



Empowering Digital Skills For The Jobs Of The Future



by



Sommario

- Overview SQL
- MAUI ToDo
- SQLite
- Data Binding

Memorizzazione dei dati

Nell'ambito di un'app web, è possibile gestire la memorizzazione dei dati, ma si incontrano limitazioni che non sono presenti in MAUI.

Memorizzazione dei dati

La caratteristica distintiva principale di un'app MAUI rispetto a un'app web, è una maggiore sicurezza.

Memorizzazione dei dati

Le app web sono vincolate dalle autorizzazioni e dalle API disponibili nei browser web, impedendo la memorizzazione di dati crittografati in modo sicuro e reversibile. Nel caso in cui un'app web necessiti di accedere a dati crittografati, è obbligata a memorizzare la chiave nella cache del browser, aumentando la vulnerabilità a vari tipi di attacchi.

Memorizzazione dei dati

D'altra parte, le app eseguibili su piattaforma godono di un accesso completo a tutte le API fornite dal sistema e sono vincolate solo dalle autorizzazioni concesse loro dall'utente.

Memorizzazione dei dati

Storage feature	Web app	.NET MAUI app
Files	A user can open and save files. Filesystem access is inconsistent between browsers and OSs, so access to specific files cannot be guaranteed. The app cannot open a file without the user.	A user can open and save files. Abstractions are provided to common locations, meaning a developer can expect a consistent result across platforms. The app can open a file without a user, so data necessary for the app's operation can be loaded in the background.
Preferences	Can be stored in the browser cache but needs to be serialized to a structured text format like JSON. Can also use the Web Storage API.	A common abstraction is provided to the app configuration location on each platform. Data is stored in key-value pairs.
Encryption	Web apps cannot store encrypted data securely. There are ways to encrypt data in a web app, but if you want your app to also be able to read that data, the app needs to store the key, which would be exposed.	A common abstraction is provided that lets you store data in an encrypted location, with a key that is managed by the OS.
Structured data	Web apps can use the IndexedDB API, an object-oriented database API available in modern web browsers.	The .NET ecosystem makes countless options available, including Entity Framework (EF) Core. Any database engine available in the .NET Standard can be used in a .NET MAUI app.

Memorizzazione dei dati

MAUI utilizza una funzionalità chiamata SecureStorage. Utilizza coppie chiave-valore, viene ottenuta una chiave crittografica dal sistema operativo per crittografare il valore dei dati memorizzati.

Memorizzazione dei dati

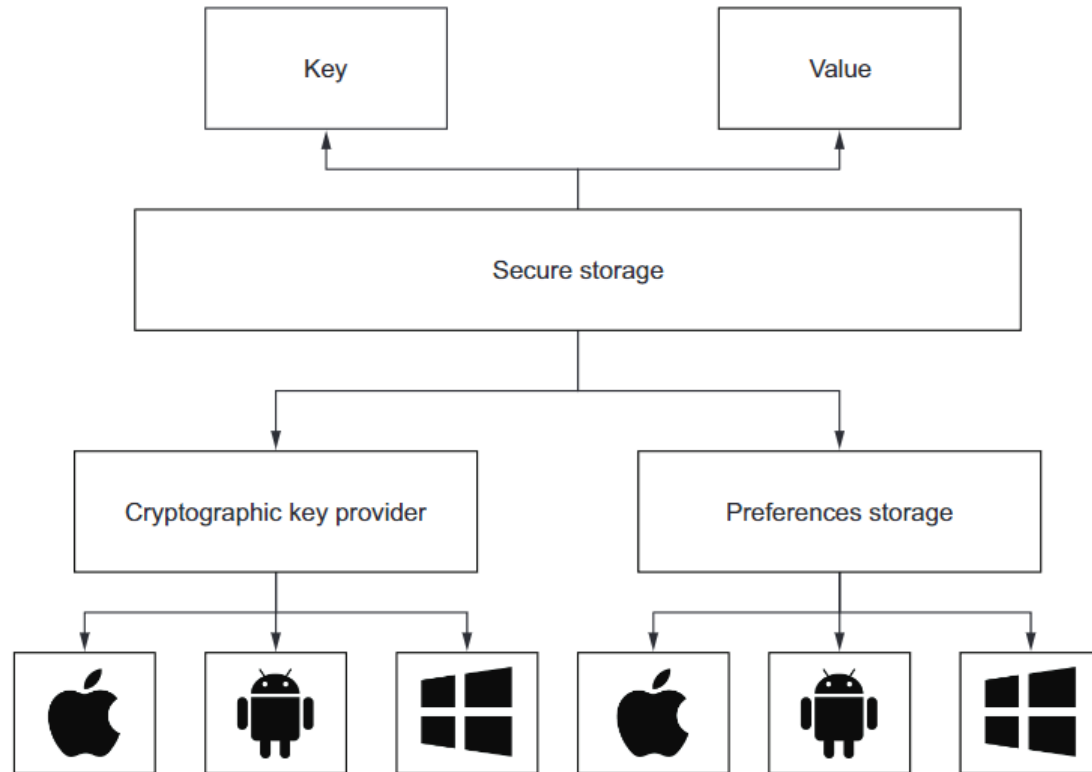
SecureStorage utilizza un'astrazione delle API di gestione dei dati crittografati, in ciascuna piattaforma. Ad esempio, su macOS e iOS, SecureStorage utilizza KeyChain, mentre su Android utilizza Keystore. La chiave crittografica è gestita dal sistema operativo e è disponibile solo per l'app, e nella maggior parte dei casi è supportata da un chip di crittografia hardware.

Memorizzazione dei dati

La crittografia basata sull'hardware è garantita su iOS e macOS grazie a un chip di crittografia integrato, mentre su Android la crittografia hardware è probabile ma non garantita, soprattutto su tablet e smartphone. Nei laptop Windows moderni, la presenza di un chip Trusted Platform Module (TPM) offre crittografia hardware, ma nei desktop questa caratteristica è meno diffusa, anche se il requisito di TPM di Windows 11 potrebbe influenzare questa situazione in futuro.



Memorizzazione dei dati



Introduzione ai Database

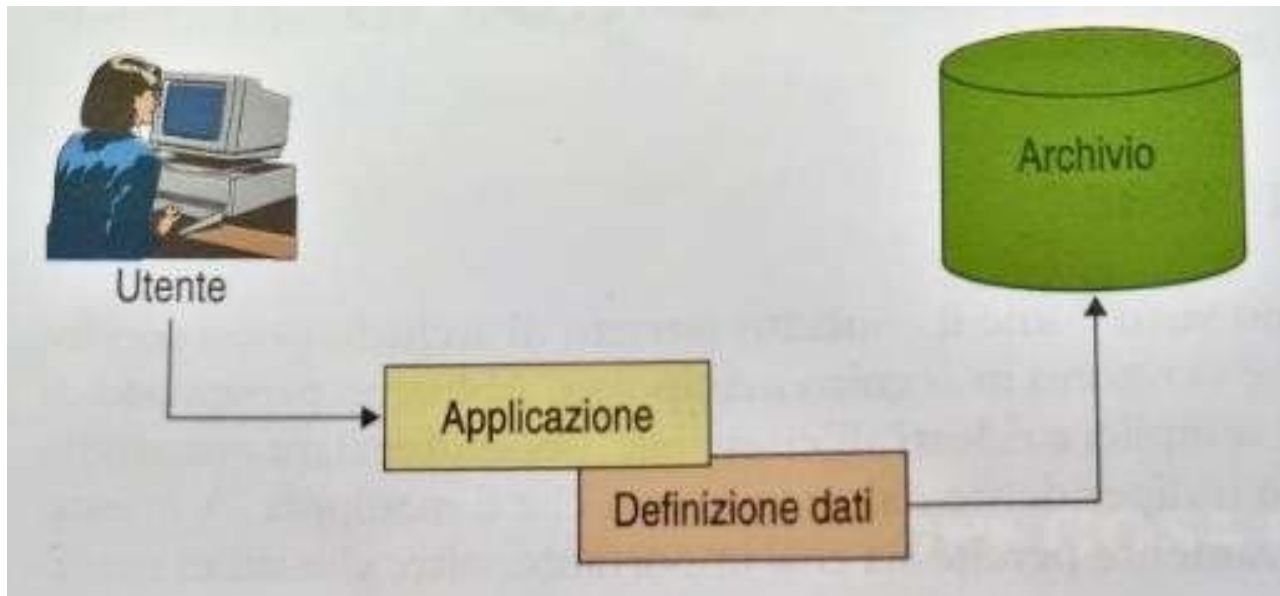
Per poter utilizzare al meglio SQLite è necessario conoscere le basi di SQL.

Perché utilizzare i database

Nei database, la definizione dei dati e i dati stessi vengono salvati all'interno dello stesso database.

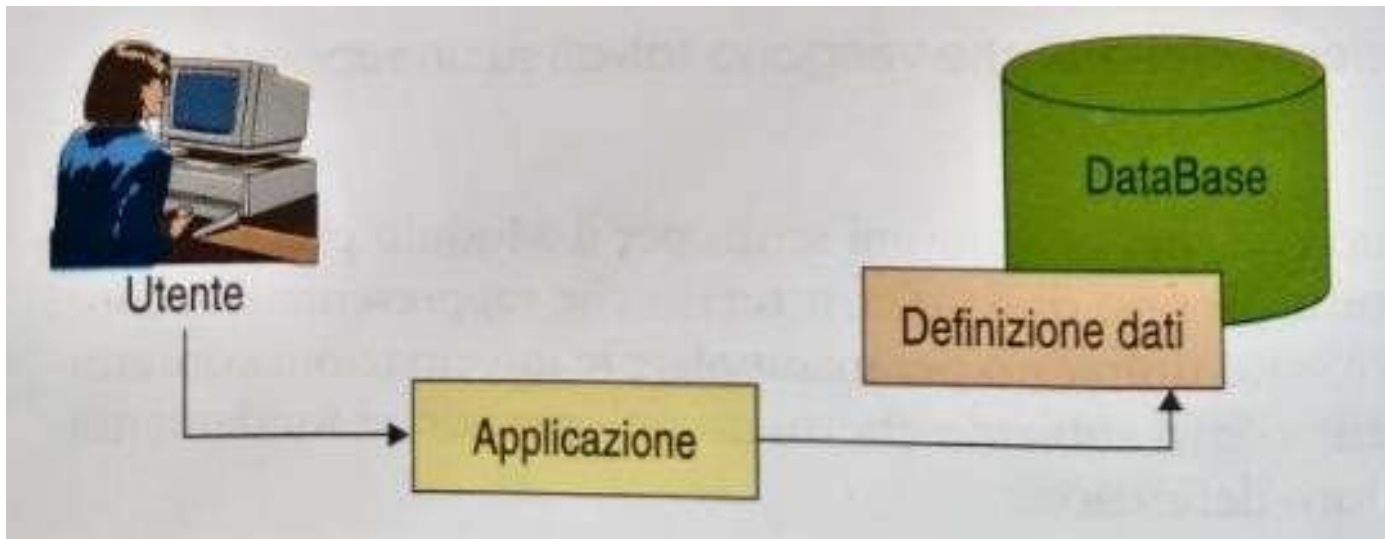
Perché utilizzare i database

Se si usano archivi convenzionali, la definizione dei dati è parte integrante dell'applicazione.



Perché utilizzare i database

Mentre con l'utilizzo di un database, la definizione della struttura dei dati è indipendente dall'applicazione.



Perché utilizzare i database

Quindi chi svolge un'azione di interfaccia tra l'applicazione, i dati e la loro definizione?

Il **DBMS**. Si definisce DBMS il sistema di gestione del database visto nel suo complesso. Il DBMS si preoccupa di gestire interamente i dati, compresa la loro definizione e il modo in cui essi vengono fisicamente archiviati.

