Android SQLite database and content provider - Tutorial

Table of Contents

* [1. SQLite and Android](http://www.vogella.com/tutorials/AndroidSQLite/article.html#sqlite-and-android)
  + [1.1. What is SQLite?](http://www.vogella.com/tutorials/AndroidSQLite/article.html#what-is-sqlite)
  + [1.2. SQLite in Android](http://www.vogella.com/tutorials/AndroidSQLite/article.html#sqlite-in-android)
* [2. SQLite architecture](http://www.vogella.com/tutorials/AndroidSQLite/article.html#sqlite-architecture)
  + [2.1. Packages](http://www.vogella.com/tutorials/AndroidSQLite/article.html#packages)
  + [2.2. Creating and updating database with SQLiteOpenHelper](http://www.vogella.com/tutorials/AndroidSQLite/article.html#creating-and-updating-database-with-sqliteopenhelper)
  + [2.3. SQLiteDatabase](http://www.vogella.com/tutorials/AndroidSQLite/article.html#sqlitedatabase)
  + [2.4. rawQuery() Example](http://www.vogella.com/tutorials/AndroidSQLite/article.html#rawquery-example)
  + [2.5. query() Example](http://www.vogella.com/tutorials/AndroidSQLite/article.html#query-example)
  + [2.6. Cursor](http://www.vogella.com/tutorials/AndroidSQLite/article.html#cursor)
  + [2.7. ListViews, ListActivities and SimpleCursorAdapter](http://www.vogella.com/tutorials/AndroidSQLite/article.html#listviews-listactivities-and-simplecursoradapter)
* [3. Tutorial: Using SQLite](http://www.vogella.com/tutorials/AndroidSQLite/article.html#tutorial-using-sqlite)
  + [3.1. Introduction to the project](http://www.vogella.com/tutorials/AndroidSQLite/article.html#introduction-to-the-project)
  + [3.2. Create Project](http://www.vogella.com/tutorials/AndroidSQLite/article.html#create-project)
  + [3.3. Database and Data Model](http://www.vogella.com/tutorials/AndroidSQLite/article.html#database-and-data-model)
  + [3.4. User Interface](http://www.vogella.com/tutorials/AndroidSQLite/article.html#user-interface)
  + [3.5. Running the apps](http://www.vogella.com/tutorials/AndroidSQLite/article.html#running-the-apps)
* [4. Content provider and sharing data](http://www.vogella.com/tutorials/AndroidSQLite/article.html#content-provider-and-sharing-data)
  + [4.1. What is a content provider?](http://www.vogella.com/tutorials/AndroidSQLite/article.html#what-is-a-content-provider)
  + [4.2. Base URI of the content provider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#base-uri-of-the-content-provider)
  + [4.3. Accessing a content provider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#accessing-a-content-provider)
  + [4.4. Custom content provider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#custom-content-provider)
  + [4.5. Security and content provider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#security-and-content-provider)
  + [4.6. Thread Safety](http://www.vogella.com/tutorials/AndroidSQLite/article.html#thread-safety)
* [5. Tutorial: Using ContentProvider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#tutorial-using-contentprovider)
  + [5.1. Overview](http://www.vogella.com/tutorials/AndroidSQLite/article.html#overview)
  + [5.2. Create contacts on your emulator](http://www.vogella.com/tutorials/AndroidSQLite/article.html#create-contacts-on-your-emulator)
  + [5.3. Using the Contact Content Provider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#using-the-contact-content-provider)
* [6. Cursors and Loaders](http://www.vogella.com/tutorials/AndroidSQLite/article.html#cursors-and-loaders)
* [7. Tutorial: SQLite, custom ContentProvider and Loader](http://www.vogella.com/tutorials/AndroidSQLite/article.html#tutorial-sqlite-custom-contentprovider-and-loader)
  + [7.1. Overview](http://www.vogella.com/tutorials/AndroidSQLite/article.html#overview-2)
  + [7.2. Project](http://www.vogella.com/tutorials/AndroidSQLite/article.html#project)
  + [7.3. Database classes](http://www.vogella.com/tutorials/AndroidSQLite/article.html#database-classes)
  + [7.4. Create ContentProvider](http://www.vogella.com/tutorials/AndroidSQLite/article.html#create-contentprovider)
  + [7.5. Resources](http://www.vogella.com/tutorials/AndroidSQLite/article.html#resources)
  + [7.6. Layouts](http://www.vogella.com/tutorials/AndroidSQLite/article.html#layouts)
  + [7.7. Activities](http://www.vogella.com/tutorials/AndroidSQLite/article.html#activities)
  + [7.8. Start your application](http://www.vogella.com/tutorials/AndroidSQLite/article.html#start-your-application)
* [8. Accessing SQLite databases directly](http://www.vogella.com/tutorials/AndroidSQLite/article.html#accessing-sqlite-databases-directly)
  + [8.1. Storage location of the SQLite database](http://www.vogella.com/tutorials/AndroidSQLite/article.html#storage-location-of-the-sqlite-database)
  + [8.2. Shell access to the database](http://www.vogella.com/tutorials/AndroidSQLite/article.html#shell-access-to-the-database)
* [9. More on ListViews](http://www.vogella.com/tutorials/AndroidSQLite/article.html#more-on-listviews)
* [10. Performance](http://www.vogella.com/tutorials/AndroidSQLite/article.html#performance)
* [11. Get the Book](http://www.vogella.com/tutorials/AndroidSQLite/article.html#get-the-book)
* [12. About this website](http://www.vogella.com/tutorials/AndroidSQLite/article.html#about-this-website)
* [13. Android SQLite resources](http://www.vogella.com/tutorials/AndroidSQLite/article.html#android-sqlite-resources)
  + [13.1. vogella GmbH training and consulting support](http://www.vogella.com/tutorials/AndroidSQLite/article.html#vogella-gmbh-training-and-consulting-support)
* [Appendix A: Copyright and License](http://www.vogella.com/tutorials/AndroidSQLite/article.html#copyright-and-license)

