Data management

SQLite

Contents

Creating the database

In this chapter we will learn how to use SQLite in Android. Firstly we have to create the database, and to do this we have to create a new class extending the *SQLOpenHelper* superclass. We also have to override the *"onCreate"* and *"onUpgrade"* methods.

The *"onCreate"* method will be executed when the database structure has to be created, and the *"onUpgrade"* method will be called when the database has to be updated to a new version.

This class would be as follows:

```
package com.example.sqliteexample;

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

public class UserSQLiteHelper extends SQLiteOpenHelper {

	public UserSQLiteHelper(Context context, String name,
			CursorFactory factory, int version) {
		super(context, name, factory, version);
		// TODO Auto-generated constructor stub
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
	}
}
```

Now we will create the database structure: We have to include a String attribute in this class with the SQL sentence for creating the database:

```
String createSQL = "CREATE TABLE user (id INTEGER, name TEXT)";
```

In this example the database structure will consist of only one table, named "user", which contains two fields: *id* and *name*.

This sentence will be called in the *"onCreate"* method with:

```
//Execute the SQL sentence for creating the table
db.execSQL(createSQL);
```

If we need to create more tables we should add more lines such as "*db.execSQL(SQLsentence)"*.

In this example, in the *"onUpgrade"* method we will delete the database and then we will create it again:

```
db.execSQL("DROP TABLE IF EXISTS user");
db.execSQL(createSQL);
```

Finally, the class would be as follows:

```
package com.example.sqliteexample;

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

public class UserSQLiteHelper extends SQLiteOpenHelper {

	String createSQL = "CREATE TABLE user (id INTEGER, name TEXT)";

	public UserSQLiteHelper(Context context, String name,
			CursorFactory factory, int version) {
		super(context, name, factory, version);
		// TODO Auto-generated constructor stub
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		//Execute the SQL sentence for creating the table
		db.execSQL(createSQL);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
		db.execSQL("DROP TABLE IF EXISTS user");
		db.execSQL(createSQL);
	}
}
```

Inserting data

At this moment we have not executed the previous code, we have only developed the class for creating the database.

Now we have to create an activity or to use one previously created. We have to add a class attribute of the "*UserSQLiteHelper*" class and another one of "*SQLiteDatabase*".

```
UserSQLiteHelper usdbh;
SQLiteDatabase db;
```

We have to initialize the "*usdbh*" attribute, so in the "*onCreate*" method of our activity we write the following code:

```
usdbh = new UserSQLiteHelper(MainActivity.this, "DBUser", null, 1);
```

This will create the database, if needed. The parameters of the *UserSQLiteHelper*'s constructor are the context, the name of the database, a CursorFactory (in this example it will be *null*) and the version of the database.

Then we add a new button called “Insert” to the activity. This button will insert five new records in the database.

In the *“setOnClickListener”* method (inside *“onClick”*) of the button we have to add the following code:

```
//We open the database in writer mode
db = usdbh.getWritableDatabase();

//Check if the database is opened correctly
if (db != null)
{
	//We will insert 5 new rows
	for(int i=1; i<=5; i++)
	{
		//Generate the data
		int id = i;
		String name = "User" + i;

		//Insert in table User
		db.execSQL("INSERT INTO user (id, name) " +
                                        "VALUES (" + id + ", '" + name
+"')");
	}

	//Close the database
	db.close();
}
```

With *getWritableDatabase()* we are accessing the database for writing, and with *db.execSQL(sentence)* we are inserting rows in the table using SQL language.

Finally, the database is closed using *db.close()*.

There is another option to insert rows in a database: using *ContentValues* and *db.insert()*. To do this, we replace the line *db.execSQL(“INSERT … “);* with the following code:

```
//Create the record using ContentValues
ContentValues newRecord = new ContentValues();
newRecord.put("id", i);
newRecord.put("name", name);

//Insert the record in the database
db.insert("user", null, newRecord);
```

The first parameter is the table; the second one is optional (“null” is usually used for this parameter) and the third one are the values of the columns.

With this we could avoid using SQL language although the result will be the same.

Deleting and updating data

These operations (deleting and updating) are similar to inserting data. The first way to do this is using SQL language as follows:

Deleting (first way to do)

```
//Delete a record
db.execSQL("DELETE FROM user WHERE id=6 ");
```

Updating (first way to do)

```
//Update a record
db.execSQL("UPDATE user SET name='newusername' WHERE id=6 ");
```

Another way for doing this and avoiding the SQL language is using *ContentValues* and the *db.delete()* and *db.update()* operations.

Updating (second way to do)

```
//We have to put the new values
ContentValues values = new ContentValues();
values.put("name","newusername");

//Update the record in the database
db.update("user", values, "id=6", null);
```

The first parameter is the table, the second parameter is the new value, the third is the condition and finally the fourth are the arguments, if used in the third parameter. For example:

```
String[] args = new String[] {"1"};
//Update the record in the database
db.update("user", values, "id=?", args);
```

Deleting (second way to do)

```
//Delete the record with id = 6
db.delete("user", "id=6", null);
```

The parameters of this sentence are similar to the update. The first parameter is the table, the second one is the condition and the third are the arguments.

It is important to highlight that in the second way we have to use a condition in the parameter (*id=6* in the examples).

Practical exercise 1

Add two buttons to the activity: one for performing the Delete operation and another one for the Update operation. Create also two EditTexts: one for the *id* (it

is needed in the Delete/Update operations) and another one for entering the new value in the case of the Update operation.

Querying data

Now we add a new button called *Query*. When the user clicks this button all records in the database will be shown.

In the *"setOnClickListener"* method (inside *"onClick"*) of this button we have to add the following code:

```
db = usdbh.getReadableDatabase();
String[] fields = new String[] {"id", "name"};
String[] args = new String[] {"User1"};

Cursor c = db.query("user", fields, "name=?", args, null, null, null);

//We have to check if exist at least one register
if (c.moveToFirst()) {
      //List all results
      do {
            String code= c.getString(0);
            String name = c.getString(1);
            Log.d("MainActivity", code + name);
      } while(c.moveToNext());
}
db.close();
```

With the line *getReadableDatabase()* we are opening the database for reading the information (it means, only read operations).

Then we are creating two String arrays: one for the fields of the query and the second one for the arguments of the query.

With *db.query* we are querying the data to the database; and the result is an object Cursor used for listing the results of the query.

The line *c.moveToFirst()* moves the cursor to the first element of the results. If there is any result this method returns true, and false otherwise.

With *c.getString(0)* and *c.getString(1)* we are getting the columns 0 and 1. It is recommended to look at the documentation (http://developer.android.com/reference/android/database/Cursor.html) and use other methods (such as *getColumnIndex*) for real applications.

The line *c.moveToNext()* allows moving the cursor to the next element of the results. When there are no more results this method returns false and the loop finishes.

Finally the database is closed with *db.close()*.

More information about SQLiteDatabase at:

http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Practical exercise 2

Add a new button called “Query” to show (e.g., in the console) the field values of the record/s searched (by *id* and/or by *name*).

Finally, add another button called "Query all" to show all the existing records in the database. Show the *id* and the *name* for each.