

# Lab13

## SQLite

本節目的：

- SQLite 資料庫用途。
- 建立 SQLite 資料庫，並對資料庫裡資料表做新增、修改、刪除和查詢的基本操作。

## 13.1 SQLite 資料庫：

SQLite 是一個由 C 語言撰寫的小型關聯式資料庫管理系統，與一般資料庫不同在於它不是一個主從關係結構的資料庫，而是被整合在應用程式中的嵌入式資料庫。Android 應用程式可以將資料儲存在手機上 SQLite 中，作為資料的快取之用，缺點是本地資料庫與伺服器的資料會有不同步的疑慮。

如圖 13-1 所示，Chrome APP 使用 SQL 資料庫儲存 Cookies、Favicons 與 History...等使用資料。

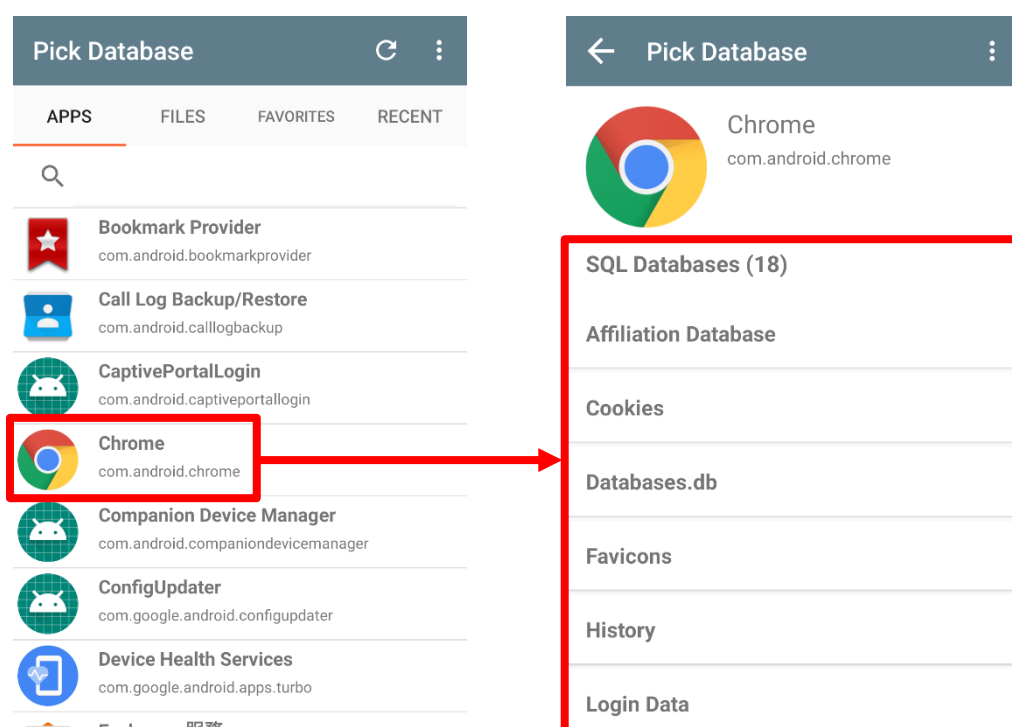


圖 13-1 使用資料庫的 APP 列表（左）與 Chrome APP 的資料庫資料表（右）

### 說明

舉凡 Line、Facebook...等應用程式，皆有使用到資料庫儲存如個人設定、聊天訊息....等大量使用資料，並且可以作為資料快取之用。

### 13.1.1 建立 SQLiteOpenHelper

Android 提供「android.database.sqlite」套件，可以處理資料庫的工作。在這個套件中的「SQLiteOpenHelper」類別，能夠讓應用程式執行建立資料庫和表格等，因此第一步我們要先建立一個 SQLiteOpenHelper 的物件。

