PROGRAMMAZIONE ORIENTATA AI MICROSERVIZI

Accesso ai dati

Tommaso Nanu
tommaso.nanu@unipr.it

UNIVERSITÀ DI PARMA
DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE E INFORMATICHE
Corso di Laurea in Informatica

Ripasso del modello relazionale

Modello relazionale

- Basato su due concetti:
  - Tabella: concetto semplice e intuitivo
  - Relazione: formale, arriva direttamente dalla matematica

Relazione matematica

Dati n insiemi , con $n > 0$, il **prodotto cartesiano**

è l'insieme delle *n*-uple

Una **relazione** su (chiamati domini della relazione) è un sottoinsieme del prodotto cartesiano

Relazione matematica

Dati gli insiemi

Prodotto cartesiano:

Un possibile sottoinsieme di può essere:

Tabella

Insieme di righe e colonne, dove:

- i-esima riga: raccolta di valori correlati ad un oggetto, ovvero un elemento del prodotto cartesiano
- i-esima colonna: «ruolo» svolto all'interno della riga, definito attributo

Attributi

Lo scopo delle relazioni è quello di organizzare i dati della nostra base dati, dove ogni n-upla contiene dei dati tra loro collegati.

Nell'ambito del modello relazionale di un DBMS è utile considerare le n-uple di una relazione come delle sequenze di dati non ordinati; usiamo gli *attributi* per distinguere i valori in esse contenuti

Relazioni, tabelle e attributi

Riprendendo l’esempio precedente:

Relazione P	
Attributo	**Attributo**
1	a
1	b
4	b

Base dati relazionale

Insieme di tabelle (o relazioni) le cui righe contengono le informazioni rilevanti per la nostra applicazione; ove necessario, sono presenti valori comuni con lo scopo di stabilire corrispondenze tra righe o tabelle differenti.

Studenti			
Matricola	**Cognome**	**Nome**	**Data di nascita**
276545	Rossi	Maria	25/11/1981
485745	Neri	Anna	23/04/1982
200768	Verdi	Fabio	12/02/1982
587614	Rossi	Luca	10/10/1981
937653	Bruni	Mario	01/12/1981

Esami		
Studente	**Voto**	**Corso**
276545	28	01
276545	27	04
937653	25	01
200768	24	04

Corsi		
Codice	**Titolo**	**Docente**
01	Analisi	Giani
03	Chimica	Melli
04	Chimica	Belli

Vincoli di integrità

Studenti			
Matricola	**Cognome**	**Nome**	**Data di nascita**
200768	Verdi	Fabio	12/02/1982
937653	Rossi	Luca	10/10/1981
937653	Bruni	Mario	01/12/1981

Esami			
Studente	**Voto**	**Lode**	**Corso**
200768	36		05
937653	28	sì	01
937653	30	sì	04
276545	25		01

Corsi		
Codice	**Titolo**	**Docente**
01	Analisi	Giani
03	Chimica	Melli
04	Chimica	Belli

Vincoli di integrità

Studenti			
Matricola	**Cognome**	**Nome**	**Data di nascita**
200768	Verdi	Fabio	12/02/1982
937653	Rossi	Luca	10/10/1981
937653	Bruni	Mario	01/12/1981

Esami			
Studente	**Voto**	**Lode**	**Corso**
200768	**36**		**05**
937653	28	**sì**	01
937653	30	sì	04
276545	25		01

Corsi		
Codice	**Titolo**	**Docente**
01	Analisi	Giani
03	Chimica	Melli
04	Chimica	Belli

Vincoli di integrità

In generale, non tutte le possibili n-uple (o tuple) sono «corrette», ovvero non tutte le tuple della nostra relazione identificano un'informazione valida per la nostra applicazione.

Introduciamo un serie di regole, ovvero i **vincoli di integrità**, che le tuple delle nostre relazioni devono rispettare; tali vincoli sono delle espressioni che assumono valore true o false rispettivamente se la tupla i-esima rispetta o meno il vincolo.

