行動應用程式設計

本週教學範圍

- Kotlin課本 Chapter 13
  - **SQLite資料庫 (13.1)**
- ~~Java課本 Part 10 儲存程式資料~~
  - ~~單元59 使用SharedPreferences儲存資料~~
  - ~~**單元60 使用SQLite資料庫儲存資料**~~
  - ~~單元61 使用Content Provider跨程式存取資料~~
  - ~~單元62 使用檔案儲存資料~~

Android資料庫簡介

- SharedPreference、檔案與資料庫各有優缺點
- 資料量大時，使用SharedPreference或檔案並不適合(若有規律結構)，這時候可以使用關聯式資料庫來做大量結構化資料的儲存。
- 若可以用資料庫，就不會用檔案(因為新增、修改、刪除、查詢上會比較方便)
- 一般在Android下都是使用SQLite資料庫
  - 一個檔案
  - 輕量化的關聯式資料庫
  - 輕薄短小，無須設定或管理(無伺服器及組態檔)

步驟0. 設計資料庫表格

- SQLite 是一個小型資料庫，架構與用法跟一般資料庫差不多
- 使用前，請先規劃資料庫與資料表(與規劃一般資料庫的方式相同)
- 應用程式需要先建立好需要的資料庫，才可以執行儲存與管理資料的工作
- SQLite 基本的資料型態：
  - INTEGER – 整數，對應Java 的byte、short、int 和long
  - REAL – 浮點數，對應Java 的float 和double
  - TEXT – 字串，對應Java 的String

SQLite相關資源

- https://zh.wikipedia.org/zh-tw/SQLite
- https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

步驟1. SQLiteOpenHelper類別

- SQLite的API都在「android.database.sqlite」套件中
- 「SQLiteOpenHelper」類別可在應用程式中執行建立資料庫和表格的工作
- 一般來言，程式設計師會在應用程式第一次在裝置中執行的時候，由它來建立資料庫與表格。之後，則使用他來新增、修改與刪除資料記錄。
- 實作：在「java/[Packet Name]」上按滑鼠右鍵，選擇「New」→「Kotlin File/Class」來建立一新類別，類別名稱可取做「MyDBHelper」。
- 然後將下一頁的程式碼複製至該檔案中。(注意第一列的package 應沿用你專案的名稱)

步驟1. SQLiteOpenHelper類別

```
package org.ntunhs.myapplication

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteDatabase.CursorFactory
import android.database.sqlite.SQLiteOpenHelper

// 基本上，這裡面的東西都不用修改
class MyDBHelper(context: Context?, name: String?, factory: CursorFactory?, version: Int) :
SQLiteOpenHelper(context, name, factory, version) {
    var sCreateTableCommand = ""

    // 建立應用程式需要的表格
    override fun onCreate(db: SQLiteDatabase) {
        if (sCreateTableCommand.isEmpty()) return
        db.execSQL(sCreateTableCommand)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVer: Int, newVer: Int) {
        // TODO Auto-generated method stub
    }

}
```

步驟2. SQLiteDatabase類別

- Create
- Query
- Insert
- Update
- Delete
- ……
- https://developer.android.com/reference/kotlin/android/database/sqlite/SQLiteDatabase

步驟2-1. 主程式 (建立資料庫)

```
val DB_FILE = "friends.db"
val DB_TABLE = "friends"
val MyDB: SQLiteDatabase

// 建立自訂的 FriendDbHelper 物件
val friDbHp = MyDBHelper(applicationContext, DB_FILE, null, 1)

// 設定建立 table 的指令
friDbHp.sCreateTableCommand = "CREATE TABLE " + DB_TABLE + "(" +
        "id INTEGER PRIMARY KEY," +
        "name TEXT NOT NULL," +
        "sex TEXT," +
        "address TEXT)"

// 取得上面指定的檔名資料庫，如果該檔名不存在就會自動建立一個資料庫檔案
MyDB = friDbHp.writableDatabase
```

