Module: Mobile Application development (Android)

Session 44: Instance State, SharedPreferences, SQLite DB and Cursors - Part 2 (Practice)

### This lab will be a continuation of session 43.

### Objectives

### At the end of this lab you will be expected to know:

### How to save & restore data as Application Preferences

### How to save & restore data as Instance State

### How to create an SQLiteDatabase

### How to manage database connections

### How to insert, update, remove, and retrieve data from an SQLite Database

### How to work with and manage Cursors

### How to use CursorAdapters

### 4. Databases & Using Persistent Data

### I strongly encourage you to read the sections of the Android Developer Guide on [Data Storage](http://developer.android.com/guide/topics/data/data-storage.html#db). In this fourth and final section you will be implementing the class that actually saves your Jokes to a database, making them persistent across multiple user sessions. The class is named the JokeDBAdapter class. JokeDBAdapter wraps all of the functionality necessary to save, update, delete, and retrieve Jokes. In addition to this, it also wraps the functionality necessary to create and update the Database itself and manage connections to it.

### You will now be tasked with implementing all of this functionality. The goal of this section of the lab is teach you how to work with an SQLiteDatabase and the general approach for wrapping data-persistence into a single class. The goal is not to teach you SQL. With this in mind, most of the actual SQL statements necessary to create and work with a Database have been written for you and placed into static-final-strings that you can use.

### 4.1 Creating an SQLiteDatabase & Managing Connections

### The JokeDBAdapter class provides an interface for performing common database operations like inserting data, updating data, removing data, and retrieving data. Additionally, it allows you to open a connection to the database as well. You can think of opening a database connection like opening a file. You can open a file for reading or you can open a file for writing. The JokeDBAdapter manages these connections as well.

### Also inside the JokeDBAdapter class is a static inner class called JokeDBHelper. This helper solves a problem that arises when trying to open a database connection. When you install the application and run it for the first time your database will not exist. You need to execute database creation code on this first run of the application before you can use the database and open a connection. However, all subsequent runs of the application don't need to execute this database creation code. A similar problem occurs if you need to upgrade your database to a new version.

### So how do you know whether your database already exists, or whether you need to execute the database creation code? The SQLiteOpenHelper abstract class solves this problem for you. By extending this class and implementing the abstract onCreate(...) and onUpgrade(...), you can test whether the database exists and conditionally execute creation or upgrade code before opening a connection. So, if you haven't guessed it by now, the static inner JokeDBHelper realizes the abstract SQLiteOpenHelper class.

### 4.1.1 Fill in the JokeDBAdapter.JokeDBHelper Static Inner Class

JokeDBAdapter will request database connections from JokeDBHelper. When a connection is requested, JokeDBHelper will test whether the database exists or needs to be updated, and take whatever actions are necessary before opening the connection. If the database needs to be created, then it will call its JokeDBHelper.onCreate(...) method. If the database needs to upgraded, it will call its JokeDBHelper.onUpgrade(...) method. For more details on this method see the Android Documentation on [SQLiteOpenHelper](http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html). You will now implement these methods.

* Fill in the **JokeDBAdapter.JokeDBHelper.onCreate(...)** method.
  + Execute the Database Creation SQL statement by calling the **execSQL(...)** method on the SQLiteDatabase parameter and passing in **JokeDBAdapter.DATABASE\_CREATE**.
    - *execSQL(...) will execute an SQL statement passed into it as a string. See the Android Documentation on* [*SQLiteDatabase.execSQL(...)*](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#execSQL(java.lang.String)) *for more details on this method.*
    - *JokeDBAdapter.DATABASE\_CREATE is a static final string that contains the Database Creation SQL statement. Go ahead and look at this SQL statement to see what the Joke database looks like.*
* Fill in the **JokeDBAdapter.JokeDBHelper.onUpgrade(...)** method.
  + *What you would normally do in this method is test which versions you are upgrading to and from. You could then conditionally modify the database to migrate it to the new schema. In our case, we will simply be deleting the existing database and creating the new one.*
  + Execute the Database Removal SQL statement by calling the **execSQL(...)** method on the SQLiteDatabase parameter and passing in **JokeDBAdapter.DATABASE\_DROP**.
  + Make a call to **onCreate(...)** to create the new database, passing in the SQLiteDatabase parameter from onUpgrade(...).

