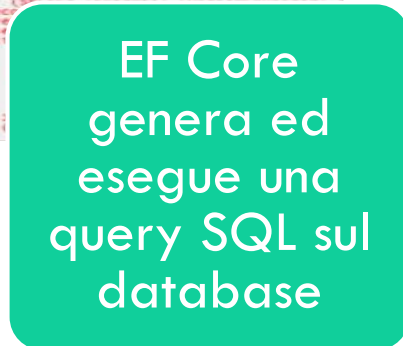


- **Database-First:** l'API EF Core crea automaticamente le classi sulla base di un database esistente.
- **Code-First:** l'API EF Core crea il database e le tabelle usando un processo di migrazione sulla base delle classi definite nel codice.

© 2013 Pearson Education, Inc. or its affiliate(s). All rights reserved. This publication is protected by copyright. Any unauthorized distribution or reproduction of this work is illegal. All other rights reserved.



Scrivere ed eseguire una query tramite l'interfaccia iQueryable



EF Core
trasforma il
risultato della
query in
oggetti .NET



Modificare
eventualmente
i dati e
richiamare il
metodo «Save
Changes»



EF Core genera ed esegue i comandi per modificare il DB

EF Core
genera ed
esegue i
comandi per
modificare il
DB

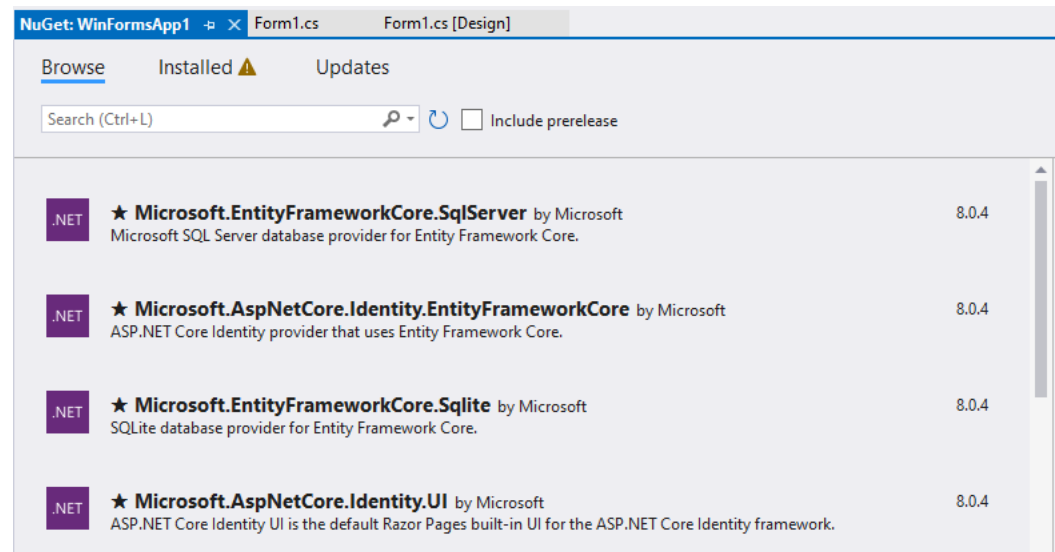
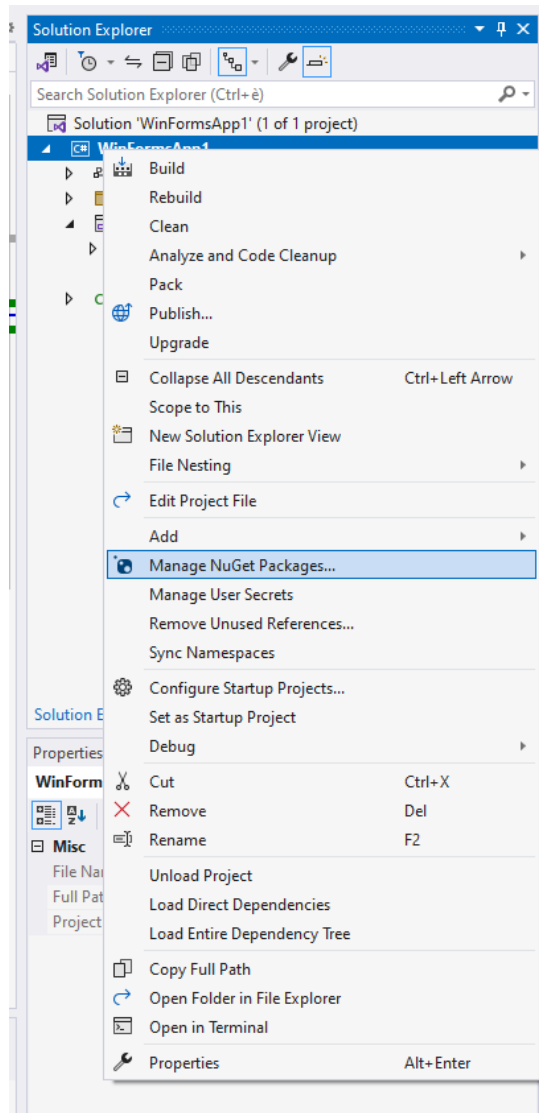
Installazione

- Consigliata l'ultima versione di Visual Studio (2022).
- È necessario installare due package nell'applicazione usando NuGet, un package manager per .NET.
- [MySQL.EntityFrameworkCore](#) (o specifico per altro DBMS): package che offre connettività ad applicazioni .NET con DBMS MySQL.