步驟2-2. 查詢記錄

- 查詢(Query)
  - Cursor!
    query(distinct: Boolean, table: String!, columns: Array<String!>!, selection: String!, selectionArgs: Array<String!>!, groupBy: String!, having: String!, orderBy: String!, limit: String!)
    Query the given table, returning a Cursor over the result set.
  - Cursor!
    query(distinct: Boolean, table: String!, columns: Array<String!>!, selection: String!, selectionArgs: Array<String!>!, groupBy: String!, having: String!, orderBy: String!, limit: String!, cancellationSignal: CancellationSignal!)
    Query the given URL, returning a Cursor over the result set.
  - Cursor!
    query(table: String!, columns: Array<String!>!, selection: String!, selectionArgs: Array<String!>!, groupBy: String!, having: String!, orderBy: String!)
    Query the given table, returning a Cursor over the result set.
  - Cursor!
    query(table: String!, columns: Array<String!>!, selection: String!, selectionArgs: Array<String!>!, groupBy: String!, having: String!, orderBy: String!, limit: String!)
    Query the given URL, returning a Cursor over the result set.

步驟2-2. Cursor

- android.database.Cursor
  - https://developer.android.com/reference/kotlin/android/database/Cursor
- Cursor Method
  - fun getCount(): Int
  - getDouble(columnIndex: Int): Double
  - getFloat(columnIndex: Int): Float
  - getInt(columnIndex: Int): Int
  - getLong(columnIndex: Int): Long
  - getPosition(): Int
  - getShort(columnIndex: Int): Short
  - getString(columnIndex: Int): String!
  - moveToFirst(): Boolean
  - moveToLast(): Boolean
  - moveToNext(): Boolean
  - moveToPrevious(): Boolean

步驟2-2. 主程式(查詢資料表)

```
val c = MyDB.query(
    true, DB_TABLE, arrayOf("name", "sex", "address"),
    null, null, null, null, null, null
)

if (c.count === 0) {
    textView1.text = ""
    Toast.makeText(this, "沒有資料", Toast.LENGTH_LONG).show()
}
else {
    c.moveToFirst();
    textView1.text = c.getString(0) + "\t" + c.getString(1)  + "\t" + c.getString(2)

    while (c.moveToNext()) {
        textView1.append("\n" + c.getString(0) + "\t" + c.getString(1)  + "\t" + c.getString(2))
    }
}
```

步驟2-3. 新增記錄

- 新增(Insert)
  - insert(**table: String!**, nullColumnHack: String!, **values: ContentValues!**)

    Convenience method for inserting a row into the database.

步驟2-3. ContentValues

- android.content.ContentValues
  - https://developer.android.com/reference/kotlin/android/content/ContentValues
- ContentValues Method
  - put(key: String!, value: String!)
  - put(key: String!, value: Int!)
  - put(key: String!, value: Long!)
  - put(key: String!, value: Float!)
  - put(key: String!, value: Double!)
  - ……

步驟2-3. 主程式 (新增記錄)

```
// 宣告一ContentValues
val newRow = ContentValues()

// 將要新增的欄位"name","sex"與"address"，放入ContentValues中
newRow.put("name", editText1.text.toString())
newRow.put("sex", "")
newRow.put("address", "")

// 將ContentValues中的資料，放至資料表中
MyDB.insert(DB_TABLE, null, newRow)
```

步驟2-4. 更新記錄

- 更新(Update)
  - update(**table: String!, values: ContentValues!, whereClause: String!**, whereArgs: Array<String!>!)

    Convenience method for updating rows in the database.

步驟2-4. 主程式 (更新記錄)

```
// 宣告一ContentValues
val newRow = ContentValues()

// 將要新增的欄位"name","sex"與"address"，放入ContentValues中
newRow.put("name", editText1.text.toString())
newRow.put("sex", "")
newRow.put("address", "")

// 將ContentValues中的資料，放至資料表中
MyDB.update(DB_TABLE, newRow, "id='" + rowid + "'", null)
```

步驟2-5. 刪除記錄

- 刪除(Delete)
  - delete(**table: String!, whereClause: String!**, whereArgs: Array<String!>!)

    Convenience method for deleting rows in the database.