**Step1** 首先選擇「File→New→Kotlin File/Class」，如圖 13-2 所示。

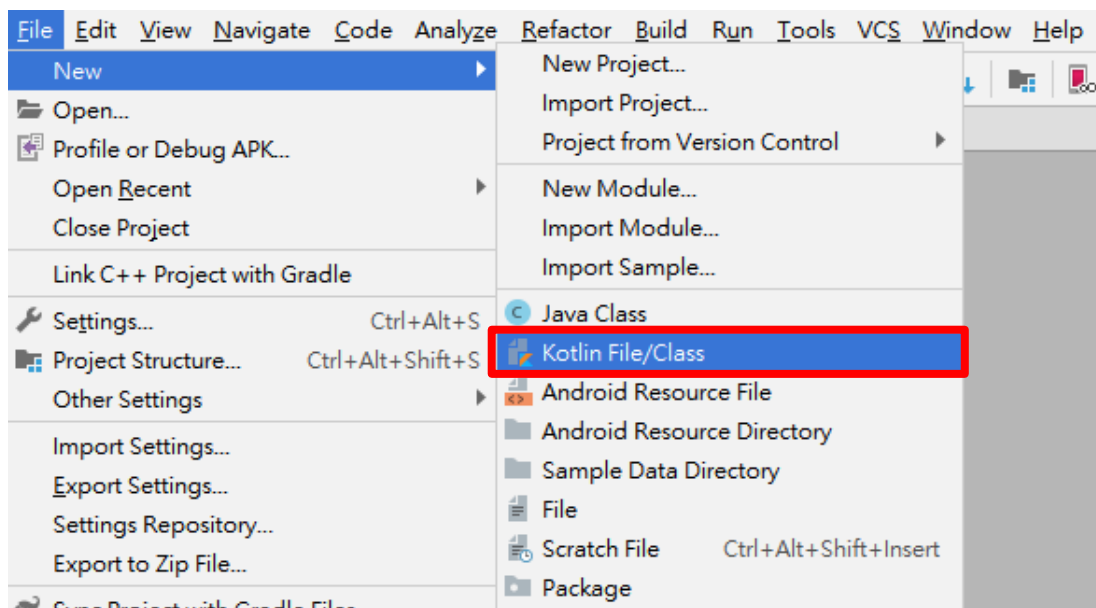


圖 13-2 產生新的 Class

**Step2** 在 Name 輸入「MyDBHelper」，並選擇建立 Class 後，點選「OK」，如圖 13-3 所示。

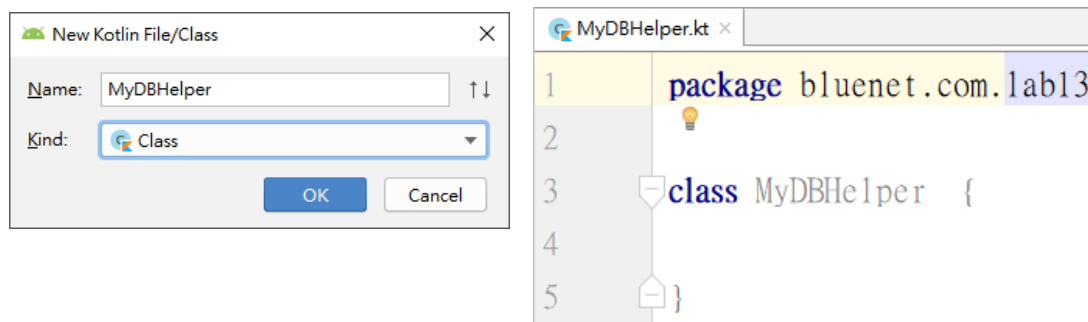


圖 13-3 創建新的類別（左）與空白的類別檔（右）

**Step3** 上步驟中會產生出一個名為 MyDBHelper 的空白 Class 檔，而我們要繼承自 SQLiteOpenHelper 來使用其功能，因此修改加入語法如下：

```

package bluenet.com.lab13

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
        //自訂建構子，只需傳入一個 Context 物件即可
class MyDBHelper (context: Context, name: String = database,
factory: SQLiteDatabase.CursorFactory? = null,
version: Int = 1): SQLiteOpenHelper(context, name, factory,
version) {        //繼承 SQLiteOpenHelper 類別
    companion object {
        private const val database = "mdatabase.db" //資料庫名稱
        private const val v = 1 //資料庫版本
    }

    override fun onCreate(db: SQLiteDatabase) {
        ... //需要加入建立資料表的 SQL 語法
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
        ... //需要加入刪除資料表的 SQL 語法
    }
}

```

應用程式第一次在裝置執行的時候，由 SQLiteOpenHelper 負責建立需要的功能，而之後執行的時候會使用已經建立好的資料庫。

## 13.1.2 設計資料庫表格

SQLite 是資料庫（Database），因此要先了解原理。資料庫代表應用程式儲存和管理資料的單位，應用程式透過資料庫來存取不同的資料。一個資料庫通常擁有數個資料表，圖 13-4 的資料庫中有乘客、司機與訂單三種資料表，分別存放三種不同類型的資料。

