Lab.04 Database in Android

1. Creating a database

1. Create a new project *Lab04.Surname* with the *dbTestActivity* activity.
2. First Let's create a database *Notes* with a single table and three fields:

   a) **Id**: integer, primary key and autoincremental

   b) **Name** and **Last name**: Both are nonzero texts

   c) **Note**: integer, not null

   For convenience, create the class of the data model (*Notes.java*) with the three attributes mentioned. Define all your set and get methods, and also their constructors [1].
3. Second, create the Data Handler: *NotesDataBaseHelper.java* that *extends SQLiteOpenHelper* which will be responsible for creating and operating the database (see the theory).

   a) Define static variables with the names of each column and each table. E.g. `COL_ID`, `COL_NOMBRE`, `COL_NOTA`, `TABLE_NOTES`, `NOTES_ALL_COL`, ...

   b) Construct the String that "creates" the database: *DATABASE_CREATE.*

   c) Implement the constructor and *onCreate()* and *onUpgrade()* methods (see the theory).

   d) Define the method to insert a record: *insertNote(Notes Notes)*

   e) Define methods to delete or update a record: *deleteNote* y *updateNote*

   f) Define the methods needed to recover the data (all or part). Use the function `getWritableDatabase().query(...)`
4. Third and last, create the presentation layer.

   a) Add to the layout a ListView.

   b) Declare in the onCreate() `dbNotas = new NotesDataBaseHelper(this);`

   c) Define two items in the menu (ActionBar): *menu_add* and *menu_update.*

   d) On *onOptionsItemSelected* identify each item/action and execute each task: **update_list** and **new_item** (see following sections).
5. **new_item**: execute `startActivity (new Intent(this, NewNote.class))`

   a) Create a new activity *NewNote* to manage the data entry.

   b) Associate a layout *new_nota.xml*: p.e. a LinearLayout with 3 rows (one for each field of the table, except the id), each row with a TextView and an EditText.

   c) At the end, add two buttons (*Add* and *Modify*). Make this last button invisible.

   d) In the *Add* handler include the code that inserts a new record into the database. **Important**: Check that all fields are well covered.
6. **update_list**: query and `update_list(dbNotes.getNotes(null));`

   a) Create a layout *list_row.xml* adapted to our data.

[1] Note: *Code* → *Generate* → {*Getter and Setter*, *Constructor*}

2. Operating a database

7. Associate the ListView with a ContextMenu with two options to edit and delete elements and that are collected in the corresponding handler:
   a) Delete selected item. Do not forget to update the ListView.
   b) To edit the selected item, send its *id* as a parameter to *NewNote*.
      1) In *NewNote* collect that *id* and get the data to initialize the EditTexts.
      2) In the layout of *NewNote* make the *Modify* button visible. That way there are two options: modify the current record or add a new one (based on current and/or modified data).
8. Include in the menu (ActionBar) two new actions: filter and sort.
   a) Implement them with each AlertDialog whose layout must have at least:
      1) An EditText to enter the reference text.
      2) Two Spinner to select the reference column and the sorting option.
      3) The button(s) for the actions to be performed.
   b) Save to *Preferences* for subsequent executions (as default values):
      1) The last reference column selected by the user.
      2) The selected sorting option (ASC or DESC).

3. Use a Content Provider

We are going to use the dictionary Content Provider (*UserDictionary.Words.CONTENT_URI*). **Note: only valid for API < 23.**

1. Create new activity *cpTestActivity* to be executed from the action *cpTest* on ActionBar.
   a) Edit the layout of this activity to have three lines.
      1) In the first a *but_search* button and a *word* EditText.
      2) In the second one a *lv* ListView.
      3) On the third line 3 buttons: Insert, Delete, Modify.
   b) Include for each button the code shown in the theory.
      - Actions are applied to **all** words that meet the search condition.
      - Check before each action that *word* is not empty.
      - Search must be "Like" and in ascending order.
      - After each action, always refresh the ListView.
   c) In the ContextMenu include 2 options: delete and edit.
      1) The *id* is extracted in the handler *onContextItemSelected*: `(int)info.id` where `info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();`
      2) After deleting, the list displayed in the ListView should be updated.