步驟2-5. 主程式 (刪除記錄)

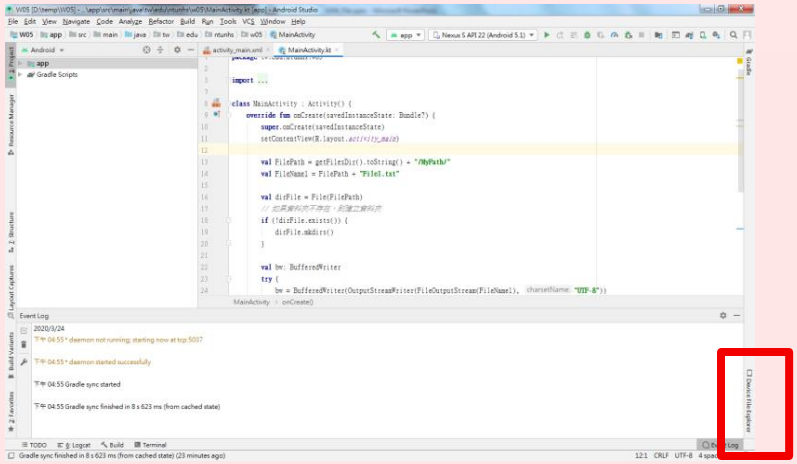
```
// 將ContentValues中的資料，放至資料表中
MyDB.delete(DB_TABLE, "id='" + rowid + "'", null)
```

Android Profiler

- 完成後，可使用Android Profiler來確認建立的資料庫
- 資料庫放於「/data/data/[Packet name]/databases/」

Android Profiler

- 提供模擬器後台管理
- 需先開啟一模擬器，才看得到東西
- 設定方法：工具列的「View」→「Tool Windows」→「 Device File Explorer 」
  - 亦會出現於右下角的「 Device File Explorer 」
- 可參考
  - https://developer.android.com/studio/profile/android-profiler



練習題(不檢查)

- 請設計一程式，包含1個EditText元件，2個Button元件與一個TextView元件
- 當使用者按下「儲存資料」的Button時，將EditText內的資料，儲存至**資料庫**中
- 當使用者按下「顯示資料」的Button時，將**資料庫**中的資料顯示出來

練習題

- 改寫前一個程式，變成可以新增、修改、刪除與查詢資料
- 請設計一程式，包含2個EditText元件(ID與Name)，4個Button元件與一個TextView元件
  - 當使用者按下「查詢」的Button時，將資料表中的所有紀錄顯示出來(此功能可捨棄，而直接做在以下三功能中)
  - 當使用者按下「新增」的Button時，新增一筆記錄至資料表中
  - 當使用者按下「修改」的Button時，將ID符合條件的紀錄，裡面的Name值予以修改
  - 當使用者按下「刪除」的Button時，將ID符合條件的紀錄，予以刪除

練習題(範例)

新增

修改

刪除

遠端資料庫

- SQLite提供我們將資料儲存在行動裝置上的環境，然而，很多時候我們希望把資料儲存在遠端，這時，我們該如何運作呢？
- 例如，將資料儲存至雲端。或者將資料存到MySQL中，或由MySQL讀出資料？

