Mobile Computing

Practice # 2d

Android Applications – Local DB

In this installment we will add persistent storage to the restaurants' application.

For that, we will create a database with a table for holding our restaurant data and switch from our **ArrayAdapter** to a **CursorAdapter**, to make use of that database. This will allow our restaurants to persist in every execution of the **LunchList** application.

1. Create a class to make the interface to a *SQLite database*, which is the SQL engine available on the Android library. We need to be able to define what our database name is, what is the schema for the table storing our restaurants, etc. These definitions should be wrapped up in a **SQLiteOpenHelper** object implementation that can open or create a database (or upgrade its version if already exists in an older form).
   a. Create a new class in a file *RestaurantsHelper.kt* extending **SQLiteOpenHelper**, defining some constants, and overriding **onCreate()** and **onUpgrade()**, as follows:

```
const DATABASE_NAME = “lunchlist.db”
const SCHEMA_VERSION = 1

class RestaurantsHelper(val ctx: Context) : SQLiteOpenHelper(ctx, DATBASE_NAME, null,
                                                        SCHEMA_VERSION) {
    override onCreate(db: SQLiteDatabase) {
    }

    override onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    }
}
```

   This says that our database is in file *lunchlist.db* and this is the first version of the schema. This version of the schema should be created in the **onCreate()** method. In **onUpgrade()** we should put code needed to convert a database and schema from an **oldVersion** to a **newVersion**. Such an upgrade, when needed, of course implies to copy all the data to some intermediate tables, fix up the tables to the new schema and copy back the data, before deleting the intermediate tables. But for now we stick with the 1st version and the SQL '**create table**' statement.

```
override onCreate(db: SQLiteDatabase) {
   db.execSQL("CREATE TABLE Restaurants (_id INTEGER PRIMARY KEY AUTOINCREMENT, ” +
                  “name TEXT, address TEXT, type TEXT, notes TEXT)")
}

override onUpgrade(db: SQLiteDatabase db, oldVersion: Int, newVersion: Int) {
   // no-op, since it will not be called until we need to define a 2nd schema version
}
```

   b. We will be using **RestaurantsHelper** as our bridge to the database, and we will not use the **Application** object anymore for sharing data.
   So, delete the **LunchApp** class and all references to it in **Main** and **Details** activities.
   Also let's get some access, in both those classes, to the **RestaurantsHelper**, in the following way:
   - Declare a **RestaurantsHelper** (dbHelper) variable in both classes.
   - Initialize in the activity classes a helper object, using the Kotlin lazy delegation (the property

will get its initial value only when it is first used):

```
val dbHelper by lazy { RestaurantsHelper(this) }
```

- Also add an override to **onDestroy()** in both classes with:

```
super.onDestroy();
dbHelper.close();
```

c. We are going to be replacing our restaurant object model restaurants (and its associated **ArrayList**) with the database and a Cursor representing the list of restaurants. This will involve adding some more logic to **RestaurantsHelper** to aid in this process.

First add an **insert()** and an **update()** methods:

```
fun insert(name: String, address: String, type: String, notes: String): Long {
  val cv = ContentValues();
  cv.put("name", name);
  cv.put("address", address);
  cv.put("type", type);
  cv.put("notes", notes);
  return writableDatabase.insert("Restaurants", "name", cv);
}

fun update(id: String, name: String, address: String, type: String, notes: String) {
  val cv = ContentValues();
  val args = arrayOf(id);
  cv.put("name", name);
  cv.put("address", address);
  cv.put("type", type);
  cv.put("notes", notes);
  writableDatabase.update("Restaurants", cv, "_id=?", args);
}
```

These methods should be called from the save button listener when a new restaurant is created or modified. Replace the listener in the Details class, after the switch statement, to contain:

```
  if (rId == null)
   currentId = dbHelper.insert(rName, rAddress, rType, rNotes)
  else
   dbHelper.update(rId!!, rName, rAddress, rType, rNotes)
  finish()
```

Notice the use of the non-null assertion operator (!!)