Perché utilizzare i database

Isolare i dati e la loro definizione offre un grande vantaggio:

l'applicazione e i suoi dati diventano indipendenti l'una dagli altri.

DBMS

Un **DBMS** deve:

- Gestire grandi quantità di dati.
- Garantire persistenza e condivisione dei dati.
- Offrire funzionalità all'utente per la gestione e lo sviluppo delle applicazioni.
- Garantire velocità di elaborazione.
- Consentire a più utenti di consultare i dati.
- Assicurare l'affidabilità dei dati .

Modellazione dei dati

Un **modello di dati** consiste in una rappresentazione astratta delle strutture dei dati di un database.

L'atto di creazione di un modello prende il nome di **modellazione dei dati**, o in inglese, di *data modeling*.

Modellazione dei dati

Le **strutture dei dati** sono tutti gli oggetti del database e le regole che governano le operazioni tra i dati.

Modellazione dei dati

Un modello dei dati è indipendente dall'hardware o dal linguaggio di programmazione con cui viene sviluppata l'applicazione.

Modellazione dei dati

La modellazione dei dati serve per tradurre i dati dal punto di vista dell'utente al punto di vista dell'applicazione/database.

Modellazione dei dati

Esistono sostanzialmente due modi per modellare i dati:

- **Il modello Entità-Relazione.**
- **Il modello a Oggetti.**

Il modello E-R

Il modello Entità-Relazione, originariamente proposto da Chen nel 1976, permette di modellare il mondo reale utilizzando esclusivamente **entità** e **relazioni**.

Il modello E-R

La sua utilità è riassumibile nei seguenti punti:

- I costrutti utilizzati nel modello E-R possono essere facilmente impiegati per la definizione dei database relazionali.
- Il modello E-R è semplice da capire, perciò può essere usato dai progettisti dei database per comunicare la struttura agli utenti finali.
- Può essere usato per implementare un modello di dati in uno specifico software di gestione di database.

Il modello E-R - Entità

Le **entità** sono gli oggetti principali su cui vengono raccolte le informazioni:

ogni entità del modello E-R serve a rappresentare graficamente un concetto, concreto o astratto, del mondo reale.

Una entità può essere una persona, una macchina, un posto o un evento che raccoglie dati.

Il modello E-R - Entità

È possibile distinguere diversi tipi di entità:

- **Entità forti:** una entità si dice **forte** se non ha bisogno di altre entità per essere identificata.
- **Entità deboli:** una entità si dice **debole** se ha bisogno di altre entità per essere identificata.
- **Entità associative:** dette anche **entità di intersezione**, sono usate per associare due o più entità allo scopo di risolvere un'associazione molti a molti.

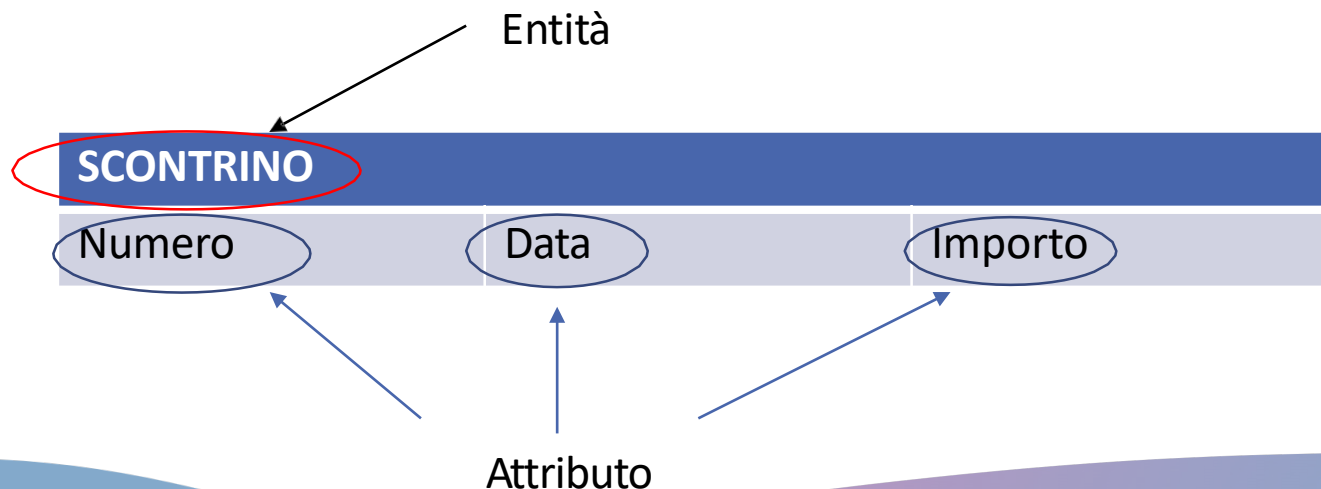
Istanze e attributi

Per le entità vengono inoltre definite le istanze:

una **istanza** di una entità consiste in un singolo oggetto descritto dall'entità.

Istanze e attributi

Gli **attributi** descrivono le entità con le quali sono associati. Una particolare istanza di un attributo è detta **valore**.



Istanze e attributi

Il **dominio** di un attributo è la collezione di tutti i valori possibili di un attributo.

Es: il dominio di Nome è una stringa di caratteri, il dominio di Numeri è l'insieme dei numeri naturali, ecc.

Istanze e attributi

Gli attributi possono essere classificati come **identificatori** o **descrittori**:

- Gli **identificatori**, chiamati più comunemente **chiavi**, identificano univocamente una istanza di una entità.
- Un **descrittore** descrive invece una caratteristica non unica di una istanza di una entità.

Relazioni

Una **relazione** rappresenta un'associazione tra una o più entità.

Le relazioni sono classificate in termini di **grado**, **cardinalità**, **direzione** ed **esistenza**.

Relazioni

Il **grado** di una relazione è il numero di entità associate alla relazione.

La relazione n -aria è la forma generale di grado n .

I casi speciali sono quelli binari e ternari, dove i gradi sono rispettivamente 2 e 3.

Relazioni

Le **relazioni binarie**, ovvero le associazioni tra due entità, sono le relazioni più comuni nel mondo reale.

Quando una relazione associa un'entità con se stessa si parla di **relazione binaria ricorsiva**.

Relazioni

Una relazione **ternaria** implica tre entità e viene usata quando quella binaria è inadeguata.

Molti approcci di modellazione riconoscono solo relazioni binarie: in questi casi, le relazioni ternarie o *n*-arie sono scomposte in due o più relazioni binarie.

Relazioni

La **cardinalità** di una relazione descrive la mappa delle istanze delle entità associate nelle relazioni. Il valore della cardinalità è "**uno**" o "**molti**".

I tipi principali di cardinalità per le relazioni sono:
uno a uno, uno a molti, molti a molti.

Relazioni

Le relazioni **molti a molti** non possono essere tradotte direttamente in tabelle relazionali, ma devono essere trasformate in due o più relazioni **uno a molti** usando delle entità associative.

Relazioni

La **direzione** di una relazione indica l'entità da cui trae origine la relazione binaria.

L'entità da cui si parte si chiama **entità padre**, e l'entità a cui si arriva si chiama **entità figlia**.

Relazioni

L'**esistenza** denota se l'esistenza di una istanza di entità è dipendente dall'esistenza dell'istanza di un'altra entità correlata.

Relazioni

L'esistenza di una entità in una relazione può essere:

- **obbligatoria**, se una istanza di una entità deve sempre essere presente perché una entità sia inclusa in una relazione.
- **opzionale**, se l'istanza di una entità non è richiesta.

Notazione E-R

Non esiste uno standard per rappresentare gli oggetti nei diagrammi E-R, ogni metodologia di modellazione usa le sue notazioni.

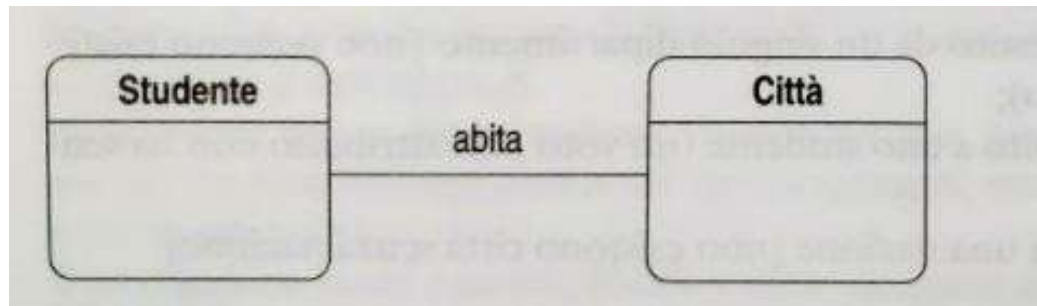
Notazione E-R

- Le entità sono rappresentate da rettangoli con gli angoli smussati e con un'etichetta che ne indica il nome.
- I nomi delle entità devono essere singolari.



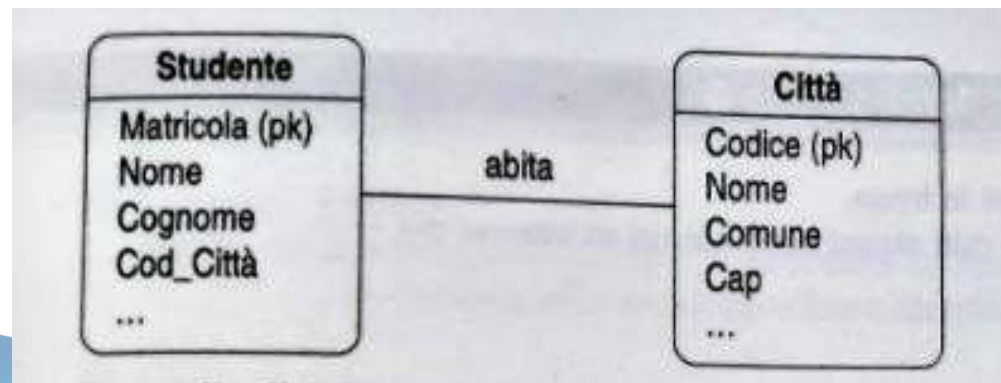
Notazione E-R

- Le relazioni sono rappresentate da una linea retta che collega due entità.
- Il nome della relazione è scritto sopra la linea e in molti casi è costituito da un verbo.



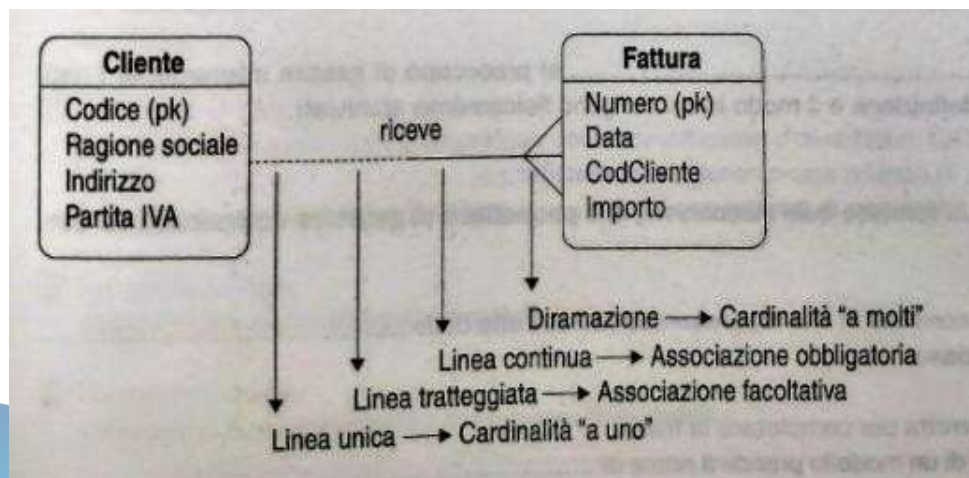
Notazione E-R

- Gli attributi, quando inclusi, sono elencati dentro il rettangolo dell'entità.
- Gli attributi che sono identificatori sono contrassegnati con la sigla "pk".
- I nomi degli attributi devono essere singolari.



