B53 – Android – SQLite

SQLite est intégrée au moteur d’exécution d’Android.

Il s’agit d’un SGBD très apprécié car il fournit une interface SQL tout en nécessitant un minimum de mémoire et en fournissant une rapidité satisfaisante.

Il est à noter que son API n’est pas JDBC (vu en B43), car celui-ci est trop lourd pour les terminaux limités en mémoire comme les téléphones. Cependant on peut faire de nombreux parallèles avec les méthodes employées avec JDBC.

Pour créer une base de données SQLite, on doit :

1. Créer notre DatabaseHelper, enfant de SQLiteOpenHelper (permet de créer nos tables)
2. Créer notre Model qui contiendra les requêtes

On peut ensuite interroger la base de données via nos activités.

## Création d’une liste de classes/locaux du CVM

### Création du DatabaseHelper

**public** **class** DatabaseHelper **extends** SQLiteOpenHelper {

 **private** **static** **final** String *DATBASE\_NAME* = "DB\_Locaux\_CVM";

 **private** **static** **final** **int** *DATABASE\_VERSION* = 1;

 **public** DatabaseHelper(Context context) {

 **super**(context, *DATBASE\_NAME*, **null**, *DATABASE\_VERSION*);

 }

 @Override

 **public** **void** onCreate(SQLiteDatabase db) {

 // Créer la table ici

 }

 @Override

 **public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {

 // Que faire lorsque l'on change de version

 }

}

### Création de la table « local »

@Override

**public** **void** onCreate(SQLiteDatabase db) {

 **try** {

 db.execSQL("CREATE TABLE local (" +

 " id INTEGER PRIMARY KEY AUTOINCREMENT," +

 " nb\_places INTEGER," +

 " nom TEXT " +

 ")");

 }

 **catch** (SQLException e) {

 e.printStackTrace();

 }

}

Types de données dans SQLite

* INTEGER
* REAL
* TEXT

\*\* boolean : on utilise integer 0 ou 1

\*\* clé primaire : INTEGER PRIMARY KEY AUTOINCREMENT

### Création du modèle (ModelRoom)

Le modèle utilise notre DataBaseHelper. Il contient les méthodes accessibles par le programme pour ajouter/modifier/supprimer/lire la base de données

**public** **class** ModelRoom {

 **private** DatabaseHelper databaseHelper;

 **private** SQLiteDatabase database;

 **public** ModelRoom(Context ctx) {

 databaseHelper = **new** DatabaseHelper(ctx);

 }

 **public** **void** openDatabase() {

 database = databaseHelper.getWritableDatabase();

 }

 **public** **void** closeDatabase() {

 database.close();

 }

 **public** **long** addRoom(**int** seats, String name) {

 ContentValues values = **new** ContentValues();

 values.put("nb\_places", seats);

 values.put("nom", name);

 **return** database.insert("local", **null**, values);

 }

}

À partir des autres classes du programme Android, on peut maintenant faire un appel à notre modèle. Voici un exemple :

Button button = (Button) findViewById(R.id.*button1*);

button.setOnClickListener(**new** OnClickListener() {

**public** **void** onClick(View view) {

 ModelRoom modelRoom = **new** ModelRoom(MainActivity.**this**);

 modelRoom.openDatabase();

 **long** id = modelRoom.addRoom(32, "C4.07");

 modelRoom.closeDatabase();

 Log.*i*("Added", "id : " + id);

 }

});

### Supprimer un local

Dans le modèle

**public** **void** deleteRoom(**long** line) {

 database.delete("local", "id = " + line, **null**);

}

Dans le programme :

modelRoom.deleteRoom(id);

### Lire un local

Le mieux serait de faire un objet Local/Room

**public** **class** Room {

 **private** **int** id;

 **private** **int** seats;

 **private** String name;

 **public** Room (**int** id, int seats, String name) {

 **this**.id = id;

 **this**.seats = seats;

 **this**.name = name;

 }

// getter setter

}

Dans le modèle

**public** Room getRoom(**long** id) {

 Cursor cursor = database.query(**true**, "local", **new** String[]{"id", "nb\_places", "nom"},

 "id = " + id, **null**, **null**, **null**, **null**, **null**, **null**);

 Room room = **null**;

 **if** (cursor != **null**) {

 cursor.moveToFirst();

 room = **new** Room(cursor.getInt(0), cursor.getInt(1), cursor.getString(2));

 }

 **return** room;

}

Dans le programme

Room room = modelRoom.getRoom(id);

Log.*i*("Afficher", "Nom : " + room.getName());

### Modifier un local

Dans le modèle

**public** **void** updateSeats(**long** id, **int** seats) {

 ContentValues values = **new** ContentValues();

 values.put("nb\_places", seats);

 database.update("local", values, "id = " + id, **null**);

}

Dans le programme

modelRoom.updateSeats(id, 50);

### Lire tous les locaux

Dans le modèle

**public** List<Room> getRoomList() {

 List<Room> roomList = **new** ArrayList<Room>();

 Cursor cursor = database.query(**true**, "local", **new** String[]{"id", "nb\_places", "nom"},

 **null**, **null**, **null**, **null**, **null**, **null**, **null**);

 **if** (cursor != **null**) {

 **if** (cursor.moveToFirst()) {

 **do** {

 Room room = **new** Room(cursor.getInt(0), cursor.getInt(1), cursor.getString(2));

 roomList.add(room);

 }

 **while** (cursor.moveToNext());

 }

 }

 **return** roomList;

 }

Dans l’activité

List<Room> roomList = modelRoom.getRoomList();

**for** (Room tmpRoom : roomList) {

 Log.*i*("Local", "id : " + tmpRoom.getId() + " (" + tmpRoom.getName() + ")");

}

### Les modifications aux données

Le problème est que si on ajoute une colonne à notre table. Que faut-il faire avec les données et la table déjà créée ? Bref, comment faire si on a une modification de la structure de la base de données ?

On peut faire deux choses:

* Détruire notre base de données et la régénérer
	+ Ceci supprimera les données actuellement dans la table (i.e. DROP suivi de CREATE)
* La mettre à jour avec des alter tables.

Cas simple (destruction et création)

Dans le databaseHelper

@Override

**public** **void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {

 db.execSQL("DROP TABLE IF EXISTS local");

 onCreate(db);

}