Module: Mobile Application development (Android)

Session 37: Data storage using SQLite 1

1. SQLite and Android

1.1. What is SQLite?

1.2. SQLite in Android

2. SQLite Architecture

2.1. Packages

2.2. SQLiteOpenHelper

2.3. SQLiteDatabase

2.4. rawQuery() Example

2.5. query() Example

2.6. Cursor

2.7. ListViews, ListActivities and SimpleCursorAdapter

3. Working with SQLite

3.1. Introduction to the project

3.2. Create Project

3.3. Database and Data Model

3.4. User Interface

3.5. Running the apps

## 1. SQLite and Android

### 1.1 What is SQLite?

SQLite is an Open Source Database which is embedded into Android. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. In addition it requires only little memory at runtime (approx. 250 KByte).

SQLite supports the data types TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before saving them in the database. SQLite itself does not validate if the types written to the columns are actually of the defined type, e.g. you can write an integer into a string column and vice versa.

### 1.2 SQLite in Android

SQLite is available on every Android device. Using an SQLite database in Android does not require any database setup or administration.

You only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed for you by the Android platform.

Access to an SQLite database involves accessing the filesystem. This can be slow. Therefore it is recommended to perform database operations asynchronously, for example inside the AsyncTask class.

If your application creates a database, this database is by default saved in the directory DATA/data/APP\_NAME/databases/FILENAME.

The parts of the above directory are constructed based on the following rules. DATA is the path which the Environment.getDataDirectory() method returns. APP\_NAME is your application name. FILENAME is the name you specify in your application code for the database.

## 2. SQLite Architecture

### 2.1 Packages

The package android.database contains all general classes for working with databases. android.database.sqlite contains the SQLite specific classes.

### 2.2 SQLiteOpenHelper

To create and upgrade a database in your Android application you usually subclass SQLiteOpenHelper. In the constructor of your subclass you call the super() method of SQLiteOpenHelper, specifying the database name and the current database version.

In this class you need to override the onCreate() and onUpgrade() methods.

onCreate() is called by the framework, if the database does not exists.

onUpgrade() is called, if the database version is increased in your application code. This method allows you to update the database schema.

Both methods receive an SQLiteDatabase object as parameter which represents the database.

SQLiteOpenHelper provides the methods getReadableDatabase() and getWriteableDatabase() to get access to an SQLiteDatabase object; either in read or write mode.

The database tables should use the identifier \_id for the primary key of the table. Several Android functions rely on this standard.

It is best practice to create a separate class per table. This class defines static onCreate() and onUpgrade() methods. These methods are called in the corresponding methods of SQLiteOpenHelper. This way your implementation of SQLiteOpenHelper will stay readable, even if you have several tables.

### 2.3 SQLiteDatabase

SQLiteDatabase is the base class for working with a SQLite database in Android and provides methods to open, query, update and close the database.

More specifically SQLiteDatabase provides the insert(), update() and delete() methods.

In addition it provides the execSQL() method, which allows to execute an SQL statement directly.

The object ContentValues allows to define key/values. The "key" represents the table column identifier and the "value" represents the content for the table record in this column. ContentValues can be used for inserts and updates of database entries.

Queries can be created via the rawQuery() and query() methods or via the SQLiteQueryBuilder class .

rawQuery() directly accepts an SQL select statement as input.

query() provides a structured interface for specifying the SQL query.

SQLiteQueryBuilder is a convenience class that helps to build SQL queries.

### 2.4 rawQuery() Example

The following gives an example of a rawQuery() call.

Cursor cursor = getReadableDatabase().

rawQuery("select \* from todo where \_id = ?", new String[] { id });

### 2.5 query() Example

The following gives an example of a query() call.

return database.query(DATABASE\_TABLE,

new String[] { KEY\_ROWID, KEY\_CATEGORY, KEY\_SUMMARY, KEY\_DESCRIPTION },

null, null, null, null, null);

The method query() has the following parameters.