Notazione E-R

La cardinalità "a molti" è rappresentata dalla diramazione in tre segmenti al termine della linea della relazione. Se la linea è unica (non c'è diramazione), la cardinalità è uno. L'esistenza obbligatoria è rappresentata da una linea continua; l'esistenza opzionale da una linea tratteggiata.



Chiavi primarie

La **chiave primaria** è un attributo o un insieme di attributi che identificano univocamente una specifica istanza di un'entità.

È buona norma che ogni entità nel modello dei dati abbia una chiave primaria il cui valore identifica univocamente l'istanza dell'entità.

Chiavi primarie

Per identificare una chiave primaria per un'entità, un attributo deve avere le seguenti proprietà:

- Il valore deve essere specificato per ogni istanza dell'entità.
- Il valore deve essere unico per ogni istanza dell'entità.
- Il valore non deve cambiare o diventare nullo durante la vita di ogni istanza dell'entità.

Chiavi primarie – Chiavi composte

Alcune volte si richiede che più di un attributo identifichi univocamente un'entità.

Una chiave primaria che è formata da più di un attributo viene detta **chiave composta**.

Chiavi primarie – Chiavi artificiali

Una **chiave artificiale** è formata da un attributo privo di significato proprio, aggiunto agli altri attributi in modo artificiale.

Solitamente la chiave artificiale è composta da un contatore che si incrementa automaticamente ad ogni istanza aggiunta all'entità.

Chiavi primarie – Chiavi artificiali

Una chiave artificiale è permessa quando una delle seguenti condizioni è verificata:

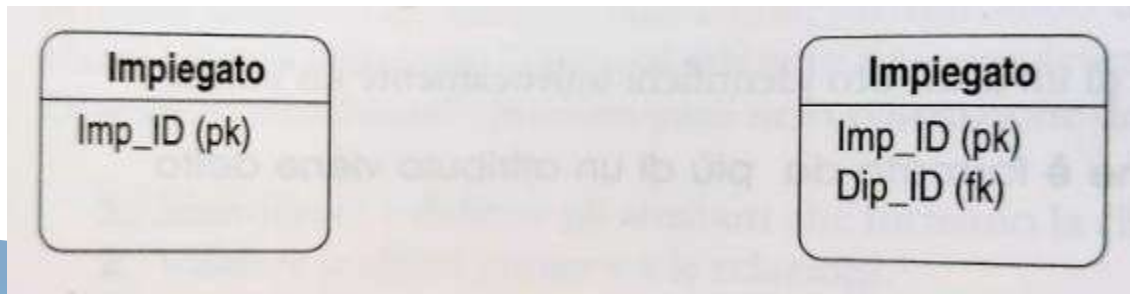
- Nessun attributo ha tutte le proprietà di una chiave primaria.
- La chiave primaria è grande e complessa.

Chiavi primarie – Migrazione di chiave primaria

Le entità dipendenti, ovvero le entità che dipendono dall'esistenza di un'altra entità per la loro identificazione, ereditano l'intera chiave primaria dell'entità padre.

Entità con chiave primaria

Entità con chiave primaria e chiave esterna



Chiavi esterne

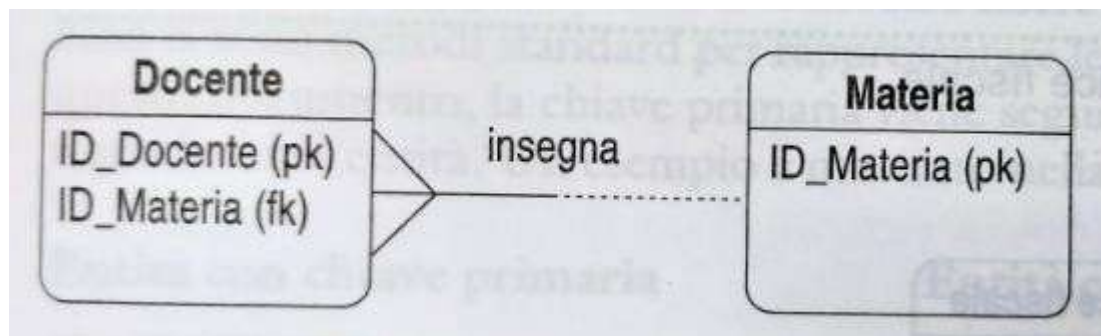
Una **chiave esterna** è un attributo che completa una relazione attraverso l'identificazione dell'entità padre.

Ogni relazione uno a molti presente nel modello deve essere supportata da una chiave esterna.

Gli attributi di chiave esterna sono affiancati dalla notazione "**fk**".

Chiavi esterne

Esempio: una materia può essere insegnata da molti docenti, un docente insegna una sola materia.



La chiave primaria **ID_Materia (pk)** dell'entità padre "migra" nell'entità figlio come chiave esterna.

Aggiungere attributi al modello

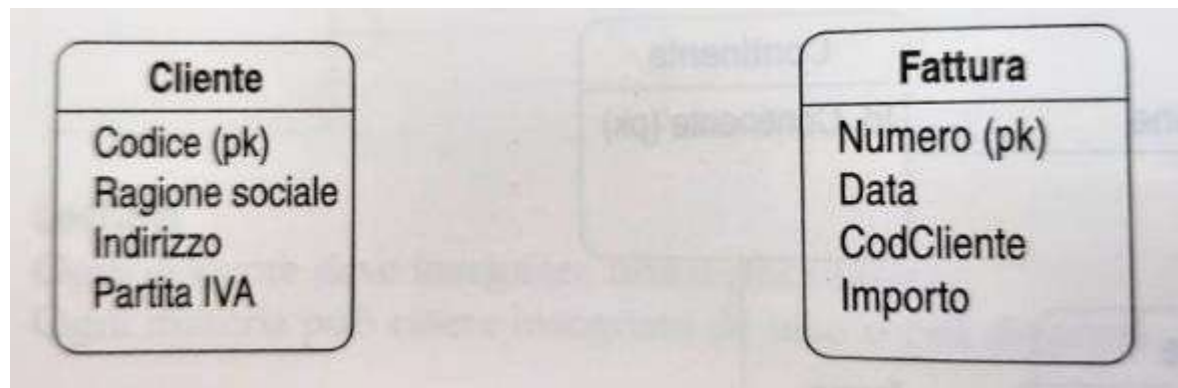
Gli **attributi non chiave** descrivono l'entità a cui appartengono.

Gli attributi non chiave non migrano e possono esistere solo in un'entità.

Aggiungere attributi al modello

Regole per l'assegnazione di attributi non chiave alle entità:

Mettere in relazione gli attributi con le entità

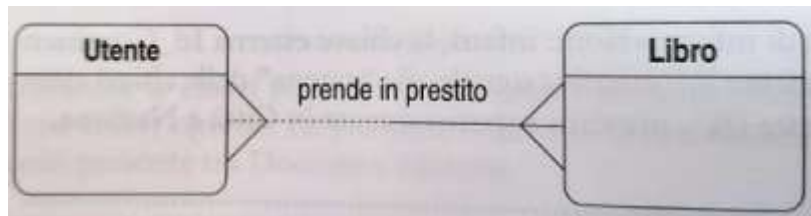


Aggiungere attributi al modello

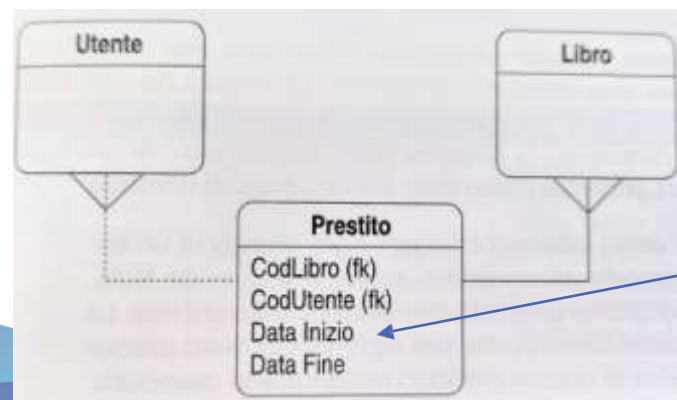
Attributi che descrivono relazioni

È possibile riclassificare una relazione con una nuova entità (entità associativa) che è figlio di entrambe le entità originali.

Es.



Diventa:



La chiave primaria non è indispensabile

Aggiungere attributi al modello

Domini

Un dominio è un insieme di valori valido per un attributo e assicura che i valori di un'inserzione o di una cancellazione d'istanze abbiano senso.

Aggiungere attributi al modello

Un dominio include i seguenti elementi:

- **Tipo di dato:** intero, decimale, carattere, data, ora, valori logici.
- **Lunghezza:** numero di cifre o caratteri per il valore.
- **Intervallo:** specifica i limiti superiori e inferiori dei valori;

Aggiungere attributi al modello

Vincoli: sono speciali restrizioni sui valori permessi.

Supporto dei valori NULL: il valore NULL è quel valore che indica che al campo non è assegnato alcun valore; NULL è diverso da zero per i campi numerici e dalla stringa vuota per i campi alfanumerici.

Valore di default: il valore di una istanza prende questo valore se non viene specificato altrimenti.

Aggiungere attributi al modello

Dominio per le chiavi primarie

Il valore delle chiavi primarie deve essere unico e il valore NULL non è ammesso

Dominio per le chiavi esterne

Il tipo di dato, la lunghezza e il formato della chiave esterna devono essere uguali a quelli della corrispondente chiave primaria. Una relazione uno a molti implica una chiave esterna non unica.

Regole di integrità dei dati

La regola dell'**integrità dell'entità** afferma che, per ogni istanza di un'entità, il valore della chiave primaria deve esistere, essere unico e non NULL. Senza l'integrità nell'entità, la chiave primaria non potrebbe essere tale.

La regola dell'**integrità referenziale** afferma che per ogni valore della chiave esterna deve esistere un valore di chiave primaria nella tabella associata.

Regole di inserzione

Inserimento dipendente: consente l'inserzione di un'istanza nell'entità figlio solo se la chiave padre esiste già.

Inserzione automatica: permette l'inserzione di un'istanza figlio. Se l'istanza dell'entità padre non esiste, viene creata.

Inserzione nulla: consente l'inserzione di un'entità figlio. Se l'istanza padre non esiste, la chiave esterna nel figlio viene impostata a NULL.

Regole di inserzione

Inserzione di default: permette l'inserzione di un'istanza di entità figlio. Se l'istanza di entità padre non esiste, la chiave esterna del figlio viene impostata a un valore predefinito.

Nessun effetto: questa regola dice che l'inserzione di un'istanza dell'entità figlio è sempre consentita. Non è richiesta in alcun modo l'esistenza dell'istanza padre, e quindi non viene eseguito alcun controllo di consistenza.

Regole di cancellazione

Cancellazione con restrizione: consente la cancellazione dell'istanza dell'entità padre solo se non ci sono istanze dell'entità figlio.

Cancellazione a cascata: permette sempre la cancellazione dell'istanza dell'entità padre e cancella tutte le istanze dell'entità figlio corrispondenti.

Regole di cancellazione

Cancellazione nulla: consente sempre la cancellazione dell'istanza dell'entità padre. Se esiste un'istanza dell'entità figlio, i valori della chiave esterna per questa istanza sono impostati a NULL.

Cancellazione di default: permette sempre la cancellazione dell'istanza dell'entità padre. Se esiste un'istanza dell'entità figlio, i valori della chiave esterna per questa istanza sono impostati a valori predefiniti.

