Tutorial 3: SQLite Databases

In this tutorial we will write a very simple application to demonstrate setting up, reading (from) and writing (to) *SQLite* databases. The database we will create is stored **locally** on the mobile. Such databases are still used nowadays, together with remote databases and web services. That however is a topic for Tutorial 4. After this tutorial you should be able to:

1. Define the structure of the database
2. Write data to tables
3. Read data from tables

Before proceeding create a new application with a basic activity called *MainActivity*. Next, add a form in the layout file, as well as the necessary string constants.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linear_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">

    <EditText
        android:id="@+id/to"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to_hint" />

    <EditText
        android:id="@+id/subject"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject_hint" />

    <EditText
        android:id="@+id/message"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/message_hint" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <Button
            android:id="@+id/show_all"
            android:layout_width="171dp"
            android:layout_height="wrap_content"
            android:layout_gravity="left"
            android:onClick="onClickShowAll"
            android:text="@string/show_all_text" />

        <Button
            android:id="@+id/send"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:layout_marginLeft="10dp"
            android:onClick="onClickSend"
            android:text="@string/send_text" />
    </LinearLayout>

</LinearLayout>
```

1. Defining the database

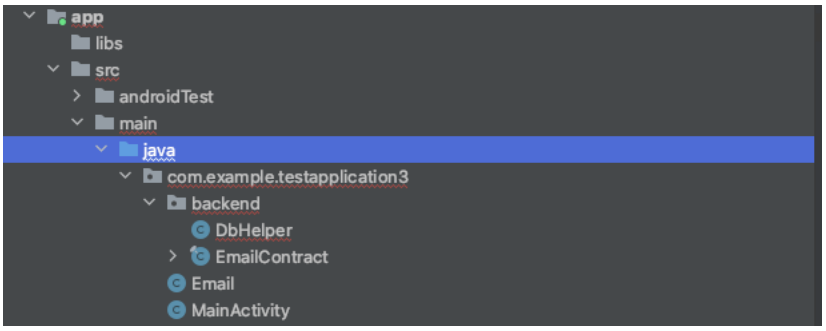
The first step is to add *SQLite* to our *gradle* dependencies. Add the following to *build.grade* under the app directory (not the one in the root) in the *dependencies* scope:

```
def sqlite_version = "2.3.0"

// Java language implementation
implementation "androidx.sqlite:sqlite:$sqlite_version"

// Implementation of the AndroidX SQLite interfaces via the Android
framework APIs.
implementation "androidx.sqlite:sqlite-framework:$sqlite_version"
```

Next create the following structure:



```
app
  libs
  src
    androidTest
    main
      java
        com.example.testapplication3
          backend
            DbHelper
            EmailContract
          Email
          MainActivity
```

The *Email* class will have four properties: id, to, subject, message – the properties we will write to our table. Add a constructor, getters and setters:

```
package com.example.testapplication3;

public class Email {
    private long id;
    private String to;
    private String subject;
    private String message;

    public Email(long id, String to, String subject, String message) {
        this.id = id;
        this.to = to;
        this.subject = subject;
        this.message = message;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Now we will focus on the *EmailContract* class. This class is used to define the table name, in this case “emails”, as well as the column names. It should look like this:

```
package com.example.testapplication3.backend;

import android.provider.BaseColumns;

public final class EmailContract {

    private EmailContract() {}

    /* Inner class that defines the table contents */
    public static class EmailEntry implements BaseColumns {
        public static final String TABLE_NAME = "emails";
        public static final String COLUMN_NAME_TO = "_to";
        public static final String COLUMN_NAME_SUBJECT = "_subject";
        public static final String COLUMN_NAME_MESSAGE = "_message";
    }
}
```

Next let’s write the *DbHelper* class. This class contains methods to create, upgrade and downgrade the database. Tables will then be created/dropped appropriately. In this class we can also write methods to insert emails into the database and fetch them. Normally databases have multiple tables, so it would make sense these methods are moved to subclasses for better management, for example, *EmailDbHelper.* In this simple example this is not required.