Dependencies:

- [Microsoft.EntityFrameworkCore](#)
- [Microsoft.EntityFrameworkCore.Relational](#)
- [MySQL.Data](#)
- [Microsoft.EntityFrameworkCore.Tools](#): package che supporta la fase di design, fornendo funzionalità per la migrazione di dati e la creazione di un DbContext tramite un processo di reverse engineer a partire da una database.

Installazione - NuGet Package Manager



- Cercare nella finestra Browse i package di interesse;
- Procedere con l'installazione;
- Accettare i termini d'uso.

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.

Scaffold-DbContext

- È il comando fondamentale che crea automaticamente il DbContext (con un insieme di Entity classes) a partire da un database (eseguire da Package Manager Console)

Scaffold-DbContext

"server=localhost;uid=student;pwd=135provaTessiland!;database=tessiland"

MySql.EntityFrameworkCore -OutputDir Tessiland -f

Connection string
Stringa di connessione al database.

Provider
Provider da usare (in genere nome del pacchetto NuGet per lo specifico DBMS)

Output folder
Directory in cui inserire i file Entity classes. I percorsi sono relativi alla directory del progetto

Per la lista completa di parametri:

<https://learn.microsoft.com/it-it/ef/core/cli/powershell#scaffold-dbcontext>

DbContext

Rappresenta il database all'interno dell'applicazione.

```
public partial class NorthwindContext : DbContext
{
    1 reference
    public NorthwindContext()
    {
    }

    0 references
    public NorthwindContext(DbContextOptions<NorthwindContext> options)
        : base(options)
    {
    }

    0 references
    public virtual DbSet<Category> Categories { get; set; }

    0 references
    public virtual DbSet<Customer> Customers { get; set; }

    0 references
    public virtual DbSet<Customerdemographic> Customerdemographics { get; set; }

    0 references
    public virtual DbSet<Employee> Employees { get; set; }

    0 references
    public virtual DbSet<Order> Orders { get; set; }

    0 references
    public virtual DbSet<OrderDetail> OrderDetails { get; set; }

    1 reference
    public virtual DbSet<Product> Products { get; set; }

    0 references
    public virtual DbSet<Region> Regions { get; set; }

    0 references
    public virtual DbSet<Shipper> Shippers { get; set; }

    0 references
    public virtual DbSet<Supplier> Suppliers { get; set; }
```

- ❑ La durata di DbContext inizia quando l'istanza viene creata e termina quando l'istanza viene eliminata. Un'istanza di DbContext è progettata per essere usata per una singola unità di lavoro.
- ❑ Un'unità di lavoro tipica quando si usa Entity Framework Core (EF Core) implica le operazioni seguenti:
 - ❑ Creazione di un'istanza di DbContext
 - ❑ Interrogazione delle istanze di entità dal DbContext.
 - ❑ Modifica alle entità rilevate in base alle esigenze per implementare la regola di business.
 - ❑ Salvataggio di eventuali modifiche apportate con SaveChanges o SaveChangesAsync. EF Core rileva le modifiche apportate e le scrive nel database.
 - ❑ Eliminazione dell'istanza di DbContext

EntityClasses

- NW
 - Category.cs
 - Customer.cs
 - Customerdemographic.cs
 - Employee.cs
 - NorthwindContext.cs
 - Order.cs
 - OrderDetail.cs
 - Product.cs
 - Region.cs
 - Shipper.cs
 - Supplier.cs
 - Territory.cs

```
7 references
public partial class Product
{
    2 references
    public int ProductId { get; set; }

    3 references
    public string ProductName { get; set; } = null!;

    3 references
    public int? SupplierId { get; set; }

    3 references
    public int? CategoryId { get; set; }

    1 reference
    public string? QuantityPerUnit { get; set; }

    3 references
    public decimal? UnitPrice { get; set; }

    1 reference
    public short? UnitsInStock { get; set; }

    1 reference
    public short? UnitsOnOrder { get; set; }

    1 reference
    public short? ReorderLevel { get; set; }

    1 reference
    public ulong Discontinued { get; set; }

    3 references
    public virtual Category? Category { get; set; }

    1 reference
    public virtual ICollection<OrderDetail> OrderDetails { get; set; } = new List<OrderDetail>();

    1 reference
    public virtual Supplier? Supplier { get; set; }
}
```

