

SQLite 实践

一、实验目的

了解 Android 中对于 SQLite 的日常增删查改（CRUD）、事务等操作。

C: Create, R: Retrieve, U: Update, D: Delete

二、准备工作

1、创建项目

创建本次实验项目：DBDemo

2、创建数据库（请迁移上次课的例子）

(1) 创建 MyDatabaseHelper

```
class MyDatabaseHelper(val context: Context, name: String, version: Int) :  
    SQLiteOpenHelper(context, name, factory: null, version) {  
  
    private val createBook = "create table Book (" +  
        "id integer primary key autoincrement," +  
        "author text," +  
        "price real," +  
        "pages integer," +  
        "name text)"  
  
    override fun onCreate(db: SQLiteDatabase) {  
        db.execSQL(createBook)  
        Toast.makeText(context, text: "Create succeeded", Toast.LENGTH_SHORT).show()  
    }  
  
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {  
    }  
}
```

MyDatabaseHelper

(2) activity_main.xml 中放置创建数据库按钮

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/createDB"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="createDB"
        android:onClick="createDB"/>
```

(3) 该按钮在 MainActivity 中对应的事件

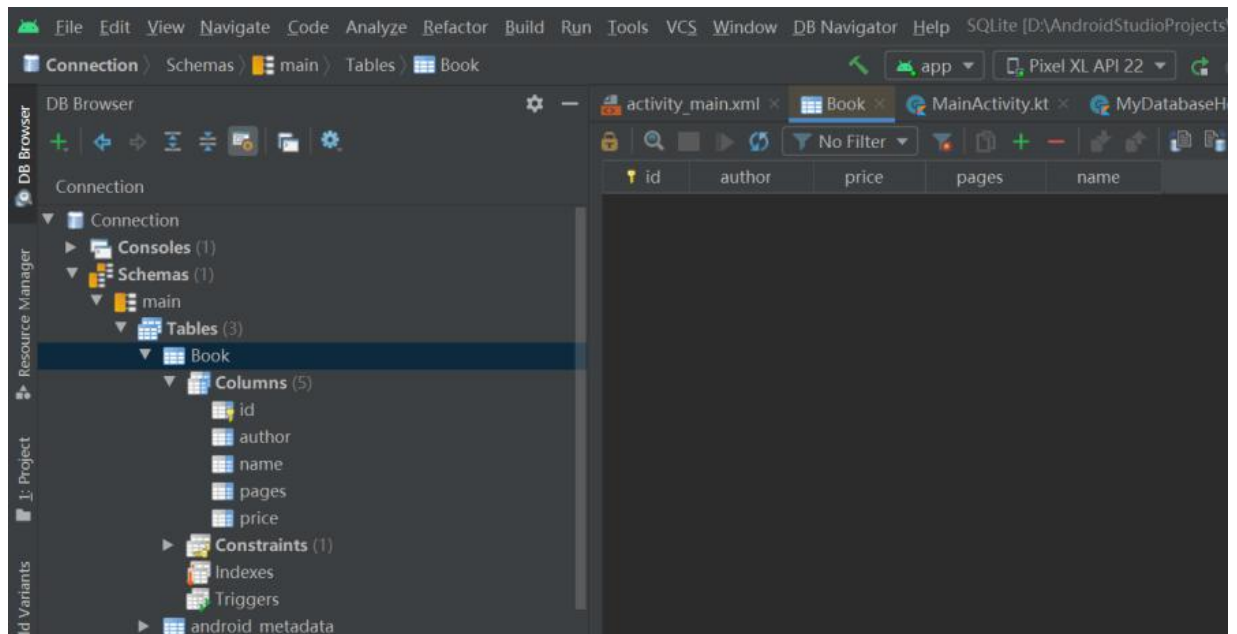
```
class MainActivity : AppCompatActivity() {

    var dbHelper : MyDatabaseHelper? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        dbHelper = MyDatabaseHelper(context = this, name = "BookStores.db", version = 1)
    }

    fun createDB(view: View) {
        dbHelper?.writableDatabase
    }
}
```

(4) 创建数据库后的最终效果



三、CRUD

1、添加数据

SQLiteOpenHelper 中提供了一个 insert () 方法，专门用于添加数据。
参数：1、表名；2、null（自动空值）；3、ContentValues 对象，提供一系列的 put 方法重载，用于向它添加数据，将表中的每个列名以及相应的待添加数据传入即可。

(1) activity_main.xml 中增加 ADD Data 按钮