Vincoli di integrità

- Vincoli di tupla
- Vincoli di chiave
- Vincoli di integrità referenziale

Vincoli di tupla

I **vincoli di tupla** esprimono condizioni sui valori all'interna della riga, indipendentemente dalle altre righe.

tra

Esami			
Studente	**Voto**	**Lode**	**Corso**
200768	**36**		**05**
937653	28	**sì**	01
937653	30	sì	04
276545	25		01

il

Il voto deve essere compreso 18 e 30

La lode è ammissibile solo se voto è pari a 30.

Vincoli di chiave

La **chiave primaria** è un insieme di colonne che identificano univocamente le righe di una tabella.

- Univocità delle righe
- Stabilire le corrispondenze tra dati in tabelle differenti

Studenti			
Matricola	**Cognome**	**Nome**	**Data di nascita**
200768	Verdi	Fabio	12/02/1982
937653	Rossi	Luca	10/10/1981
937653	Bruni	Mario	01/12/1981

Nota: identifichiamo la chiave primaria sottolineando le colonne che la compongono

Vincoli di integrità referenziale

Un **vincolo di integrità referenziale** tra un insieme di colonne C della tabella T_1 e la tabella T_2 è soddisfatto se i valori su C, per ogni riga di T_1, compaiono come valori di chiave primaria* in T_2.

In pratica, ciascuno degli attributi in C deve corrispondere ad un preciso attributo della chiave primaria di T_2.

* alcuni motori di database lo permettono anche su vincoli UNIQUE.

Vincoli di integrità referenziale

Esami				
Studente	**Voto**	**Lode**	**Corso**	**Anno**
200768	29		05	2022
937653	28		01	2023
937653	30	sì	04	2023

Corsi			
Codice	**Anno**	**Titolo**	**Docente**
01	2023	Analisi	Giani
03	2022	Chimica	Melli
04	2023	Chimica	Belli

Tipi degli attributi

- Character: char, varchar, nchar, nvarchar
- Numerici esatti: integer, decimal, bit
- Numerici approssimati: float, real, double
- Istanti temporali: date, time, timestamp
- Binary: binary

ADO.NET

Accesso ai dati

ADO.NET

Insieme di strumenti Microsoft per l'accesso ai dati. Si basa su due componenti principali:

1. Provider di dati

2. DataSet

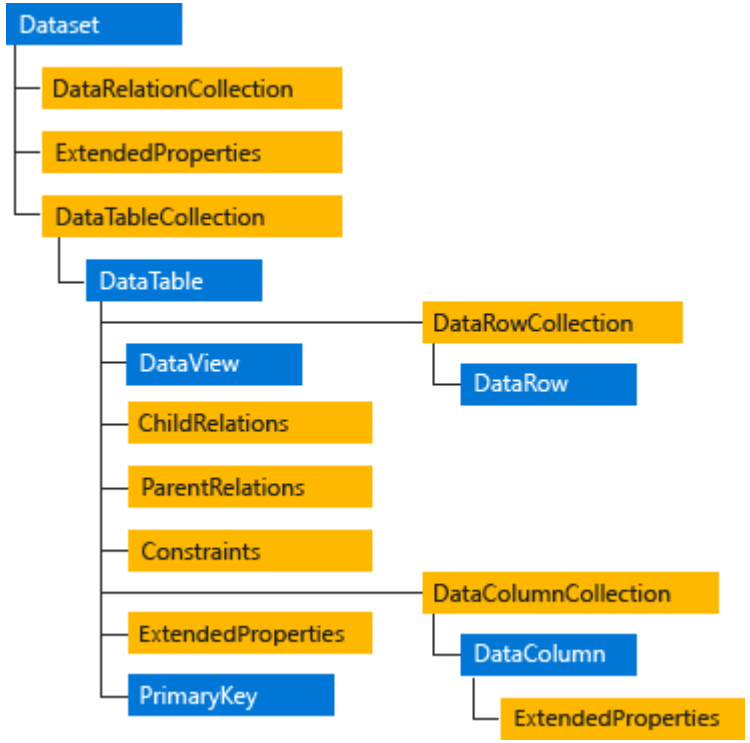
Provider di dati

Insieme di componenti per l'accesso ai dati di tipo forward-only e per la modifica dei dati

