

# JAVA ET ANDROID



## Les données SQL locales

# Sommaire

Chapitre 1 - GESTION DES DONNEES SQL.....	4
1.1 - Android et SQL.....	5
1.2 - Android et SQLite.....	6
1.2.1 - Présentation.....	6
1.2.2 - Quelques méthodes de l'API.....	7
1.2.3 - Se connecter ou créer une BD.....	8
1.2.4 - Se connecter.....	8
1.2.5 - Se déconnecter.....	8
1.2.6 - Créer une table.....	9
1.2.7 - selectAll.....	10
1.2.8 - selectOne.....	11
1.2.9 - Insérer un enregistrement.....	12
1.2.10 - Supprimer un enregistrement.....	13
1.2.11 - Modifier un enregistrement.....	13
1.2.12 - Les ordres compilés.....	14
1.2.13 - Gérer une transaction.....	15
1.2.14 - Extraire un ou plusieurs enregistrements.....	16
1.2.15 - Gestion de curseur.....	17
1.2.16 - Exemple supra-élémentaire d'utilisation d'une BD SQLite.....	18
1.2.16.1 - Objectif.....	18
1.2.16.2 - L'écran.....	18
1.2.16.3 - Codes.....	19
1.2.17 - Exemple élémentaire de gestion de BD.....	23
1.2.17.1 - Objectif.....	23
1.2.17.2 - Diagramme de séquence système.....	24
1.2.17.3 - Démarche.....	25
1.2.17.4 - Diagramme de classes générique.....	26
1.2.17.5 - L'interface statique.....	27
1.2.17.6 - Présentation des codes java.....	30
1.2.17.7 - Le bean Ville.....	32
1.2.17.8 - Le DAO.....	34
1.2.17.9 - Le « connector ».....	37
1.2.17.10 - L'activité.....	39
1.2.18 - Exercice.....	43
1.3 - Le Content Provider : fournir des données à d'autres applications.....	44
1.3.1 - Définition.....	44
1.3.2 - Schémas.....	45
1.3.3 - Diagramme.....	47
1.3.4 - Création.....	48
1.3.5 - Syntaxes.....	49
1.3.6 - Codes.....	51
1.3.7 - Exercices : ContentProvider.....	52
1.4 - Exploitation d'un ContentProvider : le ContentResolver.....	53
1.4.1 - Objectif.....	53
1.4.2 - Démarche.....	53
1.4.3 - Syntaxes.....	54
1.4.4 - Codes.....	55
1.4.5 - Exercices : ContentResolver.....	58
1.5 - TP : SMS groupés.....	59
Chapitre 2 - ANNEXES.....	60
2.1 - Le Content Resolver des Contacts.....	61
2.1.1 - Présentation.....	61
2.1.2 - Insérer un contact.....	62

2.1.2.1 - Objectif.....	62
2.1.2.2 - Syntaxes.....	63
2.1.2.3 - Le layout : contacts_insert.xml.....	64
2.1.2.4 - L'activité : ContactsInsert.java.....	66
2.1.3 - Insérer des contacts.....	70
2.1.4 - Visualiser les contacts (les noms).....	75
2.1.4.1 - Objectif.....	75
2.1.4.2 - Prémisses.....	76
2.1.4.3 - Table Contacts.....	77
2.1.4.4 - Table Email et Data.....	78
2.1.4.5 - Démarche.....	79
2.1.4.6 - Quelques syntaxes.....	80
2.1.4.7 - Le layout : contacts_select_all.xml.....	81
2.1.4.8 - L'activité : ContactsSelectAll.java.....	82
2.1.4.9 - Les noms et numéros de téléphone avec une projection.....	84
2.1.5 - Exercices : noms, téléphones et emails.....	85
2.1.6 - Visualiser un contact.....	87
2.1.7 - Supprimer un contact.....	89
2.1.8 - TD-TP : contacts to CSV.....	90
2.2 - Autres ContentResolver système.....	91
2.2.1 - L'historique du navigateur.....	92
2.2.2 - L'historique des appels téléphoniques.....	94
2.2.3 - Les media.....	95
2.2.4 - Les paramètres de sécurité.....	96
2.2.5 - Données CSV, ContentProvider et ContentResolver.....	97
2.3 - Les types de cursors Android.....	98
2.4 - Le CursorLoader d'Android.....	99

# **CHAPITRE 1 - GESTION DES DONNEES SQL**

## 1.1 - ANDROID ET SQL

Une application Android peut gérer des données SQL de différentes façons.

Le système Android intègre en natif SQLite qui est un SGBDR léger.

Son SQL est standard. Il gère les Foreign Keys et les transactions.

Pour plus de détails cf le support SQLite ou la documentation officielle (<http://www.sqlite.org/>).

Une BD **SQLite** peut être partagée entre plusieurs applications; il faut pour cela créer un **ContentProvider**. Ensuite les autres applications pourront travailler sur les données via un **ContentResolver**.

Une application Android peut accéder à une **BD distante** (MySQL, Oracle, MSS, ...) via le protocole **HTTP** et des scripts PHP, des servlets ou autres ou des Services Web REST (\*) ou SOAP (\*\*) ou via **JDBC**.

(\*) REST (Representational State Transfer) ou RESTful est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service).

<http://blog.nicolashachet.com/niveaux/confirmelarchitecture-rest-expliquee-en-5-regles/>

(\*\*) SOAP (ancien acronyme de Simple Object Access Protocol) est un protocole de RPC orienté objet bâti sur XML.

Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur. Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.

Le protocole SOAP est composé de deux parties :

une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement, un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre.

