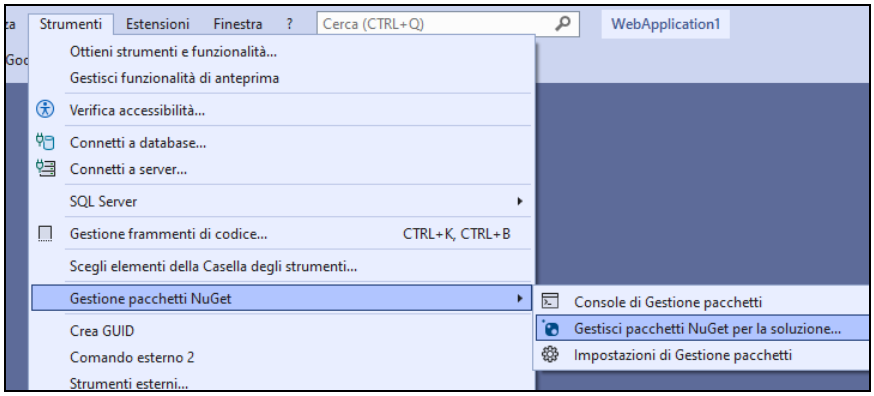
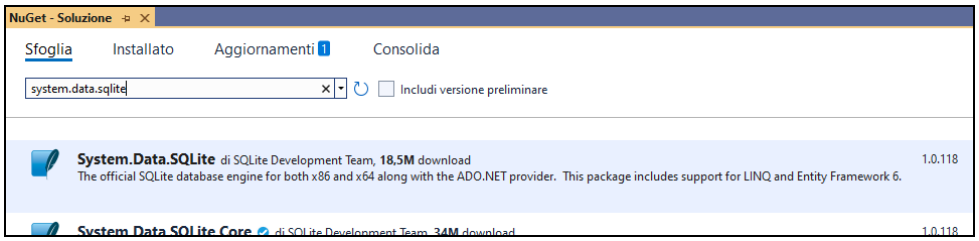
Accesso a DataBase SQLite in C#

Attivare la Libreria SQLite in un progetto Visual Studio

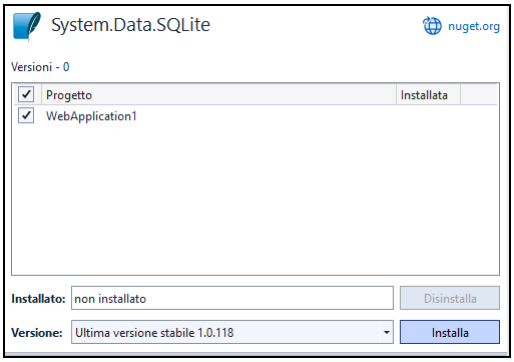
La **Libreria SQLite** deve essere installata nel Progetto Visual Studio tramite il comando "**Strumenti / Gestione Pacchetti NuGet / Gestisci pacchetti NuGet per la Soluzione**":



Nella **scheda Sfoglia**, cercare "*system.data.sqlite*" e cliccare sulla voce "**System.Data.SQLite**" risultante dalla ricerca, come in figura:



Segnare la **casella "Progetto"** e cliccare su "**Installa**" per procedere all'installazione:

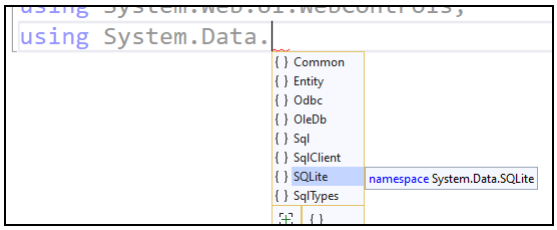


Accettare le richieste delle varie finestre di conferma che appaiono nell'installazione.

Attivare il NameSpace di SQLite

Per **Accedere a un DataBase SQLite da Codice C#** è necessario includere la **Libreria SQLite**, tramite la clausola **using**:

using System.Data.SQLite



La Connessione al DataBase

classe SQLiteConnection

La classe **SQLiteConnection** consente di gestire **la "connessione" al DataBase**, senza la quale non è possibile accedere al DataBase stesso.

```
SQLiteConnection cn = new SQLiteConnection
                    ( "DataSource = <percorso><nome-file-database>" )
```

Il parametro è una stringa (detta **Stringa di Connessione**) che specifica: il *nome del file* del DataBase (<nome-file-database>) e in che *posizione* (<percorso>) cercarlo.