Regole di cancellazione

Nessun effetto: questa regola consente sempre la cancellazione. Non viene effettuato alcun controllo di consistenza.

MauiTodo

Si procederà con la creazione di un'app di gestione delle attività chiamata MauiTodo. Un database sarà utilizzato per memorizzare i compiti, garantendo così la loro persistenza. Inoltre, il database verrà crittografato in modo sicuro per garantire la sicurezza delle informazioni.

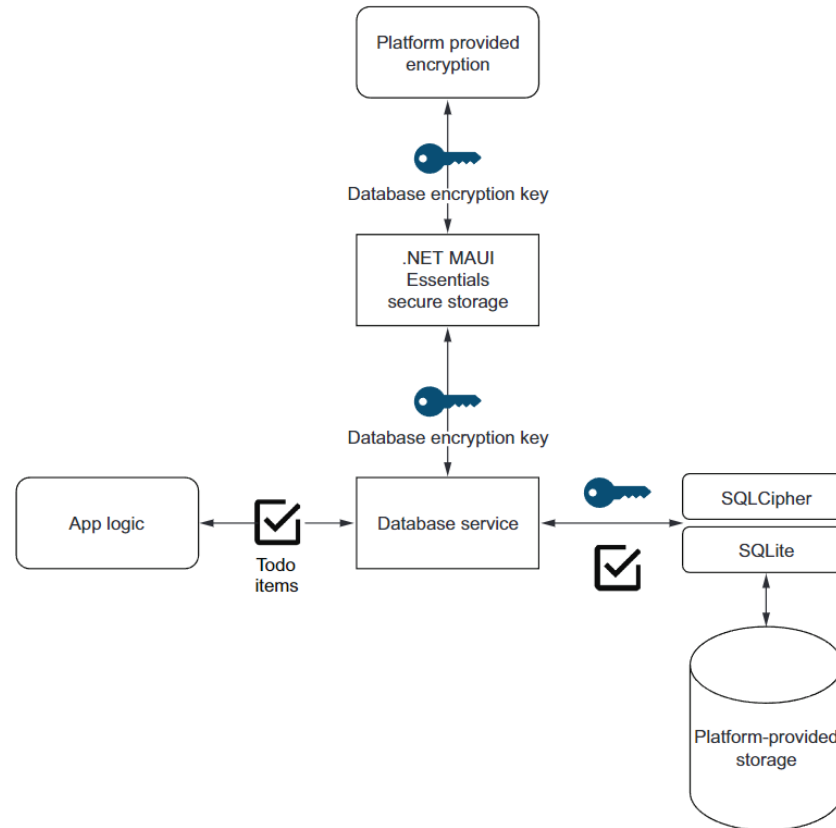
MauiTodo

Si utilizzerà **SQLite**, un motore di database open source molto popolare. Per questo progetto, si utilizzerà un pacchetto chiamato **SQLite-net**, un'implementazione di **SQLite** appositamente progettata per le app mobile.

MauiTodo

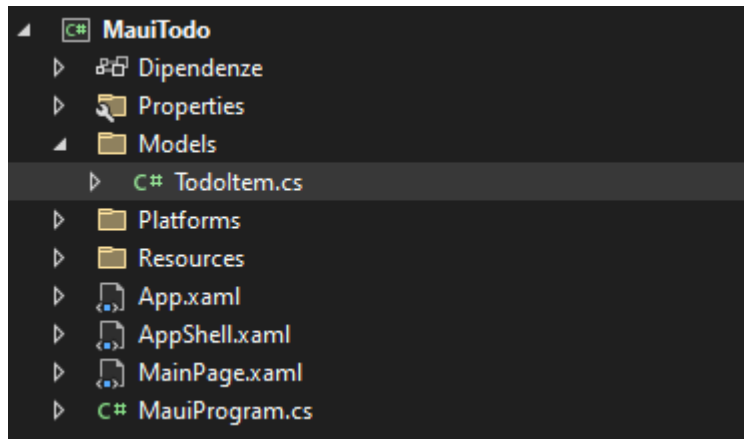
Si aggiungerà inoltre anche **SQLCipher** per crittografare e decrittografare i dati ed l'API di SecureStorage per consentire al sistema operativo di gestire la chiave di crittografia del database.

MauiTodo



MauiTodo

Si crei una nuova app MAUI, MauiTodo. Successivamente, si definisca il modello dei dati. Bisognerà creare una classe per rappresentare le task, quindi si aggiunga la cartella **Models** e crea al suo interno un nuovo file di classe chiamato `Todoltem.cs`



MauiTodo

Si inseriscano le seguenti proprietà;

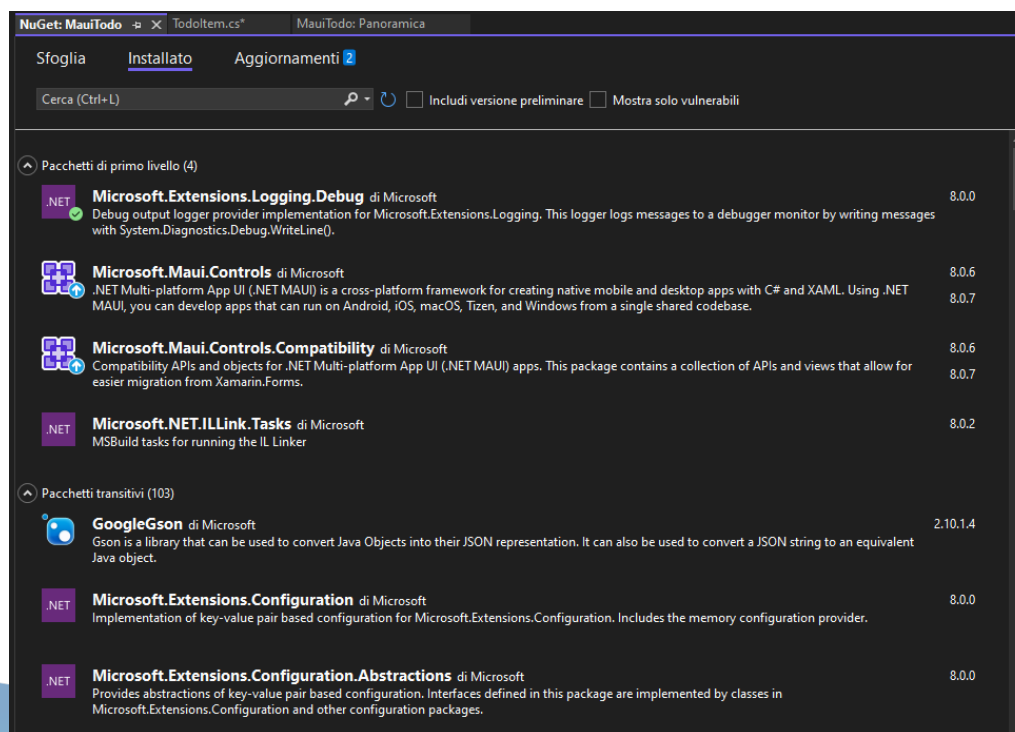
```
namespace MauiTodo.Models
{
    0 riferimenti
    internal class TodoItem
    {
        0 riferimenti
        public int Id { get; set; }
        0 riferimenti
        public string Title { get; set; }
        0 riferimenti
        public DateTime Due { get; set; }
        0 riferimenti
        public bool Done { get; set; } = false;
    }
}
```


MauiTodo

Si aggiungano le funzionalità del database per memorizzare e recuperare le task. Si aggiungano i pacchetti NuGet `sqlite-net-pcl` e `sqlite-net-sqlcipher`

MauiTodo

Per poter aggiungere i pacchetti, si clicchi con il tasto destro del mouse su **MauiTodo** e si selezioni **Aggiungi pacchetti NuGet**



MauiTodo



sqlite-net-sqlcipher di SQLite-net, 1,15 M download

1.8.116

SQLite-net is an open source and light weight library providing easy SQLite database storage for .NET, Mono, and Xamarin applications.
This version uses SQLitePCLRaw to provide platform independent versions of SQLite with the SQLCipher extension.

MauiTodo*

NuGet: MauiTodo

×

TodolItem.cs*

MauiTodo: Panoramica

Sfoglia

Installato

Aggiornamenti 2

sqlite-net-sqlcipher

×

↺

☐ Includi versione preliminare

Gestione pacchetti NuGet: MauiTodo

Origine pacchetto: nuget.org

⚙

sqlite-net-sqlcipher di SQLite-net, 1,15 M download

1.8.116

SQLite-net is an open source and light weight library providing easy SQLite database storage for .NET, Mono, and Xamarin applications.
This version uses SQLitePCLRaw to provide platform independent versions of SQLite with the SQLCipher extension.

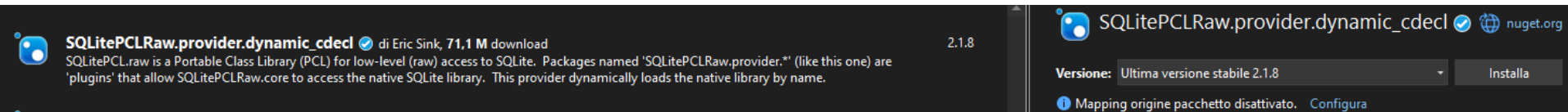
sqlite-net-sqlcipher

Versione: Ultima versione stabile 1.8.116

Installa

MauiTodo

Bisognerà inoltre installare il pacchetto **SQLitePCLRaw.provider.dynamic_cdecl** per garantire tutte le funzionalità su Android. Si installi questo pacchetto solo su Android tramite la condizione MSBuild nel file **.csproj**.



The screenshot shows the NuGet package page for **SQLitePCLRaw.provider.dynamic_cdecl** by Eric Sink. The package has 71.1 M downloads and version 2.1.8 is the latest stable version. The description states it is a Portable Class Library (PCL) for low-level (raw) access to SQLite. The interface includes a search bar, a version dropdown menu set to 'Ultima versione stabile 2.1.8', and an 'Installa' button. A message at the bottom indicates 'Mapping origine pacchetto disattivato' with a 'Configura' link.

SQLitePCLRaw.provider.dynamic_cdecl di Eric Sink, 71,1 M download 2.1.8

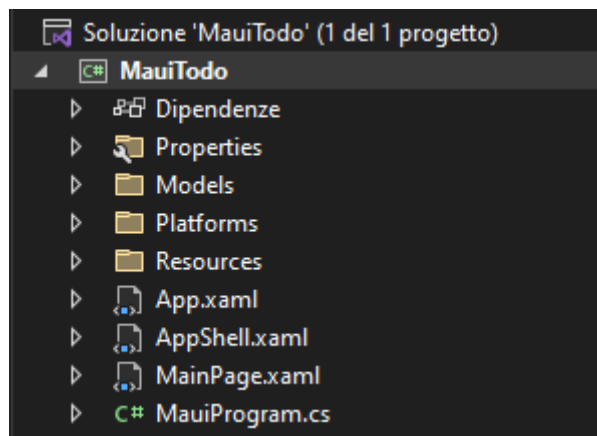
SQLitePCLRaw is a Portable Class Library (PCL) for low-level (raw) access to SQLite. Packages named 'SQLitePCLRaw.provider.*' (like this one) are 'plugins' that allow SQLitePCLRaw.core to access the native SQLite library. This provider dynamically loads the native library by name.

