Chapitre 7
Les bases de données avec SQLite

7.1 Introduction

Android offre plusieurs méthodes pour stocker les données d'une application. La solution choisie va dépendre des besoins: données privées ou publiques, un petit ensemble de données à préserver ou un large ensemble à préserver localement ou via le réseau. Ces méthodes utilisent soit des éléments propres à l'API Java ou ceux associés à l'API d'Android. Dans ce chapitre nous allons voir ces différentes méthodes et on va s'intéresser à la base de données SQLite.

7.2 Méthodes de stockage des données d'une application

7.2.1 Persistance dans l'état d'une application

On utilise pour cela la notion de « Bundle » et les différents cycles de l'activité pour sauvegarder l'information utile à l'aide du « bundle » et récupérer cette information dans un autre état de l'activité. On ne peut utiliser qu'un seul bundle, par ailleurs la donnée n'est pas persistante et n'est disponible que tant que l'application est utilisée.

7.2.2 Préférences partagées

Un ensemble de paires : clé et valeur. Clé est un « String » et Valeur est un type primitif (« boolean », « String », etc.). Ces préférences sont gérées à travers un code Java ou bien via une activité. Les données ne sont pas cryptées. Les préférences sont adaptées pour des paires simples, mais dès qu'il est question de données plus complexes, il est préférable d'utiliser des fichiers.

7.2.3 Fichiers (création et sauvegarde)

Android permet la création, la sauvegarde et la lecture de fichiers à travers un média persistant (mémorisation et disponibilité). Les fichiers peuvent être de n'importe quel type (image, XML, etc.). Les fichiers peuvent être considérés pour une utilisation interne, donc local à l'application, ou bien externe, donc partagée avec plusieurs applications.

7.2.4 Base de données relationnelle, SQLite

Android offre aussi la possibilité d'utiliser toutes les propriétés d'une base de données relationnelle. Android utilise pour cela une base de données basée sur « SQLite » (www.sqlite.org). Android stocke la base de données localement à l'application. Si l'on veut partager cette structure de données avec d'autres applications, il faudra utiliser dans ce cas, un gestionnaire de contenu (content provider) configuré à cet effet.

7.2.5 Stockage réseau

Android permet de stocker des fichiers sur un serveur distant. On peut utiliser pour cela différentes techniques.

7.3 SQLite

Android intègre le système de gestion de bases de données, SQLite. Pour plus de détails, consultez ce lien : http://www.sqlite.org/. C'est un système compact, très efficace pour les systèmes embarqués. En effet, il utilise très peu de mémoire. SQLite ne nécessite pas de serveur pour fonctionner, ce qui n'est pas le cas de MySQL par exemple.

Les opérations sur la base de données se feront donc dans le même processus que l'application. Il faudra faire attention aux opérations « lourdes », votre application va ressentir les contres coups. Il est conseillé dans ce cas d'utiliser les tâches asynchrones (ou threads).

Chaque application peut avoir donc ses propres bases. Ces bases sont stockées dans le répertoire « databases » associé à l'application (/data/data/APP_NAME/databases/nom_base). Nous pouvons les

stocker aussi sur une unité externe (sdcard).

Chaque base créée, elle le sera en mode « MODE_PRIVATE ». Aucune autre application ne peut y accéder que l'application qui l'a créée. Pour y avoir accès, il faut que la base ait été sauvegardée sur un support externe, sinon utiliser le mécanisme d'échange de données fourni par Android.

SQLite supporte les types : TEXT (chaîne de caractères), INTEGER (entiers), REAL (réels). Tous les types doivent être convertis pour être utilisés. SQLite ne vérifie pas le typage des éléments. À vous de vous en assurer que vous n'avez pas écrit un entier à la place d'une chaîne de caractères par exemple.

7.3.1 Création et mise à jour de la base

L'exemple va créer une table de commentaires. Chaque commentaire est identifié par un identificateur unique. La base va porter le nom « **commments**.db ».

Nom de la table : **comments**	
_ id	**comment**

L'organisation des fichiers permet de faciliter l'organisation de la base de données et la compréhension de l'exemple.

• Le fichier « **Comment**.java » va contenir un enregistrement d'une table et les différentes méthodes qui gravitent autour.

La classe « **Comment** » décrite dans le fichier « **Comment**.java » contient deux attributs :

private long id;

private String comment;

La base de données doit utiliser un identifiant unique « _id » comme clé primaire de la table. Des méthodes d'Android se servent de ce standard.