*Using the Android SQLite Database. This tutorial describes how to use the SQLite database in Android applications. It also demonstrates how to use existing ContentProvider and how to define new ones. It also demonstrates the usage of the Loader framework which allows to load data asynchronously. The tutorial is based on Eclipse 4.2, Java 1.6 and Android 4.2.*

1. SQLite and Android

1.1. What is SQLite?

*SQLite* is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime (approx. 250 KByte) which makes it a good candidate from being embedded into other runtimes.

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 getting saved 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.

More information about SQLite can be found on the SQLite website: [http://www.sqlite.org](http://www.sqlite.org/).

1.2. SQLite in Android

SQLite is embedded into every Android device. Using an SQLite database in Android does not require a setup procedure or administration of the database.

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 file system. This can be slow. Therefore it is recommended to perform database operations asynchronously.

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

The parts of the above directory are constructed based on the following rules. DATA is the path which theEnvironment.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 android.database package contains all necessary classes for working with databases. Theandroid.database.sqlite package contains the SQLite specific classes.

2.2. Creating and updating database with SQLiteOpenHelper

To create and upgrade a database in your Android application you create a subclass of the SQLiteOpenHelper class. 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 following methods to create and update your database.

* onCreate() - is called by the framework, if the database is accessed but not yet created.
* onUpgrade() - called, if the database version is increased in your application code. This method allows you to update an existing database schema or to drop the existing database and recreate it via the onCreate() method.

Both methods receive an SQLiteDatabase object as parameter which is the Java representation of the database.

The SQLiteOpenHelper class provides the getReadableDatabase() and getWriteableDatabase() methods 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 good practice to create a separate class per table. This class defines static onCreate() andonUpgrade() methods. These methods are called in the corresponding methods of SQLiteOpenHelper. This way your implementation of SQLiteOpenHelper stays 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 valuerepresents 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. Tutorial: Using 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 aContentProvider 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** de.vogella.android.sqlite.first;

**import** android.content.Context;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.database.sqlite.SQLiteOpenHelper;

**import** android.util.Log;

**public** **class** **MySQLiteHelper** **extends** SQLiteOpenHelper {

**public** **static** **final** String TABLE\_COMMENTS = "comments";

**public** **static** **final** String COLUMN\_ID = "\_id";

**public** **static** **final** String COLUMN\_COMMENT = "comment";

**private** **static** **final** String DATABASE\_NAME = "commments.db";

**private** **static** **final** **int** DATABASE\_VERSION = 1;

*// Database creation sql statement*

**private** **static** **final** String DATABASE\_CREATE = "create table "

+ TABLE\_COMMENTS + "( " + COLUMN\_ID

+ " integer primary key autoincrement, " + COLUMN\_COMMENT

+ " text not null);";

**public** MySQLiteHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@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 " + TABLE\_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** de.vogella.android.sqlite.first;

**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;

}

*// Will be used by the ArrayAdapter in the ListView*

@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** de.vogella.android.sqlite.first;

**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 = { MySQLiteHelper.COLUMN\_ID,

MySQLiteHelper.COLUMN\_COMMENT };

**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(MySQLiteHelper.COLUMN\_COMMENT, comment);

**long** insertId = database.insert(MySQLiteHelper.TABLE\_COMMENTS, null,

values);

Cursor cursor = database.query(MySQLiteHelper.TABLE\_COMMENTS,

allColumns, MySQLiteHelper.COLUMN\_ID + " = " + 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(MySQLiteHelper.TABLE\_COMMENTS, MySQLiteHelper.COLUMN\_ID

+ " = " + id, null);

}

**public** List<Comment> getAllComments() {

List<Comment> comments = **new** ArrayList<Comment>();

Cursor cursor = database.query(MySQLiteHelper.TABLE\_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 *main.xml* layout file in the <filename class="directory">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.

<?xml version="1.0" encoding="utf-8"?>

<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"

/>

</LinearLayout>

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

**package** de.vogella.android.sqlite.first;

**import** java.util.List;

**import** java.util.Random;

**import** android.app.ListActivity;

**import** android.os.Bundle;

**import** android.view.View;

**import** android.widget.ArrayAdapter;

**public** **class** **TestDatabaseActivity** **extends** ListActivity {

**private** CommentsDataSource datasource;

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

datasource = **new** CommentsDataSource(this);

datasource.open();

List<Comment> values = datasource.getAllComments();

*// use the SimpleCursorAdapter to show the*

*// elements in a ListView*

ArrayAdapter<Comment> adapter = **new** ArrayAdapter<Comment>(this,

android.R.layout.simple\_list\_item\_1, values);

setListAdapter(adapter);

}

*// Will be called via the onClick attribute*

*// of the buttons in main.xml*

**public** **void** onClick(View view) {

@SuppressWarnings("unchecked")

ArrayAdapter<Comment> adapter = (ArrayAdapter<Comment>) getListAdapter();

Comment comment = null;

**switch** (view.getId()) {

**case** R.id.add:

String**[]** comments = **new** String**[]** { "Cool", "Very nice", "Hate it" };

**int** nextInt = **new** Random().nextInt(3);

*// save the new comment to the database*

comment = datasource.createComment(comments[nextInt]);

adapter.add(comment);

**break**;

**case** R.id.delete:

**if** (getListAdapter().getCount() > 0) {

comment = (Comment) getListAdapter().getItem(0);

datasource.deleteComment(comment);

adapter.remove(comment);

}

**break**;

}

adapter.notifyDataSetChanged();

}

@Override

**protected** **void** onResume() {

datasource.open();

super.onResume();

}

@Override

**protected** **void** onPause() {

datasource.close();

super.onPause();

}

}

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.

