Android SQLite

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is "baked into" the Android runtime, so every Android application can create its own SQLite databases. Android SQLite native API is not JDBC, as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in **data/data//databases/** accessible from Android Device Monitor. SQLite is a typical **relational database**, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a **schema.**

Android SQLite SQLiteOpenHelper

Android has features available to handle changing database schemas, which mostly depend on using the SQLiteOpenHelper class. **SQLiteOpenHelper** is designed to get rid of two very common problems.

1. When the application runs the first time - At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.
2. When the application is upgraded to a newer schema - Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

SQLiteOpenHelper wraps up these logic to create and upgrade a database as per our specifications. For that we'll need to create a custom subclass of SQLiteOpenHelper implementing at least the following three methods.

1. **Constructor** : This takes the Context (e.g., an Activity), the name of the database, an optional cursor factory (we'll discuss this later), and an integer representing the version of the database schema you are using (typically starting from 1 and increment later).

```
public DatabaseHelper(Context context) {
super(context, DB_NAME, null, DB_VERSION);
}
```

2. **onCreate(SQLiteDatabase db)** : It's called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.

3. **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** : It's called when the schema version we need does not match the schema version of the database, It passes us a

SQLiteDatabase object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.

We define a DBManager class to perform all database CRUD(Create, Read, Update and Delete) operations.

Opening and Closing Android SQLite Database Connection
Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException {
dbHelper = new DatabaseHelper(context);
database = dbHelper.getWritableDatabase();
return this;
}
```

The **dbHelper** is an instance of the subclass of SQLiteOpenHelper. To close a database connection the following method is invoked.

```
public void close() {
dbHelper.close();
}
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc) {
ContentValues contentValue = new ContentValues();
contentValue.put(DatabaseHelper.SUBJECT, name);
contentValue.put(DatabaseHelper.DESC, desc);
database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);
}
```

Content Values creates an empty set of values using the given initial size. The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc) {
ContentValues contentValues = new ContentValues();
contentValues.put(DatabaseHelper.SUBJECT, name);
contentValues.put(DatabaseHelper.DESC, desc);
int i = database.update(DatabaseHelper.TABLE_NAME, contentValues, DatabaseHelper._ID +
" = " + _id, null);
return i; }
```

Android SQLite - Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id) {
database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);
}
```

SQLiteOpenHelper class

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.

Constructors of SQLiteOpenHelper class

Constructor	Description
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	creates an object for creating, opening and managing the database.
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)	creates an object for creating, opening and managing the database. It specifies the error handler.

Methods of SQLiteOpenHelper class

Method	Description
public abstract void onCreate(SQLiteDatabase db)	called only once when database is created for the first time.
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be upgraded.
public synchronized void close ()	closes the database object.
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be downgraded.

SQLiteDatabase class

It contains methods to be performed on sqlite database such as create, update, delete, select etc.

Methods of SQLiteDatabase class

Method	Description
void execSQL(String sql)	executes the sql query not select query.
long insert(String table, String nullColumnHack, ContentValues values)	inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
int update(String table, ContentValues values, String whereClause, String[] whereArgs)	updates a row.
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	returns a cursor over the resultset.