• Le fichier « **MySQLiteHelper**.java » contient la classe qui dérive de « **SQLiteOpenHelper** ».

La classe « MySQLiteHelper » :

Créez une nouvelle classe qui va dériver de la classe « SQLiteOpenHelper » :

public class **MySQLiteHelper** extends SQLiteOpenHelper { …}

Dans le constructeur de la classe, faites appel à la méthode « SQLiteOpenHelper » et spécifiez le nom de la base et sa version.

```
public MySQLiteHelper(Context context) {
super (context, "commments.db", null, 1);
}
```

Dans cette classe, vous devez redéfinir les méthodes « onCreate (SQLiteDatabase MaBase) » et « onUpgrade(SQLiteDatabase MaBase) ». L'argument représente votre base.

La méthode « onCreate » est appelée pour la création de la base si elle n'existe pas.

```
public void onCreate (SQLiteDatabase database) {
 database.execSQL(DATABASE_CREATE);
}
```

La variable « DATABASE_CREATE » va contenir la requête « SQL » qui permet de créer la base.

La méthode « **onUpgrade** » est appelée pour mettre à jour la version de votre base. Elle vous permet de mettre à jour le schéma de votre base.

```
public void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion) {
db.execSQL("DROP TABLE IF EXISTS " + "comments");
onCreate(db);
}
```

Il est préférable de créer une classe par table. Cette classe va définir les méthodes « onCreate » et « onUpgrade ». Vous allégez ainsi le code de la classe qui dérive de « SQLiteOpenHelper ».

• Le fichier « CommentsDataSource.java » contient la classe contrôleur. Elle contient les différentes méthodes qui vont interagir avec la base de données. C'est le **DAO** (Data Access Object)

La classe « **SQLiteOpenHelper** » fournit les deux méthodes « g**etReadableDatabase**() » et « **getWriteableDatabase**() » pour accéder à une instance « SQLiteDatabase » en mode de lecture ou écriture.

Ouverture de la base :

```
public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
}
```

Fermeture de la base :

```
public void close() {
        dbHelper.close();
}
```

Insérer un élément :

Pour insérer un élément dans la base, il faut d'abord former l'enregistrement. On utilise pour cela un objet du type « ContentValues » qui représente une collection de champs.

```
public long createComment (String comment) {
        ContentValues values = new ContentValues();
        values.put(MySQLiteHelper.COLUMN_COMMENT, comment);
        long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS,null, values);
        return insertId;
}
```

Effacer un élément :

```
public void deleteComment (Comment comment) {
        long id = comment.getId();
        database.delete(MySQLiteHelper.TABLE_COMMENTS, ySQLiteHelper.COLUMN_ID
      + " = " + id, null);
}
```

Faire une sélection :

Cursor cursor = database.query (MySQLiteHelper.TABLE_COMMENTS, allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId, null, null, null, null);

La méthode « query » retourne une instance de « Cursor » qui représente un ensemble de résultas.

- Le fichier « TestDatabaseActivity.java » contient l'activité associée à notre application.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_testdatabase);
        datasource = new CommentsDataSource(this);
       datasource.open();
}
protected void onResume() {
       super.onResume();
       datasource.open();
}

protected void onPause() {
       super.onPause();
       datasource.close();
}
```

Accès à la base de données SQLite

Le SDK d'Android inclut un programme permettant de lire une base de données SQLite.

Nous devons extraire le fichier de l'émulateur via la commande « adb/shell/pull » en tant que « root » ou bien en utilisant la vue « **Device File Explorer** », puis la commande « pull » :

```
adb pull
/data/data/ca.umontreal.iro.ift1155.testdatabaseactivity/databases/comments.db
C:> sqlite3 comments.db
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
sqlite> .tables
android metadata comments
sqlite> select * from comments;
1|Hate it
2|Cool
3|Very nice
4|Hate it
sqlite> .exit
```

On peut utiliser un adds-on dans Firefox pour accéder à la base de données : **https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager-webext/**

Comme il est possible aussi d'utiliser un de ces deux logiciels :

https://sqlitebrowser.org/ **https://sqlitestudio.pl/index.rvt**

7.4 Conclusion

A la fin de ce chapitre, nous pouvons dire que nous avons découvert la plateforme Android avec toutes ses fonctionnalités de base ainsi que la base de données SQLite qu'elle utilise. Vous avez maintenant tous les outils qui vous permettent de développer votre première application Android et qui sera le sujet du chapitre suivant.