4. Content provider and sharing data

4.1. What is a content provider?

If you want to share data with other applications you can use a *content provider* (short provider). Provider offer data encapsulation based on URI’s. Any URI which starts with content:// points to a resources which can be accessed via a provider. A URI for a resource may allow to perform the basic CRUD operations (Create, Read, Update, Delete) on the resource via the content provider.

A provider allows applications to access data. The data can be stored in an SQlite database, on the file system, in flat files or on a remote server.

While a content provider can be used within an application to access data, its is typically used to share data with other application. As application data is by default private, a content provider is a convenient to share you data with other application based on a structured interface.

A content provider must be declared in the manifest file for the application.

4.2. Base URI of the content provider

The base URI to access a content provider is defined via the combination of the content:// schema and the name space of the provider. This name space is defined

in the manifest file via the android:authorities attribute of the receiver registration. This can for example be:content://test/

The base URI represents a collection of resources. If the base URI is combined with an instance identifier, e,g.,content://test/2, it represents a single instance.

4.3. Accessing a content provider

As it is required to know the URIs of a provider to access it, it is good practice to provide public constants for the URIs to document them to other developers.

Many Android data sources, e.g. the contacts, are accessible via content providers.

4.4. Custom content provider

To create your custom content provider you have to define a class which extends android.content.ContentProvider. You must declare this class as content provider in the Android manifest file. The corresponding entry must specify theandroid:authorities attribute which allows identifying the content provider. This authority is the basis for the URI to access data and must be unique.

<provider

android:authorities="de.vogella.android.todos.contentprovider"

android:name=".contentprovider.MyTodoContentProvider" >

</provider>

Your content provider must implement several methods, e.g. query(), insert(), update(), delete(), getType()and onCreate(). In case you do not support certain methods its good practice to throw anUnsupportedOperationException().

The query() method must return a Cursor object.

4.5. Security and content provider

Until Android version 4.2 a content provider is by default available to other Android applications. As of Android 4.2 a content provider must be explicitly exported.

To set the visibility of your content provider use the android:exported=false|true parameter in the declaration of your content provider in the *AndroidManifest.xml* file.

It is good practice to always set the android:exported parameter to ensure correct behavior across Android versions.

4.6. Thread Safety

If you work directly with databases and have multiple writers from different threads you may run into concurrency issues.

A content provider can be accessed from several programs at the same time, therefore you must implement the access thread-safe. The easiest way is to use the keyword synchronized in front of all methods of the provider, so that only one thread can access these methods at the same time.

If you do not require that Android synchronizes data access to the provider, set the android:multiprocess=trueattribute in your <provider> definition in the *AndroidManifest.xml* file. This permits an instance of the provider to be created in each client process, eliminating the need to perform interprocess communication (IPC).

5. Tutorial: Using ContentProvider

5.1. Overview

The following example will use an existing ContentProvider from the *People* application.

5.2. Create contacts on your emulator

For this example we need a few maintained contacts. Select the home menu and then the People entry to create contacts.

The app will ask you if you want to login. Either login or select "Not now". Press ""Create a new contact". You can create local contacts.

Finish adding your first contact. Afterwards the app allows you to add more contacts via the + button. As a result you should have a few new contacts in your application.

5.3. Using the Contact Content Provider

Create a new Android project called *de.vogella.android.contentprovider* with the *activity* called *ContactsActivity*.

Change the corresponding layout file in the <filename class="directory">res/layout\_ folder. Rename the ID of the existingTextView to contactview. Delete the default text.

The resulting layout file should look like the following.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<TextView

android:id="@+id/contactview"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

Access to the contact ContentProvider requires a certain permission, as not all applications should have access to the contact information. Open the *AndroidManifest.xml* file, and select the Permissions tab. On that tab click the Addbutton, and select the Uses Permission. From the drop-down list select the android.permission.READ\_CONTACTSentry.

Change the coding of the activity.

**package** de.vogella.android.contentprovider;

**import** android.app.Activity;

**import** android.database.Cursor;

**import** android.net.Uri;

**import** android.os.Bundle;

**import** android.provider.ContactsContract;

**import** android.widget.TextView;

**public** **class** **ContactsActivity** **extends** Activity {

*/\*\* Called when the activity is first created. \*/*

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_contacts);

TextView contactView = (TextView) findViewById(R.id.contactview);

Cursor cursor = getContacts();

**while** (cursor.moveToNext()) {

String displayName = cursor.getString(cursor

.getColumnIndex(ContactsContract.Data.DISPLAY\_NAME));

contactView.append("Name: ");

contactView.append(displayName);

contactView.append("\n");

}

}

**private** Cursor getContacts() {

*// Run query*

Uri uri = ContactsContract.Contacts.CONTENT\_URI;

String**[]** projection = **new** String**[]** { ContactsContract.Contacts.\_ID,

ContactsContract.Contacts.DISPLAY\_NAME };

String selection = ContactsContract.Contacts.IN\_VISIBLE\_GROUP + " = '"

+ ("1") + "'";

String**[]** selectionArgs = null;

String sortOrder = ContactsContract.Contacts.DISPLAY\_NAME

+ " COLLATE LOCALIZED ASC";

**return** managedQuery(uri, projection, selection, selectionArgs, sortOrder);

}

}

If you run this application the data is read from the ContentProvider of the People application and displayed in aTextView. Typically you would display such data in a ListView.