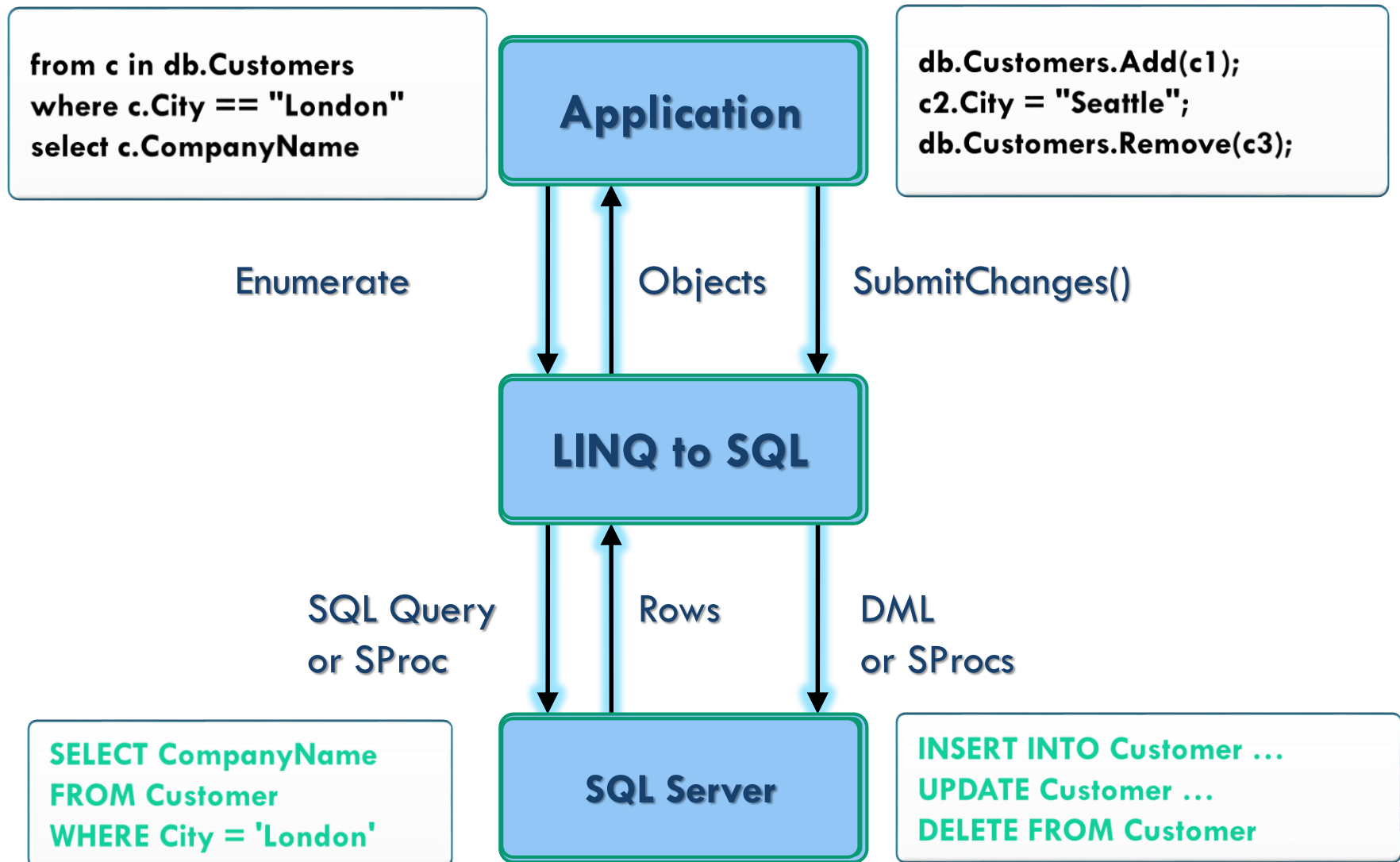
Attributi della
tabella

Oggetti collegati

Esecuzione di query: il linguaggio LINQ


- ❑ LINQ (Language-Integrated Query) è il nome di un set di tecnologie basate sull'integrazione delle funzionalità di query direttamente nel linguaggio C#.
- ❑ Le espressioni di query vengono scritte con una *sintassi di query* dichiarativa che permette di eseguire operazioni di filtro, ordinamento e raggruppamento sulle origini dati usando una quantità minima di codice.
- ❑ Le espressioni di query interrogano e trasformano i dati da qualsiasi origine dati abilitata per LINQ (es. SQL, XML).
- ❑ La query specifica le **informazioni da recuperare** dall'origine o dalle origini dati. Una query può anche specificare il modo in cui **ordinare**, **raggruppare** e definire le informazioni prima di essere restituite. Una query viene archiviata in una variabile di query e inizializzata con un'espressione di query.