SOAP a été initialement défini par Microsoft et IBM1, mais est devenu une référence depuis une recommandation du W3C, utilisée notamment dans le cadre d'architectures de type SOA (Service Oriented Architecture) pour les Services Web WS-\*.

## 1.2 - ANDROID ET SQLITE

### 1.2.1 - Présentation

Le système Android intègre en natif SQLite qui est un SGBDR léger.

SQLite gère des BD sous forme de fichiers d'extension .db.

Comme n'importe quel SGBDR, SQLite gère des données stockées dans des tables manipulables par les commandes SQL standards.

**Cependant** l'API Android, via la classe SQLiteDatabase, permet de gérer les données SQL via des méthodes spécifiques (via un DAO en somme).

Les deux packages fournis par Android sont : android.database et android.database.sqlite.


A noter qu'il est possible de gérer des transactions et des Foreign Keys.


<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

Une BD SQLite sera stockée dans le dossier suivant :

/data/data/nom.de.package/databases/

Par exemple : /data/data/fr.pb.sql/databases/cours.db

Le dossier databases pourrait être créé au préalable avec le File explorer  .

La BD pourrait être importée  avec ce même File explorer si elle a été créée avec SQLite3. Autrement elle peut être créée par script dans l'application.

## 1.2.2 - Quelques méthodes de l'API

<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

Commande	Fonctionnalité
SQLiteDatabase.openOrCreateDatabase()	Connexion ou création d'une BD
SQLiteDatabase.openDatabase()	Connexion à une BD
SQLiteDatabase.close()	Déconnexion d'une BD
SQLiteDatabase.execSQL()	Exécution d'un ordre du LDD (CREATE, DROP, ALTER) ou du LMD (INSERT, DELETE, UPDATE)
SQLiteDatabase.rawQuery()	Pour un SELECT élaboré (jointure, produit cartésien, ...)
SQLiteDatabase.query()	Exécution d'un SELECT
SQLiteDatabase.insert()	Exécution d'un INSERT
SQLiteDatabase.delete()	Exécution d'un DELETE
SQLiteDatabase.update()	Exécution d'un UPDATE

### Notes :

Une BD SQLite est le plus souvent gérée avec un **SQLiteOpenHelper** (un gestionnaire de connexion). Mais une BD SQLite peut être gérée sans SQLiteOpenHelper.

<http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

Documentation officielle :

Create a helper object to create, open, and/or manage a database.

```
SQLiteOpenHelper(Context context, String databaseName,  
SQLiteDatabase.CursorFactory factory, int version)
```

cf un exemple en 1.2.16.3 (le code de la classe GestionnaireOuvertureSQLite qui hérite de SQLiteOpenHelper).

Pour qu'une BD soit partagée entre plusieurs applications il faut créer un **ContentProvider** (cf la section 1.3). Ensuite les autres applications pourront travailler sur les données via un **ContentResolver** (cf la sections 1.4).

### 1.2.3 - Se connecter ou créer une BD

```
SQLiteDatabase.openOrCreateDatabase(String path,  
SQLiteDatabase.CursorFactory factory)
```

```
String lsChemin = "/data/data/fr.pb.sql/databases/cours.db";  
SQLiteDatabase bd = SQLiteDatabase.openOrCreateDatabase(lsChemin, null);
```

Cette méthode ouvre une BD si elle existe et la crée dans le cas contraire.

Le chemin des BDs SQLite est /data/data/nom.de.votre.package/databases/  
Pour ce projet ce sera /data/data/fr.pb.sql/databases/.

Le dossier est créé automatiquement à la création de la BD s'il n'existe pas.

```
// --- Chemin  
lsChemin = contexte.getDatabasePath("cours.db").getPath();
```

### 1.2.4 - Se connecter

```
SQLiteDatabase.openDatabase(String path, SQLiteDatabase.CursorFactory  
factory, int flags)
```

```
String lsChemin = "/data/data/fr.pb.sql/databases/cours.db";  
SQLiteDatabase bd = SQLiteDatabase.openDatabase(lsChemin, null,  
SQLiteDatabase.OPEN_READWRITE);
```

Le deuxième paramètre permettrait de passer en argument un curseur spécifique.

Quelques flags :

SQLiteDatabase.OPEN\_READONLY : ouverture en lecture seule

SQLiteDatabase.OPEN\_READWRITE : ouverture en lecture/écriture

SQLiteDatabase.CREATE\_IF\_NECESSARY : ouverture et création si nécessaire

### 1.2.5 - Se déconnecter

```
SQLiteDatabase.close()
```

```
this.bd.close();
```



---

### 1.2.6 - Créer une table

Ou tout autre instruction du LDD ou du LMD sauf un SELECT.

```
SQLiteDatabase.execSQL("SQL LDD")
```

Exemple : si la BD existe on crée la table ...

```
if(this.bd != null) {  
    bd.execSQL("CREATE TABLE IF NOT EXISTS villes(id_ville INTEGER PRIMAY  
KEY AUTOINCREMENT, cp TEXT, nom_ville TEXT)");  
}
```

Exemple : si la BD existe on supprime la table ...

```
if(this.bd != null) {  
    bd.execSQL("DROP TABLE villes");  
}
```

On exécute un ordre INSERT avec une commande SQL (méthode non standard pour Android; il est préférable d'utiliser la méthode insert de la classe SQLiteDatabase) ...

```
this.bd.execSQL("INSERT INTO villes(cp, nom_ville)  
VALUES('59000', 'Lille')");
```

---

### 1.2.7 - selectAll

**Syntaxe :**

```
SQLiteDatabase.query(String table, String[] columns, String  
whereParameter, String[] selectionArgs, String groupBy, String having,  
String orderBy)
```

La méthode query() renvoie un Cursor.