```
<Button
    android:id="@+id/addData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="addData"
    android:onClick="addData"/>
```

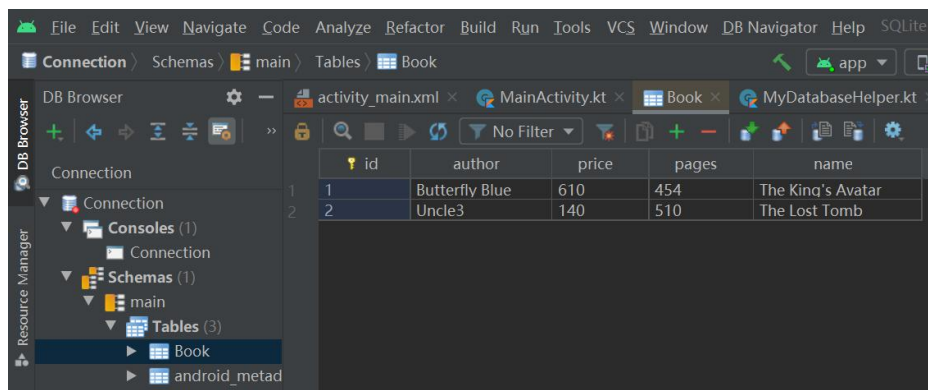
(2) MainActivity 中实现数据的插入

```
main.xml x MainActivity.kt x MyDatabaseHelper.kt x

fun addData(view: View){
    val db=dbHelper?.writableDatabase
    val values1=ContentValues().apply { this: ContentValues
        put("name","The King's Avatar")
        put("author","Butterfly Blue")
        put("pages",454)
        put("price",610.5)
    }
    db?.insert( table: "Book", nullColumnHack: null,values1)

    val values2=ContentValues().apply { this: ContentValues
        put("name","The Lost Tomb")
        put("author","Uncle3")
        put("pages",510)
        put("price",140.0)
    }
    db?.insert( table: "Book", nullColumnHack: null,values2)
}
```

(3) 打开数据库，查看结果



	id	author	price	pages	name
1	1	Butterfly Blue	610	454	The King's Avatar
2	2	Uncle3	140	510	The Lost Tomb

2、更新数据

SQLiteOpenHelper 中提供了一个 update () 方法，专门用于更新数据。

4 个参数：1、表名；2、ContentValues 对象，提供一系列的 put 方法重载，待更新的数据组装在这；3 和 4、更新条件，如果不提供则默认更新所有行

(1) 增加触发按钮

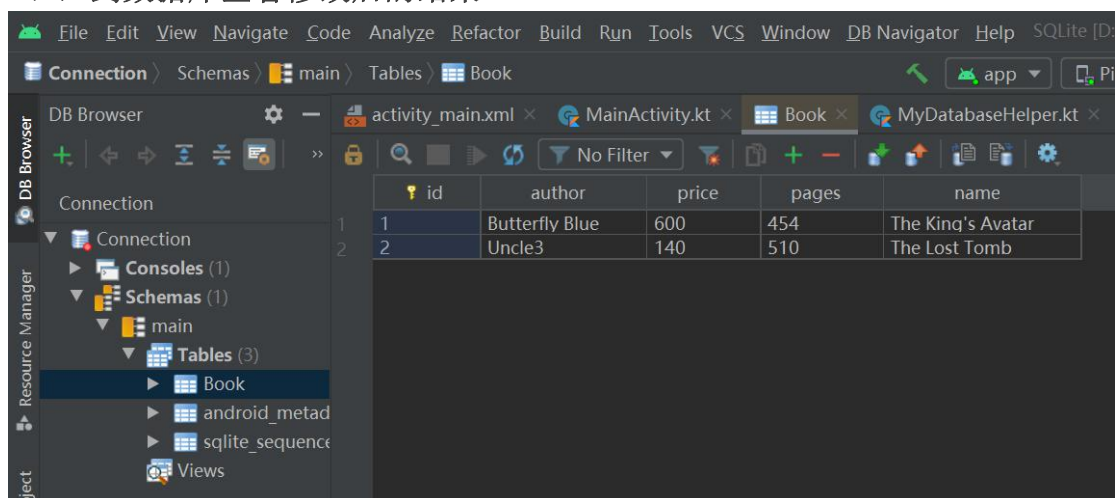
```
<Button
    android:id="@+id/updateData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="updateData"
    android:onClick="updateData"/>
```

(2) 修改 MainActivity, 模拟修改数据功能