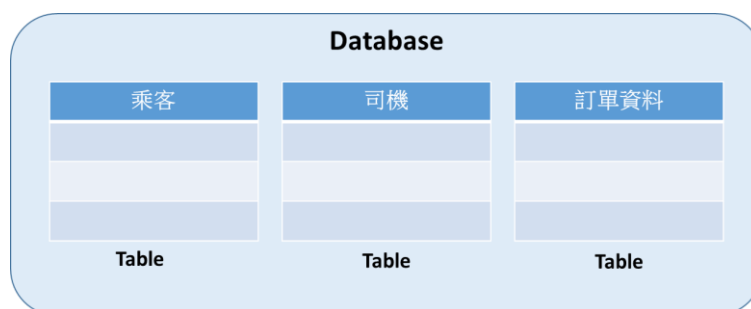


圖 13-4 資料庫與資料表示意圖

例如：一個搭車的資料庫，就需要儲存與管理乘客、司機和訂單資料。每一種定義在資料庫中的資料稱為表格（Table），例如：乘客表格可以儲存所有的乘客資料。

SQLite 資料庫必須先建立好資料庫與表格後，才可以執行存取與資料管理的工作。

建立資料庫表格使用 SQL 的「CREATE TABLE」指令，這個指令需要指定表格的名稱，還有這個表格用來儲存每一筆資料的欄位（Column）。例如以下指令會產生出一個名為 myTable 的表格：

注：SQL 語法中沒有大小寫之分。

```
CREATE TABLE myTable()
```

表格最後面的括弧中，我們要加入表格欄位的語法，每個資料庫表格中可以放入數個表格欄位，在設計表格欄位的時候，需要設定欄位名稱和型態，型態如 int、String 等會決定這欄位能夠儲存何種類型的變數，不過 SQLite 資料庫的資料型態只有下面三種，透過它們來決定表格欄位儲存的資料型態：

- **INTEGER** 整數，對應到 byte、short、int 和 long。
- **REAL** 小數，對應到的 float 和 double。
- **TEXT** 字串，對應到 String。

實現後的語法如下：

```
TITLE INTEGER
```

前者表示欄位名稱，而後後者表示變數型態。

通常在欄位中還會新增「NOT NULL」的指令，表示這個欄位不允許空值，可以避免許多資料發生問題。

```
TITLE INTEGER NOT NULL
```

此外，一個資料表必須包含一個「主鍵」欄位，這個欄位必須是唯一的值，用於索引每一筆新產生出來的資料，因此 SQLite 表格建議要包含一個欄位名稱內容唯一的主鍵、後面加上「PRIMARY KEY」的欄位。

```
book TEXT PRIMARY KEY
```

結合以上語法，我們假設要創建一個名為 myTable，有一個 book(String)、price(Integer)的欄位，我們編寫之後的字串如下：

```
CREATE TABLE myTable(book TEXT PRIMARY KEY, price INTEGER NOT NULL)
```

此即為創建表單的 SQL 語法，而 SQLite 中我們要在 MyDBHelper 裡的 onCreate(db: SQLiteDatabase)中將此語法字串傳入以產生出表單。

```
override fun onCreate(db: SQLiteDatabase) {  
    //建立一個表格 myTable，包含一個 book 字串欄位和一個 price 整數欄位  
    db.execSQL("CREATE TABLE myTable(book text PRIMARY KEY,  
                                           price integer NOT NULL)")  
}
```

onCreate()只會在創建資料表時執行，之後便不再執行，如果想要更新資料表的欄位，就需要重建資料庫。重建的流程，如圖 13-5 所示。




圖 13-5 重建資料庫流程

重建資料庫需要有三個步驟：

**Step1** 必須要修改資料庫版本，SQLiteOpenHelper 偵測到資料庫版本更新時，會調用 onUpgrade() 方法，而我們需要利用 onUpgrade() 來做刪除表格的工作。

```
private const val v = 2 //Step1：更新資料庫版本
override fun onUpgrade(db: SQLiteDatabase,
oldVersion: Int, newVersion: Int) {
    ...
}
```



**Step2** 在 onUpgrade() 中我們要加入一段 SQL 語法來刪除表格。

```
DROP TABLE IF EXISTS myTable
```

我們使用「DROP TABLE IF EXISTS」實現刪除指定的動作，這指令意思是如果 myTable 已經存在則將其刪除。

onUpgrade 修改後如下：

```
override fun onUpgrade(db: SQLiteDatabase,
oldVersion: Int, newVersion: Int) {
    //Step2：移除舊有的資料表
    db.execSQL("DROP TABLE IF EXISTS myTable")
    //Step3：重新執行 onCreate()，建立新表單
    onCreate(db)
}
```

**Step3** 刪除資料表之後，需要再次呼叫 onCreate() 來建立新的資料表。

### 13.1.3 使用資料庫

完成了建置資料庫的前置動作之後，下一步我們要實際的在程式去使用設計好的資料庫。

```
val dbrw = MyDBHelper(this).writableDatabase
```