In una **Applicazione Windows con DataBase Locale**, il file del DataBase viene cercato nella *stessa cartella ove si trova il file eseguibile* dell'applicazione (es.: **bin/debug**). Se si posiziona il DataBase in tale cartella, non è quindi necessario specificare un percorso.

In una **Applicazione Web ASPX pubblicata su un Server Web**, invece, il file del DataBase si trova generalmente *nella cartella principale del Sito* ed è comunque necessario specificarne il percorso: la proprietà **Request.PhysicalApplicationPath** consente di ottenere, da codice, il percorso fisico della cartella del Sito sul Server.

Esempio in caso di Applicazione Windows con DB nella cartella dell'eseguibile:

```
SQLiteConnection cn = new SQLiteConnection
          ( "DataSource = Tennis.db" );   ...<percorso> non necessario
```

Esempio in caso di Applicazione Web ASPX con DB sul Server Web:

```
SQLiteConnection cn = new SQLiteConnection
          ( "DataSource = " +
            Request.PhysicalApplicationPath +    ... <percorso>
            "Tennis.db" );                       ... <nome-file-db>
```

Una volta definito l'oggetto **cn** di classe **SQLiteConnection**, è possibile **Aprire la Connessione** con il metodo **Open**:

```
cn.Open( );
```

A connessione aperta, si procede ad operare con il DataBase e, a operazioni concluse, è necessario **Chiudere la Connessione** con il metodo **Close**:

```
cn.Close( );
```

Ottenere Dati
dal DataBase

Comando SQL
SELECT

classe
SQLiteCommand

Per **Ottenere Dati** da un DataBase SQLite è necessario inviare ad SQLite un **comando SQL di tipo SELECT** che, eseguito, produrrà come risposta una Tabella con i dati richiesti.

E' anzitutto necessario *definire il comando SQL*, usando la classe **SQLiteCommand**:

```
SQLiteCommand  cmd  = new SQLiteCommand
        ( <comando-select>, <connessione> )
```

<*comando-select*> è una *stringa contenente il Comando SQL di tipo SELECT* che specifica quali dati si desidera ottenere dal DataBase.

<*connessione*> è l'oggetto di classe SQLiteConnection precedentemente definito (**cn**).

Esempio:

```
SQLiteCommand  cmd  =  new SQLiteCommand
        ( "SELECT  *  FROM  Tennisti",  cn );
```

Questo comando, quando eseguito, otterrà, come risposta, l'intera tabella "Tennisti".

Una volta definito l'oggetto comando **cmd**, è possibile *inviare il comando* stesso a SQLite tramite il metodo **ExecuteReader**:

```
cn.Open( );
SQLiteDataReader  dr  =  cmd.ExecuteReader( );
```

Il metodo provoca *l'esecuzione del comando SELECT* e produce come risultato una Tabella (ossia una Sequenza di Record) che viene resa disponibile attraverso l'oggetto **dr** di classe **SQLiteDataReader**.

Leggere i Dati Ottenuti dal DataBase

classe SQLiteDataReader

Un oggetto **dr** di classe **SQLiteDataReader** serve ad **ospitare la Tabella Risultato** ottenuta dall'esecuzione del metodo *cmd.ExecuteReader* e, quindi, dall'esecuzione di un *comando SQL di tipo SELECT*.

L'oggetto **dr**, in pratica, contiene una **Sequenza di Record** risultanti da una SELECT.

L'accesso ai Record dell'oggetto **dr** può avvenire solo in modo **sequenziale** (prima si accede al primo record, poi al secondo, … e così via). Lo spostamento da un record al successivo avviene tramite il metodo **Read**:

dr.Read( ) *… restituisce un bool*

La prima volta che *dr.Read* viene richiamato, **dr** si "posiziona" sul primo record. Dopodichè, ogni volta che *dr.Read* viene richiamato, si posiziona sul record successivo.