Unresolved directive in 001\_article.adoc - include::../AndroidBackgroundProcessing/content/120\_loader.adoc[]

6. Cursors and Loaders

One of the challenges with accessing databases is that this access is slow. The other challenge is that the application needs to consider the life cycle of the components correctly, e.g. opening and closing the cursor if a configuration change happens.

To manage the life cycle you could use the managedQuery() method in *activities* prior to Android 3.0.

As of Android 3.0 this method is deprecated and you should use the Loader framework to access the ContentProvider.

The SimpleCursorAdapter class, which can be used with ListViews, has the swapCursor() method. Your *Loader* can use this method to update the Cursor in its onLoadFinished() method.

The CursorLoader class reconnect the Cursor after a configuration change.

7. Tutorial: SQLite, custom ContentProvider and Loader

7.1. Overview

The following demo is also available in the Android Market. To allow more users to play with the app, it has been downported to Android 2.3. If you have a barcode scanner installed on your Android phone, you can scan the following QR Code to go to the example app in the Android market. Please note that the app looks and behaves differently due to the different Android versions, e.g. you have an OptionMenu instead of the *ActionBar* and the theme is different.

We will create a "To-do" application which allows the user to enter tasks for himself. These items will be stored in the SQLite database and accessed via a ContentProvider.

The tasks are called "todo items" or "todos" in this tutorial.

The application consists out of two *activities*, one for seeing a list of all todo items and one for creating and changing a specific todo item. Both *activities* will communicate via *Intents*.

To asynchronously load and manage the Cursor the main *activity* will use a Loader.

The resulting application will look similar to the following.

7.2. Project

Create the project de.vogella.android.todos with the *activity* called TodosOverviewActivity. Create another *activity*called TodoDetailActivity.

7.3. Database classes

Create the package de.vogella.android.todos.database. This package will store the classes for the database handling.

As said earlier I consider having one separate class per table as best practice. Even though we have only one table in this example we will follow this practice. This way we are prepared in case our database schema grows.

Create the following class. This class also contains constants for the table name and the columns.

**package** de.vogella.android.todos.database;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.util.Log;

**public** **class** **TodoTable** {

*// Database table*

**public** **static** **final** String TABLE\_TODO = "todo";

**public** **static** **final** String COLUMN\_ID = "\_id";

**public** **static** **final** String COLUMN\_CATEGORY = "category";

**public** **static** **final** String COLUMN\_SUMMARY = "summary";

**public** **static** **final** String COLUMN\_DESCRIPTION = "description";

*// Database creation SQL statement*

**private** **static** **final** String DATABASE\_CREATE = "create table "

+ TABLE\_TODO

+ "("

+ COLUMN\_ID + " integer primary key autoincrement, "

+ COLUMN\_CATEGORY + " text not null, "

+ COLUMN\_SUMMARY + " text not null,"

+ COLUMN\_DESCRIPTION

+ " text not null"

+ ");";

**public** **static** **void** onCreate(SQLiteDatabase database) {

database.execSQL(DATABASE\_CREATE);

}

**public** **static** **void** onUpgrade(SQLiteDatabase database, **int** oldVersion,

**int** newVersion) {

Log.w(TodoTable.class.getName(), "Upgrading database from version "

+ oldVersion + " to " + newVersion

+ ", which will destroy all old data");

database.execSQL("DROP TABLE IF EXISTS " + TABLE\_TODO);

onCreate(database);

}

}

Create the following TodoDatabaseHelper class. This class extends SQLiteOpenHelper and calls the static methods of the TodoTable helper class.

**package** de.vogella.android.todos.database;

**import** android.content.Context;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.database.sqlite.SQLiteOpenHelper;

**public** **class** **TodoDatabaseHelper** **extends** SQLiteOpenHelper {

**private** **static** **final** String DATABASE\_NAME = "todotable.db";

**private** **static** **final** **int** DATABASE\_VERSION = 1;

**public** TodoDatabaseHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

*// Method is called during creation of the database*

@Override

**public** **void** onCreate(SQLiteDatabase database) {

TodoTable.onCreate(database);

}

*// Method is called during an upgrade of the database,*

*// e.g. if you increase the database version*

@Override

**public** **void** onUpgrade(SQLiteDatabase database, **int** oldVersion,

**int** newVersion) {

TodoTable.onUpgrade(database, oldVersion, newVersion);

}

}

We will use a ContentProvider for accessing the database; we will not write a data access object (DAO) as we did in the previous SQlite example.

7.4. Create ContentProvider

Create the package de.vogella.android.todos.contentprovider.

Create the following MyTodoContentProvider class which extends ContentProvider.

