Data Storage - SQLite

Agenda

- Data storage options
- How to store data in key-value pairs
- How to store structured data in a relational database

Data Storage Options

- Shared Preferences
  - Store private primitive data in key-value pairs.
- Internal Storage
  - Store private data on the device memory.
- External Storage
  - Store public data on the shared external storage.
- SQLite Databases
  - Store structured data in a private database.
- Network Connection
  - Store data on the web with your own network server.

SQLite

What is SQLite?

- SQLite is a open source SQL database that stores data to a text file on a device
- Android comes in with built in SQLite database implementation
- SQLite supports all the relational database features like SQL syntax, transactions and prepared statements

SQLite Data Types

- NULL
  - The value is a NULL value.
- INTEGER
  - The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- REAL
  - The value is a floating point value, stored as an 8-byte IEEE floating point number.
- TEXT
  - The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- BLOB
  - The value is a blob of data, stored exactly as it was input

Database on Android

- SQLite
  - Requires limited memory at runtime (approx. 250 KByte)
    - Good candidate from being embedded into other runtimes
  - Supports data types TEXT, INTEGER and REAL
    - All other types must be converted into one of these fields before getting saved
    - SQLite does not validate if the types written to the columns are actually of the defined type

Table Name: Contacts

Field	Type	Key
id	INT	PRI
name	TEXT	
phone_number	TEXT	

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.
- The database is by default saved in the directory `DATA/data/APP_NAME/databases/FILENAME`.

SQLite in Android

- The `android.database` package contains all necessary classes for working with databases.
- The `android.database.sqlite` package contains the SQLite specific classes.
- To create and upgrade a database in your Android application you create a subclass of the `SQLiteOpenHelper` class

SQLiteOpenHelper

- To create and upgrade a database in app: create a subclass of SQLiteOpenHelper
  - In the constructor, call super() method of SQLiteOpenHelper, specifying database name and current database version
    - super(context, DATABASE_NAME, null, DATABASE_VERSION);

SQLiteOpenHelper

- A helper class to manage database creation and version management
- You create a subclass implementing
  - `onCreate(SQLiteDatabase)`
    - Called when the database is created for the first time
  - `onUpgrade(SQLiteDatabase, int, int)`
    - Called when the database needs to be upgraded

SQLiteOpenHelper

```
public class DatabaseHandler extends SQLiteOpenHelper {

    /* Constructor */
    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    /* Create table on OnCreate()*/
    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    /* Update table on OnUpgrade()*/
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {

    }

}
```

```java
public class Contact {

    private int id;
    private String name;
    private String phoneNumber;

    public Contact(){

    }

    public Contact(int id, String name, String phoneNumber){
        this.id = id;
        this.name = name;
        this.phoneNumber = phoneNumber;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPhoneNumber() {
        return phoneNumber;
    }
    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
```

SQLiteOpenHelper

```
public DatabaseHandler(Context context, String name, CursorFactory factory,
        int version) {
    super(context, name, factory, version);
}
```

Parameters	
context	to use to open or create the database
name	name of the database file, or null for an in-memory database
factory	to use for creating cursor objects, or null for the default
version	- number of the database (starting at 1) - if the database is older, onUpgrade(SQLiteDatabase, int, int) will be used to upgrade the database

Create and Upgrade Table

```
public class DatabaseHandler extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1; //-- Database Version
    private static final String DATABASE_NAME = "contactsDB"; //-- Database Name
    private static final String TABLE_CONTACTS = "tblContacts"; //-- Contacts table name
    private static final String COL_KEY_ID = "id"; //-- ID Column
    private static final String COL_NAME = "name"; //-- Name Column
    private static final String COL_PHONE = "phoneNumber"; //-- Phone Column

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
                + COL_KEY_ID + " INTEGER PRIMARY KEY," + COL_NAME + " TEXT,"
                + COL_PHONE + " TEXT" + ")";
        db.execSQL(CREATE_CONTACTS_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //-- drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);
        //-- create tables again
        onCreate(db);
    }
```

SQLiteOpenHelper Methods

- **`SQLiteDatabase` `getReadableDatabase()`**
  - Create and/or open a database for read
  - Returns
    - a database object valid until `getWritableDatabase()` or `close()` is called.
- **`SQLiteDatabase` `getWritableDatabase()`**
  - Create and/or open a database that will be used for reading and writing
  - Returns
    - a read/write database object valid until `close()` is called

SQLiteDatabase

- Exposes methods to manage a SQLite database.
- SQLiteDatabase has methods to
  - create
  - delete
  - execute SQL commands
  - perform other common database management tasks.

`void execSQL ()`

- Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data.

Parameters	
sql	the SQL statement to be executed. Multiple statements separated by semicolons are not supported

Cursor rawQuery ()

- Runs the provided SQL and returns a Cursor over the result set.