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

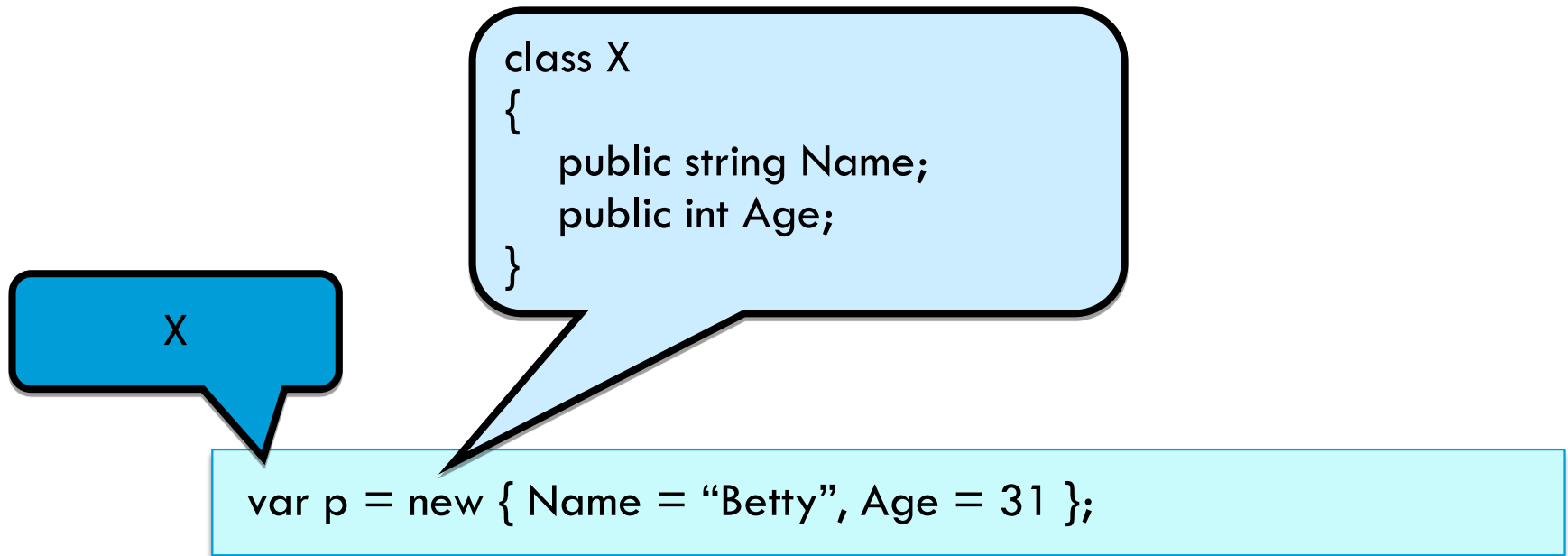
```
Point myPoint = new Point { X = 0, Y = 1 };
```