**Exemple :**

```
Cursor curseur = this.bd.query("villes", null, null, null, null, null,  
null);
```

---

### 1.2.8 - selectOne

#### Syntaxe :

```
SQLiteDatabase.query(String table, String[] columns, String  
whereParameter, String[] selectionArgs, String groupBy, String having,  
String orderBy)
```

#### Exemple :

```
// --- Avec un WHERE : une restriction  
String asWhere ="cp=?";  
String[] asArgs = {"75020"};  
Cursor curseur = this.bd.query("villes", null, asWhere, asArgs, null,  
null, null);
```

### 1.2.9 - Insérer un enregistrement

#### Syntaxe :

```
SQLiteDatabase.insert(String table, String nullColumnHack, ContentValues values)
```

ContentValues est un HashMap (liste des colonnes et valeurs des colonnes).

#### Exemple :

```
// --- Valorisation des colonnes via un HashMap
ContentValues hmValeurs = new ContentValues();
hmValeurs.put("cp", tValeurs[0]);
hmValeurs.put("nom_ville", tValeurs[1]);

// --- Insertion
try {
    llNum = this.bd.insert("villes", null, hmValeurs);
    lsMessage = String.valueOf(llNum) + " enregistrement ajouté";
}
catch (SQLException e) {
    lsMessage = "Insertion ratée";
}
```

**nullColumnHack** : optional; may be null. SQL doesn't allow inserting a completely empty row without naming at least one column name. If your provided values is empty, no column names are known and an empty row can't be inserted. If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your values is empty.

nullColumnHack: facultatif; peut être nulle. SQL ne permet pas d'insérer une ligne complètement vide sans nommer au moins un nom de colonne. Si les valeurs fournies sont vides, si les noms des colonnes sont inconnus la ligne vide ne peut être insérée. Si le paramètre nullColumnHack est fourni pour le nom de la colonne nullable alors il est possible d'insérer explicitement une valeur NULL dans le cas où la valeur est vide.

---

### 1.2.10 - Supprimer un enregistrement

```
SQLiteDatabase.delete(String table, String whereClause, String[]  
whereArgs)
```

whereClause est une chaîne du type colonne1=? and colonne2=? ...

whereArgs est un tableau de String contenant les valeurs correspondant aux colonnes de la clause WHERE.

```
l1Num = this.bd.delete("villes", "cp=?", tValeurs);
```

Pour les supprimer tous :

```
l1Num = this.bd.delete("villes", null, null);
```

---

### 1.2.11 - Modifier un enregistrement

```
SQLiteDatabase.update(String table, ContentValues values, String  
whereClause, String[] whereArgs)
```

```
this.bd.update("villes", lhm, "cp=?", tValeurs);
```

Pour les modifier tous :

```
l1Num = this.bd.update("villes", null, null, null);
```

---

### 1.2.12 - Les ordres compilés

#### Compilation

```
SQLiteStatement stmt = bd.compileStatement("Commande SQL avec des ?");
```

Valorisation des paramètres ; ça commence à 1 !!!

```
stmt.bindString(1, "valeur");
```

#### Exécution

```
stmt.execute();
```

#### Exemple :

```
String lsSQL = "INSERT INTO datecours(date_du_jour) VALUES(?)";
SQLiteStatement stmt = bd.compileStatement(lsSQL);
try {
    stmt.bindString(1, "2014-01-05");
    stmt.execute();
    Log.e("Insert compilé", "OK");
} catch (Exception e) {
    Log.e("Insert compilé", e.getMessage());
}
```

---

### 1.2.13 -

Démarre une transaction en mode EXCLUSIVE.

```
SQLiteDatabase.beginTransaction();
```

Démarre une transaction en mode IMMEDIATE.

```
SQLiteDatabase.beginTransactionNonExclusive();
```

Termine une transaction

```
SQLiteDatabase.endTransaction();
```

Pour faire un COMMIT il faut ajouter avant l'utilisation de la méthode endTransaction() l'appel à la méthode setTransactionSuccessful().

Pour faire un ROLLBACK il faut seulement appeler la méthode endTransaction().

```
db.beginTransaction();
try {
    // Ajouter l'appel a une méthode d'enregistrement dans la BD
    ordre SQL CUD
    db.setTransactionSuccessful();
} catch {
    // Ajouter une gestion des erreurs
} finally {
    db.endTransaction();
}
```

En mode SQL :

```
SQLiteDatabase.execSQL("BEGIN TRANSACTION");
```

```
SQLiteDatabase.execSQL("COMMIT");
```

```
SQLiteDatabase.execSQL("ROLLBACK");
```

### 1.2.14 - Extraire un ou plusieurs enregistrements

#### Syntaxe :

```
SQLiteDatabase.query(String table, String[] columns, String  
whereParameter, String[] selectionArgs, String groupBy, String having,  
String orderBy)
```

La clause whereParameter est une chaîne du type colonne1=? and colonne2=? ...

La méthode query() renvoie un Cursor.

