### Chapter 4: Database Connectivity Using SQLite

**1. Using Android Data and Storage APIs**

Android provides various mechanisms for data storage depending on the app's needs and data sensitivity. Here are the main storage options:

* **Shared Preferences**: A simple key-value pair storage, mainly used for storing small amounts of primitive data such as user settings or preferences.
* **Internal Storage**: Storage specific to the app, inaccessible to other apps. Data is private to the app and stored in the internal memory of the device.
* **External Storage**: Storage that can be shared between apps, including SD cards or the device's shared storage space. It's suitable for large files like images or documents.
* **SQLite Database**: A lightweight database built into Android for structured data storage. It supports SQL syntax, making it suitable for relational data.
* **Content Providers**: A mechanism for sharing data between applications securely. Content Providers use a standard interface to access data in another app.

**2. Managing Data Using SQLite**

**SQLite** is a relational database management system built into Android. It allows developers to store, retrieve, update, and manage data in a structured format using SQL queries. Below are the essentials for using SQLite in Android:

**1. Creating and Managing SQLite Database**:

* To work with SQLite, you need to create a subclass of SQLiteOpenHelper. This class helps manage database creation and version management.

**Example**:

java

public class DatabaseHelper extends SQLiteOpenHelper {

// Database version and name

private static final int DATABASE\_VERSION = 1;

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

// Table and column names

private static final String TABLE\_STUDENT = "student";

private static final String COLUMN\_ID = "id";

private static final String COLUMN\_NAME = "name";

private static final String COLUMN\_SURNAME = "surname";

private static final String COLUMN\_MARK = "mark";

// Constructor

public DatabaseHelper(Context context) {

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

}

// Creating table

@Override

public void onCreate(SQLiteDatabase db) {

String CREATE\_TABLE = "CREATE TABLE " + TABLE\_STUDENT + "("

+ COLUMN\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"

+ COLUMN\_NAME + " TEXT,"

+ COLUMN\_SURNAME + " TEXT,"

+ COLUMN\_MARK + " INTEGER)";

db.execSQL(CREATE\_TABLE);

}

// Upgrading database

@Override

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

db.execSQL("DROP TABLE IF EXISTS " + TABLE\_STUDENT);

onCreate(db);

}

}

**2. Inserting Data into SQLite**: To insert data, you can use ContentValues to map column names to values. The insert() method of SQLiteDatabase is then used.

**Example**:

java

public void addStudent(String name, String surname, int mark) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(COLUMN\_NAME, name);

values.put(COLUMN\_SURNAME, surname);

values.put(COLUMN\_MARK, mark);

// Inserting Row

db.insert(TABLE\_STUDENT, null, values);

db.close();

}

**3. Querying Data from SQLite**: To retrieve data, you can use the query() method or rawQuery() with SQL statements.

**Example**:

java

public List<Student> getAllStudents() {

List<Student> studentList = new ArrayList<>();

String selectQuery = "SELECT \* FROM " + TABLE\_STUDENT;

SQLiteDatabase db = this.getReadableDatabase();

Cursor cursor = db.rawQuery(selectQuery, null);

// Looping through all rows and adding to list

if (cursor.moveToFirst()) {

do {

Student student = new Student();

student.setId(cursor.getInt(0));

student.setName(cursor.getString(1));

student.setSurname(cursor.getString(2));

student.setMark(cursor.getInt(3));

studentList.add(student);

} while (cursor.moveToNext());

}

cursor.close();

return studentList;

}

**4. Updating Data in SQLite**: To update data, the update() method is used, specifying a ContentValues object with new values and the condition for updating.

**Example**:

java

public int updateStudent(int id, String name, String surname, int mark) {

SQLiteDatabase db = this.getWritableDatabase();

ContentValues values = new ContentValues();

values.put(COLUMN\_NAME, name);

values.put(COLUMN\_SURNAME, surname);

values.put(COLUMN\_MARK, mark);

// Updating row

return db.update(TABLE\_STUDENT, values, COLUMN\_ID + " = ?", new String[]{String.valueOf(id)});

}

**5. Deleting Data in SQLite**: To delete data, the delete() method is used with a condition.

**Example**:

java

public void deleteStudent(int id) {

SQLiteDatabase db = this.getWritableDatabase();

db.delete(TABLE\_STUDENT, COLUMN\_ID + " = ?", new String[]{String.valueOf(id)});

db.close();

}

**3. Sharing Data Between Applications with Content Providers**

Content Providers manage access to a central repository of data. They are essential when you want to share data between apps while ensuring security and permissions.

**Key Concepts**:

* **URI** (Uniform Resource Identifier): Each Content Provider is identified by a unique URI that specifies the data location.
* **ContentResolver**: A class that provides access to content from a Content Provider. You use it to query, insert, update, or delete data.
* **Cursor**: A data structure returned by querying a Content Provider, containing the results.

**Example of Content Provider**:

java

public class StudentProvider extends ContentProvider {

private static final String AUTHORITY = "com.example.school.provider";

private static final String BASE\_PATH = "students";

public static final Uri CONTENT\_URI = Uri.parse("content://" + AUTHORITY + "/" + BASE\_PATH);

@Override

public boolean onCreate() {

// Initialize the database

return true;

}

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

// Query data from database

return null;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

// Insert data into database

return null;

}

@Override

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {

// Update data in database

return 0;

}

@Override

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

// Delete data from database

return 0;

}

}

**Permissions for Content Providers**:

* Apps accessing another app's Content Provider need permissions defined in the AndroidManifest.xml.
* Permissions control read and write access using <uses-permission> tags.

**Example**:

xml

<manifest ... >

<application ... >

<provider

android:name=".StudentProvider"

android:authorities="com.example.school.provider"

android:exported="true" />

</application>

</manifest>