一開始我們需要產生 MyDBHelper 的物件實體，並且透過 writableDatabase 來建立起 SQLiteDatabase 類別，而 SQLiteDatabase 就是我們的資料庫本體，後續的新增、查詢、修改、刪除資料功能都需要使用這個物件。

#### ● 新增資料

前面我們創建了一個 myTable 的資料表，要增加一筆資料，如圖 13-6 新增百科全書。



圖 13-6 新增百科全書至資料表

而對應的語法如下：

```
//Step1：建立 ContentValues 物件，用於存放要新增的資料
val cv = ContentValues()
cv.put("book", "百科全書") //填入 book 內容
cv.put("price", 900) //填入 price 內容
//Step2：透過 insert() 放入 ContentValues 至 myTable 新增資料
dbrw.insert("myTable", null, cv) //新增資料
```

這段語法中會新增一筆百科全書、價格為 900 的資料，這邊需要使用到一個 ContentValues 物件，因為一個欄位名稱 (key) 會對應到一筆資料內容 (value)，我們需要存放資料存到對應的欄位名稱之下，因此 ContentValues 能幫我們包裝資料。

我們透過 ContentValues 分別對兩個表格填入資料，第一個參數要放入欄位名稱，第二參數要放入資料內容。之後在使用 SQLiteDatabase.insert 語法將資料存放到 myTable 之中。另外，這邊要注意，如果資料內容的型態與前面訂定的資料欄位型態不同，是無法加入的。



## ● 查詢資料

查詢是四種操作方式中最複雜的功能，如要查詢某些資料，如圖 13-7 從資料表中查詢百科全書。

book	price
百科全書	900
英文雜誌	500
歷史讀物	300



book	price
百科全書	900

圖 13-7 查詢資料表

程式中需要加入以下語法：

```
var number = ""
var book = ""
var price = ""
//Step1：建立要取得的欄位
val colum = arrayOf("book", "price")
//Step2：透過 query() 查詢[book=百科全書]的欄位後，存入輸出表格至 Cursor
val c = dbrw.query("myTable", colum, "book='百科全書'", null,
null, null, null);
if (c.count > 0) { //判斷是否有資料(總筆數不為 0)
    c.moveToFirst() //從第一筆開始輸出
    //Step3：使用迴圈將 Cursor 內的資料取出
    for(i in 0 until c.count) {
        number += "$i\n"
        book += "${c.getString(0)}\n" //取得 book 資料內容
        price += "${c.getString(1)}\n" //取得 price 資料內容
        c.moveToNext(); //移至下一筆資料
    }
}
c.close() //使用完 Cursor 後記得關閉
```

此段中我們使用 SQLiteDatabase.query() 的方法取得 book 為百科全書的資料，要查詢資料，我們需要提供查詢條件及要取得的欄位兩個重要參數：

### 1) 查詢條件

要查詢某些資料時，我們需要告知要那些資料，例如查詢書籍，需要明確說

明查詢的書名、類型等資訊。程式中的描述如下：

```
欄位名稱 = "資料內容"
```

資料庫篩選出來欄位名稱符合該筆資料內容的項目，如果沒有填入任何的條件（要填入 null），則會顯示所有資料。

## 2) 要取得的欄位

查詢到資訊之後，資料庫可以不用回傳所有的欄位，我們可以限定只取得某些欄位，例如查書時可能只需要書名與價錢，這樣就可以減少不必要的資訊。而要實現這功能，我們需要使用一組字串陣列，並填入想要回傳的資料欄位名稱，如下：

```
val column = arrayOf("欄位名稱 1", "欄位名稱 2", "欄位名稱 3")
```

而 SQLiteDatabase.query() 會回傳一個 Cursor 類別的結果，Cursor 可以想像成是一張資料表，篩選後的資料表，如圖 13-8 所示。

	get(0)	get(1)
moveToFirst()	book	price
moveToNext()	百科全書	900
moveToNext()	英文雜誌	500
moveToNext()	歷史讀物	300

圖 13-8 篩選後的資料表示意圖

Cursor.count 可以取得查詢到的總比數，我們可以使用這方法來確認是否有資料以及需要取幾次資料。

Cursor 使用 get(欄位順序) 來依序取得資料內容，由於 SQLite 比較不嚴謹，如果目的是顯示資料可以都用 Cursor.getString() 來取值，而這個欄位順序等同於上面設定的 column。也就是說，如果 column 為 arrayOf("book", "price") 的話，Cursor.getString(0) 可以取得 book 的資料內容，而 Cursor.getString(1) 可以取得 price 的資料內容。

當要移動至其他筆資料時，Cursor 提供一種非常簡單的方式移動。那就是每一次使用 Cursor.moveToNext() 來移動至下一筆項目，直到資料的最後一筆為止，因此一開始需要使用 Cursor.moveToFirst() 移動到第一筆資料，以確保不會遺漏任