d. We need also to query the database and put the result in a cursor for all the restaurants and also for a single restaurant, given its _id. For that we need two more methods in our **RestaurantsHelper** class, as well as some other methods to retrieve the individual pieces of data out of a cursor. These new methods should be as follows:

```
 fun getAll(): Cursor {
  return readableDatabase.rawQuery(
   "SELECT _id, name, address, type, notes FROM Restaurants ORDER BY name", null)
 }
```

```
fun getById(id: String): Cursor {
  val args = arrayOf(id)
  return readableDatabase.rawQuery(
    "SELECT _id, name, address, type, notes FROM Restaurants WHERE _id=?", args)
}

fun getName(c: Cursor): String {
  return c.getString(1)
}

fun getAddress(c: Cursor): String {
  return c.getString(2)
}

fun getType(c: Cursor): String {
  return c.getString(3)
}

fun getNotes(c: Cursor): String {
  return c.getString(4)
}
```

e. Declare a new String property rId, eliminate the rPos property, and replace the data from the intent, and the last if (…) in **onCreate()**, and also the **load()** method of the **DetailsActivity** class, by the following:

```
    private val rId: String? by lazy { intent.getStringExtra(ID_EXTRA) }
      . . .
        if (rId != null) load()
      . . .

  private fun load() {
    val c = dbHelper.getById(rId!!)
    c.moveToFirst()
    edName.setText(dbHelper.getName(c))
    edAddress.setText(dbHelper.getAddress(c))
    edNotes.setText(dbHelper.getNotes(c))
    when (dbHelper.getType(c)) {
      "sit" -> rgTypes.check(R.id.sit)
      "take" -> rgTypes.check(R.id.take)
      "delivery" -> rgTypes.check(R.id.delivery)
    }
    c.close()
  }
```

Now the **Details** activity expects to be invoked from the **Main** with an **Intent** transporting the _id of the selected restaurant, or nothing if we want to add a new restaurant (from the Main menu).

f. On the Main activity replace now the **onRestItemClick()** listener to send the current restaurant id and start the Details activity:

```
private fun onRestItemClick(id: Long) {
  currentId = id
  startActivity(Intent(this, DetailsActivity::class.java).putExtra(ID_EXTRA, id.toString()))
 }
```

We need a new property in the **MainActivity** class to store the current id, like:

```
var currentId: Long = -1L
```

and correct the toast listener in **onOptionsItemSelected()** using:

```
 override onOptionsItemSelected(item: MenuItem) {
  . . .
    if (currentId != -1L) {
     val c = dbHelper.getById(currentId.toString())
     c.moveToNext()
     message = String.format("%s:\n%s", dbHelper.getName(c), dbHelper.getNotes(c))
     c.close()
    }
  . . .
 }
```

g. Next we need to replace, in the **Main** activity, our model containing the restaurants by a **Cursor**, and make our **RestaurantAdapter** a cursor adapter:

   Replace the initialization in the **onCreate()** method (see that the model is initialized with all restaurants available in the database). The call to **startManagingCursor()** allows the activity to retrieve automatically again the model cursor, in the case it has to be recreated.

```
  val restaurantsCursor = dbHelper.getAll()
  startManagingCursor(restaurantsCursor)
  . . .
  list.adapter = RestaurantAdapter(restaurantsCursor)
```

   Finally we have to adapt the **RestaurantAdapter** to be a **CursorAdapter**:

```
 inner class RestaurantAdapter(c: Cursor) : CursorAdapter(this@MainActivity, c, true) {
  override fun newView(ctx: Context, c: Cursor, parent: ViewGroup): View {
   val row: View = layoutInflater.inflate(R.layout.row, parent, false)
   row.findViewById<TextView>(R.id.title).text = dbHelper.getName(c)
   row.findViewById<TextView>(R.id.address).text = dbHelper.getAddress(c)
   val symbol = row.findViewById<ImageView>(R.id.symbol)
   when (dbHelper.getType(c)) {
    "sit" -> symbol.setImageResource(R.drawable.ball_red)
    "take" -> symbol.setImageResource(R.drawable.ball_yellow)
    "delivery" -> symbol.setImageResource(R.drawable.ball_green)
   }
   return row
  }

  override fun bindView(v: View, ctx: Context, c: Cursor) {}
 }
```

h. Remove the **Restaurant** class, also the manifest reference to **LunchApp**, compile, install and use.