Se **dr.Read ha restituito TRUE** significa che **era ancora disponibile un record successivo** su cui posizionarsi e che ora **dr** si è posizionato su di esso.

Se **dr.Read ha restituito FALSE**, allora **non ci sono più altri record**: quindi, tutti i record presenti in *dr* sono stati scanditi e *dr* non è più accessibile.

Quando *dr* è posizionato su un record, è possibile accedere ai suoi Campi scrivendo:

dr ["<nome-campo>"] *… restituisce un "object"* (ossia un oggetto generico)

Per processare tutti i record, si usa in genere un **ciclo while** del tipo:

```
while ( dr.Read( ) ) {
   … <istruzioni che processano il record su cui è posizionato dr> …
}
```

Esempio:

```
cn.Open( );
SQLiteDataReader dr = cmd.ExecuteReader( );
while ( dr.Read( ) ) {
   string Cognome = Convert.ToString ( dr["Cognome"] );
   lblElenco.Text = lblElenco.Text + Cognome + " ";
}
```

Fin quando *dr.Read* restituisce TRUE, allora c'è un record da processare e il ciclo esegue le istruzioni concatenando il cognome relativo al tennista del record attivo.

Visualizzato l'ultimo record, non ci sono più altri record: dr.Read restituisce FALSE interrompendo così l'esecuzione del ciclo.

Si noti che, essendo *dr["Cognome"] un object generico*, è necessario convertirlo in stringa con una *conversione esplicita* (Convert).

Alla fine, tutti i Cognomi vengono concatenati e visualizzati in un'unica label *lblElenco*, separati da uno spazio (" ").

Una volta processati i record ottenuti, è necessario chiudere sia **dr** che la connessione **cn**, con i soliti metodi **Close**:

```
dr.Close( );
cn.Close( );
```

Usare i Parametri nei Comandi SQL

Nei comandi SQL è possibile inserire dei "**Parametri**", ossia delle "variabili" alle quali assegnare un valore diverso ad ogni esecuzione del comando.

Se si usano Parametri, si parla di **Query Parametriche**.

In SQL, il **Nome di un Parametro** inizia sempre con il **simbolo @.**

Esempio:

```
SELECT  Nominativo, Altezza
FROM  Tennisti
WHERE  Altezza > @AltezzaData
```

Questa SELECT restituisce solo i Record dei Tennisti più alti di una data altezza.
Il Parametro (@AltezzaData) si riconosce perché inizia con il simbolo @.

Per impostare il valore del Parametro si usa il metodo **AddWithValue** dell'insieme **Parameters** dell'oggetto **cmd**:

```
cmd.Parameters.AddWithValue ( <nome-parametro>, <valore> )
```

Esempio:

```
SQLiteCommand  cmd  =  new SQLiteCommand
            ( "SELECT  Nominativo  FROM  Tennisti " +
              "WHERE  Altezza > @AltezzaData ",  cn );

cmd.Parameters.AddWithValue ( "@AltezzaData",  1.87 )
```

Il Valore attribuito al parametro @AltezzaData) è pari a 1.87

Inviare Comandi SQL che non restituiscono Tabelle Risultato	Se il comando SQL **non è di tipo SELECT** (quindi *CREATE TABLE, INSERT, DELETE, UPDATE* o altro), allora **non viene restituita nessuna Tabella Risultato** ed è necessario utilizzare il metodo **ExecuteNonQuery:** **cmd.ExecuteNonQuery( );** Esempio: SQLiteCommand cmd = new SQLiteCommand ("DELETE FROM Tennisti WHERE idTennista = 2", cn); cn.Open( ); **cmd.ExecuteNonQuery( );** Essendo un comando DELETE, esso non restituisce una Tabella e deve essere eseguito con il *metodo ExecuteNonQuery* (e non con *ExecuteReader*)
Ottenere un Singolo Dato invece che una Tabella	Se si desidera recuperare **solo il Primo Campo del Primo Record** della Tabella Risultato generata da un comando SELECT, è comodo usare il metodo **ExecuteScalar**. **ExecuteScalar** esegue il comando SELECT ma non restituisce tutta la Tabella Risultato bensì **un Singolo Valore**, quello situato nel *Primo Campo del Primo Record*. **cmd.ExecuteScalar( );** *… restituisce un "object"* (ossia un oggetto generico) Il valore restituito è un *object* generico, quindi è necessaria una conversione esplicita. Esempio: SQLiteCommand cmd = new SQLiteCommand ("SELECT Nominativo FROM Tennisti ORDER BY Altezza DESC", cn); cn.Open( ); string TennistaPiuAlto = **Convert.ToString (cmd.ExecuteScalar( ));** Il comando SELECT fornisce *l'elenco dei tennisti ORDINATO secondo le Altezze decrescenti*, quindi, il Primo Campo (Nominativo) del Primo Record, contiene il nome del tennista più alto: *ExecuteReader* restituisce solo questo dato e non la Tabella intera. Da notare la conversione esplicita in stringa.