Versione: Ultima versione stabile 2.1.8 [nuget.org](#) [Installa](#)

[Mapping origine pacchetto disattivato. Configura](#)

MauiTodo

Si clicchi due volte su «MauiTodo» per poter aprire il file .csproj



MauiTodo

Si inseriscano le seguenti righe poco prima della fine del nodo **Project**

```
<ItemGroup  
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==  
'android'">  
<PackageReference Include="SQLitePCLRaw.provider.dynamic_cdecl" Version="2.1.4" />  
</ItemGroup>
```

MauiTodo

```
<ItemGroup Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) == 'android'">
  <PackageReference Include="SQLitePCLRaw.provider.dynamic_cdecl" Version="2.1.4" />
</ItemGroup>

</Project>
```

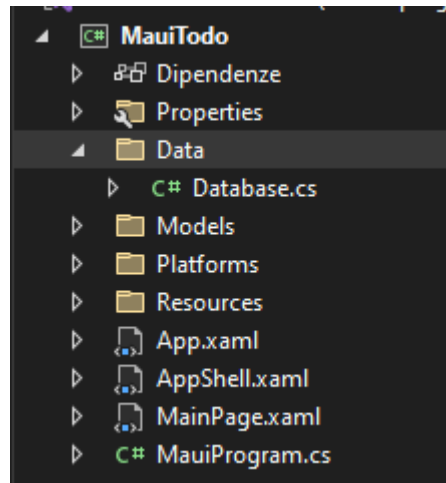
MauiTodo

Si indichi nella classe `TodoItem` tramite SQLite che la proprietà `ID` è la chiave primaria e deve essere incrementata automaticamente dal database;

```
[PrimaryKey, AutoIncrement]  
0 riferimenti  
public int Id { get; set; }
```


MauiTodo

Successivamente, si crei una cartella **Data**. Si inserisca al suo interno un nuovo file di classe, **Database.cs**, dove bisognerà aggiungere i metodi per effettuare le varie operazioni alle task da un database SQLite.



MauiTodo

Si inserisca la stringa per poter effettuare il collegamento al database.

```
string dbOpons = new SQLiteConneconString(databasePath, true,  
key: _dbEncryponKey);
```

MauiTodo

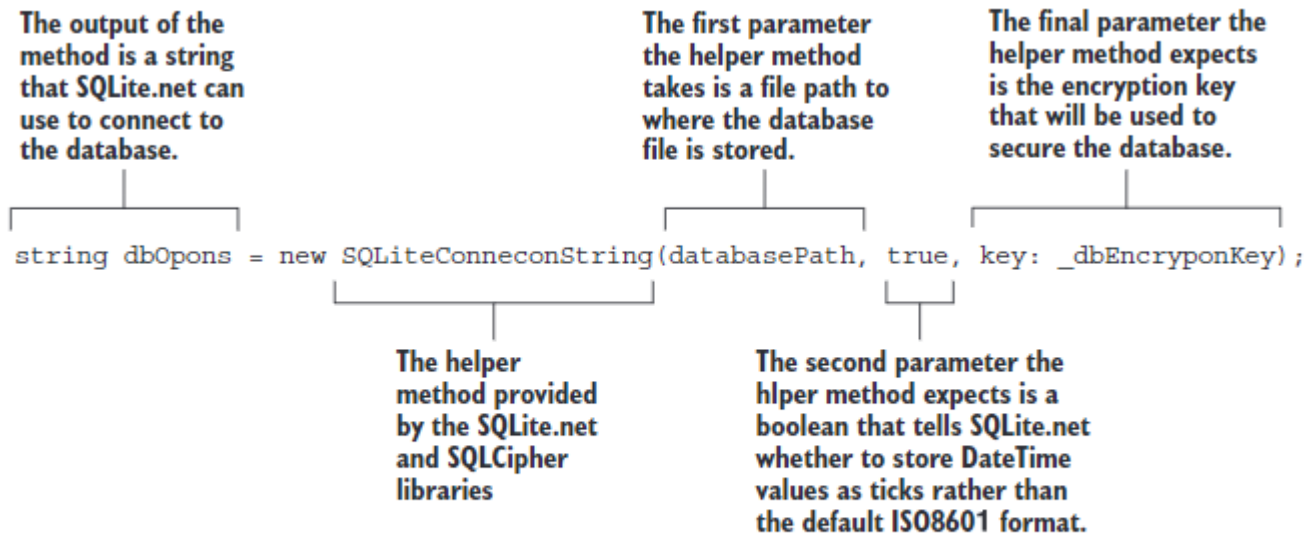


Figure 3.11 SQLite-net provides helper methods for constructing connection strings. SQLCipher provides additional extensions that enable the construction of connection strings that include encryption keys.

MauiTodo

SQLite include un ORM (Object-Relational Mapping) leggero che permette l'interazione con il database utilizzando oggetti POCO (Plain Old CLR Objects) invece di dover scrivere istruzioni SQL ogni volta che si desidera scrivere o recuperare dati dal database.

MauiTodo

Questi oggetti POCO possono anche essere utilizzati come definizioni delle tabelle. Nel contesto dell'app to-do, è stato definito un modello utilizzando un tipo C# che rappresenta gli elementi to-do, e si lascia che SQLite generi automaticamente le tabelle basate su questa definizione.

MauiTodo

È importante assicurarsi che la tabella sia stata creata nel database prima di iniziare a leggere o scrivere dati su di essa. Anche se la presenza della tabella è prerequisito per il suo utilizzo, non è necessario bloccare l'esecuzione dell'app mentre la tabella viene configurata.

MauiTodo

Il metodo fornito da SQLite-net per la creazione della tabella è asincrono, e per ovviare al fatto che non si possono avere costruttori asincroni, si avvolge la chiamata che crea la tabella in un metodo asincrono dedicato.

MauiTodo

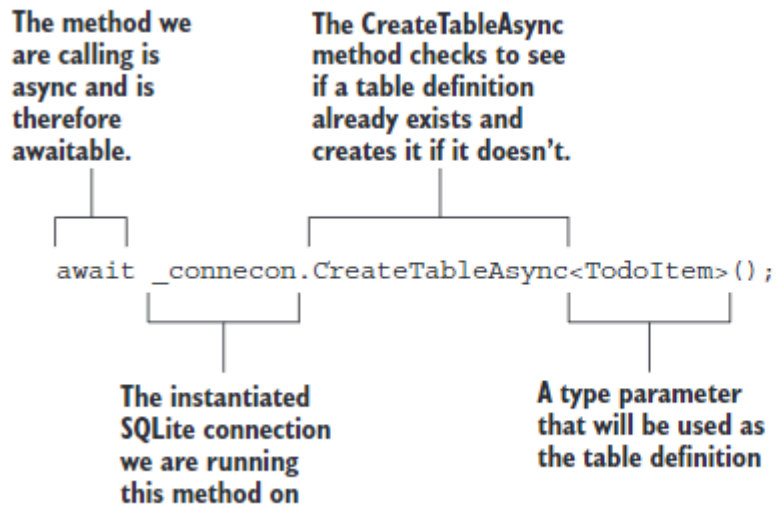


Figure 3.12 SQLite features an ORM that lets us define tables using POCOs. An asynchronous method is provided to verify that a table exists and create it if it doesn't.

MauiTodo

E' quindi possibile chiamare questo metodo dal costruttore usando (`_`), il quale avvierà il metodo in un thread senza bloccare la UI.

MauiTodo

Una volta creata la tabella, è possibile continuare a utilizzare gli oggetti POJO e l'ORM per interagire con essa.

MauiTodo

L'aggiunta di elementi al database è semplice come eseguire il comando `_connection.InsertAsync(item);` l'ORM è in grado di inferire automaticamente la tabella in cui inserire l'oggetto in base al suo tipo. Anche interrogare una tabella per ottenere le voci è un processo semplice, utilizzando una query integrata nel linguaggio (LINQ) e un'espressione lambda.

MauiTodo

Listing 3.9 Database class

```
using MauiTodo.Models;  
using SQLite;  
  
namespace MauiTodo.Data  
{  
    public class Database
```

MauiTodo

Queries SecureStorage for a value with a key of "dbKey". In this case, "key" refers to the identifier of this securely stored item (as in key-value pair), and not the cryptographic key used to secure it. As the SecureStorage API is async only, we need to call .Result because we can't wait for the call in the class constructor.

```
{
    private readonly SQLiteAsyncConnection _connection;

    public Database()
    {
        var dataDir = FileSystem.AppDataDirectory;
        var databasePath = Path.Combine(dataDir, "MauiTodo.db");

        string _dbEncryptionKey = SecureStorage.
            GetAsync("dbKey").Result;
```

The `FileSystem.AppDataDirectory` helper returns the application data path for the platform the app is running on.

```
        if (string.IsNullOrEmpty(_dbEncryptionKey))
        {
            Guid g = new Guid();
            _dbEncryptionKey = g.ToString();
            SecureStorage.SetAsync("dbKey", _dbEncryptionKey);
        }
```

Creates a new GUID if the database encryption key is empty

As no encryption key was returned from SecureStorage, sets the value to the new GUID

MauiTodo

```
var dbOptions = new SQLiteConnectionString(
    databasePath, true, key: _dbEncryptionKey);

_connection = new SQLiteAsyncConnection(dbOptions);

_ = Initialise();

private async Task Initialise()
{
    await _connection.CreateTableAsync<TodoItem>();
}

public async Task<List<TodoItem>> GetTodos()
{
    return await _connection.Table<TodoItem>().ToListAsync();
}
```

Sets a read-only field
to hold the async
database connection

Using the ORM, we can use a type
parameter to request a table with a
design matching the specified type. We
await the result and cast it to a List.

SQLite-net includes a lightweight ORM, so we
can initialize a table based on a model. The
CreateTableAsync method takes a type
parameter that serves as the model for the
table and then creates a table based on that
model if one does not exist already.

Uses a helper method from SQLite-net to create a
connection string for the database datafile. This
helper method takes a path, a Boolean that
determines whether DateTime values are stored as
ticks, and an encryption key used to encrypt and
decrypt the database.

“Throw away” call to an async
method. We need to return a
constructed class with an initialized
connection, but we don’t need to
wait for the database engine to
initialize the tables.

MauiTodo

```
public async Task<TodoItem> GetTodo(int id)
{
    var query = _connection.Table<TodoItem>().
    ➔ Where(t => t.Id == id);
    return await query.FirstOrDefaultAsync();
}

public async Task<int> AddTodo(TodoItem item)
{
    return await _connection.InsertAsync(item);
}

public async Task<int> DeleteTodo(TodoItem item)
{
    return await _connection.DeleteAsync(item);
}

public async Task<int> UpdateTodo(TodoItem item)
{
    return await _connection.UpdateAsync(item);
}
}
```

Awaits the query's result, casts it to the specified type, and returns this awaitable call

Uses a LINQ lambda expression to create a query definition, using a type to represent a table design

The ORM has an InsertAsync method that is smart enough to determine which table to insert the provided parameter into based on the type of the provided parameter

MauiToDo

Si procederà ora a creare un'interfaccia utente per visualizzare e inserire elementi to-do e scrivere il codice per collegare tale UI al database.

MauiToDo

Nel file MainPage.xaml, sarà aggiornato il titolo della pagina e aggiunto un campo di testo in cui l'utente può specificare il titolo per i nuovi elementi to-do, insieme a un selettore di date per la scadenza.

MauiToDo

Sarà necessario un pulsante per confermare l'aggiunta dei nuovi elementi to-do e, infine, si desidera un modo per visualizzare un elenco degli elementi to-do memorizzati nel database.

MauiToDo

Questi elementi verranno aggiunti direttamente, piuttosto che modificare i controlli esistenti. Nel codice XAML, verrà eliminato lo ScrollView e tutto ciò che contiene, lasciando solo i tag `<ContentPage...>...</ContentPage>`. Lo ScrollView rimosso, che è un layout, sarà sostituito con un altro tipo di layout chiamato Grid.

MauiToDo

Per il momento, si aggiungeranno i tag di apertura e chiusura del Grid con le relative proprietà ed elementi figli.