#### Exemples :

```
// --- Sans WHERE : une projection  
String[] cols = {"cp", "nom_ville"};  
Cursor curseur = this.bd.query("villes", cols, null, null, null, null,  
null);
```

```
// --- Avec un WHERE : une projection et une restriction  
String[] cols = {"cp", "nom_ville"};  
String asWhere = "cp=?";  
String[] asArgs = {"75020"};  
Cursor curseur = this.bd.query("villes", cols, asWhere, asArgs, null,  
null, null);
```

Il est possible d'outrepasser la syntaxe objet et d'exécuter un ordre SQL standard :

```
SQLiteDatabase.rawQuery("SELECT ...", ArgumentsDuWhere);
```

```
Cursor curseur = this.bd.rawQuery("SELECT * FROM villes", null);
```

```
String[] asArgs = {"75011"};  
Cursor curseur = this.bd.rawQuery("SELECT * FROM villes WHERE cp = ?",  
asArgs);
```



### 1.2.15 - Gestion de curseur

Les numéros des colonnes commencent à 0.

<http://developer.android.com/reference/android/database/Cursor.html>

#### Quelques méthodes de Cursor

Méthode	Description
Cursor.moveToFirst()	No comment !
Cursor.moveToPrevious()	No comment !
Cursor.moveToNext()	No comment !
Cursor.moveToLast()	No comment !
Cursor.move(rang)	No comment !
Cursor.isFirst()	No comment !
Cursor.isLast()	No comment !
Cursor.isBeforeFirst()	No comment !
Cursor.isAfterLast()	No comment !
Cursor.getCount()	No comment !
Cursor.getString(index)	No comment ! Index commence à 0.
Cursor.getInt(index)	No comment ! Index commence à 0.
Cursor.getDouble(index)	No comment ! Index commence à 0.
Cursor.getColumnCount()	No comment !
Cursor.getColumnName(i)	No comment !
Cursor.close()	No comment !

Note : toutes les méthodes moveToXXX() et isXXX() renvoient un booléen.

Cf l'exemple de la méthode selectAll().

---

## 1.2.16 - Exemple supra-élémentaire d'utilisation d'une BD SQLite

### 1.2.16.1 - Objectif

Ajouter un enregistrement dans une table que l'on va créer dans une nouvelle BD.

Puisque le dossier /data/data/n.d.p/databases n'existe pas il va être créé par le script.

Puisque la BD n'existe pas elle va être créée par le script.

Puisque la table « pays » n'existe pas elle va être créée par le script.

La table « pays » a la structure suivante : pays(id\_pays INTEGER, nom\_pays VARCHAR(50))

La PK est id\_pays. La valeur est auto incrémentée.

Les valeurs dans la colonne nom\_pays sont uniques.

Les codes SQL qui seront exécutés sont les suivants :

```
CREATE TABLE pays(  
  id_pays INTEGER PRIMARY KEY AUTOINCREMENT,  
  nom_pays VARCHAR(50) NOT NULL UNIQUE  
);
```

```
INSERT INTO pays(nom_pays) VALUES('France');
```

### 1.2.16.2 - L'écran



## 1.2.16.3 - Codes

## Le Helper

```
package fr.pb.sqllocal;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

/*
 * Classe Helper pour la gestion de l'ouverture de la BD
 * La connexion a la BD en somme ou la creation et connexion si elle
n'existe pas
 * Methodes : constructeur, onCreate, onUpgrade
 */

// -----
public class GestionnaireOuvertureSQLite extends SQLiteOpenHelper {
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "cours.db";

    private static final String DROP_TABLE_PAYS = "DROP TABLE IF EXISTS
pays";
    private static final String CREATE_TABLE_PAYS = "CREATE TABLE IF NOT
EXISTS pays(id_pays INTEGER PRIMARY KEY AUTOINCREMENT, nom_pays
VARCHAR(50) NOT NULL UNIQUE)";

    // --- Constructeur
    // -----
    public GestionnaireOuvertureSQLite(Context contexte, CursorFactory
fabrique) {
        // --- Cree la BD si elle n'existe pas
        // --- et de ce fait execute le code de onCreate()
        // --- Se connecte si elle existe
        super(contexte, DB_NAME, fabrique, DB_VERSION);
    } // GestionnaireOuvertureSQLite()

    @Override
    // -----
    public void onCreate(SQLiteDatabase abd) {
        // --- Creation de la table pays
        abd.execSQL(CREATE_TABLE_PAYS);

        /*
         * Pour les tests
         */
        // --- Remplissage de la table pays
        abd.execSQL("INSERT INTO pays(nom_pays) VALUES('France')");
    } // onCreate()

    @Override
    // -----
    public void onUpgrade(SQLiteDatabase abd, int ancienneVersion, int
nouvelleVersion) {
        // --- Suppression de la table puis appel a la creation des tables
        abd.execSQL(DROP_TABLE_PAYS);
    }
}
```

```
        onCreate(abd);  
    } /// onUpgrade()  
  
} /// class GestionnaireOuvertureSQLite
```

**Le layout : bd\_create.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp">

    <Button
        android:id="@+id/buttonBDCreate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BD Create" />

    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</LinearLayout>
```

**L'activité**

```
package fr.pb.sqllocal;

import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class BDCreate extends Activity implements View.OnClickListener {

    private Button buttonBDCreate;
    private TextView textViewMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.bd_create);

        buttonBDCreate = findViewById(R.id.buttonBDCreate);
        buttonBDCreate.setOnClickListener(this);

        textViewMessage = findViewById(R.id.textViewMessage);
    }

    @Override
    public void onClick(View v) {

        StringBuilder lsbMessage = new StringBuilder();
        GestionnaireOuvertureSQLite gos;
        SQLiteDatabase bd;
    }
```

```
        try {
            // Connexion
            // --- GestionnaireOuvertureSQLite(Contexte, Fabrique de
curseur);
            gos = new GestionnaireOuvertureSQLite(this, null);
            bd = gos.getWritableDatabase();

            lsbMessage.append("Connexion réussie");
            lsbMessage.append("\n");