何筆資料。

## ● 修改資料

當某筆資料需要做修正，如圖 13-9 更正百科全書的價格為 200。

book	price	 Update	book	price
百科全書	900		百科全書	200
英文雜誌	500		英文雜誌	500
歷史讀物	300		歷史讀物	300

圖 13-9 更新資料庫的資料

我們會需要使用到 SQLiteDatabase.update() 的語法如下：

```
//Step1：建立 ContentValues 物件，用於存放要修改的資料
val cv = ContentValues()
cv.put("price", 200) //填入新價格
//Step2：查詢 book 為百科全書的欄位，透過 update() 修改資料
dbrw.update("myTable", cv, "book='百科全書'", null)
```

這段語法會先找出所有的 book 為百科全書的資料，並且將 price 為 900 的資料寫入進去，因此只要是 book 為百科全書的資料，其價格都會變成 200。

## ● 刪除資料

當某筆資料需要移除時，如圖 13-10 從資料庫中刪除百科全書的資料


book	price	 Delete	book	price
百科全書	900		英文雜誌	500
英文雜誌	500		歷史讀物	300
歷史讀物	300			

圖 13-10 刪除資料表中的資料

我們可以使用 SQLiteDatabase.delete() 將其刪除。要實作的語法如下：

```
//查詢 book 為百科全書的欄位後，透過 delete() 刪除資料
dbrw.delete("myTable", "book='百科全書'", null)
```

語法使用上與查詢類似，需要描述要查詢的資料為何，如此語法中會篩選出所有的 book 為百科全書的資料，並且將其刪除。

### 13.1.4 使用結構化查詢語言 SQL

除了使用 SQLiteOpenHelper 提供的基礎函式，SQLiteOpenHelper 也支援直接使用結構化查詢語言(SQL)對資料庫進行管理，分為資料查詢與資料異動兩種使用方式。

#### 1) 資料查詢

當我們要查詢某筆資料時，可以使用 SQLiteDatabase.rawQuery()的語法，與 SQLiteDatabase.query()一樣會回傳一個 Cursor 類別的結果。

```
//搜尋 myTable 資料表中的所有資料
val c = dbrw.rawQuery("SELECT * FROM myTable", null)
Toast.makeText(this, "共有${c.count}筆資料",
                Toast.LENGTH_SHORT).show()
... //判斷並輸出 Cursor 內容
//使用完後記得關閉 Cursor
c.close()
```

#### 2) 資料異動（新增、刪除、修改、取代...等）

當我們要更動資料庫的資料時，可以使用 SQLiteDatabase.execSQL()的語法，execSQL()並沒有任何回傳值，通常會搭配 Try Catch 一同使用，當指令成功時程式會繼續運作，而失敗時則會拋出 Exception 錯誤。

```
try{//新增一筆 book 為“百科全書” price 為 900 的資料進 myTable 資料表中
    dbrw.execSQL("INSERT INTO myTable(book, price)
                  VALUES(?,?)", arrayOf("百科全書", 900))

    //更新 myTable 資料表中符合 book 為“百科全書”的所有資料的 price 為 200
    dbrw.execSQL("UPDATE myTable SET price = 200
                  WHERE book LIKE '百科全書'")

    //刪除 myTable 資料表中符合 book 為“百科全書”的所有資料
    dbrw.execSQL("DELETE FROM myTable
                  WHERE book LIKE '百科全書'")
}catch (e: Exception){...}
```

## 13.2 圖書管理系統

- 使用 SQLite 資料庫創建一個圖書管理系統，可以增加、查詢、修改、刪除書籍資訊（書名、價格），如圖 13-11 所示。



The screenshot shows a web application titled "Lab13" in a teal header. Below the header, there are two input fields: "書名：" (Book Name) and "價格：" (Price), each followed by a text input line. Underneath these fields is a row of four buttons: "查詢" (Query), "新增" (Add), "修改" (Modify), and "刪除" (Delete). The buttons are light gray with black text. The background of the form area is light gray.