### 4.1.2 Fill JokeDBAdapter Constructor, Open, & Close

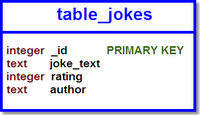
Now that the JokeDBHelper class has been established, JokeDBAdapter can use it to open database connections. JokeDBAdapter has a JokeDBHelper member variable aptly name m\_dbHelper. You will have to maintain this member variable via the constructor. JokeDBAdapter also has an SQLiteDatabase member variable named m\_db that represents your connection to the database. You will manage this connection via the open() and close() methods.

* In the JokeDBAdapter constructor:
  + Initialize m\_dbHelper to a new JokeDBHelper object.
    - Pass in the same application Context object that the JokeDBAdapter constructor received as a parameter.
    - Pass in the JokeDBAdapter.DATABASE\_NAME string constant as the name of the database.
    - Pass in null for the CursorFactory object. This tells SQLiteOpenHelper to use the default CursorFactory (*Don't worry about this)*.
    - Pass in the JokeDBAdapter.DATABASE\_VERSION integer constant as the database version number. SQLiteOpenHelper will compare this version number against the existing database's version number *(if it exists)* to determine whether JokeDBHelper.onUpdate(...) needs to be called.
* In JokeDBAdapter.open():
  + Retrieve a writable SQLiteDatabase object from m\_dbHelper.
    - Do this by calling getWritableDatabase() on m\_dbHelper.
    - *Alternatively, you could retrieve a read-only SQLiteDatabase by calling getReadableDatabase().*
  + set m\_db equal to the SQLiteDatabase you just retrieved.
  + *The reason why you don't open the database connection in the constructor is that you may initialize the JokeDBAdapter long before you ever need or want to use the database connection. You may also open and close the database multiple times to minimize the length of time in which the connection is open. Doing it this way allows you to explicitly open and close the database while only instantiating a single JokeDBAdapter object.*
* In JokeDBAdapter.close():
  + Close the SQliteDatabase, m\_db, by calling its close() method.

Try running your application to make sure that you don't get any errors. You won't be able to view, add, remove, or update jokes, but you shouldn't get any errors at this point.

### 4.2 Adding Data to an SQLiteDatabase You will now implement functionality for adding Jokes. You won't be able to test this until you finish section 4.3.1 which enables you to retrieve jokes, but it's coming up quick so sit tight. Functionality for adding Jokes to the database is encapsulated in the JokeDBAdapter.insertJoke(...) method which you used when updating your AdvancedJokeList.addJoke(...) method.

### Jokes are stored in the Database in a single table. The table has an attribute, or column, for each member variable of the Joke. Uniqueness of a Joke is determined by its ID member variable. This value is maintained by the database. When a Joke gets added to the database, the database generates a unique id for it. You can see what the table looks like in the diagram below:



### When inserting Jokes into the database you will use an android.content.ContentValues object. For complete background on this class see the Android Documentation on [ContentValues](http://developer.android.com/reference/android/content/ContentValues.html). Essentially, a ContentValues object is a key/value map in which you store each piece of the Joke using its corresponding database table attribute name, or column name, as the key. For example, you would store the Joke's rating value under a key of "rating".