**package** de.vogella.android.todos.contentprovider;

**import** java.util.Arrays;

**import** java.util.HashSet;

**import** android.content.ContentProvider;

**import** android.content.ContentResolver;

**import** android.content.ContentValues;

**import** android.content.UriMatcher;

**import** android.database.Cursor;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.database.sqlite.SQLiteQueryBuilder;

**import** android.net.Uri;

**import** android.text.TextUtils;

**import** de.vogella.android.todos.database.TodoDatabaseHelper;

**import** de.vogella.android.todos.database.TodoTable;

**public** **class** **MyTodoContentProvider** **extends** ContentProvider {

*// database*

**private** TodoDatabaseHelper database;

*// used for the UriMacher*

**private** **static** **final** **int** TODOS = 10;

**private** **static** **final** **int** TODO\_ID = 20;

**private** **static** **final** String AUTHORITY = "de.vogella.android.todos.contentprovider";

**private** **static** **final** String BASE\_PATH = "todos";

**public** **static** **final** Uri CONTENT\_URI = Uri.parse("content://" + AUTHORITY

+ "/" + BASE\_PATH);

**public** **static** **final** String CONTENT\_TYPE = ContentResolver.CURSOR\_DIR\_BASE\_TYPE

+ "/todos";

**public** **static** **final** String CONTENT\_ITEM\_TYPE = ContentResolver.CURSOR\_ITEM\_BASE\_TYPE

+ "/todo";

**private** **static** **final** UriMatcher sURIMatcher = **new** UriMatcher(

UriMatcher.NO\_MATCH);

**static** {

sURIMatcher.addURI(AUTHORITY, BASE\_PATH, TODOS);

sURIMatcher.addURI(AUTHORITY, BASE\_PATH + "/#", TODO\_ID);

}

@Override

**public** **boolean** onCreate() {

database = **new** TodoDatabaseHelper(getContext());

**return** false;

}

@Override

**public** Cursor query(Uri uri, String**[]** projection, String selection,

String**[]** selectionArgs, String sortOrder) {

*// Uisng SQLiteQueryBuilder instead of query() method*

SQLiteQueryBuilder queryBuilder = **new** SQLiteQueryBuilder();

*// check if the caller has requested a column which does not exists*

checkColumns(projection);

*// Set the table*

queryBuilder.setTables(TodoTable.TABLE\_TODO);

**int** uriType = sURIMatcher.match(uri);

**switch** (uriType) {

**case** TODOS:

**break**;

**case** TODO\_ID:

*// adding the ID to the original query*

queryBuilder.appendWhere(TodoTable.COLUMN\_ID + "="

+ uri.getLastPathSegment());

**break**;

**default**:

**throw** **new** IllegalArgumentException("Unknown URI: " + uri);

}

SQLiteDatabase db = database.getWritableDatabase();

Cursor cursor = queryBuilder.query(db, projection, selection, selectionArgs, null, null, sortOrder);

*// make sure that potential listeners are getting notified*

cursor.setNotificationUri(getContext().getContentResolver(), uri);

**return** cursor;

}

@Override

**public** String getType(Uri uri) {

**return** null;

}

@Override

**public** Uri insert(Uri uri, ContentValues values) {

**int** uriType = sURIMatcher.match(uri);

SQLiteDatabase sqlDB = database.getWritableDatabase();

**long** id = 0;

**switch** (uriType) {

**case** TODOS:

id = sqlDB.insert(TodoTable.TABLE\_TODO, null, values);

**break**;

**default**:

**throw** **new** IllegalArgumentException("Unknown URI: " + uri);

}

getContext().getContentResolver().notifyChange(uri, null);

**return** Uri.parse(BASE\_PATH + "/" + id);

}

@Override

**public** **int** delete(Uri uri, String selection, String**[]** selectionArgs) {

**int** uriType = sURIMatcher.match(uri);

SQLiteDatabase sqlDB = database.getWritableDatabase();

**int** rowsDeleted = 0;

**switch** (uriType) {

**case** TODOS:

rowsDeleted = sqlDB.delete(TodoTable.TABLE\_TODO, selection,

selectionArgs);

**break**;

**case** TODO\_ID:

String id = uri.getLastPathSegment();

**if** (TextUtils.isEmpty(selection)) {

rowsDeleted = sqlDB.delete(

TodoTable.TABLE\_TODO,

TodoTable.COLUMN\_ID + "=" + id,

null);

} **else** {

rowsDeleted = sqlDB.delete(

TodoTable.TABLE\_TODO,

TodoTable.COLUMN\_ID + "=" + id

+ " and " + selection,

selectionArgs);

}

**break**;

**default**:

**throw** **new** IllegalArgumentException("Unknown URI: " + uri);

}

getContext().getContentResolver().notifyChange(uri, null);

**return** rowsDeleted;

}

@Override

**public** **int** update(Uri uri, ContentValues values, String selection,

String**[]** selectionArgs) {

**int** uriType = sURIMatcher.match(uri);

SQLiteDatabase sqlDB = database.getWritableDatabase();

**int** rowsUpdated = 0;

**switch** (uriType) {

**case** TODOS:

rowsUpdated = sqlDB.update(TodoTable.TABLE\_TODO,

values,

selection,

selectionArgs);

**break**;

**case** TODO\_ID:

String id = uri.getLastPathSegment();

**if** (TextUtils.isEmpty(selection)) {

rowsUpdated = sqlDB.update(TodoTable.TABLE\_TODO,

values,

TodoTable.COLUMN\_ID + "=" + id,

null);

} **else** {

rowsUpdated = sqlDB.update(TodoTable.TABLE\_TODO,

values,

TodoTable.COLUMN\_ID + "=" + id

+ " and "

+ selection,

selectionArgs);

}

**break**;

**default**:

**throw** **new** IllegalArgumentException("Unknown URI: " + uri);

}

getContext().getContentResolver().notifyChange(uri, null);

**return** rowsUpdated;

}

**private** **void** checkColumns(String**[]** projection) {

String**[]** available = { TodoTable.COLUMN\_CATEGORY,

TodoTable.COLUMN\_SUMMARY, TodoTable.COLUMN\_DESCRIPTION,

TodoTable.COLUMN\_ID };

**if** (projection != null) {

HashSet<String> requestedColumns = **new** HashSet<String>(

Arrays.asList(projection));

HashSet<String> availableColumns = **new** HashSet<String>(

Arrays.asList(available));

*// check if all columns which are requested are available*

**if** (!availableColumns.containsAll(requestedColumns)) {

**throw** **new** IllegalArgumentException(

"Unknown columns in projection");

}

}

}

}

MyTodoContentProvider implements update(), insert(), delete() and query() methods. These methods map more or less directly to the SQLiteDatabase interface.