            // Creation de la table
            // cf GOS

            // Insertion dans la table
            // cf GOS + ce qui suit
            ContentValues hmValeurs = new ContentValues();
            hmValeurs.put("nom_Pays", "Italie");
            long llNum = bd.insert("pays", null, hmValeurs);

            // Visualisation de la table
            // --- Sans WHERE : une projection
            String[] cols = {"id_pays", "nom_pays"};
            Cursor curseur = bd.query("pays", cols, null, null, null,
null, null);
            while (curseur.moveToNext()) {
                lsbMessage.append(curseur.getInt(0));
                lsbMessage.append("-");
                lsbMessage.append(curseur.getString(1));
                lsbMessage.append("\n");
            }
            curseur.close();

            // Deconnexion
            gos.close();
            lsbMessage.append("Vous êtes déconnecté(e) !");
            lsbMessage.append("\n");
            bd = null;

        } catch (Exception e) {
            lsbMessage.append("Aïe, aïe, aïe ! ");
            lsbMessage.append(e.getMessage());
            lsbMessage.append("\n");
        }

        textViewMessage.setText(lsbMessage.toString());
    } /// onClick
} /// classe
```

## 1.2.17 - Exemple élémentaire de gestion de BD

### 1.2.17.1 - Objectif

CRUD sur la table Ville.

Se connecter et se déconnecter de la BD.

Ajouter un enregistrement.

Supprimer un enregistrement en fonction du CP.

Afficher tous les enregistrements.

Afficher un enregistrement en fonction du CP.

Écran d'insertion et d'affichage des enregistrements (pour afficher tous les enregistrements il faut que le champ CP soit vide, pour n'afficher qu'un enregistrement il faut saisir une valeur dans le champ CP).

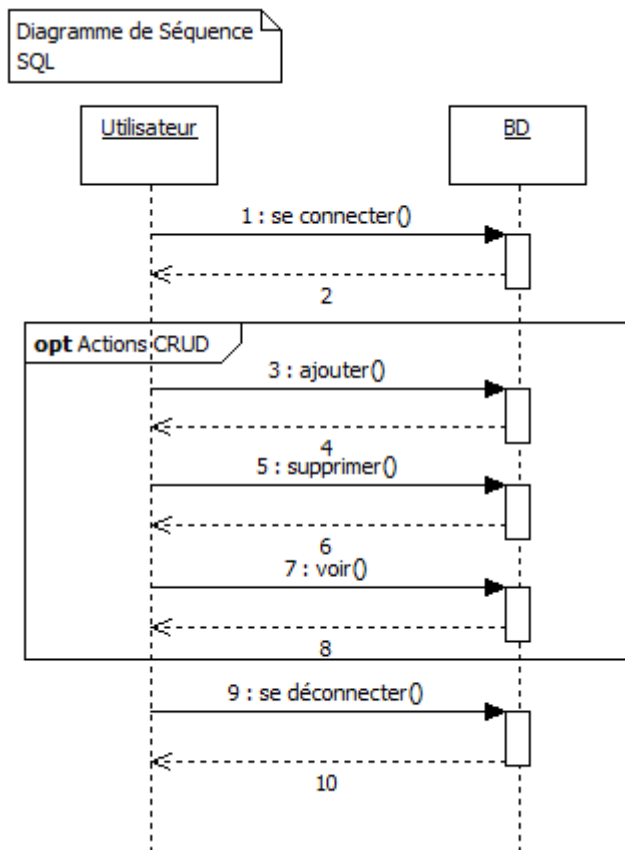
Ajout	Affichage
<p>SQL</p> <p>11:15</p> <p>+</p> <p>-</p> <p>?</p> <p>Cp 75010</p> <p>Ville Paris 10</p> <p>ID Pays 33</p> <p>Insertion OK</p>	<p>SQL</p> <p>11:31</p> <p>+</p> <p>-</p> <p>?</p> <p>Cp</p> <p>Ville</p> <p>ID Pays  </p> <p>75001 - Paris I - 33 75011 - Paris XI - 33 75012 - Paris XII - 33 75020 - Paris XX - 33</p>

**Note :**

Il serait possible de créer la BD et la table avec sqlite3.exe "à l'extérieur" puis d'importer la BD dans le dossier /data/data/fr.pb.sql/databases/.

## 1.2.17.2 - Diagramme de séquence système

L'utilisateur doit se connecter en premier s'il veut ajouter un enregistrement ou visualiser un ou plusieurs enregistrements.





### 1.2.17.3 - Démarche

Dans le même projet,  
Créez une classe Activity nommée **VilleCRUD**.  
Créez un layout nommé **ville\_crud.xml**.

Ajoutez ces String dans le fichier /res/values/strings.xml.