圖 13-11 圖書管理系統實機畫面

- 輸入書名、價格後按下新增可以新增一本書，如圖 13-12 所示。

Lab13

書名： 百科全書

價格： 900

查詢 新增 修改 刪除

Lab13

書名：

價格：

查詢 新增 修改 刪除

新增書名百科全書 價格900

圖 13-12 新增百科全書到資料庫（左）與新增成功畫面（右）

- 按下查詢可以列出所有的書，而如果有輸入書名，僅會列出符合書名的書，如圖 13-13 所示。

Lab13

書名：

價格：

查詢 新增 修改 刪除

書名	價格
書名:百科全書	價格:900
書名:英文雜誌	價格:500
書名:歷史讀物	價格:300

共有3筆資料

Lab13

書名： 英文雜誌

價格：

查詢 新增 修改 刪除

書名	價格
書名:英文雜誌	價格:500

共有1筆資料

圖 13-13 查詢資料庫的所有書籍（左）與查詢英文雜誌（右）

- 輸入書名、價格後按下修改可以修改一本書的價格，再按下查詢後可以看到書本的價格被修改，如圖 13-14 所示。

Lab13

書名：英文雜誌

價格：250

查詢 新增 修改 刪除

Lab13

書名：

價格：

查詢 新增 修改 刪除

書名:百科全書	價格:900
書名:英文雜誌	價格:250
書名:歷史讀物	價格:300

共有3筆資料

圖 13-14 修改英文雜誌的價格（左）與查詢修改結果（右）

- 輸入書名按下刪除可以刪除一本書，再按下查詢後可以看到書本被刪除，如圖 13-15 所示。

Lab13

書名：英文雜誌

價格：

查詢 新增 修改 刪除

Lab13

書名：

價格：

查詢 新增 修改 刪除

書名:百科全書	價格:900
書名:歷史讀物	價格:300

共有2筆資料

圖 13-15 刪除資料庫的英文雜誌（左）與使用查詢確認刪除結果（右）

## 13.2.1 圖書管理畫面設計

Step1 建立新專案，以及圖 13-16 對應的 class 與 xml 檔：

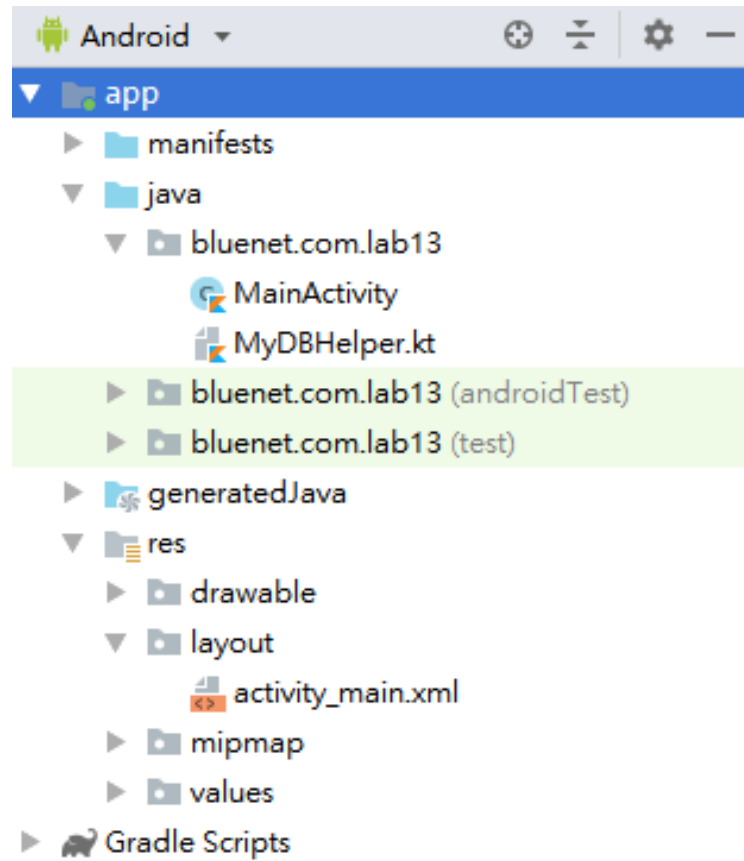


圖 13-16 圖書管理系統專案架構



**Step2** 繪製 activity\_main.xml 檔，如圖 13-17 所示。

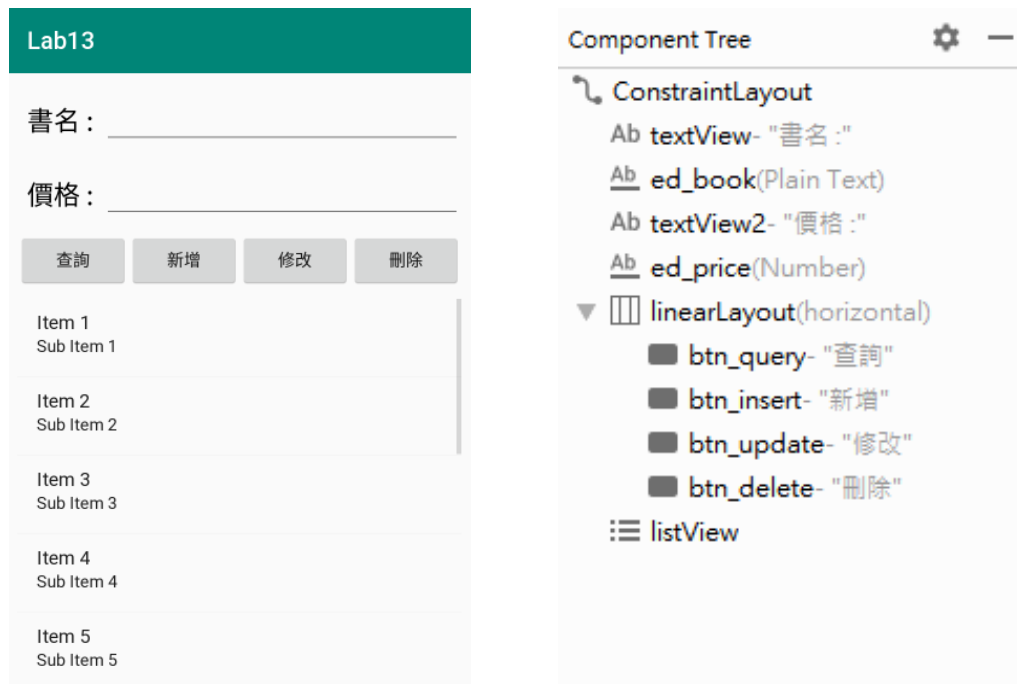


圖 13-17 圖書管理系統預覽畫面（左）與布局元件樹（右）

對應的 xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:text="書名 : "
        android:textSize="22sp"
        android:textColor="@android:color/black"
    />
</android.support.constraint.ConstraintLayout>
```

```
app:layout_constraintBottom_toBottomOf="@+id/ed_book"  
app:layout_constraintLeft_toLeftOf="parent"  
app:layout_constraintTop_toTopOf="@+id/ed_book" />
```

#### <EditText

```
android:id="@+id/ed_book"  
android:layout_width="0dp"  
android:layout_height="wrap_content"  
android:layout_marginStart="8dp"  
android:layout_marginTop="16dp"  
android:layout_marginEnd="8dp"  
android:ems="10"  
android:inputType="textPersonName"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintStart_toEndOf="@+id/textView"  
app:layout_constraintTop_toTopOf="parent" />
```

#### <TextView

```
android:id="@+id/textView2"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="價格 :"  
android:textSize="22sp"  
android:textColor="@android:color/black"  
app:layout_constraintBottom_toBottomOf="@+id/ed_price"  
app:layout_constraintStart_toStartOf="@+id/textView"  
app:layout_constraintTop_toTopOf="@+id/ed_price" />
```

#### <EditText

```