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



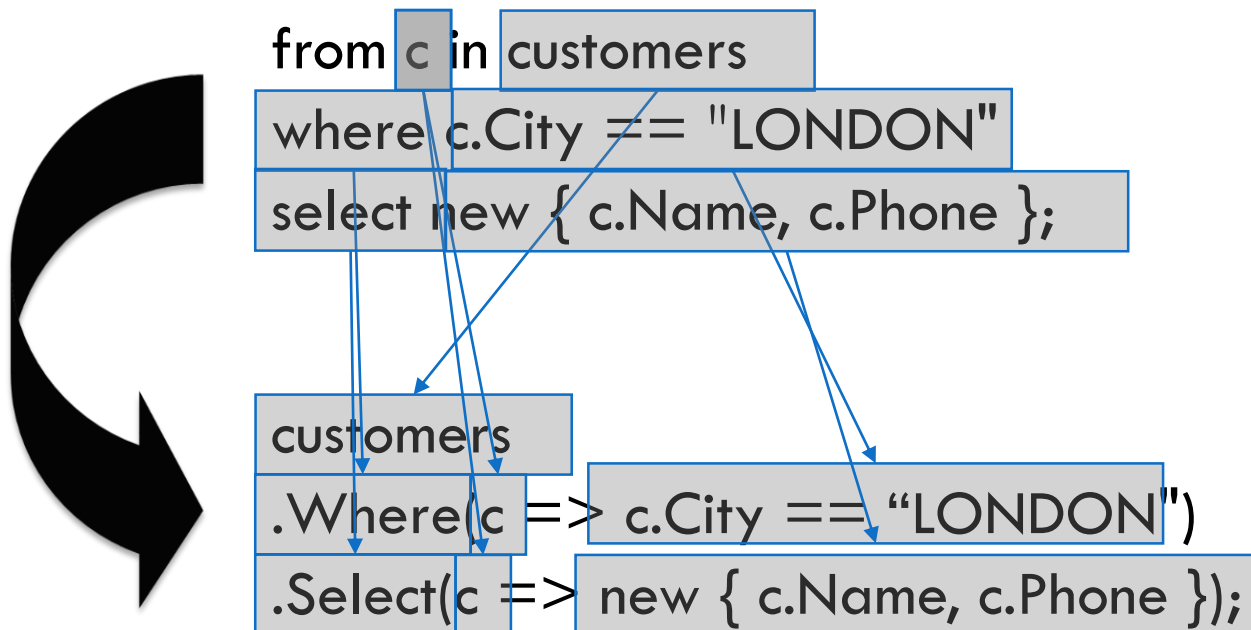
Lambda Expressions

- ❑ 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
}
```

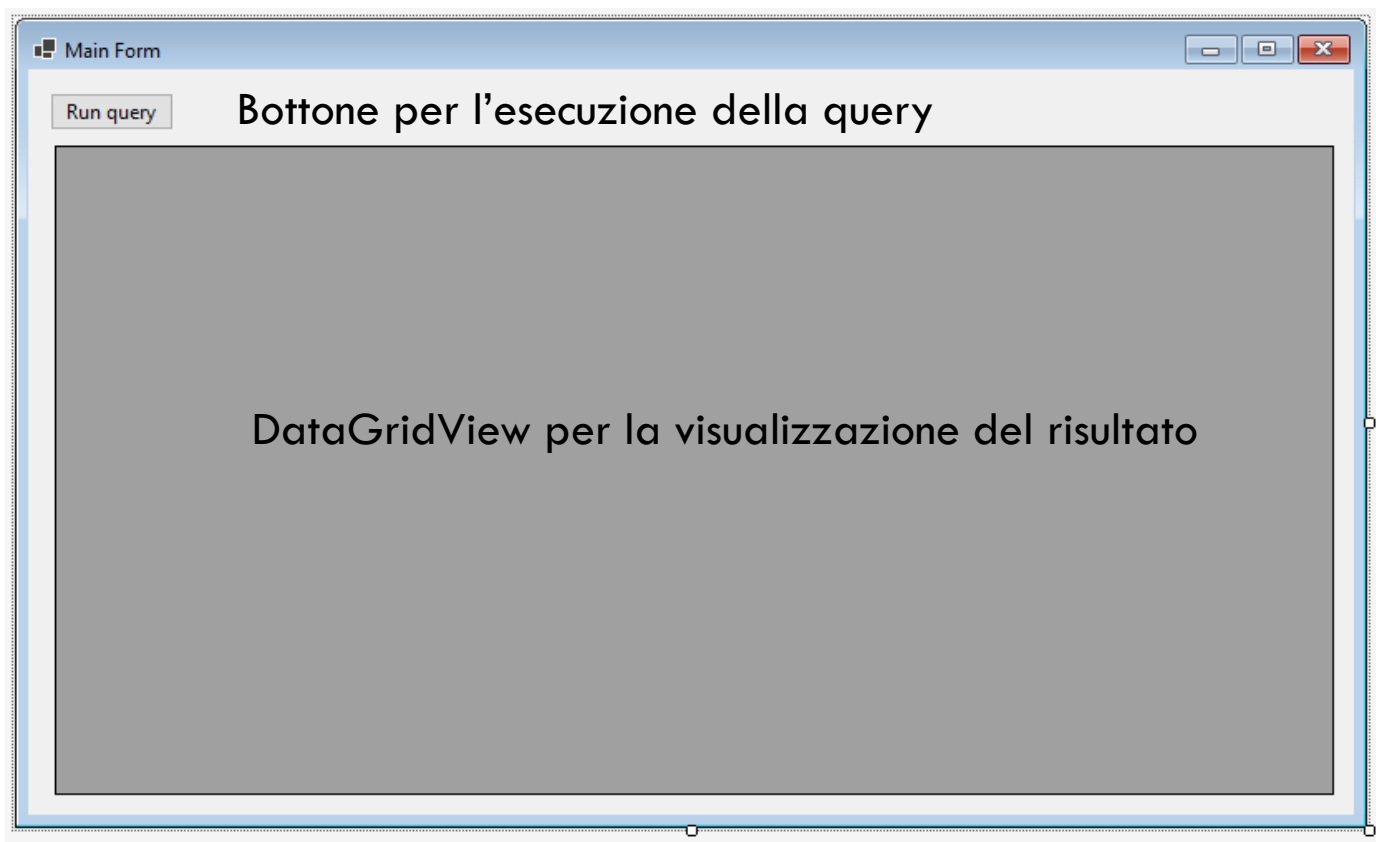
Query Expressions → LINQ

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



L'applicazione

- ❑ Dichiarare e istanziare un opportuno DbContext
`var db = new NorthwindContext();`
- ❑ Inserire nella form gli opportuni controlli.



Query – 1

Tutti i prodotti per i quali 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;
```

```
dataGridViewResult.DataSource = query.ToList();
```

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

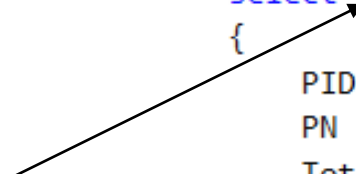
```
var query = (from p in db.Products
            where p.Category.CategoryName.StartsWith("D")
            select p).Take(5);
```

Query – 3 (definizione)

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)
             };

```

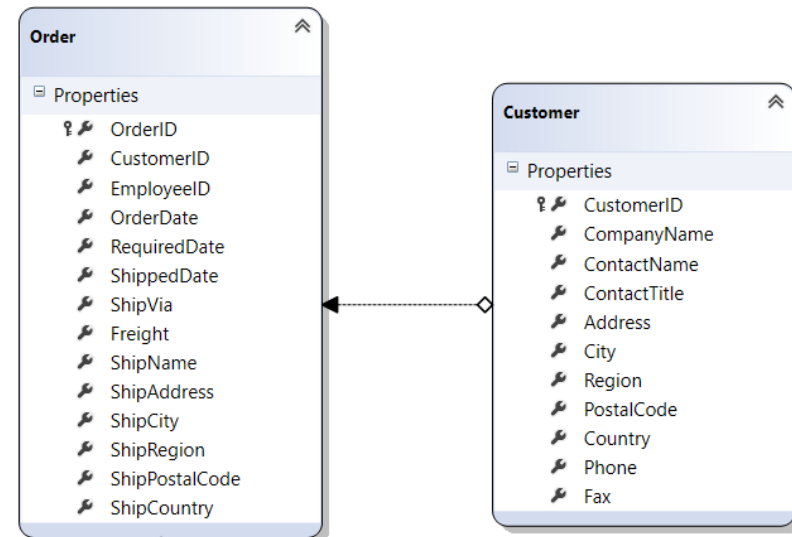


Implicitly Typed
Local Object

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)
    };
```

