# Esercitazione di Fine Settimana – Week 5

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Roberta |
|  |  | Cognome | Beretta |
|  |  | Data | 8/10/2021 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.

1. Descrivere le modalità di utilizzo di ADO.NET

ADO.NET è un framework progettato per far comunicare uno strato di persistenza con una app.

ADO.NET si può utilizzare in due modalità, ovvero la modalità connected e quella disconnected.

Nella prima si richiede l’accesso al database, si fanno le operazioni desiderate e poi si chiude la connessione. Le modifiche al database vengono fatte dunque “in tempo reale”.

Nella seconda modalità invece si richiede l’accesso al database per farne una copia locale, dopodichè si chiude la connessione.

Le modifiche che si vogliono fare al database vengono fatte dunque nella copia locale, dopodichè si procede all’allineamento con il database remoto riaprendo la connessione.

Tutte queste operazioni vengono fatte tramite delle classi definite in un pacchetto nuGet apposito.

In entrambe le modalità di utilizzo di ADO.NET (nel nostro caso abbiamo usato come strato di persistenza SQLServer) utilizzano le classi SqlConnection per tentare la connessione al DBm SqlCommand che permette le operazioni di CRUD tramite Query.

Inoltre nel disconnected mode si utilizzano le classi DataSet e DataAdapter per creare la copia in locale del DB remoto.

1. Quali sono i metodi di esecuzione della classe DbCommand disponibili e in quali casi vanno utilizzati?

SqlCommand è la classe che va istanziata qualora si voglia interagire con il DB tramite query.

Questa settimana abbiamo visto tre metodi di esecuzione della classe: ExecuteReader(), ExecuteNonQuery() ed ExecuteScalar().

Il primo si utilizza con il comando SELECT che restituisce una proiezione. ExecuteReader legge una riga della tabella e la restituisce un oggetto che contiene le informazioni del record letto.

Il secondo metodo invece si utilizza con comandi tipo INSERT, UPDATE e DELETE che non restituiscono una proiezione ma vanno a modificare i record del DB. ExecuteReader() restituisce il numero di righe che sono state impattate dal comando.

Infine ExecuteScalar() restituisce il valore del primo campo della prima riga della tabella che viene restituita dalla query. Questo metodo è dunque utile quando ad esempio si vuole mostrare il risultato di una funzione scalare come ad esempio Count(), Max(), Min() ecc..

1. Descrivere l’utilizzo della classe DataAdapter

Come dicevo prima, DataAdapter è una classe che viene utilizzata nella modalità disconnected di ADO.NET e lavora in modo molto ravvicinato con la classe DataSet.

DataSet serve per creare una copia del DB remoto in locale. É la classe responsabile della creazione di tabelle, campi, record, relazioni e vincoli tramite classi derivate.

DataAdapter invece è la classe che mantiene l’allineamento tra i record della copia locale del DB e lo strato di persistenza in remoto.

Ad esempio abbiamo visto il metodo Fill() che serve a riempire i campi di una tabella creata con DataSet.

1. Come si definisce il modello dati in Entity Framework?

Il modello dati di EF si chiama Object Relational Mapping (ORM) ovvero, cercando di descrivelo con una immagine, si ha che il framework si pone in mezzo tra la app e il database e il suo compito è mappare ogni istanza di un certo oggetto in C# con un record della corrispettiva tabella nel DB.

Questa corrispondenza viene fatta in modo pressochè automatico, quindi a differenza di ADO.NET, non è necessatia la conoscenza del lingiaggio SQL.

1. Descrivere l’utilizzo delle Migration e i vantaggi che ne derivano

Abbiamo visto che EntityFramework lo si può approcciare in modalità CodeFirst, ovvero tutto il progetto di database lo si realizza tramite codice C#.

Ciò che si fa in Sql è creare il database fisico, dopodichè tutta la struttura, tutte le relazioni le si creano tramite codice.

Le migration servono dunque a tradurre e passare tale stuttura da codice a implementazione nel DB.

Le migration servono anche ad aggiornare il database qualora vengano fatte modifiche successive alla prima implementazione, come ad esempio se vengono aggiunte o rimosse entità, se vengono create nuove relazioni ecc..

Uno dei vantaggi che mi viene in mente è il fatto che il lavoro viene in sostanza fatto tutto in un linguaggio e in un programma.

**Esercitazione Pratica**

Realizzare un sistema di gestione delle spese che si basi su:

* Un database **GestioneSpese** (SQL Server), costituito dalle tabelle
  + **Spese**
    - *Id* (int, PK, auto-incrementale)
    - *Data* (datetime)
    - *CategoriaId* (int, FK)
    - *Descrizione* (varchar(500))
    - *Utente* (varchar(100))
    - *Importo* (decimal)
    - *Approvato* (bit)
  + **Categorie**
    - *Id* (int, PK, auto-incrementale)
    - *Categoria* (varchar(100))
* Una **Console app** che consenta di:
  + Inserire nuove Spese
  + Approvare le Spese esistenti (impostare il campo *Approvato*)
  + Cancellare le Spese esistenti
  + Mostri
    - l'elenco delle Spese Approvate
    - L'elenco delle Spese di uno specifico Utente
    - Il totale delle Spese per Categoria

VINCOLI TECNICI

* Utilizzare Entity Framework
* Utilizzare l'approccio Code-First e attivare le Migrations

**OPZIONALE**: Implementare una o più delle funzionalità utilizzando ADO.NET (Connected o Disconnected mode, a scelta)

*Es. Inserire una nuova spesa*