Mobile Programming

[week 09]

[Database Connection]

Hilmy A. T.

hilmi.tawakal@gmail.com

Saving Data in SQL Databases

- Ideal for repeating or structured data, such as contact information.
- The APIs you'll need to use a database on Android are available in the:
  - android.database.sqlite package.
- This class assumes that you are familiar with SQL databases in general and helps you get started with SQLite databases on Android.

Using Databases

- Android provides full support for SQLite databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.
- The recommended method to create a new SQLite database is to create a subclass of SQLiteOpenHelper and override the onCreate() method, in which you can execute a SQLite command to create tables in the database.

Define a Schema and Contract

- One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized.
- The schema is reflected in the SQL statements that you use to create your database.
- You may find it helpful to create a companion class, known as a contract class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

Define a Schema and Contract

- A contract class is a container for constants that define names for URIs, tables, and columns.
- The contract class allows you to use the same constants across all the other classes in the same package.
- This lets you change a column name in one place and have it propagate throughout your code.

Define a Schema and Contract

- A good way to organize a contract class is to put definitions that are global to your whole database in the root level of the class. Then create an inner class for each table that enumerates its columns.

Example: Create Contract Class

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the contract class,
    // give it an empty constructor.
    public FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static abstract class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
        public static final String COLUMN_NAME_ENTRY_ID = "entryid";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";
        ...
    }
}
```

Example:Using Contract Class

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE + COMMA_SEP +
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +
    ... // Any other options for the CREATE command
    " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

Create a Database Using a SQL Helper

- Just like files that you save on the device's internal storage, Android stores your database in private disk space that's associated application. Your data is secure, because by default this area is not accessible to other applications.
- A useful set of APIs is available in the SQLiteOpenHelper class. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. All you need to do is call getWritableDatabase() or getReadableDatabase().

Create a Database Using a SQL Helper

- Because they can be long-running, be sure that you call getWritableDatabase() or getReadableDatabase() in a background thread, such as with AsyncTask or IntentService.
- To use SQLiteOpenHelper, create a subclass that overrides the onCreate(), onUpgrade() and onOpen() callback methods. You may also want to implement onDowngrade(), but it's not required.

Example: Create Database

```
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);
    }
}
```

Access Database

- To access your database, instantiate your subclass of SQLiteOpenHelper
- Example:

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

Put Information into a Database

- Insert data into the database by passing a ContentValues object to the insert() method
- Example:

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);

// Insert the new row, returning the primary key value of the new row
long newRowId;
newRowId = db.insert(
         FeedEntry.TABLE_NAME,
         FeedEntry.COLUMN_NAME_NULLABLE,
         values);
```

Read Information from a Database

- To read from a database, use the query() method, passing it your selection criteria and desired columns.
- The method combines elements of insert() and update(), except the column list defines the data you want to fetch, rather than the data to insert.
- The results of the query are returned to you in a Cursor object.

Read Information from a Database

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_UPDATED,
    ...
    };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
    FeedEntry.TABLE_NAME,  // The table to query
    projection,                               // The columns to return
    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
    );
```

Read Information from a Database

- To look at a row in the cursor, use one of the Cursor move methods, which you must always call before you begin reading values.
- Generally, you should start by calling moveToFirst(), which places the "read position" on the first entry in the results.
- For each row, you can read a column's value by calling one of the Cursor get methods, such as getString() or getLong().
- For each of the get methods, you must pass the index position of the column you desire, which you can get by calling getColumnIndex() or getColumnIndexOrThrow().

Read Information from a Database

Example:

```
cursor.moveToFirst();
long itemId = cursor.getLong(
    cursor.getColumnIndexOrThrow(FeedEntry._ID)
);
```

Delete Information from a Database

- To delete rows from a table, you need to provide selection criteria that identify the rows.
- The database API provides a mechanism for creating selection criteria that protects against SQL injection.
- The mechanism divides the selection specification into a selection clause and selection arguments.

Delete Information from a Database

- The clause defines the columns to look at, and also allows you to combine column tests.
- The arguments are values to test against that are bound into the clause.
- Because the result isn't handled the same as a regular SQL statement, it is immune to SQL injection.

Delete Information from a Database

Example:

```
// Define 'where' part of query.
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { String.valueOf(rowId) };
// Issue SQL statement.
db.delete(table_name, selection, selectionArgs);
```

Update a Database

When you need to modify a subset of your database values, use the update() method. Updating the table combines the content values syntax of insert() with the where syntax of delete().

Example:

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// New value for one column
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the ID
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
String[] selectionArgs = { String.valueOf(rowId) };

int count = db.update(
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

Question?