MauiToDo

Listing 3.10 UI for the MauiToDo app

```
<Grid RowDefinitions="1*, 1*, 1*, 1*, 8*"
      MaximumWidthRequest="400"
      Padding="20">
  <Label Grid.Row="0"
        Text="Maui Todo"
        SemanticProperties.HeadingLevel="Level1"
        SemanticProperties.Description="Maui Todo"
        HorizontalTextAlignment="Center"
        FontSize="Title"/>
  <Entry Grid.Row="1"
        HorizontalOptions="Center"
        Placeholder="Enter a title"
        SemanticProperties.Hint="Title of the new todo item"
        WidthRequest="300"
        x:Name="TodoTitleEntry" />
  <DatePicker Grid.Row="2"
              WidthRequest="300"
              HorizontalOptions="Center"
              SemanticProperties.Hint="Date the todo item is due"
              x:Name="DueDatepicker" />
```

← Adds a grid layout with five rows

← Adds a label with the page title

← Adds an entry where the user can enter the title of a new to-do item

← Adds a date picker. A date picker shows the native platform's standard control for choosing a date and returns the selected value as a DateTime type.

MauiToDo

```
<Button Grid.Row="3"
        Text="Add"
        SemanticProperties.Hint="Adds the todo item to the database"
        WidthRequest="100"
        HeightRequest="50"
        HorizontalOptions="Center"
        Clicked="Button_Clicked"/>

<ScrollView Grid.Row="4">
    <Label HorizontalTextAlignment="Center"
           SemanticProperties.Description="The list of todo items
    in the database"
           x:Name="TodosLabel" />
</ScrollView>

</Grid>
```

Adds a button that the user can click to add a new to-do item

Adds a ScrollView inside the grid on the fifth row (row 4). We want a ScrollView because we will fill it with the to-do items that the user adds, which may extend beyond the page.

Adds a label that will show the list of to-do items the user has added

MauiToDo

È stata definita un'interfaccia utente semplice per inserire ed elencare gli elementi to-do dell'utente. Tutti i controlli in questa interfaccia sono familiari eccetto il nuovo DatePicker, che consente all'utente di scegliere una data tramite un selettore grafico anziché inserirla in un campo di testo.

MauiToDo

Per aggiungere funzionalità, verranno apportate modifiche al code-behind: si aggiungeranno proprietà e campi per mantenere un'istanza del database, i valori degli elementi to-do presenti nel database e i valori che l'utente desidera aggiungere a un nuovo elemento to-do.

MauiToDo

Si aggiornerà il costruttore della classe e si aggiungerà un metodo per inizializzare la pagina al caricamento. Infine, si implementerà un metodo per rispondere ai clic sui pulsanti e aggiungere nuovi elementi to-do al database. Il file code-behind (MainPage.xaml.cs) sarà aggiornato di conseguenza.

MauiToDo

Listing 3.11 Code for the MauiToDo main page

```
using MauiToDo.Data;
using MauiToDo.Models;

namespace MauiToDo;

public partial class MainPage : ContentPage
{
    string _todoListData = string.Empty;

    readonly Database _database;

    public MainPage()
    {
        InitializeComponent();
        _database = new Database();
        _ = Initialize();
    }

    private async Task Initialize()
    {
        var todos = await _database.GetTodos();
    }
}
```

Contains the values of the to-do items that we want to display onscreen

Stores an instance of our database class

In the constructor, we create an instance of the database class and assign it to the `_database` field.

Gets the list of to-do items in the database

Uses the discard variable to call our Initialize method

MauiToDo

```
        foreach (var todo in todos)
        {
            _todoListData += $"{todo.Title} - {todo.
➡ Due:f}{Environment.NewLine}";
        }

        TodosLabel.Text = _todoListData;
    }

```

Assigns the value of the `_todoListData` field to the text property of the `TodosLabel` in the UI

Adds the title and date of each to-do item (using the `:f` syntax to format it to a culture-formatted string) and a new line to the `_todoListData` field

MauiToDo

```
private async void Button_Clicked(object sender, EventArgs e)
{
    var todo = new TodoItem
    {
        Due = DueDatepicker.Date,
        Title = TodoTitleEntry.Text
    };

    var inserted = await _database.AddTodo(todo);

    if (inserted != 0)
    {
        _todoListData += $"{todo.Title} - {todo.Due:f}{Environment.NewLine}";

        TodosLabel.Text = _todoListData;

        TodoTitleEntry.Text = String.Empty;
        DueDatepicker.Date = DateTime.Now;
    }
}
```

Adds the newly created to-do item to the database

In the button clicked delegate, creates a new instance of the TodoItem class, assigning the text value of the Entry in the UI to the Title property and the date value of the date picker in the UI to the Due property

Adds the new to-do item to the list

Checks the Id value of the to-do item to make sure it was inserted


Updates the value of the TodosLabel in the UI

Resets the two input controls, ready for a new to-do

MauiToDo

Si avvii ora l'applicazione:

MauiToDo



Maui Todo

Enter a title

18/12/2021

Add

Write - Thursday, 7 April 2022 12:00 AM
Finish .NET MAUI in Action - Saturday, 1 January 2022 12:00 AM

Figure 3.13 MauiToDo running on Windows. When you click the Add button, an event handler gets the values from the Entry and DatePicker and uses them to create a new to-do item with those values for the title and due date, respectively. It then adds them to the list of to-do items on the screen.

Esercitazione 1

>>> runcodenovelli@gmail.com

Data Binding: Connettere UI al codice

Finora, il code-behind è stato utilizzato per ottenere valori da e assegnare valori alle proprietà degli elementi dell'interfaccia utente.

Data Binding: Connettere UI al codice

E' possibile utilizzare `x:...` per assegnare nomi agli elementi dell'interfaccia utente (ad esempio, `x:Name="TodoTitleEntry"`) e fare riferimento a questi elementi nell'interfaccia utente tramite il nome assegnato nel codice.

Data Binding: Connettere UI al codice

Ciò consente di utilizzare `TodoTitleEntry.Text` per leggere il valore della proprietà di testo dell'elemento dell'interfaccia utente chiamato `TodoTitleEntry`.

Data Binding: Connettere UI al codice

Sebbene XAML sia un linguaggio di markup, gli elementi dell'interfaccia utente utilizzati in XAML sono comunque classi. Aggiungendo un `<Entry.../>` alla vista, a runtime, la vista (anch'essa una classe) istanzia un oggetto di tipo `Entry`.

Data Binding: Connettere UI al codice

Esaminando le proprietà della classe `Entry`, si può notare che essa dispone di una proprietà chiamata `Text` di tipo `string`, da cui si ottiene il valore inserito dall'utente. La capacità di accedere alle proprietà degli elementi XAML nel codice è potente e offre un modo ordinato per ottenere e impostare valori dall'interfaccia utente.

Data Binding: Connettere UI al codice

Tuttavia, un metodo migliore è l'uso del data binding, che consente di collegare una proprietà a un'altra in modo che, quando una cambia, anche l'altra cambia.

Data Binding: Connettere UI al codice

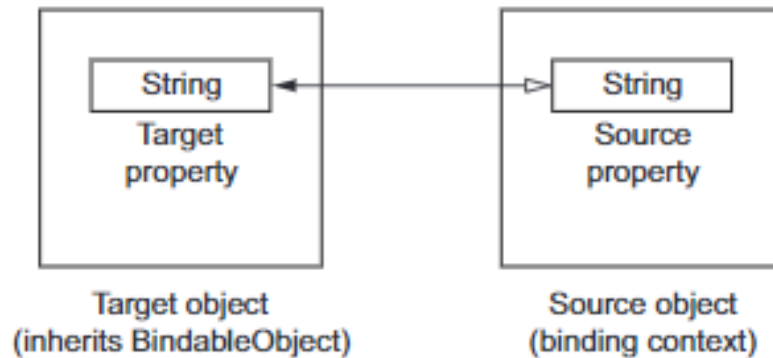


Figure 3.14 A property on a class that inherits from `BindableObject` can be bound to a property of the same type.

Data Binding: Connettere UI al codice

Il data binding collega una proprietà sorgente a una proprietà destinazione. La proprietà destinazione deve appartenere a un oggetto che eredita dalla classe base `BindableObject` (tutti i controlli dell'interfaccia utente predefiniti in .NET MAUI ereditano `BindableObject`).

Data Binding: Connettere UI al codice

La proprietà destinazione viene collegata alla sorgente, quindi è nella destinazione che viene impostato il binding. La proprietà sorgente può appartenere a un oggetto di qualsiasi tipo, ma deve corrispondere al tipo della proprietà destinazione.

Data Binding: Connettere UI al codice

Se le proprietà sorgente e destinazione non sono dello stesso tipo, può essere utilizzato un convertitore di valori.

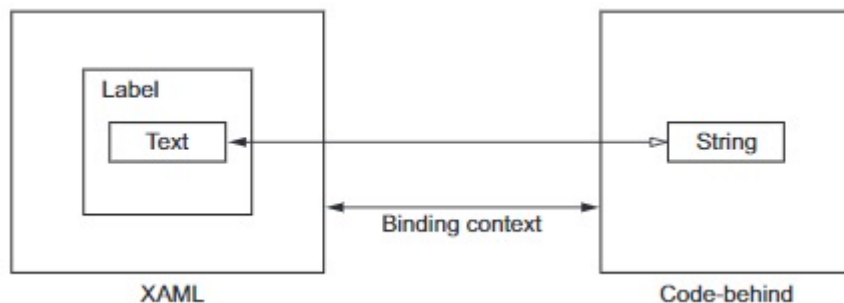


Figure 3.15 Each view (target object) can have a single binding context (source object). Properties of the target view are bound to properties of the same type on the source object.

Data Binding: Connettere UI al codice

Ogni vista può avere un unico contesto di binding, che rappresenta l'oggetto sorgente. Una volta impostato il binding, le proprietà di quella vista (target) possono essere collegate alle proprietà dell'oggetto sorgente semplicemente facendo riferimento al nome della proprietà.

Data Binding: Connettere UI al codice

Le singole proprietà del controllo possono comunque essere collegate in modo arbitrario a qualsiasi proprietà di qualsiasi altro oggetto, ma ciò richiede un riferimento esplicito.

Data Binding: Connettere UI al codice

Impostare il contesto di binding per un controllo significa che qualsiasi proprietà del controllo può essere collegata a qualsiasi proprietà dell'oggetto sorgente utilizzando solo il nome della proprietà.

Data Binding: Connettere UI al codice

Il contesto di binding si eredita e si propaga dai componenti genitori a quelli figli all'interno di una vista. I contesti di binding possono essere definiti esplicitamente per qualsiasi vista, ma se non si definisce esplicitamente un contesto di binding per un controllo in una `ContentPage`, il contesto di binding sarà lo stesso della `ContentPage`.

Data Binding: Connettere UI al codice

Ad esempio, aggiungendo un layout o una collezione a una `ContentPage`, il layout erediterà il contesto di binding della `ContentPage`, così come tutti i suoi controlli figli.

Data Binding: Connettere UI al codice

Se si imposta esplicitamente il contesto di binding del layout o della collezione, i controlli figli ereditano quest'ultimo. Nel caso specifico, il contesto di binding di una pagina è stato impostato sul code-behind della pagina stessa.

Data Binding: Connettere UI al codice

Il titolo e il pulsante ereditano questo contesto di binding e si collegano alle proprietà nel code-behind della `ContentPage`. Il contesto di binding per il `VerticalStackLayout` è stato impostato su una proprietà oggetto nel code-behind.

Data Binding: Connettere UI al codice

Il sottotitolo e il pulsante all'interno di questo layout ereditano questo contesto di binding e si collegano alle proprietà dell'oggetto.

Data Binding: Connettere UI al codice

Le viste possono anche essere collegate ad altre viste. Ad esempio, sia Label che Entry hanno una proprietà chiamata Text di tipo stringa, quindi queste proprietà possono essere collegate. Collegando queste due proprietà, si può fare in modo che una aggiorni automaticamente l'altra.