```
fun updateData(view: View){
    val db=dbHelper?.writableDatabase
    val values=ContentValues()
    values.put("price",600)
    val rows=db?.update( table: "Book",values, whereClause: "name=?",arrayOf("The King's Avatar")) //?代表占位符,arrayOf代表为所有的占位符提供值
    Toast.makeText( context: this, text: "rows is $rows",Toast.LENGTH_SHORT).show()
}
```

这里注意: `val rows = db.update("Book", values, "name = ?", arrayOf("The Da Vinci Code"))` 这行代码, `?` 代表占位符, `arrayOf` 代表为所有的占位符提供值, 这里只有一个占位符, 就提供一个值, 该值被打包为数组一并传入 SQLite。

(3) 到数据库查看修改后的结果



The screenshot shows an IDE interface with the SQLite database browser open. The 'Book' table is selected, and its contents are displayed in a table view. The table has five columns: id, author, price, pages, and name. There are two rows of data.

id	author	price	pages	name
1	Butterfly Blue	600	454	The King's Avatar
2	Uncle3	140	510	The Lost Tomb

3、删除数据

SQLiteOpenHelper 中提供了一个 `delete()` 方法, 专门用于删除数据。
3 个参数: 1、表名; 2 和 3、删除条件, 如果不提供则默认更新所有行

(1) 增加触发按钮

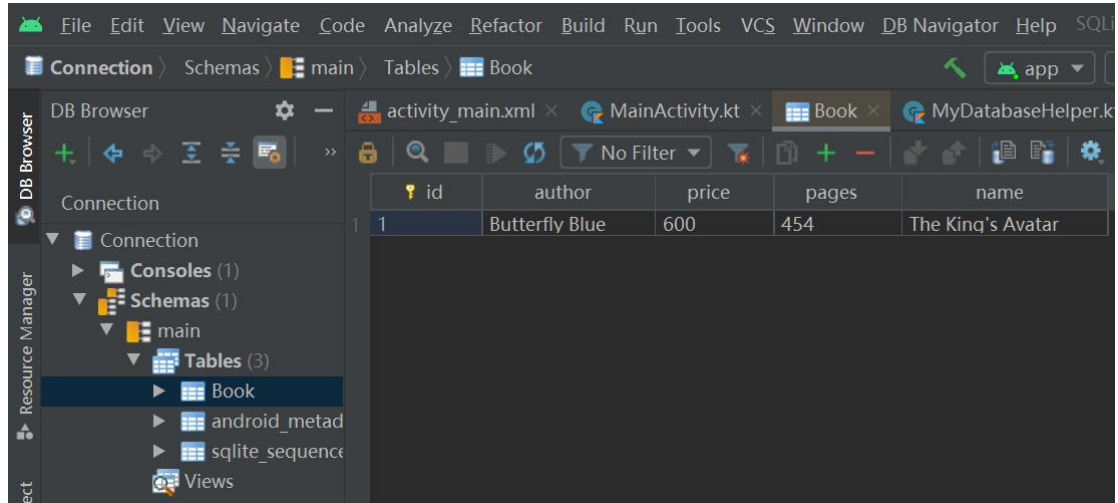
```
<Button
    android:id="@+id/deleteData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="deleteData"
    android:onClick="deleteData"/>
```

(2) 修改 MainActivity

