Android programming: Flipped Classroom: SQL

For this assignment, you have an app that will search a database that contains restaurant information. We are giving you a database that we populated by scraping yelp. That database file is in the assets directory. You can find it by looking for restaurants.db. You can run the command line program `sqlite3` on the database file to find out what tables it has and the contents of those tables. Your app will read this database and display its contents in various ways. You never update the database.

Behavior.

- When you click the "Go" button without changing the default spinners, the app will return all restaurants from its table. Try to get this functionality working first, because it involves making a very simple query on the database object. By selecting a cuisine type and/or city, the app will show restaurants with the specific type and/or located in the specified city. If you select the "# limit?" spinner, the app will restrict the number of restaurants returned to the specified number. The limit spinner is maybe the second piece of functionality to implement, because it is conceptually simple and easily verified. Finally by checking the "price," box, the app will display its result ordered by price in either ascending or descending order (the "up/down?" spinner).

- The result of the `SQLiteDatabase.query` call is a Cursor. You need to set that Cursor as the data source for the RestaurantAdapter. Look up a function to do that. That is all you need to do for the display of the listview (yes, it is a ListView, but you don't need to know much about it). To find the right function, look at the documentation for `CursorAdapter`. There are two candidate functions, so google the difference between them.

- The app will show the name, address, phone number, website and price for each restaurant. Price is an integer in the range of 0 to 4 (inclusive). The app will show the price in unary using the "$" character, e.g. "$$" represents a price value of 2. This is done in `RestaurantItemAdapter.bindView`. This bindView is like any other bindView, it takes a view and some information and puts the information in the view. Work on bindView either first or after you have an initial query so you can see the results of queries.

CursorAdapter.

- This FC introduces you to a CursorAdapter, which is like the list adapters we have been using, but a little different. With list adapters, the adapter's job is to mediate between a list and the recyclerview. We would create an adapter with a particular list, and then if the list changes, we could call `submitList` (this function applies primarily to adapters derived from `ListAdapter` which we saw in `ListViewFrag`).

- The same kind of thing is going on here with your restaurantAdapter. But now Cursors are playing the role of lists. They are the data structure your adapter is adapting for the view. We create the adapter with a Cursor, and then there is a function to update the adapter with a new Cursor when the user does a new query. Don't create a new adapter for each query. The function to update the adapter with a new Cursor is not called `submitList`, but if you look at the documentation for `CursorAdapter` (which is the base class for `RestaurantItemAdapter`) you will find the right function. (As we say above, there are two candidate functions, you should google the difference between them).

Understanding the database. Below is the schema of the tables you will use. You can get it from running `sqlite3` on your database file and typing ".schema". In the business table, each business has an _id, which is the primary key for the table. Each row has attributes of the business, like its web URL and its phone number. A schema is the names of the columns and their types.

The categories table has a reference to the business _id as a foreign key. That allows a single business to be in multiple categories. But if a business is deleted, its entries in the categories table are removed, because they no longer refer to a valid business _id. Note how the the primary key is specified. **Please:** explain what that notation ("primary key (_id, category_name)") means and why the primary key is what it is all in one sentence in your README.

Rules and submission Recall the rules for flipped classrooms: do the work alone or with a partner that is unique for the semester. You can find the github link on piazza.

Only one student needs to submit this code, but both students may do so if they wish. This assignment should be submitted through github (sorry, partners can't share the repository). Include a README at the top directory level that contains both student's EIDs.

```
CREATE TABLE businesses
            (_id INTEGER PRIMARY KEY,
            name TEXT,
            full_address TEXT,
            phone TEXT,
            rating DECIMAL(1,1),
            review_count INTEGER,
            price INTEGER,
            url TEXT,
            city TEXT);
CREATE TABLE categories
            (_id INTEGER,
            category_name text,
            primary key (_id, category_name)
            FOREIGN KEY(_id) REFERENCES businesses(_id) ON DELETE CASCADE
            );
```

Consider these files.

- **AndroidManifest.xml** You do not need to change this file.
- **DatabaseHelper.kt** You do not need to change this file.
- **MainActivity.kt** The notation `XXX Write Me` indicates code that you should write. Don't change `onCreate` and don't change `handleRestaurant`. `handleRestaurant` is provided as an example for how to generate the select clause.

  Take a look at `onCreate`. First it does the normal content view stuff. Then it creates a bunch of spinners, which provide the drop down menus in the app, like the limit dropdown that has values 10, 20, and 50. It creates these spinners from the values in XML resource files (see the directory res/values). It also loads these resource arrays into memory using lazy initialization (see the top of the class, with values like variables like `cities` that is initialized from `R.array.city`.

  Notice that the `queryButton`'s on click listener is `::doQuery`.

  `onCreate` then creates a DatabaseHelper and a `RestaurantItemAdapter`. But it creates an adapter with a cursor that generates an empty list. `defaultBadCursor` is named that way because I don't want you to pay too much attention to it. It is there because we need a valid Cursor object to create the `RestaurantItemAdapter`.

  doQuery. Look at this function. Its ultimate goal is to call `restaurantDb.query`, which is a complex function with many string parameters. The first part of doing a valid query is to create the selection and its arguments. This is done by `handleRestaurant`, which is given to you, and `handleCity`, which you must write. Note that the parameters to these functions are Mutable.

  We then provide the code to create the selection string and to create an array of argument strings. We also log these, so you can look at them to see if you are on the right track. Then you need to construct the order by string and the limit string before you can all query and then update the `restaurantAdapter`.
- **RestaurantItemAdapter.kt** This class contains code that creates a view for list items using the result returned from the database. Fill in the missing parts indicated by `XXX Write Me`.
- **activity_main.xml** No changes.

Hints.

- Use `SQLiteDatabase.query`, don't use `rawQuery`.
- Start with the simplest query you can to print all the restaurants. You can pass null for a bunch of the arguments. Then add a limit clause. Then either an OrderBy clause or select a particular city. Finally, combine everything including cuisine types.
- Do not use database views.
- In `RestaurantItemAdapter.bindView` a Cursor is like a pointer to a single item. Use it to get the data for a single row. Use column names to "index" the cursor, don't use integer position because that is fragile (the columns might get reordered for some queries).
- We use the function `toTypedArray()` to go from an `ArrayList<String>` to an `Array<String>`. `https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/to-typed-array.html`