Android Advance
Lesson 6
SQLite,
Content Provider

Outline

I. Introduction to SQLite

II. Working with Android SQLite Database

III. Introduction to Content Provider

IV. Working with Content Provider

I. Introduction to SQLite

1. What is SQLite?

- SQLite is an 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. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

II. Working with Android SQLite Database

1. Define a Schema and Contract

```
public final class FeedReaderContract {
  // To prevent someone from accidentally instantiating the contract class,
  // make the constructor private.
  private FeedReaderContract() {}
  /* Inner class that defines the table contents */
  public static class FeedEntry implements BaseColumns {
     public static final String TABLE_NAME = "entry";
     public static final String COLUMN_NAME_TITLE = "title";
     public static final String COLUMN_NAME_SUBTITLE = "subtitle";
  }
}
```

2. Create a Database Using a SQL Helper

```
public class FeedReaderDbHelper 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 = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

To access your database, instantiate your subclass of `SQLiteOpenHelper`:

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
```

3. Put Information into a Database

```
long insert (String table,
             String nullColumnHack,
             ContentValues values)
```

table	String: the table to insert the row into
nullColumnHack	String: optional; may be null. SQL doesn't allow inserting a completely empty row without naming at least one column name. If your provided values is empty, no column names are known and an empty row can't be inserted. If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your values is empty.
values	ContentValues: this map contains the initial column values for the row. The keys should be the column names and the values the column values

4. Read Information from a Database

table	String: The table name to compile the query against.
columns	String: A list of which columns to return.
selection	String: A filter declaring which rows to return
selectionArgs	String: You may include ?s in selection, which will be replaced by the values from selectionArgs
groupBy	String: A filter declaring how to group rows
having	String: A filter declare which row groups to include in the cursor
orderBy	String: How to order the rows
limit	String: Limits the number of rows returned by the query

5. Delete Information from a Database

```
int delete (String table,
            String whereClause,
            String[] whereArgs)
```

table	String: the table to delete from
whereClause	String: the optional WHERE clause to apply when deleting. Passing null will delete all rows
whereArgs	String: You may include ?s in the where clause, which will be replaced by the values from whereArgs. The values will be bound as Strings.
int	the number of rows affected if a whereClause is passed in, 0 otherwise.

6. Update a Database

```
int update (String table,
            ContentValues values,
            String whereClause,
            String[] whereArgs)
```

table	String: the table to update in
values	ContentValues: a map from column names to new column values. null is a valid value that will be translated to NULL.
whereClause	String: the optional WHERE clause to apply when updating. Passing null will update all rows.
whereArgs	String: You may include ?s in the where clause, which will be replaced by the values from whereArgs. The values will be bound as Strings.
int	the number of rows affected

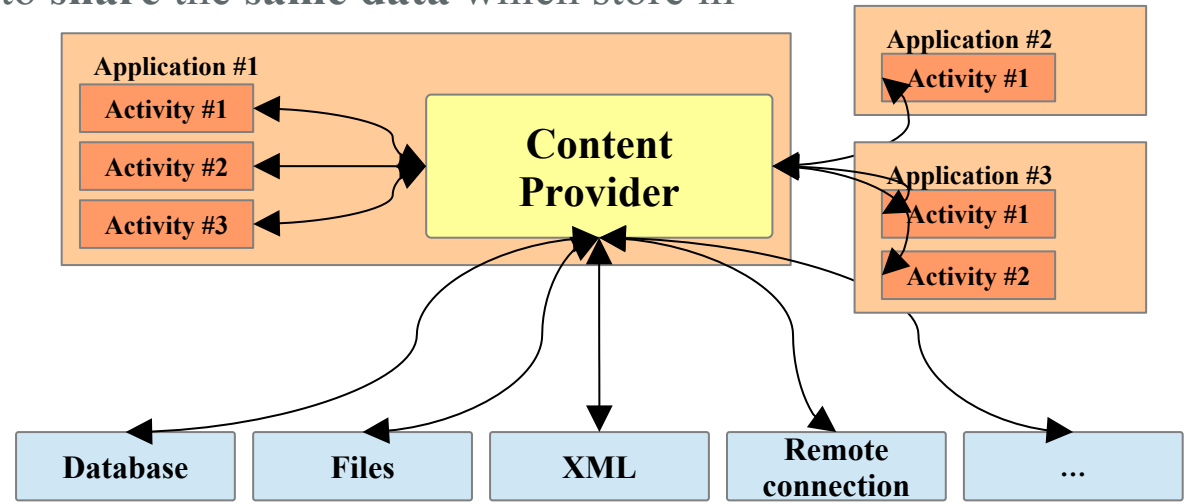
7. Sample

https://github.com/DoanVanToan/Sqlite_Demo

III. Introduction to Content Provider

1. What is content provider

- Content providers can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps. They encapsulate the data, and provide mechanisms for defining data security
- Allow multiple app to **share** the **same data** which store in
  - a database
  - Files
- **Sample**
  - Contacts
  - Calendar
  - Settings
  - Bookmarks
  - Media...