```
<string name="cp">Cp</string>
<string name="ville">Ville</string>
<string name="idpays">Code Pays</string>

<string name="message">Message</string>
```

Ajoutez les icônes dans le dossier dans le dossier /res/drawable/.

Créez une classe Bean nommée **Ville**.  
Créez une classe DAO nommée **VilleDAO**.  
Créez une classe provider ou helper nommée **GestionnaireOuvertureSQLite**.

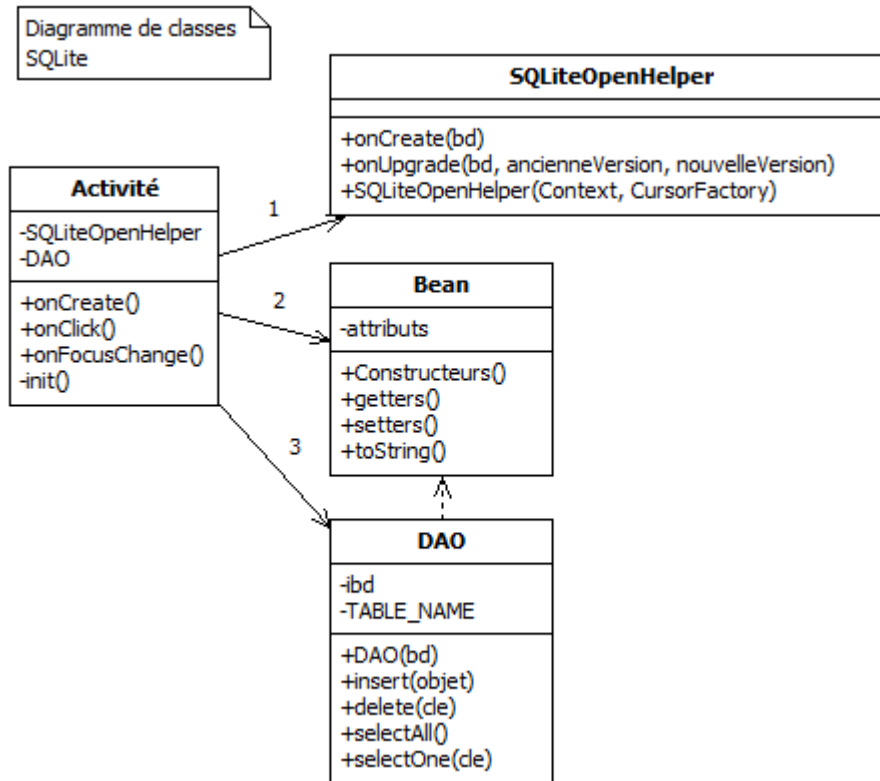
La gestion d'une table nécessite :

- ✓ L'existence d'un dossier /data/data/nom.du.package/databases/,
- ✓ L'existence d'une BD,
- ✓ L'existence d'une table.

Si ces 3 éléments n'existent pas, ils seront créés par script dans **GestionnaireOuvertureSQLite**.

## 1.2.17.4 - Diagramme de classes générique

Une activité Android basique avec SQLite utilise un SQLiteOpenHelper et un DAO et un JavaBean.



**Note** : `initInterface()` et pas `init()`.

## 1.2.17.5 - L'interface statique

**ville\_crud.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="10dp">

        <!-- Boutons -->
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="7"
            android:orientation="horizontal">

            <ImageButton
                android:id="@+id/buttonSeConnecter"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="20"
                android:contentDescription="Connexion"
                android:src="@drawable/in_noir_24_24" />

            <ImageButton
                android:id="@+id/buttonAjouter"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="20"
                android:contentDescription="Ajouter"
                android:src="@drawable/plus_noir_24_24" />

            <ImageButton
                android:id="@+id/buttonSupprimer"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="20"
                android:contentDescription="Supprimer"
                android:src="@drawable/minus_noir_24_24" />

            <ImageButton
                android:id="@+id/buttonVoir"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="20"
                android:contentDescription="Voir"
                android:src="@drawable/voir_noir_24_24" />

            <ImageButton
                android:id="@+id/buttonSeDeconnecter"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
```

```
        android:layout_weight="20"
        android:contentDescription="Déconnexion"
        android:src="@drawable/out_noir_24_24" />

</LinearLayout>

<!-- Saisie CP -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="7"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textViewCp"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="20"
        android:text="@string/cp" />

    <EditText
        android:id="@+id/editTextCp"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="80"
        android:inputType="number">

        <requestFocus />
    </EditText>
</LinearLayout>

<!-- Saisie ville -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="7"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textViewNomVille"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="20"
        android:text="@string/ville" />

    <EditText
        android:id="@+id/editTextNomVille"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="80" />
</LinearLayout>

<!-- Saisie ID Pays -->

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="7"
    android:orientation="horizontal">
```

```
<TextView
    android:id="@+id/textViewIdPays"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="20"
    android:text="@string/idpays" />

<EditText
    android:id="@+id/editTextIdPays"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="80"
    android:inputType="number" />

</LinearLayout>

<!-- Autres -->

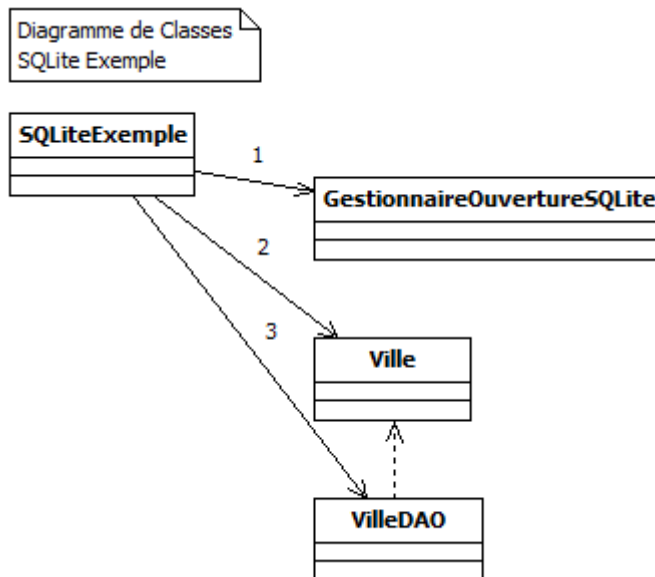
<TextView
    android:id="@+id/textViewMessage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/message" />

</LinearLayout>

</ScrollView>
```

## 1.2.17.6 - Présentation des codes java

Les quatre classes : SQLiteExemple (l'activité), Ville (le Bean), VilleDAO (le DAO), GestionnaireOuvertureSQLite (le Helper).