Data Binding: Connettere UI al codice

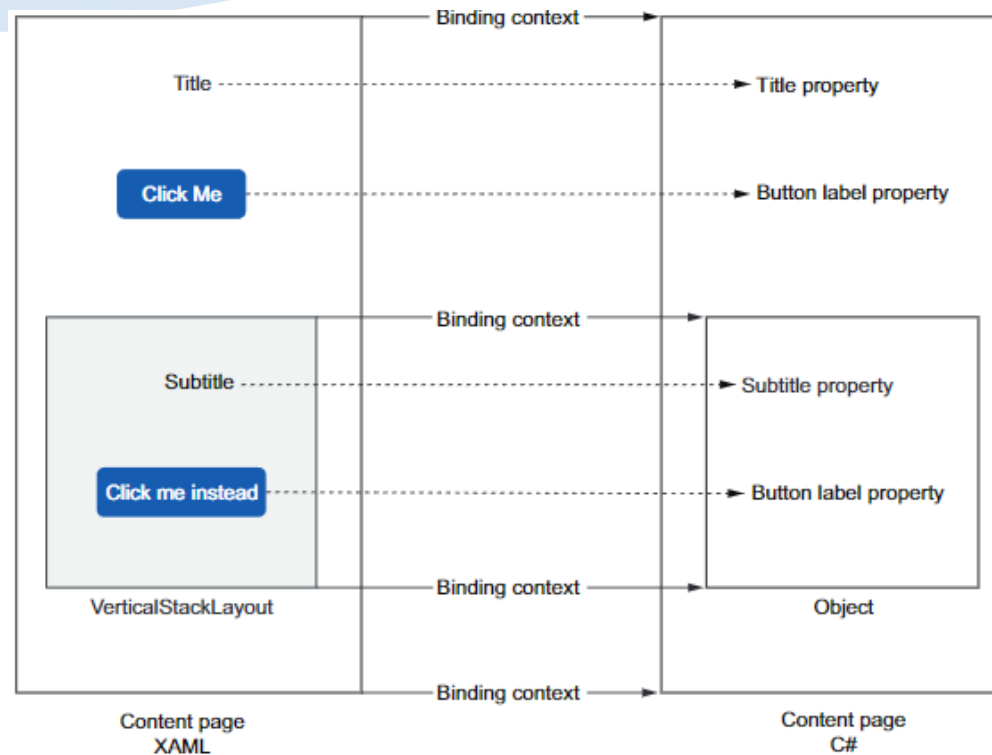


Figure 3.16 The binding context can be set at any level and is inherited by child components from that point down.

Data Binding: Connettere UI al codice

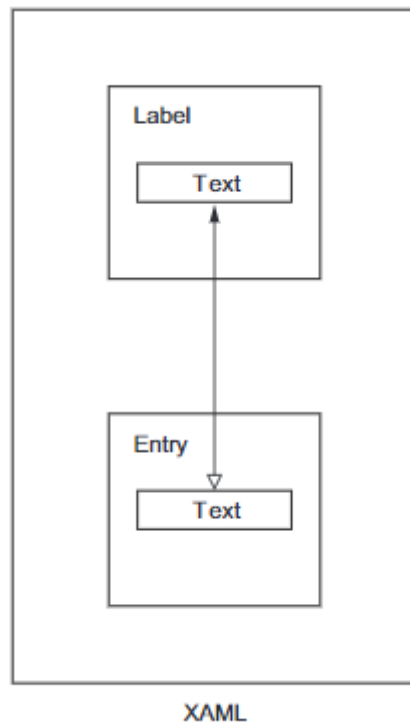


Figure 3.17 Properties of views can be bound to properties of other views.

Binding view-to-view

I binding view-to-view possono essere impostate in XAML e consentono di collegare proprietà in una vista senza alcun intervento nel code-behind.

Binding view-to-view

Creando un nuovo progetto .NET MAUI chiamato Bindings, e rimuovendo lo ScrollView e il suo contenuto da MainPage.xaml, è possibile sostituire lo ScrollView con del codice specifico, posizionandolo all'interno dei tag ContentPage.

Binding view-to-view

Listing 3.12 Markup for view-to-view bindings

```
<VerticalStackLayout VerticalOptions="Center"
    HorizontalOptions="Center"
    Spacing="20"
    WidthRequest="200">
```

Places our controls inside a `VerticalStackLayout`. We learn more about layouts in chapter 4.

```
    <Label FontSize="Title"
        BindingContext="{x:Reference TextEntry}"
        Text="{Binding Text}"
        HorizontalTextAlignment="Center"/>
```

Sets the binding context for this view (in this case, the `Label` control) to another view named `TextEntry`. The `x:Reference` syntax refers to other views on the same page by name.

```
    <Entry x:Name="TextEntry"
        Placeholder="Enter some text..." />
</VerticalStackLayout>
```

Sets the binding source (the property on the binding context that we want to bind this target property to) to the `Text` property of the binding context

We've named this `Entry TextEntry` so we can refer to it by name in other views.

Binding view-to-view

Prima di avviare il progetto, si elimini il campo **count** e il metodo **OnCounterClicked** dal code-behind. Si avii il programma e si inserisca del testo.

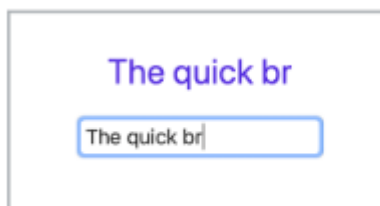


Figure 3.18 The `Text` property of a `Label` bound to the `Text` property of an `Entry`. When the value of `Text` on the `Entry` changes, the `Label` changes automatically.

Binding view-to-view

Si noti come, digitando, il testo dell'etichetta cambi. Negli esempi precedenti, si è fatto affidamento al code-behind per ottenere proprietà dai controlli dell'interfaccia utente e quindi impostare proprietà su altri controlli.

Binding view-to-view

Con il data binding, si collega una proprietà all'altra, e il resto viene gestito automaticamente. Il binding da vista a vista è utile quando si hanno proprietà dello stesso tipo su due viste interconnesse.

Binding view-to-view

Un comportamento simile che si potrebbe aver visto in un'app reale è l'aggiornamento dei numeri su un mock-up di una carta di credito man mano che l'utente li inserisce.

Binding view-to-view

Un altro esempio pratico potrebbe essere la modifica del livello di zoom su un'immagine o una mappa in risposta a un cursore (slider). Si procederà quindi ad aggiungere questa funzionalità all'app Bindings, inserendo i due controlli menzionati sotto l'Entry nel VerticalStackLayout nel file MainPage.xaml dell'app.

Binding view-to-view

Listing 3.13 Controls to add to the Bindings app

We add a Slider control and name it ZoomSlider so we can refer to it elsewhere.

```
<Slider x:Name="ZoomSlider" />
```

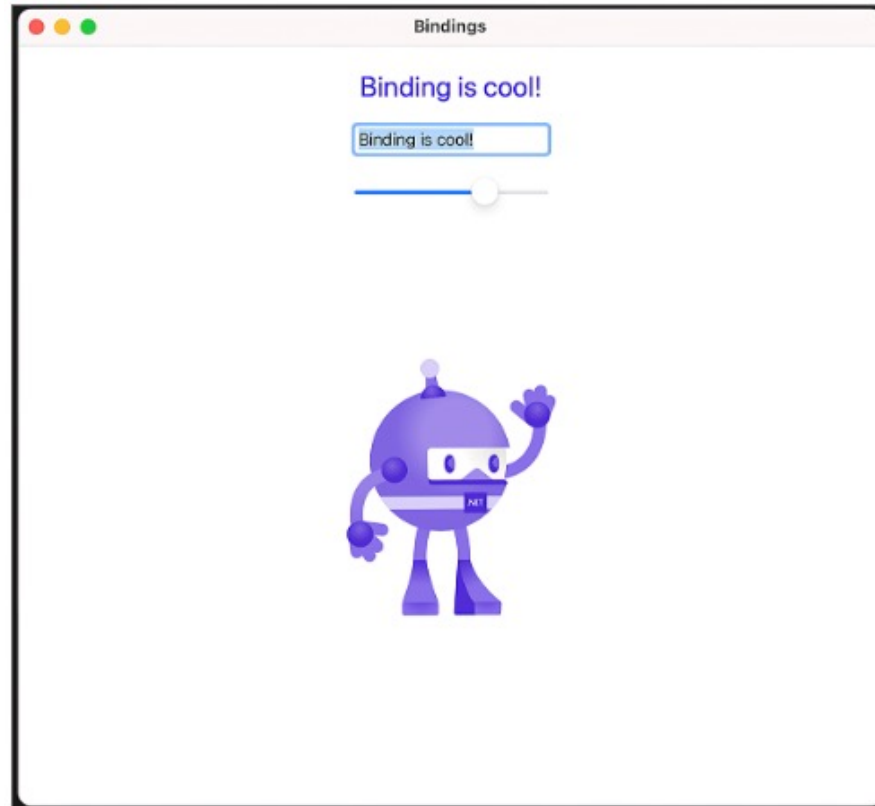
Adds the
Dotnet Bot
image back in

```
<Image Source="dotnet_bot.png"  
WidthRequest="300"  
HorizontalOptions="Center"  
BindingContext="{x:Reference ZoomSlider}"  
Scale="{Binding Value}" />
```

Sets the binding context
for the image control to
the ZoomSlider. This means
any properties on the
Image control that we bind
will get their values from
the ZoomSlider object.

Set the Scale property of the Image. We use a binding rather than an explicit value, and the binding is to a property called Value. The binding context has been set to the ZoomSlider, so the Scale property of this Image is bound to ZoomSlider.Value.

Binding view-to-view



Binding view-to-view

Muovendo lo Slider avanti e indietro, si osserva che la scala dell'immagine varia in tempo reale. Questo è stato possibile grazie all'uso del data binding per collegare la proprietà Scale dell'Immagine alla proprietà Value dello Slider. Quando il valore dello Slider cambia, cambia anche la scala dell'Immagine.

Binding view-to-view

Questo risultato avrebbe potuto essere ottenuto anche senza data binding, ma il processo sarebbe stato più complesso. Sarebbe stato necessario creare un metodo invocato dall'evento ValueChanged dello Slider e, ad ogni cambiamento di valore, impostare la proprietà Scale dell'Immagine.

Collezioni e binding in codice

Un utilizzo avanzato del data binding è l'impostazione di una sorgente di dati per una `CollectionView` tramite la proprietà `ItemsSource`. Una `CollectionView` visualizza una raccolta di elementi, come una `List<T>`, sullo schermo.

Collezioni e binding in codice

Le raccolte o le liste sono al centro di molte applicazioni, e con .NET MAUI è possibile utilizzare il data binding per definire la modalità di visualizzazione di ogni singolo elemento nella raccolta.

Collezioni e binding in codice

La caratteristica più significativa della `CollectionView` è l'uso di un `DataTemplate`, che è la definizione di come gli elementi devono essere visualizzati a schermo.

Collezioni e binding in codice

Sebbene in un esempio attuale si possa utilizzare una Label solo per visualizzare testo, con un DataTemplate si può definire qualsiasi layout immaginabile e collegare aspetti di quel layout a proprietà nel contesto di binding.

Collezioni e binding in codice

Mentre un `DataTemplate` definisce come visualizzare un elemento a schermo, un `ItemTemplate` è il `DataTemplate` specifico utilizzato nella `CollectionView`. L'`ItemTemplate` è una proprietà di una `CollectionView` a cui viene assegnato un `DataTemplate`. I `DataTemplates` possono essere definiti inline ma possono anche essere viste indipendenti e riutilizzabili

Collezioni e binding in codice

A differenza dell'ereditarietà dell'intera raccolta a cui è legata la `CollectionView`, il contesto di binding per un `ItemTemplate` è l'elemento stesso.