```
fun deleteData(view: View){
    val db=dbHelper?.writableDatabase
    db?.delete( table: "Book", whereClause: "pages>?", arrayOf("500"))
}
```

删掉页码大于 500 页的数据。

(3) 查看数据库数据确认是否成功



4、查询数据

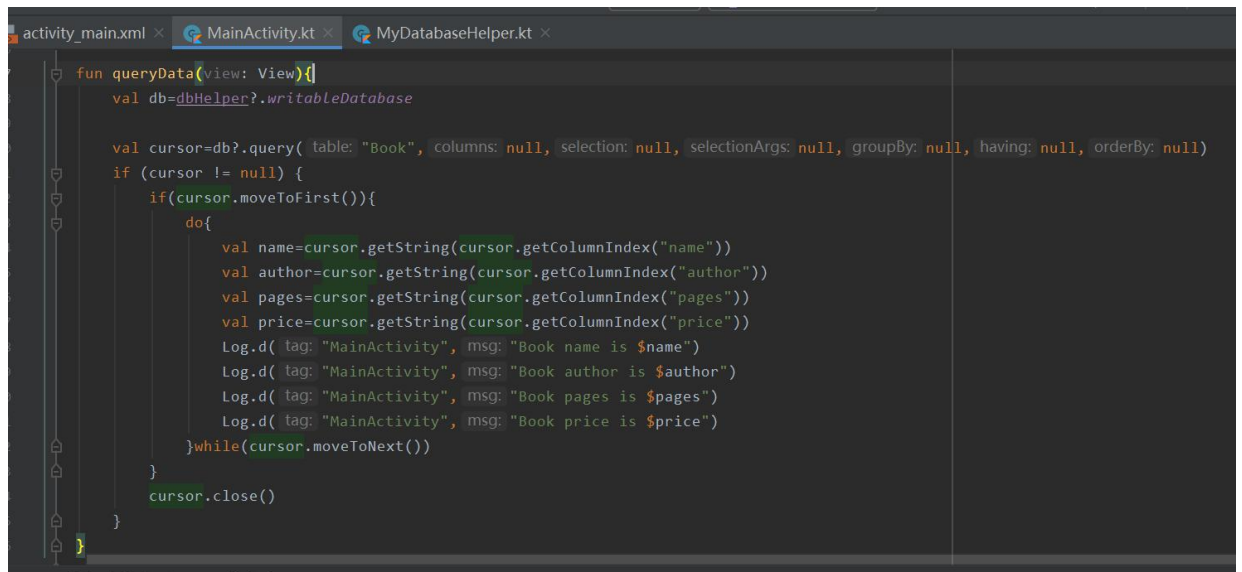
SQLiteOpenHelper 中提供了一个 query () 方法，专门用于查询数据。该方法非常复杂，最少一个重载都有 7 个参数。请参见下表。

query()方法参数	对应 SQL 部分	描 述
table	from table_name	指定查询的表名
columns	select column1, column2	指定查询的列名
selection	where column = value	指定 where 的约束条件
selectionArgs	-	为 where 中的占位符提供具体的值
groupBy	group by column	指定需要 group by 的列
having	having column = value	对 group by 后的结果进一步约束
orderBy	order by column1, column2	指定查询结果的排序方式

(1) 增加查询按钮

```
<Button
    android:id="@+id/queryData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="queryData"
    android:onClick="queryData"/>
```

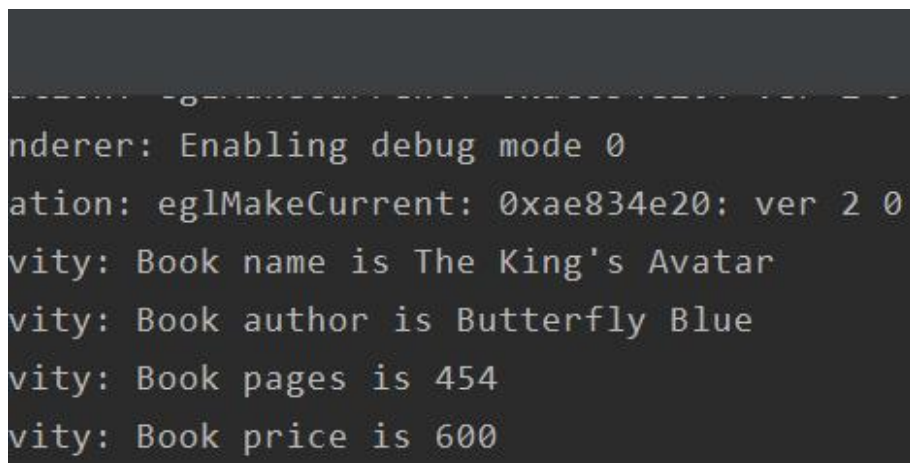
(2) 修改 MainActivity