第十三章 SQLite

課前指引

- 認識SQLite
- 建立SQLite資料庫，學習資料庫基本操作

章節大綱

13.1 SQLite 資料庫

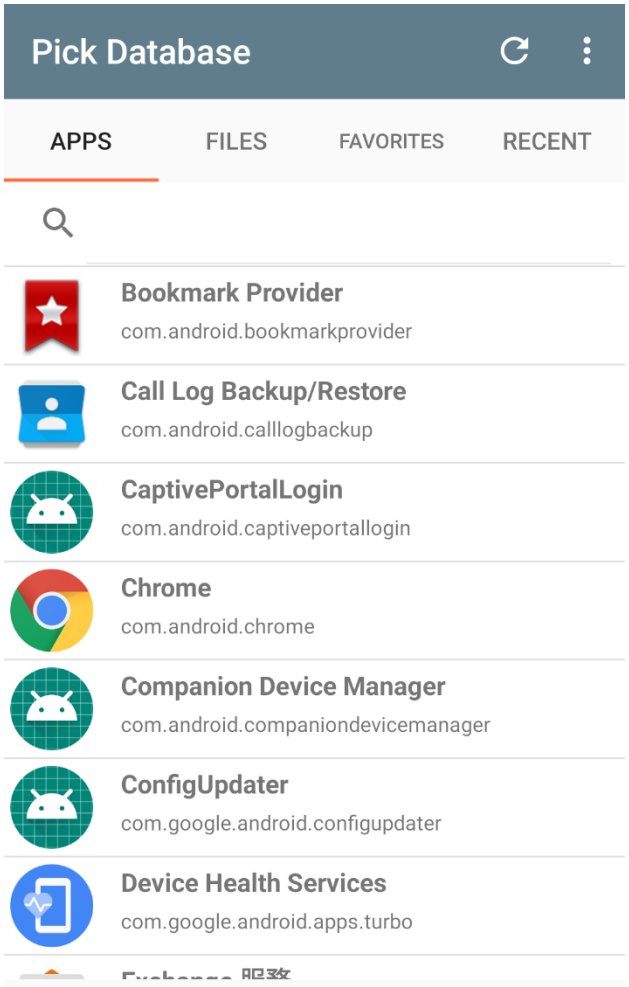
13.2 圖書管理系統

備註：可依進度點選小節

13.1 SQLite 資料庫

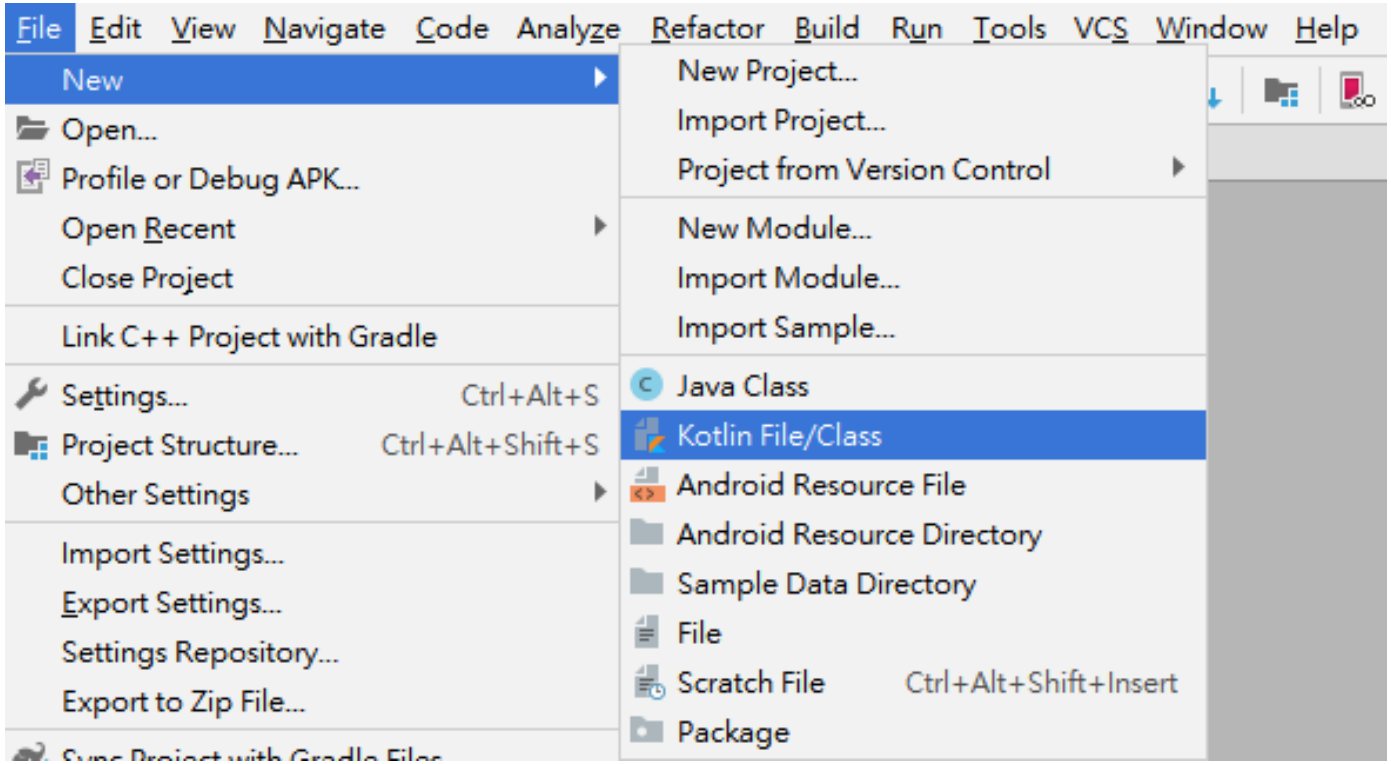
SQLite是一個由C語言撰寫的小型關聯式資料庫管理系統，與一般資料庫不同點在於，它不是一個主從關係結構的資料庫，而是被整合在應用程式中的嵌入式資料庫。