Collezioni e binding in codice

Ciò significa che per una raccolta di oggetti con una proprietà stringa chiamata Name, è possibile impostare il binding per la proprietà Text di una Label nell'ItemTemplate su Name, e questo visualizzerà il valore desiderato.

Collezioni e binding in codice

Rifacendosi all'app MauiTodo, è possibile migliorare significativamente il modo in cui vengono visualizzati gli elementi to-do. Attualmente, una Label visualizza il titolo di ogni elemento to-do nel database, separati da un carattere di nuova linea.

Collezioni e binding in codice

Questo metodo è primitivo e limitato; per esempio, non consente di visualizzare la data di scadenza o di formattare diversamente gli elementi in base alle loro proprietà (scaduti, completati, ecc.).

Collezioni e binding in codice

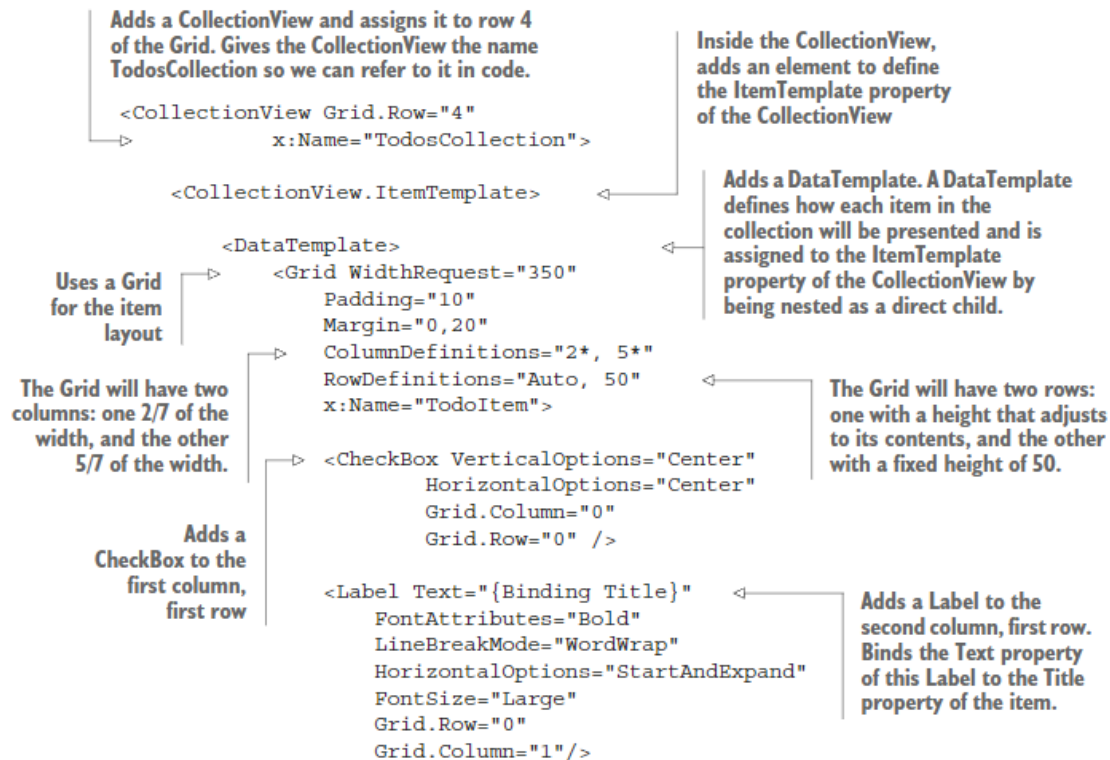
L'app verrà aggiornata per utilizzare una `CollectionView`, consentendo di definire un layout specifico per la presentazione di ogni elemento nella lista to-do, e il binding della `CollectionView` sarà impostato su una `ObservableCollection` di `TodoItems`

Collezioni e binding in codice

Una `ObservableCollection` è una collezione generica simile a una `List`, ma è automaticamente configurata tramite il motore XAML per fornire una notifica quando gli elementi vengono aggiunti o rimossi, aggiornando automaticamente la `CollectionView` nell'interfaccia utente se si aggiungono o rimuovono elementi `to-do`.

Collezioni e binding in codice

Listing 3.14 To-dos CollectionView



Collezioni e binding in codice

```
        <Label Text="{Binding Due, StringFormat=
➡ '{0:dd MMM yyyy}'}"
            Grid.Column="1"
            Grid.Row="1"/>
        </Grid>
    </DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
```

← Adds a Label to the second column, second row. Binds the Text property of this Label to the Due property of the item and, as the Due property is a DateTime, supplies a formatting rule.

Collezioni e binding in codice

È stata aggiunta una `CollectionView` con una definizione di come visualizzare ciascun elemento della raccolta, ma non è stata ancora collegata a una collezione effettiva.

Collezioni e binding in codice

Questa operazione verrà eseguita ora nel codice. Nel file `MainPage.xaml.cs`, all'inizio della classe, prima delle definizioni dei membri privati, è stata aggiunta una `ObservableCollection` di `TodoItems`.

Collezioni e binding in codice

ObservableCollection si trova nello spazio dei nomi System.Collections.ObjectModel, quindi è necessario aggiungere l'istruzione using richiesta all'inizio del file. Successivamente, nel costruttore, impostare la proprietà ItemsSource della CollectionView sull'ObservableCollection:

```
TodosCollection.ItemsSource = Todos;
```


Collezioni e binding in codice

Si elimini ora il codice non più utilizzato, compresa la stringa privata `_todoListData` e ogni riferimento ad essa: l'assegnazione all'interno del ciclo `foreach` nel metodo `Initialise`, l'assegnazione del valore alla `Label` nel metodo `Initialise`, l'aggiunta al ciclo nel metodo `Button_Clicked` e l'assegnazione alla `Label` nel metodo `Button_Clicked`.

Collezioni e binding in codice

Successivamente, è stata reintrodotta la logica per inizializzare la collezione e aggiornarla quando l'utente aggiunge un nuovo elemento to-do.

Collezioni e binding in codice

Nell'inizializzazione, dentro il ciclo foreach del metodo Initialise, ogni elemento to-do viene aggiunto all'ObservableCollection. La stessa logica è stata applicata nel metodo Button_Clicked, nell'istruzione if, dove in precedenza l'elemento veniva aggiunto alla stringa.

Collezioni e binding in codice

Con queste modifiche, è stata aggiunta alla UI una `CollectionView` con un template dati che specifica come visualizzare gli elementi. È stata aggiunta al code-behind una `ObservableCollection`, un tipo speciale di collezione che invia notifiche quando il suo contenuto cambia, e questa è stata collegata come sorgente degli elementi da visualizzare nella `CollectionView`.

Collezioni e binding in codice

È stato inoltre aggiunto del codice per inserire elementi to-do nella `CollectionView` sia all'inizializzazione della pagina sia quando l'utente aggiunge un nuovo elemento to-do. Avviando l'app e aggiungendo alcuni elementi to-do, si dovrebbe ottenere un risultato simile a quanto descritto.

ItemsSource bindings in XAML

Impostare i contesti di binding tramite codice è efficace, ma considerando che il data binding è ciò che conferisce a XAML la logica, è utile esaminare anche come impostare l'ItemsSource per la CollectionView in XAML

ItemsSource bindings in XAML

È possibile definire il contesto di binding per qualsiasi vista direttamente in XAML, quando il contesto di binding di un controllo è stato impostato sulle proprietà di un altro.

ItemsSource bindings in XAML

L'app MauiTodo verrà rifattorizzata per realizzare tutto il binding in XAML. Nel file code-behind MainPage.xaml.cs, nel costruttore, si rimuove la riga aggiunta in precedenza per impostare la sorgente degli elementi: `TodosCollection.ItemsSource = Todos;` invece, questa associazione verrà configurata in XAML. Nel file MainPage.xaml, nel tag di apertura `<ContentPage...>`, si aggiunge un nome per consentirne il riferimento.

ItemsSource bindings in XAML

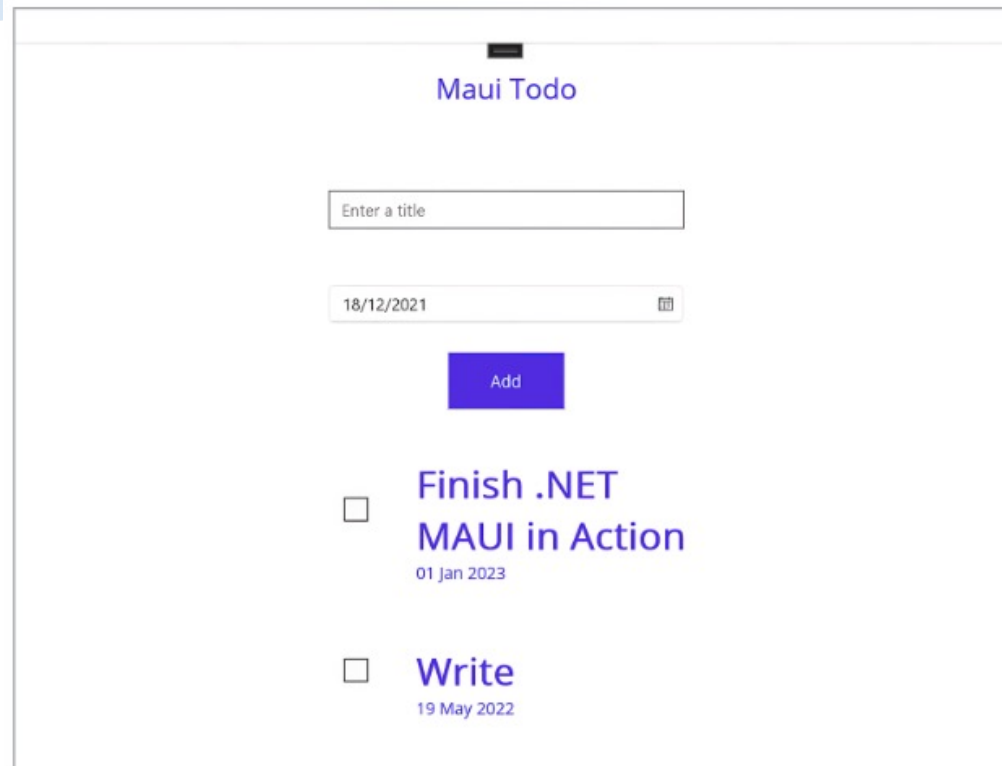


Figure 3.20 MauiTodo with a CollectionView running on Windows

ItemsSource bindings in XAML

Successivamente, nella `CollectionView`, collega la proprietà `ItemsSource` alla `ObservableCollection` `Todos` nel contesto di binding. La seguente immagine mostra il nuovo tag di apertura `ContentPage` e il nuovo tag di apertura `CollectionView`, con le righe aggiunte evidenziate in grassetto.

ItemsSource bindings in XAML

Listing 3.15 MainPage.xaml ContentPage tag changes

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiTodo.MainPage"
             BackgroundColor="{DynamicResource SecondaryColor}"
             x:Name="PageTodo"
             BindingContext="{x:Reference PageTodo}">
    ...
    <CollectionView Grid.Row="4"
                    ItemsSource="{Binding Todos}"
                    x:Name="TodosCollection">
```

Uses the Name attribute from the XAML namespace (aliased to x) to specify a name for the page

...

Binds the ItemsSource property of the CollectionView to the Todos property of the binding context

Sets the binding context for the page. Use the name you created for the page as a reference, setting the binding context for the page to itself.

Esercitazione 2

>>> runcodenovelli@gmail.com



Domande e approfondimenti