# Esercitazione di Fine Settimana – Week 5

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Laura |
|  |  | Cognome | Martines |
|  |  | 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 si può utilizzare in due modalità diverse:

* Connessa 🡪 così chiamata perché necessita che la connessione con il Database resti aperta per tutta la durata delle operazioni. Permette di eseguire tutte le operazioni di CRUD sui dati, ma la lettura avviene in modalità forward-only, ossia permette di leggere i record in ordine, ma non esiste un cursore che mi permetta di spostarmi da un record all’altro. Utililizza principalmente i comandi di Connection (per creare la connessione al DB), Commands (pere generare i comandi da inviare al DB), DataReader(per leggere le informazioni dal dDB).
* Disconnessa🡪 in questo caso è possibile lavorare offline, quindi ci si connette al DB per scaricare in locale una copia dei dati per poi lavorare senza occupare la connessione con il database. Infine, si attua una riconciliazione dei record modificati con quelli nel DB remoto. In questo caso i comandi utilizzati sono diversi, perché oltre ai precedenti esiste il DataSet, che è la copia in locale del DB; essa è una struttura dati formata da diversi altri elementi come DataTable, DataColumn, DataRow... ognuno dei quali rappresenta una collezione di dati (tabelle, colonne delle tabelle, righe..); infine è molto importante il DataAdapter, che connette il DB remoto con quello locale, gestisce i commandi da eseguire sul DB e la riconciliazione (aprendo in automatico la connessione con il DB per il tempo necessario).

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

La classe DbCommand è la classe padre di tutta una serie di classi specifiche per ogni tipo di Database utilizzabile; nel caso di Microsoft SQL server, si utilizza la classe SqlCommand, che deriva appunto da DbCommand.

Essa ha diversi comandi a disposizione; i principali sono:

* CommandType, che ci permette di identificare il tipo di comando che vogliamo inviare al DB; abbiamo quindi TableDirect, per leggere direttamente i dati di una tabella; oppure StoredProcedure per richiamare una StoredProcedure appunto o infine quello più comunemente usato (e più generale) che è text, che permette di scrivere direttamente sotto forma di stringa lo statement SQL da eseguire.
* CommandText è un comando il cui contenuto dipende dalla scelta effettuata per il commandType: se si è scelto TableDirect, il commandText conterrà il nome della tabella da leggere; se si è scelto StoredProcedure si avrà il nome della stored procedure da eseguire; infine, se il commandType scelto è text, il commandText conterrà la stringa con lo statement SQL.
* Connection 🡪 in cui si specifica la connessione da utilizzare

1. Descrivere l’utilizzo della classe DataAdapter

La classe DataAdapter è, come nel caso precedente, una classe che padre da cui derivano classi specifiche per il singolo DB (nel caso di Microsoft SQL Server, si usa SqlDataAdapter). Essa gestisce le comunicazioni tra DB remoto e copia in locale (DataSet), quindi si usa per popolare il dataset, ma anche per fare la riconciliazione dei dati con il DB remoto. Essa tiene traccia si tutte le modifiche effettuate in locale sui record, e al momento della riconciliazione le applica al DB remoto (a patto che siano stati definiti dal programmatore i metodi da eseguire per ogni operazione di CRUD).

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

Il modello dati in EF si definisce in maniera diversa a seconda del tipo di approccio che si utilizza:

* nel DB-First, il modello deriva dalla struttura del DB;
* nel Model-First, come dice il nome stesso, il modello è il primo ad essere creato e poi da esso si derivano le classi OOP ma anche le tabelle del DB;
* nel Code-First, il modello viene creato dal codice, e da esso poi si genera il DB.

Prendendo in considerazione l’approccio Code-First, il modello viene definito a partire dal codice, quindi dalla generazione delle classi che rappresenteranno i record, dalla creazione del DbSet; per definire la mappatura tra OOP e DB si usano nei casi più semplici le convenzioni di Type Discovery, altrimenti è possibile usare le Data Annotations (che permettono di limitare la quantità di codice da scrivere, ma per contro non possono modellare tutti gli aspetti) oppure le Fluent API, che richiedono una maggiore quantità di codice ma sono più complete.

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

Le migrations permettono di aggiornare/creare il DB in modo automatico quando il modello viene modificato. Posso essere svolte in maniera automatica (scelta poco consigliata perché invasiva) oppure possso essere triggerate manualmente dallo sviluppatore. Quando una migration viene creata tramite il comando “add-migration” seguito dal nome della migration, in automatico viene generata una classe (inserita all’interno della cartella Migrations) che contiene due funzioni, chiamate Up e down. La prima contiene tutti i comandi per aggiornare il DB all’ultima versione del modello, la seconda invece i comandi per eliminare tali aggiornamenti e quindi riportare il DB alla versione precedente. Con il comando update-database è possibile utilizzare la migration per aggiornare il db.

Tale approccio è molto comodo, perché permette di scrivere tutto il codice in C#, quindi da un lato il programmatore utilizza un solo linguaggio. In aggiunta, il codice è tutto contenuto all’interno di Visual studio e quindi è più facile condividerlo con il team e anche inserirlo e monitorarlo in repository come gitHub o Azure DevOps.

**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*