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。



13.1.1建立 SQLiteOpenHelper

Android提供SQLiteOpenHelper類別，協助應用程式管理並使用資料庫，第一步要先建立一個SQLiteOpenHelper的物件，點選工具列的「File → New → Kotlin File/Class」。



13.1.1建立 SQLiteOpenHelper

繼承SQLiteOpenHelper類別，並覆寫onCreate()與onUpdate()方法，onCreate()在創建資料庫架構（資料表、資料欄位）時呼叫，而onUpgrade()在版本變更時呼叫，通常是資料表格式異動。

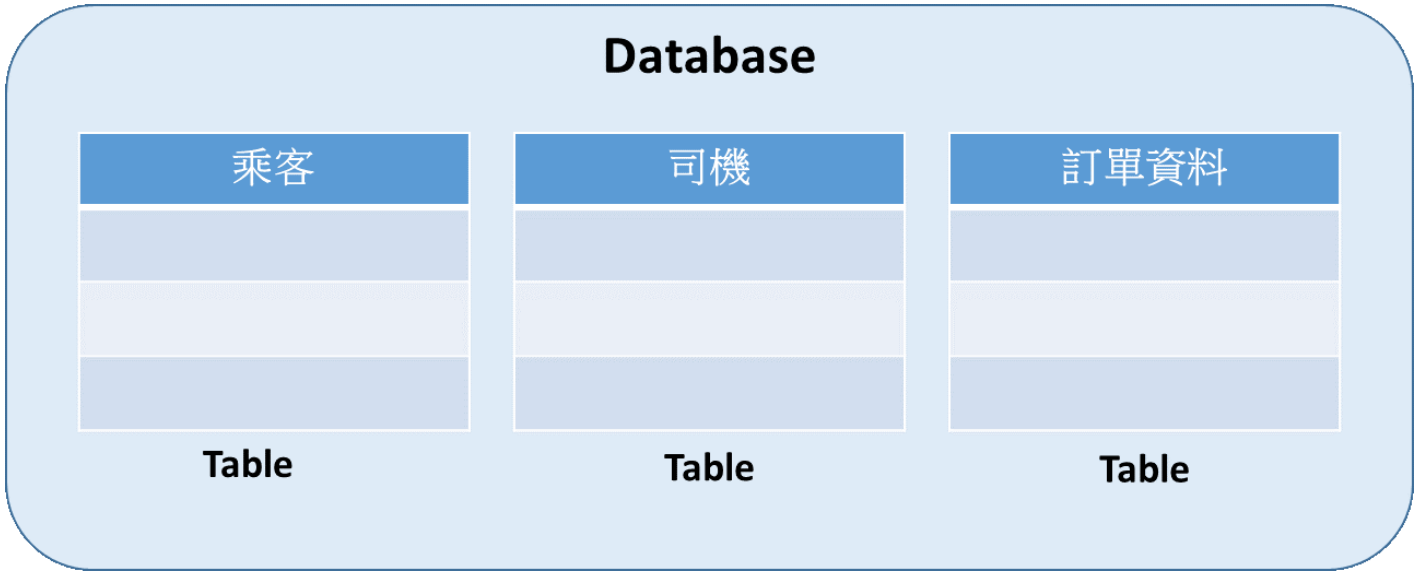
```
class MySQLiteOpenHelper (context: Context): SQLiteOpenHelper(context, name,
null, version) {
    companion object {
        private const val name =  “mdatabase.db”  //資料庫名稱
        private const val version = 1 //資料庫版本
    }

    override fun onCreate(db: SQLiteDatabase) {
        //建立資料庫架構（定義資料表）
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion:
Int) {
        //更新資料庫架構（資料表格式異動）
    }
}
```