**Table 1. Parameters of the query() method**

| Parameter | Comment |
| --- | --- |
| String dbName | The table name to compile the query against. |
| String[] columnNames | A list of which table columns to return. Passing "null" will return all columns. |
| String whereClause | Where-clause, i.e. filter for the selection of data, null will select all data. |
| String[] selectionArgs | You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array. |
| String[] groupBy | A filter declaring how to group rows, null will cause the rows to not be grouped. |
| String[] having | Filter for the groups, null means no filter. |
| String[] orderBy | Table columns which will be used to order the data, null means no ordering. |

If a condition is not required you can pass null, e.g. for the group by clause.

The "whereClause" is specified without the word "where", for example a "where" statement might look like: "\_id=19 and summary=?".

If you specify placeholder values in the where clause via ?, you pass them as the selectionArgs parameter to the query.

### 2.6 Cursor

A query returns a Cursor object. A Cursor represents the result of a query and basically points to one row of the query result. This way Android can buffer the query results efficiently; as it does not have to load all data into memory.

To get the number of elements of the resulting query use the getCount() method.

To move between individual data rows, you can use the moveToFirst() and moveToNext() methods. The isAfterLast() method allows to check if the end of the query result has been reached.

Cursor provides typed get\*() methods, e.g. getLong(columnIndex), getString(columnIndex) to access the column data for the current position of the result. The "columnIndex" is the number of the column you are accessing.

Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table.

A Cursor needs to be closed with the close() method call.

### 2.7 ListViews, ListActivities and SimpleCursorAdapter

ListViews are Views which allow to display a list of elements.

ListActivities are specialized activities which make the usage of ListViews easier.

To work with databases and ListViews you can use the SimpleCursorAdapter. The SimpleCursorAdapter allows to set a layout for each row of the ListViews.

You also define an array which contains the column names and another array which contains the IDs of Views which should be filled with the data.

The SimpleCursorAdapter class will map the columns to the Views based on the Cursor passed to it.

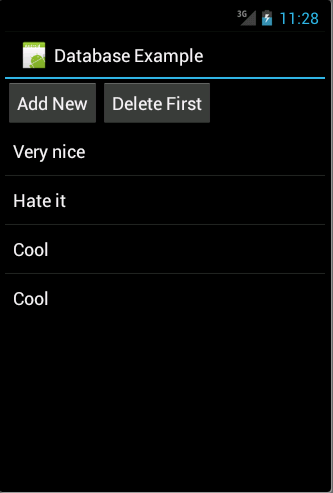
To obtain the Cursor you should use the Loader class.

## 3. Working with SQLite

### 3.1 Introduction to the project

The following demonstrates how to work with an SQLite database. We will use a data access object (DAO) to manage the data for us. The DAO is responsible for handling the database connection and for accessing and modifying the data. It will also convert the database objects into real Java Objects, so that our user interface code does not have to deal with the persistence layer.

The resulting application will look like the following.



Using a DAO is not always the right approach. A DAO creates Java model objects; using a database directly or via a ContentProvider is typically more resource efficient as you can avoid the creation of model objects.

I still demonstrate the usage of the DAO in this example to have a relatively simple example to begin with. Use the latest version of Android 4.0. This is currently API Level 15. Otherwise I would have to introduce the Loader class, which should be used as of Android 3.0 for managing a database Cursor. And this class introduces additional complexity.

### 3.2 Create Project

Create the new Android project with the name de.vogella.android.sqlite.first and an Activity called TestDatabaseActivity.

### 3.3 Database and Data Model

Create the MySQLiteHelper class. This class is responsible for creating the database. The onUpgrade() method will simply delete all existing data and re-create the table. It also defines several constants for the table name and the table columns.

|  |
| --- |
| **package** com.example.sqliteapplication;  **import** android.content.Context;  **import** android.database.sqlite.SQLiteDatabase;  **import** android.database.sqlite.SQLiteDatabase.CursorFactory;  **import** android.database.sqlite.SQLiteOpenHelper;  **import** android.util.Log;  **public** **class** MySQLiteHelper **extends** SQLiteOpenHelper {  **private** **static** **final** String *DBNAME* = "commments.db";  **private** **static** **final** **int** *DBVERSION* = 1;  **private** **static** **final** String *DATABASE\_CREATE* = "create table comments (cid integer primary key autoincrement, cname text not null)";    **public** MySQLiteHelper(Context context){  **super**(context, *DBNAME*, **null**, *DBVERSION*);  // **TODO** Auto-generated constructor stub  }  @Override  **public** **void** onCreate(SQLiteDatabase database) {  database.execSQL(*DATABASE\_CREATE*);  }  @Override  **public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  Log.*w*(MySQLiteHelper.**class**.getName(),  "Upgrading database from version " + oldVersion + " to "  + newVersion + ", which will destroy all old data");  db.execSQL("DROP TABLE IF EXISTS comments");  onCreate(db);  }  } |