### 4.2.1 Fill in insertJoke(Joke joke)

* Create a new ContentValues object.
* Put the text of the Joke in the ContentValues object using JokeDBAdapter.JOKE\_KEY\_TEXT as the key.
  + Use the ContentValues.Put(...) method.
* Put the rating of the Joke in the ContentValues object using JokeDBAdapter.JOKE\_KEY\_RATING as the key.
* Put the author of the Joke in the ContentValues object using JokeDBAdapter.JOKE\_KEY\_AUTHOR as the key.
* *Since this is a new Joke, the ID hasn't been assigned to it by the database yet, so you do not put the ID in the ContentValues object.*
* Insert the new Joke into the database by calling **insert(...)** on **m\_db**.
  + Pass into this method JokeDBAdapter.DATABASE\_TABLE\_JOKE as the table name to insert the joke into.
  + Pass in null as the "nullColHack" parameter *(Don't worry about this)*.
  + Pass in the ContentValues object you just created.
  + Read the Android Documentation on [SQLiteDatabase](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html) for a complete description of the method.
* The insert(...) method returns the ID of the newly inserted Joke, or -1 if the insert was unsuccessful. Make insertJoke(...) return this value.

### 4.3 Querying an SQLiteDatabase In order to retrieve jokes from the database, you must execute what is known as a query. A query tells the database to find specific rows in the database as well as which columns should be returned in the result of the query. The result of an SQLiteDatabase query is a Cursor object, which you used in previous changes you made to the JokeList Application. Another way to think of the Cursor is as a subset of the Joke database table. This subset may contain all, none, or some of the rows from the actual Joke database table. Additionally, it may contain all, none, or some of the columns from the Joke database table.

### The result of the query depends entirely on the query itself. To perform a query, you use the overloaded SQLiteDatabase.query(...) method. The simplest form of this method takes in the name of the table you want to query and an array containing the names of the columns for which you want information returned. It takes a number of other arguments as well, which have the option of being null. See the Android Documentation on [SQLiteDatabase](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html) for a complete description of this method.

### There is one other argument that this method takes that is of importance. This is the *selection* string argument. The *selection* argument allows you to tell the database to only return joke rows that meet a certain criteria. In particular, you can build a string to tell the database to only return joke rows that have a certain rating.

### 4.3.1 Fill in getAllJokes(String ratingFilter)

### This method returns all rows of Jokes in the database that have a rating equal to ratingFilter. However, if the ratingFilter parameter is null, this method should return every Joke row in the database with no exclusions.

* Create a local selection string variable.
* Test the ratingFilter parameter for null.
  + If null, then set your selection string to null.
    - *If the selection string is null, then all rows will be returned.*
  + Else, set your selection string to:

JOKE\_KEY\_RATING + "=" + ratingFilter;

* + - *This is evaluated by the database like an "if" statement. If the row in the database has a JOKE\_KEY\_RATING column value equal to rating filter it will be returned.*
* Call the following method on m\_db

#### SQLiteDatabase.query ([String](http://developer.android.com/reference/java/lang/String.html) table, [String[]](http://developer.android.com/reference/java/lang/String.html) columns, [String](http://developer.android.com/reference/java/lang/String.html) selection, [String[]](http://developer.android.com/reference/java/lang/String.html) selectionArgs, [String](http://developer.android.com/reference/java/lang/String.html) groupBy, [String](http://developer.android.com/reference/java/lang/String.html) having, [String](http://developer.android.com/reference/java/lang/String.html) orderBy)

* + Pass in JokeDBAdapter.DATABASE\_TABLE\_JOKE as the table name.
  + Pass in a string array containing the JokeDBAdapter.JOKE\_KEY\_ID, JokeDBAdapter.JOKE\_KEY\_TEXT, JokeDBAdapter.JOKE\_KEY\_RATING, and JokeDBAdapter.JOKE\_KEY\_AUTHOR column names.
  + Pass in your local selection string variable.
  + Pass in null for everything else.
* The query method will return a Cursor object containing the results of the query. Make getAllJokes(...) return the result of the query method.

### 4.3.2 Fill in getAllJokes()

### This is an overloaded version of the previous method that returns a cursor containing all Joke rows in the database. Fill in this method so that it returns the result of getAllJokes(null).