Parameters	
sql	the SQL query
selectionArgs	You may include ?s in where clause in the query, which will be replaced by the values from selectionArgs

Return
A Cursor object, which is positioned before the first entry

Query

- Cursor
  - A query returns a Cursor object
  - A Cursor represents the result of a query and points to one row of the query result
    - 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 getCount()
  - To move between individual data rows, use moveToFirst() and moveToNext()

rawQuery Example

```
public List<Contact> getAllContacts() {
    SQLiteDatabase db = this.getWritableDatabase();
    List<Contact> contactList = new ArrayList<Contact>();
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS; //-- select all query

    //-- run query on db
    Cursor cursor = db.rawQuery(
            selectQuery,   // the SQL query.
            null);

    //-- loop through all rows and adding to list
    if (cursor.moveToFirst()) { //-- Move the cursor to the first row
        do {
            Contact contact = new Contact();
            contact.setId(cursor.getInt(0));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));

            contactList.add(contact);  //-- add contact to list
        } while (cursor.moveToNext()); // Move the cursor to the next row
    }

    //-- close resources
    cursor.close();
    db.close();

    return contactList;  //-- return contact list
}
```

long insert ()

- Insert a row into the database

Parameters	
table	the table to insert the row into
nullColumnHack	optional; may be null
values	this map contains the initial column values for the row. The keys should be the column names and the values the column values

Return
the row ID of the newly inserted row, or -1 if an error occurred

`insert` Example

```
public long addContact(Contact contact) {
    // SQL Statement: INSERT tblContacts (name,phoneNumber) VALUES ('a','1')
    SQLiteDatabase db = this.getWritableDatabase();

    // Create ContentValues to pass insert method
    ContentValues values = new ContentValues();
    values.put(COL_NAME, contact.getName());
    values.put(COL_PHONE, contact.getPhoneNumber());

    //-- insert new contact row
    long result = db.insert(
            TABLE_CONTACTS, // the table to insert the row into
            null,
            values);      // this map contains the column values for the row.

    //-- close database connection
    db.close();

    return result;
}
```

Cursor query ()

- Query the given table, returning a Cursor over the result set.

Parameters	
table	The table name to compile the query against
columns	A list of which columns to return. Passing null will return all columns
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection

Cursor query ()

Parameters	
groupBy	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped
having	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself).
orderBy	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself).

Return
A Cursor object, which is positioned before the first entry

query Example

```
public Contact getContact(int id) {
    // SELECT id,name,phoneNumber FROM tblContacts
    SQLiteDatabase db = this.getReadableDatabase();
    Contact contact = new Contact();

    Cursor cursor = db.query(
            TABLE_CONTACTS,   //  The table name to compile the query against.
            new String[] { COL_KEY_ID, COL_NAME, COL_PHONE },   //  A list of columns
            COL_KEY_ID + "=?",    // SQL WHERE clause
            new String[] { String.valueOf(id) }, // selectionArgs for WHERE clause
            null,       // SQL GROUP BY
            null,       // SQL HAVING clause
            null);      // SQL ORDER BY clause

     if (cursor != null) { //-- Check cursor object for null
        cursor.moveToFirst();  //-- Move the cursor to the first row

        contact.setId(cursor.getInt(0));            //-- read id (0th col) from cursor
        contact.setName(cursor.getString(1));       //-- read name (1th col) from cursor
        contact.setPhoneNumber(cursor.getString(2)) //-- read phone (2th col) from cursor
    }

    cursor.close(); //-- close cursor
    db.close(); //-- close database connection

    return contact;
}
```

int update ()

- Update rows in the database.

Parameters	
table	the table to update in
values	a map from column names to new column values
whereClause	the optional WHERE clause to apply when updating. Passing null will update all rows
whereArgs	You may include ?s in the where clause, which will be replaced by the values from whereArgs

Return
the number of rows affected

update Example

```
public int updateContact(Contact contact) {
   SQLiteDatabase db = this.getWritableDatabase();

    // Create ContentValues to pass update method
    ContentValues values = new ContentValues();
    values.put(COL_NAME, contact.getName());
    values.put(COL_PHONE, contact.getPhoneNumber());

    int result = db.update(
            TABLE_CONTACTS,        // table
            values,                // values
            COL_KEY_ID + " = ?",   // whereClause
            new String[] { String.valueOf(contact.getId()) });  // whereArgs

    //-- close resources
    db.close();

    return result;
}
```

int delete ()