- **Connection**: stabilisce la connettività verso un'origine dati
- **Command**: accesso ai comandi del database (SELECT; UPDATE; stored procedure)
- **DataReader**: componente per leggere i dati in arrivo
- **DataAdapter**: definisce il collegamento tra origine dati e DataSet

DataSet

- Contenitore di dati indipendente dall'origine dati (SQL Server, XML, …). È composto da una o più DataTable, definiti come un insieme di righe e colonne; comprende anche le informazioni tipiche dei database relazionali quali vincoli di chiave primaria, vincoli di integrità referenziale tra gli oggetti DataTable del DataSet.



https://learn.microsoft.com/it-it/dotnet/framework/data/adonet/ado-net-architecture

Connessione a SQL Server

```
using System.Data.SqlClient;

string connectionString =
"Server=localhost,2433;Database=MyDB;User Id=sa;Password=p4ssw0rD;Encrypt=False";

using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
}
```

Lettura dati da SQL Server

```
    string connectionString = "Server=localhost,2433;Database=MyDB;User
Id=sa;Password=*;Encrypt=False";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        using (SqlCommand command = new SqlCommand(
            "SELECT Matricola, Cognome FROM Studenti;",
            connection))
        {
            connection.Open();

            using (SqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    Console.WriteLine("{0}\t{1}", reader.GetInt32(0), reader.GetString(1));
                }
            }
        }
    }
```

Caricamento di un DataSet

```
    string connectionString = "Server=localhost,2433;Database=MyDB;User
Id=sa;Password=*;Encrypt=False";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string queryString = "SELECT Matricola, Cognome FROM Studenti";

        using (SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection))
        {
            DataSet customers = new DataSet();
            adapter.Fill(customers, "Studenti");
        }
    }
```

SQL Server su Docker

SQL Server su Docker

```
version: '3.4'

services:
mssql-server:
   image: "mcr.microsoft.com/mssql/server:2019-latest"
   environment:
     ACCEPT_EULA: "Y"
     MSSQL_PID: Developer
     MSSQL_SA_PASSWORD: p4ssw0rD
   ports:
     - 2433:1433
   volumes:
     - "mssql-server:/var/opt/mssql"
adminer:
   image: adminer:4.8.1
   ports:
     - 2431:8080
   environment:
     TZ: Europe/Rome
     ADMINER_DEFAULT_SERVER: mssql-server

volumes:
mssql-server:
```

ORM: Object Relational Mapping

ORM

Tecnica di programmazione per la manipolazione dei dati, favorendo l'integrazione tra software OO e una basi di dati relazionali.

Di fatto, gli oggetti del database vengono «mappati» come oggetti di un linguaggio OO.

Corsi		
Codice	**Titolo**	**Docente**
01	Analisi	Giani
03	Chimica	Melli
04	Chimica	Belli

```
public class Corsi
{
    [Key]
    public int Codice { get; set; }

    public string Titolo { get;
set; }

    public string Docente { get; set;
}
}
```

ORM

Vantaggi:

- Portabilità del software rispetto al DBSM utilizzato
- Scrittura di codice ad alto livello (C# nel nostro caso)
- Stratificazione del software, isolando la logica di accesso al DB
- Riduzione del codice sorgente: esempio, le operazioni CRUD possono essere scritte in pochissime righe
- Protezione da attacchi informatici come SQL injection

Svantaggi:

- Per alcune tipologie di query molto complesse, possiamo riscontrare una riduzione delle prestazioni rispetto alla scrittura diretta di query SQL; alcune volte è proprio impossibile passare dall'ORM
- Apprendimento iniziale non banale

Entity Framework Core

Entity Framework Core

- Framework open source e multipiattaforma di Microsoft
- Sviluppato a partire da Entity Framework (non più sviluppato attivamente)
- Supporta più provider di database (SQL Server, Oracle, PostgreSQL, Sqlite, MongoDB, …)
- Interazione con i dati mediante oggetti .NET fortemente tipizzati

Modello di EF

Il modello è un componente di EF che mappa le entità del database. Tipicamente, un'applicazione ha un insieme di modelli, ovvero un insieme di classi C#, opportunamente configurati.