```
activity_main.xml x MainActivity.kt x MyDatabaseHelper.kt x
fun queryData(view: View){
    val db=dbHelper?.writableDatabase

    val cursor=db?.query( table: "Book", columns: null, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null)
    if (cursor != null) {
        if(cursor.moveToFirst()){
            do{
                val name=cursor.getString(cursor.getColumnIndex("name"))
                val author=cursor.getString(cursor.getColumnIndex("author"))
                val pages=cursor.getString(cursor.getColumnIndex("pages"))
                val price=cursor.getString(cursor.getColumnIndex("price"))
                Log.d( tag: "MainActivity", msg: "Book name is $name")
                Log.d( tag: "MainActivity", msg: "Book author is $author")
                Log.d( tag: "MainActivity", msg: "Book pages is $pages")
                Log.d( tag: "MainActivity", msg: "Book price is $price")
            }while(cursor.moveToNext())
        }
        cursor.close()
    }
}
```

3) 到 Logcat 中查看查询结果



```
nderer: Enabling debug mode 0
ation: eglMakeCurrent: 0xae834e20: ver 2 0
vity: Book name is The King's Avatar
vity: Book author is Butterfly Blue
vity: Book pages is 454
vity: Book price is 600
```

四、思考

1、如果 Query 中要实现更为复杂的查询，应该如何编写代码？比如我要查询价格大于 17, 页码少于 500 的书，并且要按照书名进行排序，应如何编写代码？

使用 rawQuery()

```
//查询价格大于17，页码少于500的书，并且要按照书名进行排序
fun complexQuery(view: View){
    val db=dbHelper?.writableDatabase
    val cursor=db.rawQuery( SQL: "select * from Book where price>? and pages<? order by name", arrayOf("17","500"))
    if (cursor != null) {
        if(cursor.moveToFirst()){
            do{
                val name=cursor.getString(cursor.getColumnIndex("name"))
                val author=cursor.getString(cursor.getColumnIndex("author"))
                val pages=cursor.getString(cursor.getColumnIndex("pages"))
                val price=cursor.getString(cursor.getColumnIndex("price"))
                Log.d( tag: "MainActivity", msg: "Book name is $name")
                Log.d( tag: "MainActivity", msg: "Book author is $author")
                Log.d( tag: "MainActivity", msg: "Book pages is $pages")
                Log.d( tag: "MainActivity", msg: "Book price is $price")
            }while(cursor.moveToNext())
        }
        cursor.close()
    }
}
```

查询结果:

```
D/MainActivity: Book name is The King's Avatar
                Book author is Butterfly Blue
                Book pages is 454
                Book price is 610.5
D/MainActivity: Book name is The Unique Legend
                Book author is huXuan
                Book pages is 210
                Book price is 55
```

2、db.execSQL 是个更为简单的实现方式，请用 db.execSQL 将本次实验涉及到对数据库的操作，重新写一次

(1) 查询数据

```
fun queryData(view: View){
    val db=dbHelper?.writableDatabase
    val cursor=db?.rawQuery( sql: "select * from Book", selectionArgs: null)
    //val cursor=db?.query("Book",null,null,null,null,null,null)
    if (cursor != null) {
        if(cursor.moveToFirst()){
            do{
                val name=cursor.getString(cursor.getColumnIndex("name"))
                val author=cursor.getString(cursor.getColumnIndex("author"))
                val pages=cursor.getString(cursor.getColumnIndex("pages"))
                val price=cursor.getString(cursor.getColumnIndex("price"))
                Log.d( tag: "MainActivity", msg: "Book name is $name")
                Log.d( tag: "MainActivity", msg: "Book author is $author")
                Log.d( tag: "MainActivity", msg: "Book pages is $pages")
                Log.d( tag: "MainActivity", msg: "Book price is $price")
            }while(cursor.moveToNext())
        }
        cursor.close()
    }
}
```

(2) 插入数据

代码:

```
fun addData(view: View){
    val db=dbHelper?.writableDatabase
    db?.execSQL( sql: "insert into Book(name,author,pages,price) values('The King's Avatar','Butterfly Blue',454,610.5)")
    db?.execSQL( sql: "insert into Book(name,author,pages,price) values('The Lost Tomb','Uncle3',510,140.0)")
    db?.execSQL( sql: "insert into Book(name,author,pages,price) values('The Unique Legend','huXuan',210,55.0)")
    db?.execSQL( sql: "insert into Book(name,author,pages,price) values('Harry Potter','J.K.Rowling',2576,118.0)")
}
```

查询效果:

```
ainActivity: Book name is The King's Avatar
Book author is Butterfly Blue
Book pages is 454
Book price is 610.5
Book name is The Lost Tomb
Book author is Uncle3
Book pages is 510
Book price is 140
Book name is The Unique Legend
Book author is huXuan
Book pages is 210
Book price is 55
Book name is Harry Potter
Book author is J.K.Rowling
Book pages is 2576
Book price is 118
```

共 4 本书

(3) 更新数据

代码:

```
fun updateData(view: View){
    val db=dbHelper?.writableDatabase
    db?.execSQL( sql: "update Book set price=600 where name='The King's Avatar'")
    val cursor=db?.rawQuery( sql: "select * from Book where name='The King's Avatar'", selectionArgs: null)
    cursor?.moveToFirst()
    val rows=cursor?.count
    cursor?.close()
    Toast.makeText( context: this, text: "rows is $rows",Toast.LENGTH_SHORT).show()
}
```

通过使用 `cursor.count` 来计算修改的行数

效果:



```
D/MainActivity: Book name is The King's Avatar
Book author is Butterfly Blue
D/MainActivity: Book pages is 454
Book price is 600
```

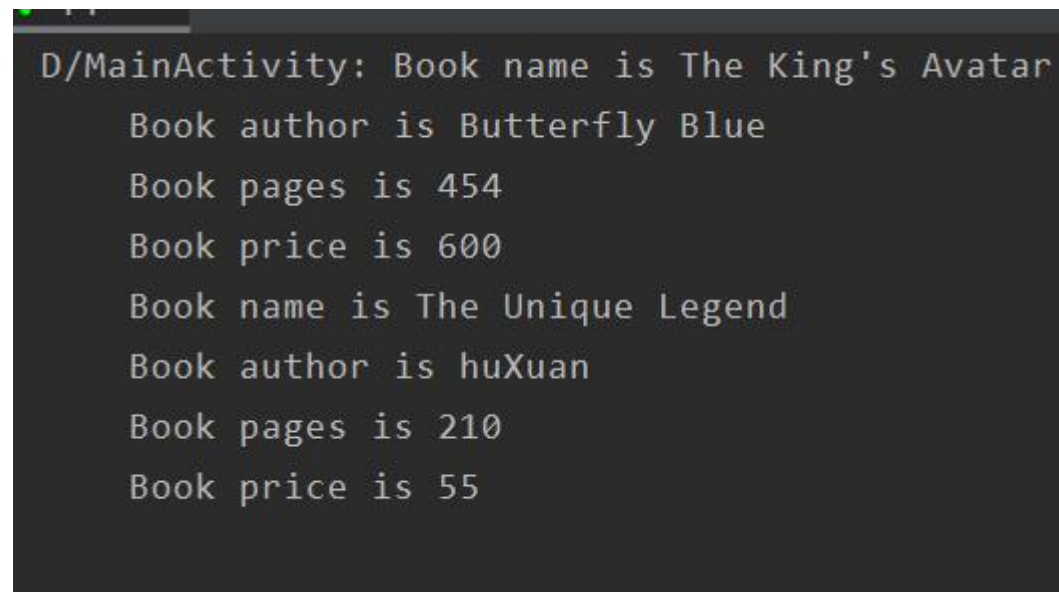
更改行为 1，价格改成了 600

(4) 删除数据

代码:

```
fun deleteData(view: View){
    val db=dbHelper?.writableDatabase
    db?.execSQL( sql: "delete from Book where pages>500")
    //db?.delete("Book","pages>", arrayOf("500"))
}
```

查询效果:



```
D/MainActivity: Book name is The King's Avatar  
Book author is Butterfly Blue  
Book pages is 454  
Book price is 600  
Book name is The Unique Legend  
Book author is huXuan  
Book pages is 210  
Book price is 55
```

只剩 2 本书