Note: To change the database schema or clear all the data, just increment the version.

The class should look as follows:

```
public class DbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database
version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "test.db";
    private final String _to = EmailContract.EmailEntry.COLUMN_NAME_TO;
    private final String _message =
EmailContract.EmailEntry.COLUMN_NAME_MESSAGE;
    private final String _subject =
EmailContract.EmailEntry.COLUMN_NAME_SUBJECT;

    public DbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(createTables());
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        db.execSQL(dropTables());
        onCreate(db);
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }

    private String createTables() {
        return "CREATE TABLE " + EmailContract.EmailEntry.TABLE_NAME + " ("
+
                EmailContract.EmailEntry._ID + " INTEGER PRIMARY KEY, " +
                _to + " varchar, " +
                _subject + " varchar, " +
                _message + " varchar)";
    }

    private String dropTables() {
        return "DROP TABLE IF EXISTS " +
EmailContract.EmailEntry.TABLE_NAME;
    }

    public long insertEmail(Email email) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();

        values.put(_to, email.getTo());
        values.put(_subject, email.getSubject());
        values.put(_message, email.getMessage());

        long id = db.insert(EmailContract.EmailEntry.TABLE_NAME, null,
values);

        return id;
    }

    public ArrayList<Email> getEmails() {
        ArrayList<Email> emails = new ArrayList<>();
        SQLiteDatabase db = this.getReadableDatabase();

        String[] projection = {
                BaseColumns._ID,
                _to,
                _subject,
                _message
        };

        // How you want the results sorted in the resulting Cursor
        String sortOrder = _subject + " ASC";

        Cursor cursor = db.query(
                EmailContract.EmailEntry.TABLE_NAME,   // The table to
query
                projection,             // The array of columns to return
(pass null to get all)
                null,              // The columns for the WHERE clause
                null,          // The values for the WHERE clause
                null,                   // don't group the rows
                null,                   // don't filter by row groups
                sortOrder               // The sort order
        );

        while(cursor.moveToNext()) {
            long id =
cursor.getLong(cursor.getColumnIndexOrThrow(BaseColumns._ID));
            String to =
cursor.getString(cursor.getColumnIndexOrThrow(_to));
            String subject =
cursor.getString(cursor.getColumnIndexOrThrow(_subject));
            String message =
cursor.getString(cursor.getColumnIndexOrThrow(_message));
            Email email = new Email(id, to, subject, message);
            emails.add(email);
        }
        cursor.close();

        return emails;
    }

    public Email getEmailById(long id) {
        SQLiteDatabase db = this.getReadableDatabase();

        String[] projection = {
                BaseColumns._ID,
                _to,
                _subject,
                _message
        };

        // Filter results WHERE "id" = condition
        String selection = BaseColumns._ID + " = ?";
        String[] selectionArgs = { Long.toString(id) };

        // How you want the results sorted in the resulting Cursor
        String sortOrder = _subject + " ASC";

        Cursor cursor = db.query(
                EmailContract.EmailEntry.TABLE_NAME,   // The table to
query
                projection,              // The array of columns to return
(pass null to get all)
                selection,               // The columns for the WHERE clause
                selectionArgs,           // The values for the WHERE clause
                null,                    // don't group the rows
                null,                    // don't filter by row groups
                sortOrder                // The sort order
        );

        Email email = null;

        while(cursor.moveToNext()) {
            String to =
cursor.getString(cursor.getColumnIndexOrThrow(_to));
cursor.getString(cursor.getColumnIndexOrThrow(_subject));
            String message =
cursor.getString(cursor.getColumnIndexOrThrow(_message));
            email = new Email(id, to, subject, message);
        }
        cursor.close();

        return email;
    }

}
```