android:id="@+id/ed_price"  
android:layout_width="0dp"  
android:layout_height="wrap_content"  
android:layout_marginStart="8dp"  
android:layout_marginTop="16dp"  
android:layout_marginEnd="8dp"  
android:ems="10"  
android:inputType="number"  
app:layout_constraintEnd_toEndOf="parent"
```

```
app:layout_constraintStart_toEndOf="@+id/textView2"
app:layout_constraintTop_toBottomOf="@+id/ed_book" />
```

#### <LinearLayout

```
android:id="@+id/linearLayout"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:layout_marginStart="8dp"
android:layout_marginTop="8dp"
android:layout_marginEnd="8dp"
android:orientation="horizontal"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/ed_price">
```

#### <Button

```
android:id="@+id/btn_query"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="查詢" />
```

#### <Button

```
android:id="@+id/btn_insert"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="新增" />
```

#### <Button

```
android:id="@+id/btn_update"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="修改" />
```

#### <Button

```
android:id="@+id/btn_delete"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="删除" />
    </LinearLayout>

    <ListView
        android:id="@+id/listView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayout" />
</android.support.constraint.ConstraintLayout>
```

## 13.2.2 SQL 存取資料庫

**Step1** 撰寫 MyDBHelper，需要建立 myTable 資料表，包含 book 字串欄位、一個 price 整數欄位。

```
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
        //自訂建構子，只需傳入一個 Context 物件即可
class MyDBHelper (context: Context, name: String = database,
factory: SQLiteDatabase.CursorFactory? = null,
version: Int = v): SQLiteOpenHelper(context, name, factory,
version) { //繼承 SQLiteOpenHelper 類別
    companion object {
        private const val database = "mdatabase.db" 資料庫名稱
        private const val v = 1 資料庫版本
    }

    override fun onCreate(db: SQLiteDatabase) {
        //建立資料表 myTable，包含一個 book 字串欄位和一個 price 整數欄
        //位
        db.execSQL("CREATE TABLE myTable(book text PRIMARY KEY,
price integer NOT NULL)")
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS myTable")
        onCreate(db)
    }
}
```

**Step2** 撰寫 MainActivity，建立 MyDBHelper 實體，並透過 writableDatabase 來取得 SQLiteDatabase 實體。