13.1.2 設計資料庫表格

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，資料庫中有乘客、司機與訂單等三種資料表，分別存放三種不同類型的資料。



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「資料表（Table）」，例如：乘客資料表可以儲存所有的乘客資料。

13.1.2 設計資料庫表格

在使用資料庫前必須先建立好資料表，使用「CREATE TABLE」指令，可以生成指定名稱的資料表，以及這個資料表用來儲存每一筆資料的欄位（Column）。

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

myTable為資料表名稱，括弧中是資料表欄位的屬性，每個資料表可以放入數個資料欄位，在設計欄位時需要考慮到儲存變數的資料型態，如integer、real、text等會決定這欄位能夠儲存何種類型的變數。

- INTEGER（整數）：byte、short、int、long。
- REAL（小數）：float、double。
- TEXT（字串）：char、String。

13.1.2 設計資料庫表格

欄位中的「NOT NULL」指令，表示這個欄位不允許空值。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。

此即為創建資料表的SQL語法，而在SQLiteOpenHelper中，我們要在MyDBHelper裡的onCreate()中，將此語法字串傳入以產生出表單。

```
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)")
}
```

13.1.2 設計資料庫表格

當我們需要修改原先的資料庫架構時，需要透過onUpgrade更新或重建資料庫。SQLiteOpenHelper偵測到資料庫版本更新時，會呼叫onUpgrade()方法，而我們需要利用onUpgrade()來做刪除表格的工作。


```
private const val version = 2 //資料庫版本
```

```
override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    //刪除資料表
    db.execSQL("DROP TABLE IF EXISTS myTable")
    //重建資料庫
    onCreate(db)
}
```

13.1.3 使用資料庫

■ 新增資料

要在SQLite中新增資料，需要使用ContentValues物件，以欄位名稱（key）對應資料內容（value）的方式填入資料，再使用SQLiteDatabase.insert語法將資料存放到myTable之中。

book	price

Insert

book	price
百科全書	900

```
//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
//透過insert()放入ContentValues至myTable新增資料
dbrw.insert("myTable", null, cv) //新增資料
```

13.1.3 使用資料庫

■ 查詢資料

要在SQLite中查詢資料，需要使用SQLiteDatabase.query()語法，提供查詢條件及要取得的欄位等兩個重要參數，篩選出符合條件資料內容的項目，如果沒有填入任何的條件（null），則會顯示所有資料。

book	price
百科全書	900
英文雜誌	500
歷史讀物	300

Query

book	price
百科全書	900

13.1.3 使用資料庫

Cursor.count可以取得查詢到的總筆數，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。

```
//要從資料庫取得的欄位
val colum = arrayOf("book","price")
//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
    //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
```

13.1.3 使用資料庫

使用get(欄位順序)依序取得資料內容，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。

使用Cursor.moveToNext()可以移動至下一筆項目，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。

13.1.3 使用資料庫

■ 修改資料

要在SQLite中修改資料，需要使用SQLiteDatabase.update()語法，update()語法會先找出所有符合條件的資料，並且將新的資料寫入進去。

book	price
百科全書	900
英文雜誌	500
歷史讀物	300

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

```
//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
//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)
```

13.1.3 使用資料庫

■ 刪除資料

要在SQLite中刪除資料，需要使用SQLiteDatabase.delete()語法，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符合資料，並且將其刪除。

book	price
百科全書	900
英文雜誌	500
歷史讀物	300

Delete

book	price
英文雜誌	500
歷史讀物	300

```
//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)
```

13.1.4 使用結構化查詢語言 SQL

SQLiteOpenHelper除了提供基礎的語法函式，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。

- 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
c.close() //使用完後記得關閉Cursor
```

13.1.4 使用結構化查詢語言 SQL

- 資料異動：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

實作一個圖書管理系統APP，可以將書籍資料（書名、價格）保存在手機中，並且提供查詢、新增、刪除與刪改書籍資料等基本服務。

- 按下「查詢」按鈕，可以列出所有的書，若有輸入書名，僅會列出符合書名的資料。
- 按下「新增」按鈕，可以新增一本書至資料庫。
- 按下「修改」按鈕，可以修改資料庫中指定書籍的價格。
- 按下「刪除」按鈕，可以刪除資料庫中指定書籍的資料。