It also has the checkColumns() method to validate that a query only requests valid columns.

Register your ContentProvider in your AndroidManifest.xml file.

<application

<!-- Place the following after the Activity

Definition

-->

<provider

android:name=".contentprovider.MyTodoContentProvider"

android:authorities="de.vogella.android.todos.contentprovider" >

</provider>

</application>

7.5. Resources

Our application requires several resources. First define a menu listmenu.xml ` in the folder `res/menu. If you use the Android resource wizard to create the "listmenu.xml" file, the folder will be created for you; if you create the file manually you also need to create the folder manually.

This XML file will be used to define the option menu in our application. The android:showAsAction="always" attribute will ensure that this menu entry is displayed in the *ActionBar* of our application.

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android" >

<item

android:id="@+id/insert"

android:showAsAction="always"

android:title="Insert">

</item>

</menu>

The user will be able to select the priority for the todo items. For the priorities we create a string array. Create the following file priority.xml in the res/values folder .

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string-array name="priorities">

<item>Urgent</item>

<item>Reminder</item>

</string-array>

</resources>

Define also additional strings for the application. Edit strings.xml under res/values.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="hello">Hello World, Todo!</string>

<string name="app\_name">Todo</string>

<string name="no\_todos">Currently there are no Todo items maintained</string>

<string name="menu\_insert">Add Item</string>

<string name="menu\_delete">Delete Todo</string>

<string name="todo\_summary">Summary</string>

<string name="todo\_description">Delete Todo</string>

<string name="todo\_edit\_summary">Summary</string>

<string name="todo\_edit\_description">Description</string>

<string name="todo\_edit\_confirm">Confirm</string>

</resources>

7.6. Layouts

We will define three layouts. One will be used for the display of a row in the list, the other ones will be used by our*activities*.

The row layout refers to an icon called *reminder*. Paste an icon of type "png" called "reminder.png" into yourres/drawable folders ( drawable-hdpi, drawable-mdpi, drawable-ldpi )

If you do not have an icon available you can copy the icon created by the Android wizard (ic\_launcher.png in the res/drawable\* folders) or rename the reference in the layout file. Please note that the Android Developer Tools sometimes change the name of this generated icon, so your file might not be called "ic\_launcher.png".

Alternatively you could remove the icon definition from the "todo\_row.xml" layout definition file which you will create in the next step.

Create the "todo\_row.xml" layout file in the folder <filename class="directory">res/layout\_.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

<ImageView

android:id="@+id/icon"

android:layout\_width="30dp"

android:layout\_height="24dp"

android:layout\_marginLeft="4dp"

android:layout\_marginRight="8dp"

android:layout\_marginTop="8dp"

android:src="@drawable/reminder" >

</ImageView>

<TextView

android:id="@+id/label"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="6dp"

android:lines="1"

android:text="@+id/TextView01"

android:textSize="24dp"

>

</TextView>

</LinearLayout>

Create the todo\_list.xml layout file. This layout defines how the list looks like.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<ListView

android:id="@android:id/list"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

</ListView>

<TextView

android:id="@android:id/empty"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/no\_todos" />

</LinearLayout>

Create the todo\_edit.xml layout file. This layout will be used to display and edit an individual todo item in theTodoDetailActivity *activity*.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<Spinner

android:id="@+id/category"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:entries="@array/priorities" >

</Spinner>

<LinearLayout

android:id="@+id/LinearLayout01"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" >

<EditText

android:id="@+id/todo\_edit\_summary"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:hint="@string/todo\_edit\_summary"

android:imeOptions="actionNext" >

</EditText>

</LinearLayout>

<EditText

android:id="@+id/todo\_edit\_description"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_weight="1"

android:gravity="top"

android:hint="@string/todo\_edit\_description"

android:imeOptions="actionNext" >

</EditText>

<Button

android:id="@+id/todo\_edit\_button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/todo\_edit\_confirm" >

</Button>

</LinearLayout>

7.7. Activities

Change the coding of your activities to the following. First TodosOverviewActivity.java.

**package** de.vogella.android.todos;

**import** android.app.ListActivity;

**import** android.app.LoaderManager;

**import** android.content.CursorLoader;

**import** android.content.Intent;

**import** android.content.Loader;

**import** android.database.Cursor;

**import** android.net.Uri;

**import** android.os.Bundle;

**import** android.view.ContextMenu;

**import** android.view.ContextMenu.ContextMenuInfo;

**import** android.view.Menu;

**import** android.view.MenuInflater;

**import** android.view.MenuItem;

**import** android.view.View;

**import** android.widget.AdapterView.AdapterContextMenuInfo;

**import** android.widget.ListView;

**import** android.widget.SimpleCursorAdapter;

**import** de.vogella.android.todos.contentprovider.MyTodoContentProvider;

**import** de.vogella.android.todos.database.TodoTable;

*/\**

*\* TodosOverviewActivity displays the existing todo items*

*\* in a list*

*\**

*\* You can create new ones via the ActionBar entry "Insert"*

*\* You can delete existing ones via a long press on the item*

*\*/*

**public** **class** **TodosOverviewActivity** **extends** ListActivity **implements**

LoaderManager.LoaderCallbacks<Cursor> {

**private** **static** **final** **int** ACTIVITY\_CREATE = 0;

**private** **static** **final** **int** ACTIVITY\_EDIT = 1;

**private** **static** **final** **int** DELETE\_ID = Menu.FIRST + 1;

*// private Cursor cursor;*

**private** SimpleCursorAdapter adapter;

*/\*\* Called when the activity is first created. \*/*

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.todo\_list);

this.getListView().setDividerHeight(2);

fillData();

registerForContextMenu(getListView());