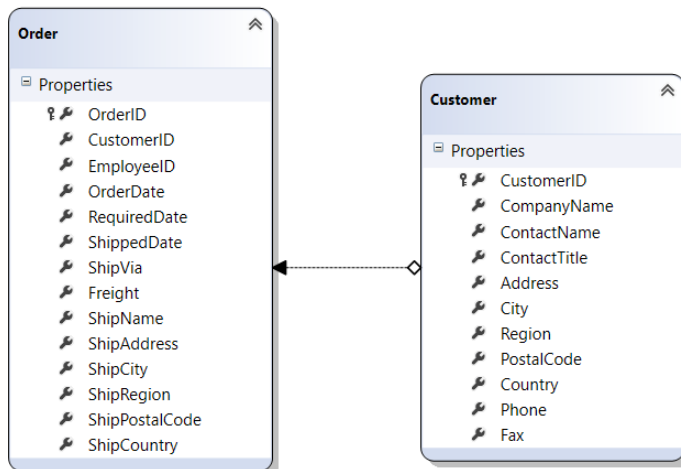


| | CustomerID | Name | OrderCount | SumFreight |
|---|------------|----------------|------------|------------|
| ► | ERNSH | Roland Mendel | 30 | 6205,3900 |
| | QUICK | Horst Kloss | 28 | 5605,6300 |
| | SAVEA | Jose Pavarotti | 31 | 6683,7000 |

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 };
```



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 Braunschweiger | 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.SaveChanges();
```