2 + 3. Writing to and reading from the database

Next let's build the functionality to insert emails from the user's input. We've already created the form layout. Let's create the layout for the Confirmation screen (to be shown when Send is tapped). Add a new activity – *ConfirmActivity*. Apply the following 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="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/id_confirm"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <TextView
        android:id="@+id/to_confirm"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <TextView
        android:id="@+id/subject_confirm"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <TextView
        android:id="@+id/message_confirm"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

</LinearLayout>
```

Next let's read the user's input when Send button is tapped. In doing so, the email is written to the database, the new *id* is returned and sent to the *ConfirmActivity* via an *intent*. This activity will use this *id* to fetch the record from the database. In *MainActivity* add the following method:

```
public void onClickSend(View view) {
    DbHelper dbHelper = new DbHelper(this);

    EditText to = (EditText)findViewById(R.id.to);
    String toValue = to.getText().toString();
    EditText subject = (EditText)findViewById(R.id.subject);
    String subjectValue = subject.getText().toString();
    EditText message = (EditText)findViewById(R.id.message);
    String messageValue = message.getText().toString();

    // we use -1 as a dummy value as the id will be generated automatically
    Email email = new Email(-1, toValue, subjectValue, messageValue);
    // actual id is returned after insert is successful
    long id = dbHelper.insertEmail(email);

    Intent intent = new Intent(this, ConfirmActivity.class);
    intent.putExtra("ID", id);
    startActivity(intent);
}
```

In *ConfirmActivity* we'll first fetch the email record using the *id*, then populate the screen with its data:

```
public class ConfirmActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_confirm);
        populateScreen();
    }

    private void populateScreen() {
        DbHelper dbHelper = new DbHelper(this);
        Intent intent = getIntent();
        long id = intent.getLongExtra("ID", -1); //-1 is default value

        // gets email from db using id
        Email email = dbHelper.getEmailById(id);

        TextView idConfirm = (TextView) findViewById(R.id.id_confirm);
        idConfirm.setText(Long.toString(email.getId()));
        TextView toConfirm = (TextView) findViewById(R.id.to_confirm);
        toConfirm.setText(email.getTo());
        TextView subjectConfirm = (TextView)
findViewById(R.id.subject_confirm);
        subjectConfirm.setText(email.getSubject());
        TextView messageConfirm = (TextView)
findViewById(R.id.message_confirm);
        messageConfirm.setText(email.getMessage());
    }
}
```

At this point if you run the application, populate the form and click Send, you should be shown the details of the inputted email. This email is stored in the *Emails* table.

Next, let's show all the emails in the table. Create a new *ShowAllActivity*. In the layout file set the id of the *LinearLayout* to *emails_container*. We're going to use this *id* to programmatically add *TextViews* to this container.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/emails_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">

</LinearLayout>
```

In *MainActivity*, add the following method to start the *ShowAllActivity* when *Show All* is clicked:

```
public void onClickShowAll(View view) {
    Intent intent = new Intent(this, ShowAllActivity.class);
    startActivity(intent);
}
```

Finally, *ShowAllActivity* should read the emails from the database, and for each one it adds a simple *TextView* to show the fields.

```
public class ShowAllActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_show_all);
        populateScreen();
    }

    private void populateScreen() {
        DbHelper dbHelper = new DbHelper(this);
        ArrayList<Email> emails = dbHelper.getEmails();
        LinearLayout container = (LinearLayout)
findViewById(R.id.emails_container);

        for (Email email : emails) {
            TextView text = new TextView(this);
            text.setText(email.getTo() + " - " + email.getSubject() + " - "
+ email.getMessage());
            container.addView(text);
        }
    }
}
```

4. Challenge

Your challenge is to add the following functionality to the app:

1. In the *ShowAll* screen add a button Remove All. When clicked, the records the *Emails* table should be deleted, and the user is taken back to the *MainActivity*. To do this, it is suggested to add a method in *DbHelper* class – which should be invoked in an *onClickRemove* method.
2. Instead of showing the emails as a simple list, show them in a *RecyclerView* – as was done in Tutorial 2.