}

*// create the menu based on the XML defintion*

@Override

**public** **boolean** onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.listmenu, menu);

**return** true;

}

*// Reaction to the menu selection*

@Override

**public** **boolean** onOptionsItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** R.id.insert:

createTodo();

**return** true;

}

**return** super.onOptionsItemSelected(item);

}

@Override

**public** **boolean** onContextItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** DELETE\_ID:

AdapterContextMenuInfo info = (AdapterContextMenuInfo) item

.getMenuInfo();

Uri uri = Uri.parse(MyTodoContentProvider.CONTENT\_URI + "/"

+ info.id);

getContentResolver().delete(uri, null, null);

fillData();

**return** true;

}

**return** super.onContextItemSelected(item);

}

**private** **void** createTodo() {

Intent i = **new** Intent(this, TodoDetailActivity.class);

startActivity(i);

}

*// Opens the second activity if an entry is clicked*

@Override

**protected** **void** onListItemClick(ListView l, View v, **int** position, **long** id) {

super.onListItemClick(l, v, position, id);

Intent i = **new** Intent(this, TodoDetailActivity.class);

Uri todoUri = Uri.parse(MyTodoContentProvider.CONTENT\_URI + "/" + id);

i.putExtra(MyTodoContentProvider.CONTENT\_ITEM\_TYPE, todoUri);

startActivity(i);

}

**private** **void** fillData() {

*// Fields from the database (projection)*

*// Must include the \_id column for the adapter to work*

String**[]** from = **new** String**[]** { TodoTable.COLUMN\_SUMMARY };

*// Fields on the UI to which we map*

**int[]** to = **new** **int[]** { R.id.label };

getLoaderManager().initLoader(0, null, this);

adapter = **new** SimpleCursorAdapter(this, R.layout.todo\_row, null, from,to, 0);

setListAdapter(adapter);

}

@Override

**public** **void** onCreateContextMenu(ContextMenu menu, View v,

ContextMenuInfo menuInfo) {

super.onCreateContextMenu(menu, v, menuInfo);

menu.add(0, DELETE\_ID, 0, R.string.menu\_delete);

}

*// creates a new loader after the initLoader () call*

@Override

**public** Loader<Cursor> onCreateLoader(**int** id, Bundle args) {

String**[]** projection = { TodoTable.COLUMN\_ID, TodoTable.COLUMN\_SUMMARY };

CursorLoader cursorLoader = **new** CursorLoader(this,

MyTodoContentProvider.CONTENT\_URI, projection, null, null, null);

**return** cursorLoader;

}

@Override

**public** **void** onLoadFinished(Loader<Cursor> loader, Cursor data) {

adapter.swapCursor(data);

}

@Override

**public** **void** onLoaderReset(Loader<Cursor> loader) {

*// data is not available anymore, delete reference*

adapter.swapCursor(null);

}

}

And TodoDetailActivity.java

**package** de.vogella.android.todos;

**import** android.app.Activity;

**import** android.content.ContentValues;

**import** android.database.Cursor;

**import** android.net.Uri;

**import** android.os.Bundle;

**import** android.text.TextUtils;

**import** android.view.View;

**import** android.widget.Button;

**import** android.widget.EditText;

**import** android.widget.Spinner;

**import** android.widget.Toast;

**import** de.vogella.android.todos.contentprovider.MyTodoContentProvider;

**import** de.vogella.android.todos.database.TodoTable;

*/\**

*\* TodoDetailActivity allows to enter a new todo item*

*\* or to change an existing*

*\*/*

**public** **class** **TodoDetailActivity** **extends** Activity {

**private** Spinner mCategory;

**private** EditText mTitleText;

**private** EditText mBodyText;

**private** Uri todoUri;

@Override

**protected** **void** onCreate(Bundle bundle) {

super.onCreate(bundle);

setContentView(R.layout.todo\_edit);

mCategory = (Spinner) findViewById(R.id.category);

mTitleText = (EditText) findViewById(R.id.todo\_edit\_summary);

mBodyText = (EditText) findViewById(R.id.todo\_edit\_description);

Button confirmButton = (Button) findViewById(R.id.todo\_edit\_button);

Bundle extras = getIntent().getExtras();

*// check from the saved Instance*

todoUri = (bundle == null) ? null : (Uri) bundle .getParcelable(MyTodoContentProvider.CONTENT\_ITEM\_TYPE);

*// Or passed from the other activity*

**if** (extras != null) {

todoUri = extras

.getParcelable(MyTodoContentProvider.CONTENT\_ITEM\_TYPE);

fillData(todoUri);

}

confirmButton.setOnClickListener(**new** View.OnClickListener() {

**public** **void** onClick(View view) {

**if** (TextUtils.isEmpty(mTitleText.getText().toString())) {

makeToast();

} **else** {

setResult(RESULT\_OK);

finish();

}

}

});

}

**private** **void** fillData(Uri uri) {

String**[]** projection = { TodoTable.COLUMN\_SUMMARY,

TodoTable.COLUMN\_DESCRIPTION, TodoTable.COLUMN\_CATEGORY };