Create the Comment class. This class is our model and contains the data we will save in the database and show in the user interface.

|  |
| --- |
| **package** com.example.sqliteapplication;  **public** **class** Comment {  **private** **long** id;  **private** String comment;  **public** **long** getId() {  **return** id;  }  **public** **void** setId(**long** id) {  **this**.id = id;  }  **public** String getComment() {  **return** comment;  }  **public** **void** setComment(String comment) {  **this**.comment = comment;  }    @Override  **public** String toString() {  **return** comment;  }  } |

Create the CommentsDataSource class. This class is our DAO. It maintains the database connection and supports adding new comments and fetching all comments.

|  |
| --- |
| **package** com.example.sqliteapplication;  **import** java.util.ArrayList;  **import** java.util.List;  **import** android.content.ContentValues;  **import** android.content.Context;  **import** android.database.Cursor;  **import** android.database.SQLException;  **import** android.database.sqlite.SQLiteDatabase;  **public** **class** CommentsDataSource {  // Database fields  **private** SQLiteDatabase database;  **private** MySQLiteHelper dbHelper;  **private** String[] allColumns = { "cid","cname"};  **public** CommentsDataSource(Context context) {  dbHelper = **new** MySQLiteHelper(context);  }  **public** **void** open() **throws** SQLException {  database = dbHelper.getWritableDatabase();  }  **public** **void** close() {  dbHelper.close();  }  **public** Comment createComment(String comment) {  ContentValues values = **new** ContentValues();  values.put("cname", comment);  **long** insertId = database.insert("comments", **null**, values);  Cursor cursor = database.query("comments", allColumns, "cid = " + insertId, **null**,  **null**, **null**, **null**);  cursor.moveToFirst();  Comment newComment = cursorToComment(cursor);  cursor.close();  **return** newComment;  }    **public** **void** deleteComment(Comment comment) {  **long** id = comment.getId();  System.*out*.println("Comment deleted with id: " + id);  database.delete("comments","cid = " + id, **null**);  }  **public** List<Comment> getAllComments() {  List<Comment> comments = **new** ArrayList<Comment>();  Cursor cursor = database.query("comments", allColumns, **null**, **null**, **null**, **null**, **null**);  cursor.moveToFirst();  **while** (!cursor.isAfterLast()) {  Comment comment = cursorToComment(cursor);  comments.add(comment);  cursor.moveToNext();  }  // Make sure to close the cursor  cursor.close();  **return** comments;  }  **private** Comment cursorToComment(Cursor cursor) {  Comment comment = **new** Comment();  comment.setId(cursor.getLong(0));  comment.setComment(cursor.getString(1));  **return** comment;  }  } |

### 3.4 User Interface

Change your activity\_test\_database.xml layout file in the res/layout folder to the following. This layout has two buttons for adding and deleting comments and a ListView which will be used to display the existing comments. The comment text will be generated later in the Activity by a small random generator.

|  |
| --- |
| <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"*  android:orientation=*"vertical"* >  <LinearLayout  android:id=*"@+id/group"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"* >  <Button  android:id=*"@+id/add"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:text=*"Add New"*  android:onClick=*"onClick"*/>  <Button  android:id=*"@+id/delete"*  android:layout\_width=*"wrap\_content"*  android:layout\_height=*"wrap\_content"*  android:text=*"Delete First"*  android:onClick=*"onClick"*/>    </LinearLayout>  <ListView  android:id=*"@android:id/list"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"wrap\_content"*  android:text=*"@string/hello\_world"* />  </LinearLayout> |

Change your TestDatabaseActivity class. to the following. We use here a ListActivity for displaying the data.

### 3.5 Running the apps

Install your application and use the Add and Delete button. Restart your application to validate that the data is still there.