Vengono messe a disposizione del programmatore due approcci:

1. Generare un modello a partire da una base dati esistente
2. Scrivere un modello (in C#) e usare le migrazioni per creare una base dati

Modello di EF

```
public class Corsi
{
    public int Codice { get; set; }
    public required string Titolo { get; set; }
    public required string Docente { get;
set; }

    public List<Esami> ListaEsami { get; set; }
}
```

```
public class Studenti
{
    public int Matricola { get; set; }
    public required string Cognome { get; set; }
    public required string Nome { get; set; }
    public DateTime DataDiNascita { get; set; }

    public List<Esami> ListaEsami { get; set; }
}
```

```
public class Esami
{
    public int CorsiId { get; set; }
    public int StudentiId { get; set; }
    public int Voto { get; set; }
    public bool Lode { get; set; }

    public Corsi Corso { get; set; }
    public Studenti Studente { get; set;
}
}
```

DbContext

Il DbContext consente la manipolazione dei dati presenti sulla base dati attraverso le entità C#; una sua istanza rappresenta una sessione di lavoro con la base dati.

Contiene al suo interno, come proprietà public, degli oggetti *DbSet<TEntity>:* consentono la manipolazione dei dati sulle corrispondenti entità *TEntity* presenti nella base dati. Di fatto tutte le query LINQ che agiscono su un *DbSet<TEntity>* verranno tradotte in query SQL ed eseguite sulla base dati

DbContext

Progettata per essere usata per una singola sessione di lavoro; ciò significa che la sua durata è molto breve.

È un tipo che implementa *IDisposable*, dobbiamo quindi ricordarci di invocare il metodo *Dispose()* sull'istanza in uso; d'altra parte, può sfruttare il meccanismo di *dependency injection*, per cui non dobbiamo preoccuparci dell'eliminazione dell'istanza. In particolare verrà registrato come un servizio *scoped* e, tipicamente, verrà configurato per leggere la stringa di connessione dalla configurazione della nostra applicazione.

Attenzione: non è thread-safe!

DbContext

```
public class UniprDbContext : DbContext
{
    public UniprDbContext(DbContextOptions<UniprDbContext> options) : base(options) { }

    protected override void OnModelCreating(ModelBuilder torea)
    {
        // configurazione del modello tramite API fluent
    }

    public DbSet<Studenti> Studenti { get; set; }
    public DbSet<Corsi> Corsi { get; set; }
    public DbSet<Esami> Esami { get; set; }
}
```

DbContext – Ciclo di vita

- Creazione dell'istanza
- Tracking delle entità:
  - Query in lettura
  - Inserimento di nuove entità
- Modifica delle entità sotto tracking (se la nostra istanza di logica di business contempla una modifica)
- Chiamata al metodo *SaveChanges():* le entità sotto tracking vengono rese persistenti nella base dati
- Eliminazione dell'istanza

Configurazione di un modello

Entity Framework Core utilizza un modello di metadati per effettuare il mapping degli oggetti .NET verso il database sottostante. La sua configurazione può essere fatta attraverso:

1. Fluent API, tramite override del metodo *OnModelCreating* senza modificare le classi entità

2. Data Annotation, direttamente nelle classi entità

Fluent API

```
public class UniprDbContext : DbContext
{
    public UniprDbContext(DbContextOptions<UniprDbContext> options) : base(options) { }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // configurazione del modello tramite API fluent

        modelBuilder.Entity<Studenti>().ToTable("Studenti");

        modelBuilder.Entity<Studenti>().HasKey(s => s.Matricola);
        modelBuilder.Entity<Corsi>().HasKey(c => c.Codice);
        modelBuilder.Entity<Esami>().HasKey(e => new { e.CorsiId, e.StudentiId });
    }

    public DbSet<Studenti> ListaStudenti { get; set; }
    public DbSet<Corsi> Corsi { get; set; }
    public DbSet<Esami> Esami { get; set; }
}
```

Data Annotation

```
[Table("Studenti")]
[PrimaryKey("Matricola")]
public class Studenti
{
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Matricola { get; set; }

    public required string Cognome { get; set; }

    public required string Nome { get; set; }

    public DateTime DataDiNascita { get; set; }

    public List<Esami> ListaEsami { get; set; } = new List<Esami>();
}
```