### 

### 4.3.3 Fill in getJokeFromCursor(Cursor cursor)

This static helper method is used to translate a Joke database table row into a Joke object. It should translate the row where the Cursor is currently pointing.

* First and foremost, there is no guarantee that the Cursor given to you as a parameter is not null and not empty. You should test for both of these conditions and return null in the event that either are true.
* Construct a new Joke object and initialize all of its member variables with data from the cursor.
  + To retrieve data from the Cursor, call the appropriate get method, passing in the Joke table column **index** corresponding to the member variable data that you want.
  + The column **indexes** are JOKE\_COL\_ID, JOKE\_COL\_TEXT, JOKE\_COL\_AUTHOR, JOKE\_COL\_RATING.
* Return the Joke object you just created.

You can now run your application. You should now be able to add Jokes and have them appear in your application. If you close the application and restart it, the Jokes that you previously entered should still be there.

### 4.3.4 Fill in getJoke(long id)

### This method retrieves a single Joke table row from the database. The row retrieved is the row whose JOKE\_KEY\_ID column value equals the id parameter passed in. This is guaranteed to return one row since no two rows share the same JOKE\_KEY\_ID column value. This method should return a Joke object constructed from the row returned by the query.

### Fill in this method on your own.

### Execute a query using a selection string like you did for the getAllJokes(String ratingFilter) method.

### NOTE: The Cursor that is returned from query(...) is positioned *before* the first result/row. You must call cursor.moveToNext() before using the cursor or you will get an IndexOutOfBoundsException for trying to use a cursor positioned at -1.

### Construct the Joke object to return like you did in getJokeFromCursor(Cursor cursor).

### After filling this method in, you should be able to upload jokes to the Server, which is the only piece of functionality that uses this method.

### 

### 4.4 Editing Data in an SQLiteDatabase

### You are almost done. You only need to implement two more pieces of functionality. You need to be able to update the contents of a Joke in the database and you need to be able to remove a Joke from the Database.

### 

### 4.4.1 Fill in removeJoke(Joke joke)

The remove method operates similarly to the getJoke(...) method in that your goal is to identify a single Joke row in the database.

* Create a selection string to identify the Joke row in the database whose JOKE\_KEY\_ID column value matches the id of the Joke parameter passed into removeJoke(...).
* Call the **delete(...)** method on m\_db.
  + Pass in DATABASE\_TABLE\_JOKE as the table name.
  + Pass in your selection string.
  + Pass in null for the final argument.
* The delete(...) method returns a count of the number of rows that were deleted as a result of making this call. If the result is greater than 0, removeJoke(...) should return true, otherwise it should return false.
* Run your application. You should now be able to permanently remove Jokes from the database via the Remove Context MenuItem.

### 4.4.2 Fill in updateJoke(Joke joke)

### The update method should identify a single Joke row in the database and replace all of its column values with the values in the Joke parameter passed into updateJoke(...).

### Create a ContentValues object and place the contents of the Joke parameter into it like you did for the insertJoke(...) method.

### Do not put the ID value into your ContentValues object. You do not want to change this.

### Create a selection string to identify the Joke row in the database whose JOKE\_KEY\_ID column value matches the id of the Joke parameter.

### Call the update(...) method on m\_db.

### Pass in DATABASE\_TABLE\_JOKE as the table name.

### Pass in your ContentValues object.

### Pass in your selection string.

### Pass in null for the final argument.

### The update(...) method returns a count of the number of rows that were updated as a result of making this call. If the result is greater than 0, updateJoke(...) should return true, otherwise it should return false.

### Run your application. Changes to Joke ratings should now be persistant.

### 5. Deliverables

### Put your entire project directory into a .zip file and submit the archive. This effectively provides time-stamped evidence that you submitted the lab on time. The name of your archive should be lab4<yourname>.zip. So if your username is RAM and you created a zip file, then your file would be named lab4RAM.zip.