Cursor cursor = getContentResolver().query(uri, projection, null, null,

null);

**if** (cursor != null) {

cursor.moveToFirst();

String category = cursor.getString(cursor

.getColumnIndexOrThrow(TodoTable.COLUMN\_CATEGORY));

**for** (**int** i = 0; i < mCategory.getCount(); i++) {

String s = (String) mCategory.getItemAtPosition(i);

**if** (s.equalsIgnoreCase(category)) {

mCategory.setSelection(i);

}

}

mTitleText.setText(cursor.getString(cursor

.getColumnIndexOrThrow(TodoTable.COLUMN\_SUMMARY)));

mBodyText.setText(cursor.getString(cursor

.getColumnIndexOrThrow(TodoTable.COLUMN\_DESCRIPTION)));

*// always close the cursor*

cursor.close();

}

}

**protected** **void** onSaveInstanceState(Bundle outState) {

super.onSaveInstanceState(outState);

saveState();

outState.putParcelable(MyTodoContentProvider.CONTENT\_ITEM\_TYPE, todoUri);

}

@Override

**protected** **void** onPause() {

super.onPause();

saveState();

}

**private** **void** saveState() {

String category = (String) mCategory.getSelectedItem();

String summary = mTitleText.getText().toString();

String description = mBodyText.getText().toString();

*// only save if either summary or description*

*// is available*

**if** (description.length() == 0 && summary.length() == 0) {

**return**;

}

ContentValues values = **new** ContentValues();

values.put(TodoTable.COLUMN\_CATEGORY, category);

values.put(TodoTable.COLUMN\_SUMMARY, summary);

values.put(TodoTable.COLUMN\_DESCRIPTION, description);

**if** (todoUri == null) {

*// New todo*

todoUri = getContentResolver().insert(

MyTodoContentProvider.CONTENT\_URI, values);

} **else** {

*// Update todo*

getContentResolver().update(todoUri, values, null, null);

}

}

**private** **void** makeToast() {

Toast.makeText(TodoDetailActivity.this, "Please maintain a summary",

Toast.LENGTH\_LONG).show();

}

}

The resulting AndroidManifest.xml looks like the following.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="de.vogella.android.todos"

android:versionCode="1"

android:versionName="1.0" >

<uses-sdk android:minSdkVersion="15" />

<application

android:icon="@drawable/icon"

android:label="@string/app\_name" >

<activity

android:name=".TodosOverviewActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity

android:name=".TodoDetailActivity"

android:windowSoftInputMode="stateVisible|adjustResize" >

</activity>

<provider

android:name=".contentprovider.MyTodoContentProvider"

android:authorities="de.vogella.android.todos.contentprovider" >

</provider>

</application>

</manifest>

7.8. Start your application

Start your application. You should be able to enter a new todo item via the "Insert" button in the ActionBar.

An existing todo item can be deleted on the list via a long press.

To change an existing todo item, touch the corresponding row. This starts the second *activity*.

8. Accessing SQLite databases directly

8.1. Storage location of the SQLite database

SQlite stores the whole database in a file. If you have access to this file, you can work directly with the data base. Accessing the SQlite database file only works in the emulator or on a rooted device.

A standard Android device will not grant read-access to the database file.

8.2. Shell access to the database

It is possible to access an SQLite database on the emulator or a rooted device via the command line. For this use the following command to connect to the device.

adb shell

The command adb is located in your Android SDK installation folder in the "platform-tools" subfolder.

Afterwards you use the "cd" command to switch the database directory and use the "sqlite3" command to connect to a database. For example in my case:

# Switch to the data directory

cd /data/data

# Our application

cd de.vogella.android.todos

# Switch to the database dir

cd databases

# Check the content

ls

# Assuming that there is a todotable file

# connect to this table

sqlite3 todotable.db

The most important commands are:

| *Table 2. SQlite commands* | |
| --- | --- |
| **Command** | **Description** |
| .help | List all commands and options. |
| .exit | Exit the sqlite3 command. |
| .schema | Show the CREATE statements which were used to create the tables of the current database. |

You find the complete documentation of SQlite at <http://www.sqlite.org/sqlite.html>.

9. More on ListViews

Please see [Android ListView Tutorial](http://www.vogella.com/tutorials/AndroidListView/article.html) for an introduction into ListViews and ListActivities.

10. Performance

Changes in SQLite are ACID (atomic, consistent, isolated, durable). This means that every update, insert and delete operation is ACID. Unfortunately this requires some overhead in the database processing therefore you should wrap updates in the SQLite database in an transaction and commit this transaction after several operations. This can significantly improve performance.

The following code demonstrates that performance optimization.

db.beginTransaction();

**try** {

**for** (**int** i= 0; i< values.lenght; i++){

*// TODO prepare ContentValues object values*

db.insert(your\_table, null, values);

*// In case you do larger updates*

yieldIfContededSafely()

}

db.setTransactionSuccessful();

} **finally** {

db.endTransaction();

}

For larger data updates you should use the yieldIfContededSafely() method. SQLite locks the database during an transaction. With this call, Android checks if someone else queries the data and if finish automatically the transaction and opens a new one. This way the other process can access the data in between.

11. Get the Book

This tutorial is part of a book available in electronic form for your [Kindle](http://www.vogella.com/books/android.html).

13. Android SQLite resources

[SQlite website](http://www.sqlite.org/)

[SQL Tutorial](http://www.vogella.com/tutorials/SQL/article.html)

[SQLiteManager Eclipse Plug-in](http://coderzheaven.blogspot.de/2012/03/sqlitemanager-plugin-for-eclipse.html?m=1)