**SQLiteExemple.java : l'Activity**

VilleCRUD
(-) GestionnaireOuvertureSQLite gos
(-) SQLiteDatabase ibd
(-) VilleDAO idao
(-) Widgets
(+) onCreate()
(+) onClick()
(-) initInterface()

**Ville.java : le Bean**

Ville
(-) idVille
(-) cp
(-) nomVille
(-) idPays
(+) Ville()
(+) Ville(cp, nomVille, idPays)
(+) Ville(idVille, cp, nomVille, idPays)
(+) getters()
(+) setters()

**VilleDAO.java : gestion de la table (CRUD)**

VilleDAO
(-) TABLE_NAME
(-) ibd
(+) VilleDAO(bd)
(+) insert(ville)
(+) delete(ville)
(+) selectAll()
(+) selectOne(id)

**GestionnaireOuvertureSQLite.java : un Helper pour la connexion et/ou la création de la BD**

GestionnaireOuvertureSQLite
(-) DB_NAME
(-) DB_VERSION
(-) CREATE_TABLE_PAYS
(-) DROP_TABLE_PAYS
(-) CREATE_TABLE_VILLE
(-) DROP_TABLE_VILLE
(+) GestionnaireOuvertureSQLite()
(+) onCreate()
(+) onUpgrade()

**Note** : le Bean n'est utilisé pour l'instant que pour l'insertion (comme paramètre) et la sélection unique (comme objet retourné).

## 1.2.17.7 - Le bean Ville

**Ville**

```
package fr.pb.sqllocal;

/*
 * JavaBean Ville
 */
public class Ville {

    // --- Attributs
    private int idVille;
    private String cp;
    private String nomVille;
    private String idPays;

    // --- Methodes

    // --- Constructeurs
    public Ville() {
        super();
    }

    public Ville(String cp, String nomVille, String idPays) {
        super();
        this.cp = cp;
        this.nomVille = nomVille;
        this.idPays = idPays;
    }

    public Ville(int idVille, String cp, String nomVille, String idPays)
    {
        super();
        this.idVille = idVille;
        this.cp = cp;
        this.nomVille = nomVille;
        this.idPays = idPays;
    }

    // --- Accesseurs et modificateurs
    public int getIdVille() {
        return idVille;
    }

    public void setIdVille(int idVille) {
        this.idVille = idVille;
    }

    public String getCp() {
        return cp;
    }

    public void setCp(String cp) {
        this.cp = cp;
    }

    public String getNomVille() {
```



```
        return nomVille;
    }

    public void setNomVille(String nomVille) {
        this.nomVille = nomVille;
    }

    public String getIdPays() {
        return idPays;
    }

    public void setIdPays(String idPays) {
        this.idPays = idPays;
    }

    @Override
    public String toString() {
        return idVille + "-" + cp + "-" + nomVille + "-" + idPays;
    }
} // / class Ville
```

## 1.2.17.8 - Le DAO

**VilleDAO**

```
package fr.pb.sqllocal;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.Cursor;
import android.content.ContentValues;

/*
 * DAO
 * Methodes : constructeur, insert, delete, selectOne, selectAll
 */
public class VilleDAO {

    private static final String TABLE_NAME = "ville";
    private SQLiteDatabase ibd;

    // --- CONSTRUCTEUR
    public VilleDAO(SQLiteDatabase bd) {
        this.ibd = bd;
    } // / VilleDAO()

    /**
     *
     * @param ville
     * @return
     */
    public boolean insert(Ville ville) {

        boolean lbOK = false;

        // --- Un enregistrement sous forme de HashMap
        ContentValues hmColonnesValeurs = new ContentValues();
        hmColonnesValeurs.put("cp", ville.getCp());
        hmColonnesValeurs.put("nom_ville", ville.getNomVille());
        hmColonnesValeurs.put("id_pays", ville.getIdPays());

        // --- Insertion
        try {
            // public long insert (String table, String
            nullColumnHack, ContentValues values)
            this.ibd.insert(TABLE_NAME, null, hmColonnesValeurs);
            lbOK = true;
        } catch (SQLiteException e) {
        }

        return lbOK;
    } // / insert()

    /**
     *