- Delete rows in the database

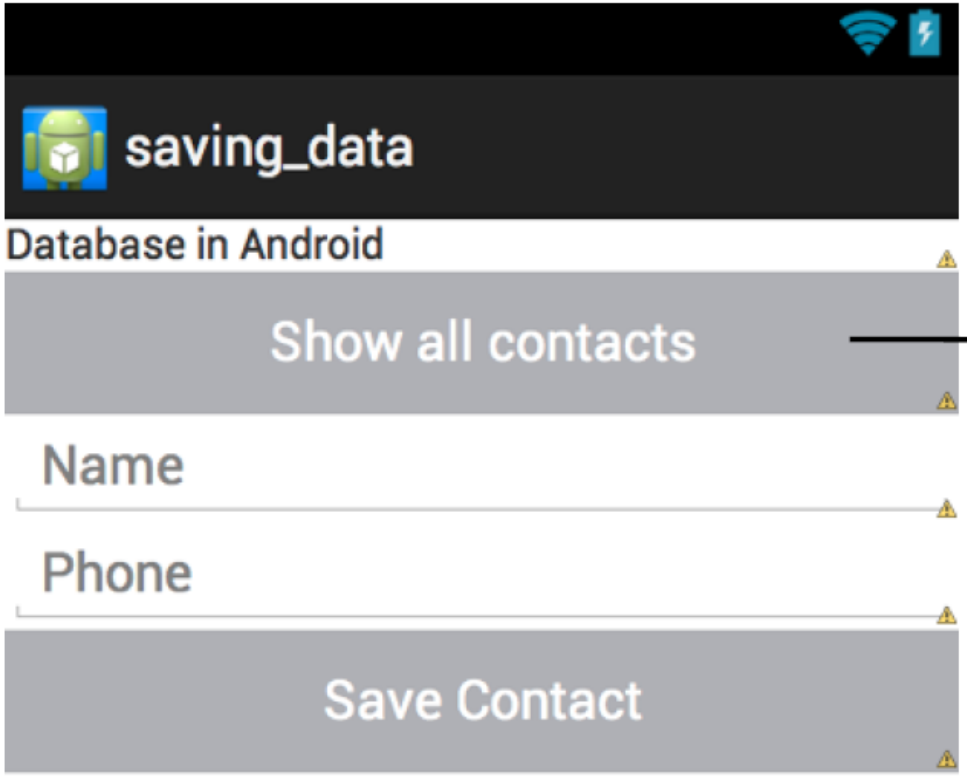
Parameters	
table	the table to delete from
whereClause	the optional WHERE clause to apply when deleting. Passing null will delete all rows
whereArgs	You may include ?s in the where clause, which will be replaced by the values from whereArgs

Return
the number of rows affected if a whereClause is passed in, 0 otherwise. To remove all rows and get a count pass "1" as the whereClause

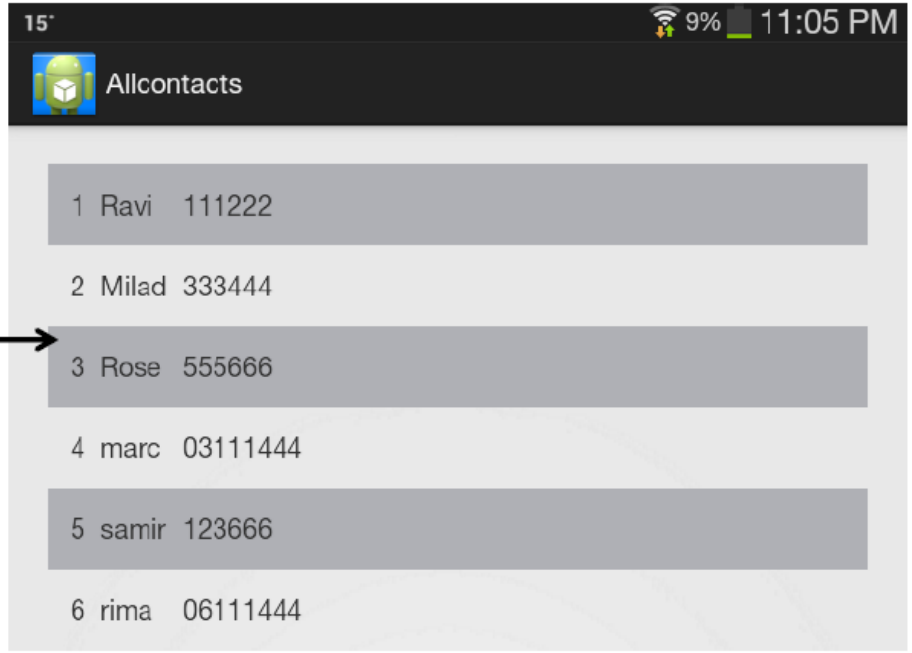
delete Example

```
public int deleteContact(Contact contact) {

    // SQL Statement: DELETE FROM tblContacts WHERE id = ?
    SQLiteDatabase db = this.getWritableDatabase();

    int result = db.delete(
            TABLE_CONTACTS,        // table
            COL_KEY_ID + " = ?",   // whereClause
            new String[] { String.valueOf(contact.getId()) }); // whereArgs

    //-- close resources
    db.close();

    return result;
}
```

Let's do this!



MainActivity



Allcontacts