IV. Working with Content Provider

1. URIs for Content Providers

- **URI** - *Uniform Resource Identifier* include
  - Provider **name** (the **authority**)
  - **Path** to the content
  - **Id**

content://com.example.provider/articles/id

2. MIME types for Content Provider

- A MIME type is an Internet standard for definning types of content
- MIME types for Content Providers
  - type path ***vnd***
  - Subtype part:
    - If the URI pattern is for a single row: **android.cursor.item/**
    - If the URI pattern is for more than one row: a**ndroid.cursor.dir/**
  - Provider-specific part: **vnd**.<**name**>.<**type**>

Sample

if a provider's authority is **com.example.app.provider**, and it exposes a table named **table1**

- MIME type for multiple rows *vnd.android.cursor.**dir**/vnd.com.example.provider.table1*
- MIME type For a single row *vnd.android.cursor.**item**/vnd.com.example.provider.table1*

3. Add supporting methods to the database class

- Include some supporting method in database class to provide for query, update, delete operations
- Should not close the database connection after performing the operation

4. How to create a Content Provider

- Class for a content provider must inherit the ContentProvider class
- Override the onCreate method of ContentProvider class to initialize variable

Constructor/ Method	Description
UriMatcher(code)	Creates a new UriMatcher object and specifies the code to return when no matching URI is found
addURI(authority, path, code)	Adds a URI to the UriMatcher object and specifies the code to return when a matching URI is foind for this URI

5. How to provide for querying

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

Params	Description
uri	Uri: The URI to query. This will be the full URI sent by the client; if the client is requesting a specific record, the URI will end in a record number that the implementation should parse and add to a WHERE or HAVING clause, specifying that _id value. This value must never be null.
rpobjection	String: The list of columns to put into the cursor. If null all columns are included. This value may be null.
selection	String: A selection criteria to apply when filtering rows. If null then all rows are included. This value may be null.
selectionArgs	String: You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings. This value may be null.
sortOrder	String: How the rows in the cursor should be sorted. If null then the provider is free to define the sort order. This value may be null.

6. How to provide for insert rows

➢ Uri **insert** (Uri *uri*, ContentValues *values*)

- Return the URI for the **newly inserted** item. This value may be null.

Parrams	Description
uri	Uri: The content:// URI of the insertion request. This must not be null. This value must never be null.
values	ContentValues: A set of column_name/value pairs to add to the database. This must not be null. This value may be null.

7. How to provide for updating rows

- int **update** (Uri *uri*, ContentValues *values*, String *selection*, String[] *selectionArgs*)
  - Return the number of rows affected

Params	Description
uri	Uri: The URI to query. This can potentially have a record ID if this is an update request for a specific record. This value must never be null.
values	ContentValues: A set of column_name/value pairs to update in the database. This must not be null. This value may be null.
selection	String: An optional filter to match rows to update. This value may be null.
selectionArgs	String This value may be null.

8. How to provide for deleting rows

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

- Return the number of rows affected

Parrams	Description
uri	Uri: The full URI to query, including a row ID (if a specific record is requested). This value must never be null.
selection	String: An optional restriction to apply to rows when deleting. This value may be null.
selectionArgs	String This value may be null.

9. How to register Content Provider

➢ Define in AndroidManifest.xml file

```
<provider
        android:name=”com<name>ProviderName”
        android:authorities=”com.<name>”
        android:exported=”true”>
</provider>
```

Attribute	Description
name	Specifies the package and name of the content provider class
authorities	Specifies the authority for the content provider
exported	Specifies whether the content provider can be used by another app

10. How to use a custom Content Provider

```
public class TaskListProvider extends ContentProvider {

    public static final String AUTHORITY = "com.murach.tasklist.provider";

    public static final int NO_MATCH = -1;
    public static final int ALL_TASKS_URI = 0;
    public static final int SINGLE_TASK_URI = 1;

    private TaskListDB db;
    private UriMatcher uriMatcher;

    @Override
    public boolean onCreate() {
        db = new TaskListDB(getContext());

        uriMatcher = new UriMatcher(NO_MATCH);
        uriMatcher.addURI(AUTHORITY, "tasks", ALL_TASKS_URI);
        uriMatcher.addURI(AUTHORITY, "tasks/#", SINGLE_TASK_URI);

        return true;
    }
```

Q&A

Exercise

- Apply knowledge about Content Provider do List Contact App Load all contact from Local show in a List