Uso della Chiave Primaria nelle ListBox/ComboBox delle pagine ASPX

In una Pagina ASPX, quando si **elencano i Record di una Tabella in una ListBox** (o **ComboBox**), è bene inviare al browser non solo il **Testo da Visualizzare** per ogni Record, ma anche le relative **Chiavi Primarie** che individuano i Record univocamente.

Per questo motivo è possibile aggiungere alla ListBox oggetti di classe **ListItem**, che contengono sia il *Testo da Visualizzare*, sia la relativa *Chiave Primaria*.

```
ListItem  VoceElenco = new ListItem ( <Testo>, <Chiave>)
lstElenco.Items.Add ( VoceElenco );
```

Quando, nel browser, viene selezionato un elemento della ListBox, il successivo PostBack invia al server *non solo il Testo, ma anche la Chiave del Record* scelto: questo permette di individuare il Record univocamente per l'operazione da effettuare su di esso.

Esempio che genera una ListBox con Testo e Chiave:

```
lstElenco.Items.Clear( );
SQLiteCommand  cmd  =  new SQLiteCommand
            ( "SELECT  idTennista, Nominativo  FROM  Tennisti",  cn );
cn.Open( );
SQLiteDataReader  dr  =  cmd.ExecuteReader( );
while ( dr.Read( ) )  {
   ListItem VoceElenco = new ListItem
            ( Convert.ToString ( dr.["Nominativo"],          // ... testo da visualizzare
              Convert.ToString ( dr.["idTennista"] );        // ... chiave primaria
   lstElenco.Items.Add ( VoceElenco )
}
dr.Close( );
cn.Close( )
```

Per ogni Record della Tabella Tennisti viene creata (e aggiunta alla ListBox) una "VoceElenco" (di classe *ListItem*) che contiene sia il "Nominativo" (testo che sarà visualizzato nella ListBox), sia la relativa Chiave Primaria "idTennista" (che sarà presente nella ListBox, ma non visibile)

Effettuata la selezione sulla ListBox nel Browser, al successivo PostBack, è possibile recuperare, da codice, la voce selezionata nella ListBox tramite le seguenti proprietà:

```
lstElenco.SelectedValue     ... restituisce una Stringa
lstElenco.SelectedItem      ...restituisce un oggetto ListItem
```

SelectedValue restituisce la **Chiave Primaria** della voce selezionata. Essendo una stringa è necessario convertirla in numero, se la chiave primaria è di tipo Integer.

SelectedItem restituisce l'intero oggetto di classe ListItem relativo alla voce selezionata.

N.B.: è da **evitare l'uso di SelectedIndex**, perché durante l'invio della pagina, la selezione e il postback, altri utenti potrebbero modificare la tabella, provocando un cambiamento delle "posizioni" dei record.

Esempio che elimina il record selezionato nella ListBox:

```
int Chiave = lstElenco.SelectedValue // ... recupero la chiave primaria del tennista selezionato
SQLiteCommand  cmd  =  new SQLiteCommand
            ( "DELETE  FROM  Tennisti  WHERE  idTennista = @id",  cn );
cmd.Parameters.AddWithValue ( "@id",  Chiave )
cn.Open( );
cmd.ExecuteNonQuery( );
cn.Close( )
```