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];
```

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

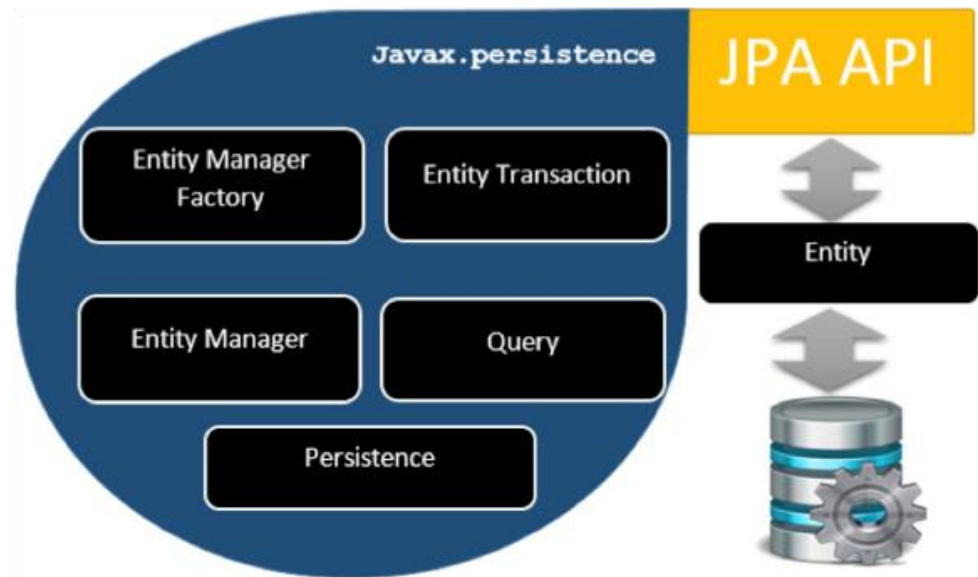
```
// Ask the DataContext to save all the changes.
db.SaveChanges();
```

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

Jn

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](https://www.tutorialspoint.com/jpa/jpa_quick_guide.htm), Toplink, Spring Data JPA, etc.



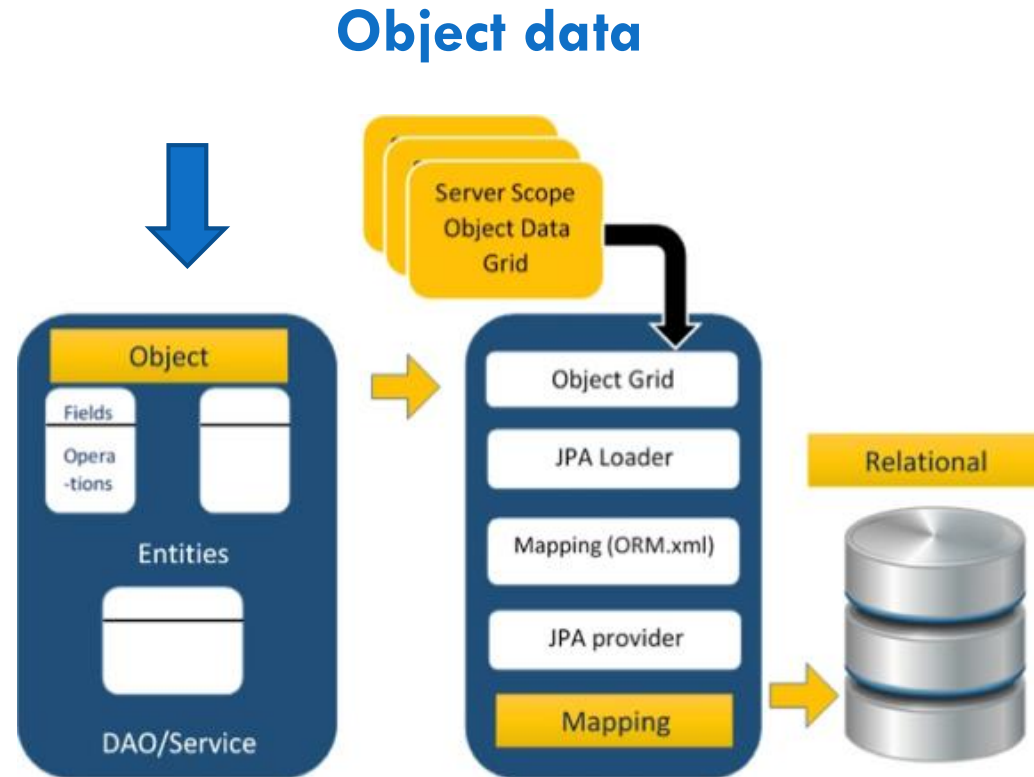
https://www.tutorialspoint.com/jpa/jpa_quick_guide.htm

<https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>

<https://www.objectdb.com/java/jpa/getting/started>

ORM Mapping in JPA – livello 1

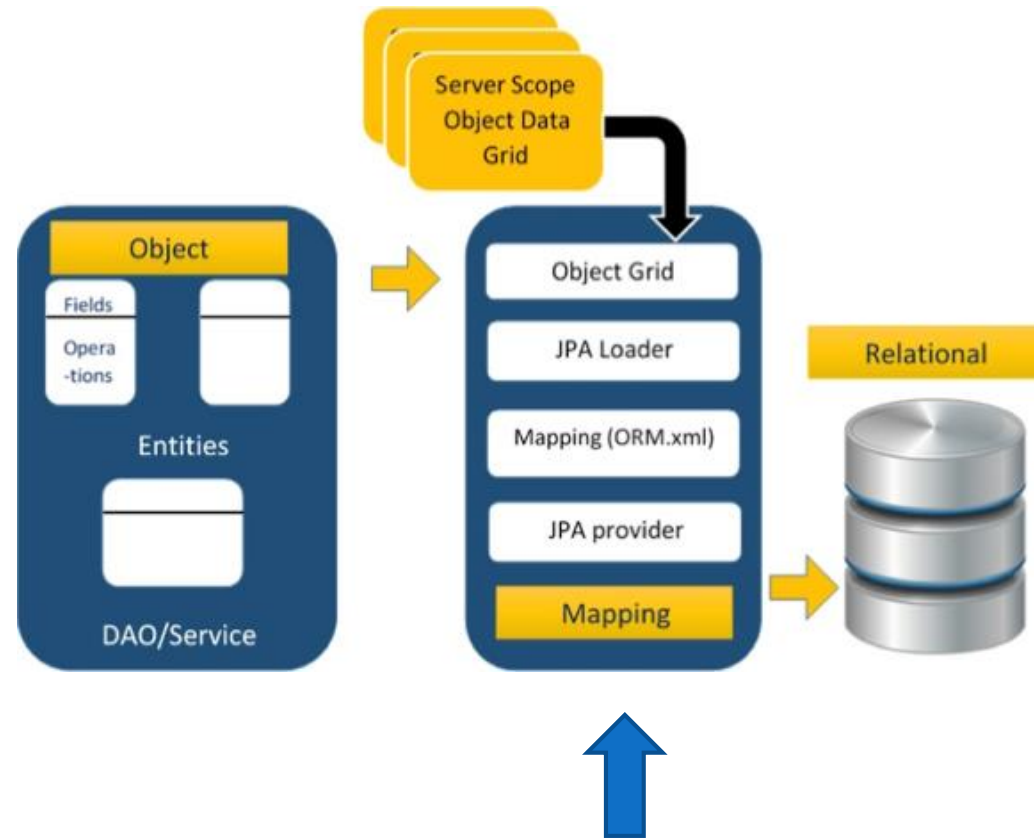
- Contiene classi, servizi e interfacce.
- Esempio (impiegati):