Configurazione delle proprietà delle entity

- Nome di colonna
- Tipo della colonna
- Lunghezza massima (string o byte[])
- Precisione (decimal o DateTime)
- Nullable

Tali configurazioni oltre ad essere utilizzate dal DbContext per manipolare i dati presenti su database, sono indispensabili alle routine di *migrations* in tutti quegli scenari dove lo schema del database viene creato/modificato a partire dal codice sorgente C#, ovvero a partire dai model.

Configurazione della chiave primaria

Ogni model deve obbligatoriamente avere una chiave primaria, implicita o esplicita. È possibile, inoltre, informare EF che la chiave primaria non verrà impostata dall'utilizzatore finale bensì verrà autogenerata dalla base dati.

- Implicita, chiamando il campo chiave *Id* oppure *<NomeEntità>Id*
- Esplicita, attraverso l'attributo di data annotation *[Key]* oppure la fluent API *.HasKey()*

Nota: i campi autogenerati possono essere configurati anche in campi non chiave.

Relazioni in EF Core

Una relazione ci permette di correlare due entità. Nei nostri esempi precedenti, esiste una correlazione tra le entità Studente ed Esami.

Uno studente può registrare 0, 1 o più esami mentre, viceversa, un Esame deve obbligatoriamente essere sostenuto da uno Studente.

In tal caso la relazione tra Studenti ed Esami è definita essere 1 a Molti.

Relazioni in EF Core

In EF Core le proprietà evidenziale dono denominate «Naviations» o «Navigation Properties»

```
public class Studenti
{
    public int Matricola { get; set; }
    public required string Cognome { get; set; }
    public required string Nome { get; set; }
    public DateTime DataDiNascita { get; set; }

    public List<Esami> ListaEsami { get; set; }
}
```

```
public class Esami
{
    public int CorsiId { get; set; }
    public int StudentiId { get; set; }
    public int Voto { get; set; }
    public bool Lode { get; set; }

    public Corsi Corso { get; set; }
    public Studenti Studente { get; set; }
}
```

Relazioni in EF Core

Analogamente alla chiave primaria, la configurazione di una relazione viene applicata automaticamente da entity framework quando i nomi delle proprietà sono «parlanti», cioè costruiti con un prefisso uguale al nome del model concatenato alla stringa «Id»: *StudentiId, CorsiId*.

Si dovrà configurare esplicitamente la relazione qualora i nomi delle proprietà non rispettino le condizioni descritte qui sopra.

```
    public class Esami
    {
        public int ID_CORSO { get; set; }
        public int ID_STUDENTE { get;
set; }
        public int Voto { get; set; }
        public bool Lode { get; set; }
        public Corsi Corso { get; set; }
        public Studenti Studente { get;
set; }
    }
    public class Studenti
    {
        public int Matricola { get; set; }
        public required string Cognome { get;
    set; }
        public required string Nome { get;
    set; }
        public DateTime DataDiNascita { get;
    set; }

        public List<Esami> ListaEsami { get;
set; }
    }
```

```
modelBuilder.Entity<Studenti>()
.HasMany(e => e.ListaEsami)
.WithOne(e => e.Studente)
.HasForeignKey(e => new
{ e.ID_STUDENTE });
```

Query

EF Core utilizza LINQ per l'esecuzione di query sulla base dati. Attraverso i model e il context, LINQ produce una rappresentazione della query (indipendente dal tipo di base dati utilizzata) che poi passa al provider specifico (SQL Server, Oracle, …).

Il provider provvede a convertire la query nel linguaggio specifico del motore DB, T-SQL per SQL Server, SQL per Oracle, …

Query

```
var contextOptions = new DbContextOptionsBuilder<UniprDbContext>()
    .UseSqlServer(
"Server=localhost,2433;Database=UNIPR;User
Id=sa;Password=p4ssw0rD;Encrypt=False")
    .Options;

using var context = new UniprDbContext(contextOptions);

var studenti = context.Studenti.ToList();
```