```
class MainActivity : AppCompatActivity() {  
    //建立 MyDBHelper 物件  
    private lateinit var dbrw: SQLiteDatabase  
  
    private var items: ArrayList<String> = ArrayList()  
    private lateinit var adapter: ArrayAdapter<String>  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        //取得資料庫實體  
        dbrw = MyDBHelper(this).writableDatabase  
        //宣告 Adapter，使用 simple_list_item_1 並連結 listView  
        adapter = ArrayAdapter(this,  
                                android.R.layout.simple_list_item_1, items)  
        listView.adapter = adapter  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        //資料庫不使用時記得關閉  
        dbrw.close()  
    }  
}
```

**Step3** 為 4 個按鈕分別建立監聽事件，並在按下時執行對應的副程式。

```
btn_query.setOnClickListener {  
    //查詢 myTable 資料表，全部或 book 欄位為輸入字串（ed_book）的資料  
    val c = dbrw.rawQuery(if(ed_book.length()<1) "SELECT * FROM  
myTable" else "SELECT * FROM myTable WHERE book LIKE  
'${ed_book.text}'",null)  
    //從第一筆開始輸出  
    c.moveToFirst()  
    items.clear()  
    Toast.makeText(this,"共有${c.count}筆資料",  
        Toast.LENGTH_SHORT).show()  
    for (i in 0 until c.count) {  
        //填入書名與價格  
        items.add("書名:${c.getString(0)}\t\t\t\t\t  
                價格:${c.getString(1)}")  
        //移動到下一筆  
        c.moveToNext()  
    }  
    //更新 listView 內容  
    adapter.notifyDataSetChanged()  
    //使用完後記得關閉 Cursor  
    c.close()  
}
```

書名:百科全書	價格:900
書名:英文雜誌	價格:500
書名:歷史讀物	價格:300

```
btn_insert.setOnClickListener {
    //判斷是否沒有填入書名或價格
    if (ed_book.length()<1 || ed_price.length()<1)
        Toast.makeText(this,"欄位請勿留空",
                                Toast.LENGTH_SHORT).show()
    else
        try{
            //新增一筆 book 與 price 資料進入 myTable 資料表
            dbrw.execSQL("INSERT INTO myTable(book, price)
VALUES(?,?)", arrayOf<Any?>(ed_book.text.toString(),
ed_price.text.toString()))
            Toast.makeText(this,"新增書名${ed_book.text}    價格
${ed_price.text}", Toast.LENGTH_SHORT).show()
            //清空輸入框
```

```

        ed_book.setText("")
        ed_price.setText("")
    }catch (e: Exception){
        Toast.makeText(this, "新增失敗:$e",
            Toast.LENGTH_LONG).show()
    }
}

btn_update.setOnClickListener {
    //判斷是否沒有填入書名或價格
    if (ed_book.length()<1 || ed_price.length()<1)
        Toast.makeText(this, "欄位請勿留空",
            Toast.LENGTH_SHORT).show()
    else
        try{
            //更新 book 欄位為輸入字串 (ed_book) 的資料的 price 欄位數值
            dbrw.execSQL("UPDATE myTable SET price =
                ${ed_price.text} WHERE book LIKE '${ed_book.text}'")
            Toast.makeText(this, "更新書名${ed_book.text}    價格
                ${ed_price.text}", Toast.LENGTH_SHORT).show()
            //清空輸入框
            ed_book.setText("")
            ed_price.setText("")
        }catch (e: Exception){
            Toast.makeText(this, "更新失敗:$e",
                Toast.LENGTH_LONG).show()
        }
    }
}

btn_delete.setOnClickListener {
    //判斷是否沒有填入書名
    if (ed_book.length()<1)
        Toast.makeText(this, "書名請勿留空",
            Toast.LENGTH_SHORT).show()
    else
        try{
            //從 myTable 資料表刪除 book 欄位為輸入字串 (ed_book) 的資料

```

更新書名英文雜誌 價格250



```
        dbrw.execSQL("DELETE FROM myTable WHERE book LIKE  
'${ed_book.text}''")  
        Toast.makeText(this, "刪除書名${ed_book.text}",  
                        Toast.LENGTH_SHORT).show()  
  
        //清空輸入框  
        ed_book.setText("")  
        ed_price.setText("")  
    }catch (e: Exception){  
        Toast.makeText(this, "刪除失敗:$e",  
                        Toast.LENGTH_LONG).show()  
    }  
}
```