 - La classe Impiegato contiene attribute come ID, nome, stipendio e metodi (set e get degli attribute).
 - Le classi DAO/Service per Impiegato contengono metodi di servizio per la creazione, ricerca e cancellazione degli impiegati.



ORM Mapping in JPA – livello 2

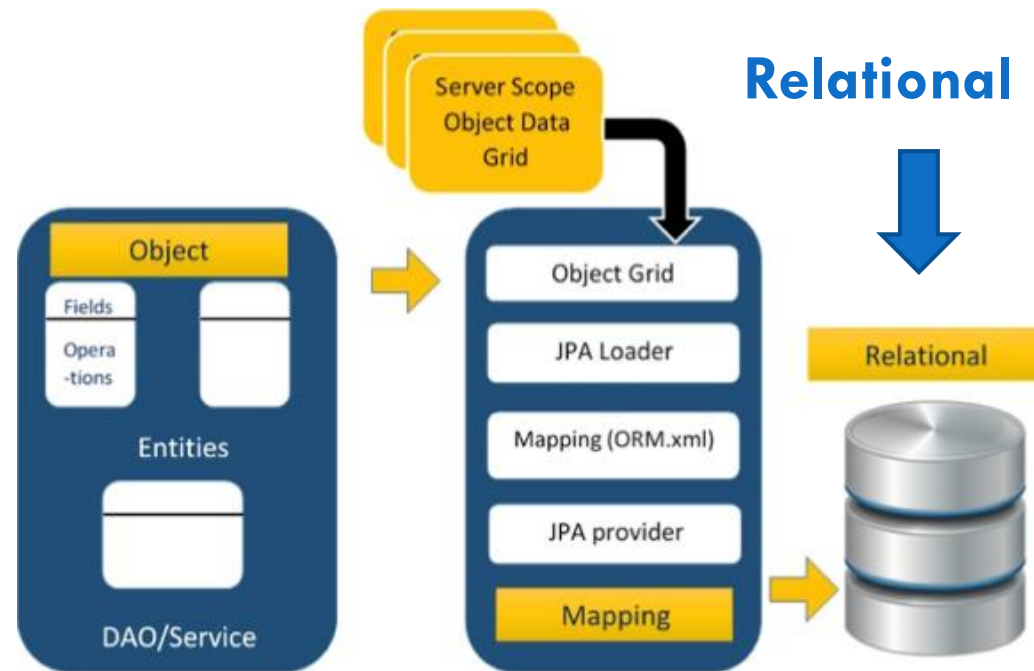
- Contiene:
 - ▣ Il provider JPA: software che implementa JPA (es. Hibernate)
 - ▣ Il file di mapping ORM.xml (tra classi Java e db relazionale)
 - ▣ Il loader JPA: memoria cache per il caricamento dei dati
 - ▣ Object Grid: posizione temporanea che può contenere una copia del db relazionale. Ogni query 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 – livello 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).

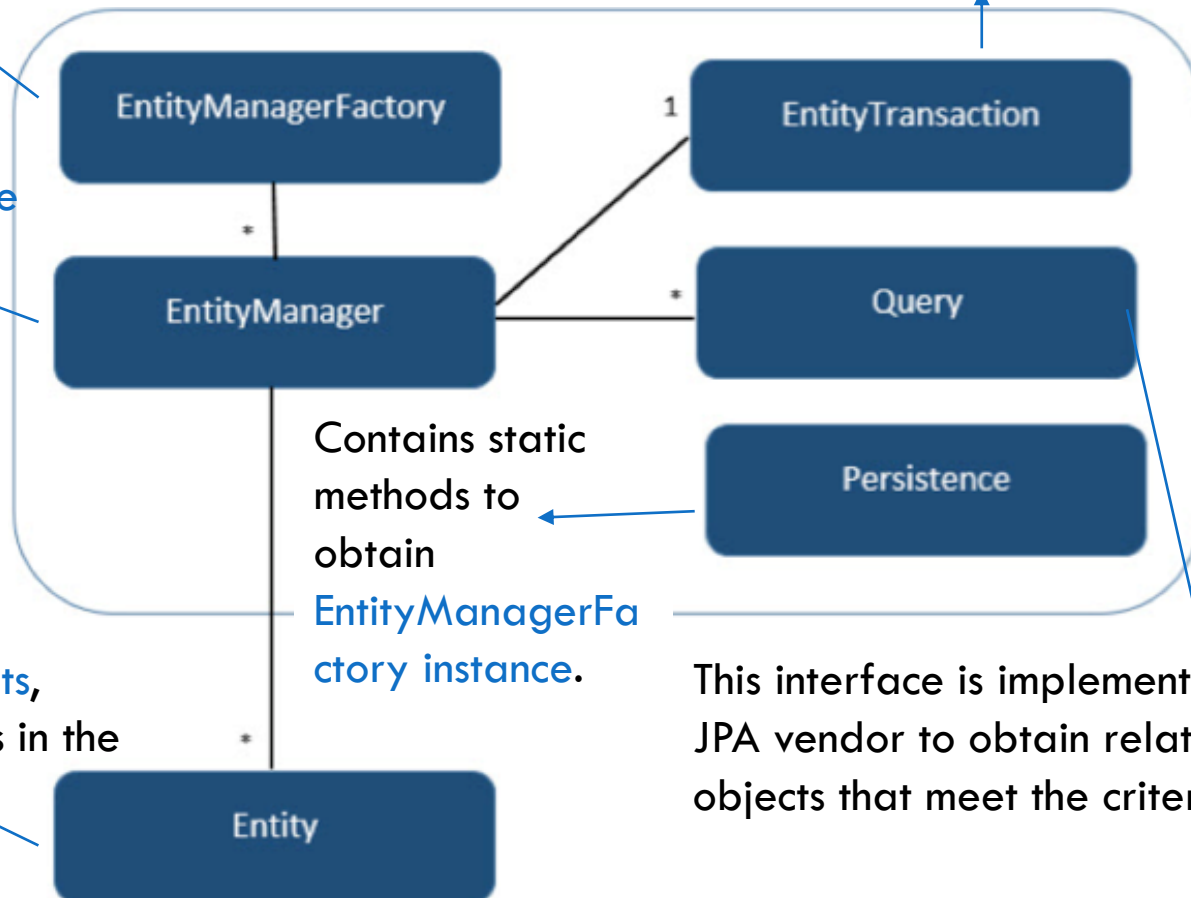


JPA: architettura delle classi

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?