```

```
* @param asCP
* @return
*/
@Deprecated
public boolean delete(String asCP) {

    boolean lbOK = false;
    String[] tValeurs = new String[1];
    tValeurs[0] = asCP;

    try {
        // public int delete (String table, String whereClause,
String[] whereArgs)
        this.ibd.delete(TABLE_NAME, "cp=?", tValeurs);
        lbOK = true;
    } catch (SQLException e) {
    }

    return lbOK;

} // / delete()

/**
 * selectAll : renvoie tous les enregistrements
 *
 * @return une String
 * (devrait renvoyer une List d'objets,
 * une List de Ville en l'occurrence)
 */
@Deprecated
public String selectAll() {

    StringBuilder lsbResultat = new StringBuilder();

    // --- Tous les enregistrements
    try {
        String[] cols = { "cp", "nom_ville" , "id_pays"};

        Cursor curseur = this.ibd.query(TABLE_NAME, cols, null,
null, null, null, null);

        // Cursor curseur =
// this.ibd.rawQuery("SELECT cp, nom_ville FROM villes",
null);

        while (curseur.moveToNext()) {
            lsbResultat.append(curseur.getString(0));
            lsbResultat.append(" - ");
            lsbResultat.append(curseur.getString(1));
            lsbResultat.append(" - ");
            lsbResultat.append(curseur.getString(2));
            lsbResultat.append("\n");
        }
    } catch (SQLException e) {
        lsbResultat.append("Erreur de lecture ");
    }

    return lsbResultat.toString();

} // / selectAll()
```

```
/**
 * selectOne : un enregistrement
 *
 * @param asCP
 * @return : une ville
 */
public Ville selectOne(String asCP) {

    Ville ville;
    // --- Un enregistrement
    try {
        String[] cols = { "id_ville", "cp", "nom_ville" , "id_pays"};
        String[] tValeurs = { asCP };

        // query(String table, String[] columns, String selection,
String[]
        // selectionArgs, String groupBy, String having, String
orderBy)
        Cursor curseur = this.ibd.query(TABLE_NAME, cols, "cp=?",
tValeurs, null, null, null);

        if (curseur.moveToNext()) {
            ville = new Ville(curseur.getInt(0), asCP,
curseur.getString(2),curseur.getString(3));
        } else {
            ville = new Ville(0, " ", "", "");
        }
    } catch (SQLException e) {
        ville = new Ville(0, "Erreur de lecture ", e.getMessage(),
        "");
    }
    return ville;
} // / selectOne()

} // / class VilleDAO
```

## 1.2.17.9 - Le « connector »

**GestionnaireOuvertureSQLite.java**

```
package fr.pb.sqllocal;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

/*
 * Classe Helper pour la gestion de l'ouverture de la BD
 * La connexion a la BD en somme ou la creation et connexion si elle
n'existe pas
 * Methodes : constructeur, onCreate, onUpgrade
 */

// -----
public class GestionnaireOuvertureSQLite extends SQLiteOpenHelper {
    private static final int DB_VERSION = 2;
    private static final String DB_NAME = "cours.db";

    private static final String DROP_TABLE_PAYS = "DROP TABLE IF EXISTS
pays";
    private static final String CREATE_TABLE_PAYS = "CREATE TABLE IF NOT
EXISTS pays(id_pays TEXT PRIMARY KEY, nom_pays TEXT)";

    private static final String DROP_TABLE_VILLE = "DROP TABLE IF EXISTS
ville";
    private static final String CREATE_TABLE_VILLE = "CREATE TABLE IF NOT
EXISTS ville(id_ville INTEGER PRIMARY KEY AUTOINCREMENT, cp TEXT,
nom_ville TEXT, id_pays TEXT, FOREIGN KEY(id_pays) REFERENCES
pays(id_pays))";

    // --- Constructeur
    // -----
    public GestionnaireOuvertureSQLite(Context contexte, CursorFactory
fabrique) {
        // --- Cree la BD si elle n'existe pas
        // --- et de ce fait execute le code de onCreate()
        // --- Se connecte si elle existe
        super(contexte, DB_NAME, fabrique, DB_VERSION);
    } // GestionnaireOuvertureSQLite()

    @Override
    // -----
    public void onCreate(SQLiteDatabase abd) {
        // --- Creation de la table pays
        abd.execSQL(CREATE_TABLE_PAYS);
        // --- Creation de la table villes
        abd.execSQL(CREATE_TABLE_VILLE);

        /*
         * Pour les tests
         */
        // --- Remplissage table pays
    }
}
```

```
        abd.execSQL("INSERT INTO pays(id_pays, nom_pays)
VALUES('33','France')");
        abd.execSQL("INSERT INTO pays(id_pays, nom_pays)
VALUES('39','Italie')");
        // --- Remplissage table ville
        abd.execSQL("INSERT INTO ville(cp, nom_ville, id_pays)
VALUES('75001','Paris I','33')");
        abd.execSQL("INSERT INTO ville(cp, nom_ville, id_pays)
VALUES('75011','Paris XI','33')");
        abd.execSQL("INSERT INTO ville(cp, nom_ville, id_pays)
VALUES('75012','Paris XII','33')");
        abd.execSQL("INSERT INTO ville(cp, nom_ville, id_pays)
VALUES('75020','Paris XX','33')");
    } /// onCreate()

    @Override
    // -----
    public void onUpgrade(SQLiteDatabase abd, int ancienneVersion, int
nouvelleVersion) {
        // --- Suppression des tables puis appel a la creation des tables
        abd.execSQL(DROP_TABLE_VILLE);
        abd.execSQL(DROP_TABLE_PAYS);
        onCreate(abd);
    } /// onUpgrade()

} /// class GestionnaireOuvertureSQLite
```

## 1.2.17.10 - L'activité

**VilleCRUD.java**

```
package fr.pb.sqllocal;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.TextView;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;

/*
 * Ptit CRUD!!!
 * Methodes : onCreate, onClick, initInterface
 */
public class VilleCRUD extends Activity implements OnClickListener {
    /*
     * Attributs
     */
    private GestionnaireOuvertureSQLite gos;
    private SQLiteDatabase ibd;
    private VilleDAO idao;
    private Context contexte;

    private boolean ibOK;

    private ImageButton buttonSeConnecter;
    private ImageButton buttonSeDeconnecter;
    private ImageButton buttonAjouter;
    private ImageButton buttonSupprimer;
    private ImageButton buttonVoir;

    private EditText editTextCp;
    private EditText editTextNomVille;
    private EditText editTextIdPays;

    private TextView textViewMessage;

    @Override
    // -----
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.ville_crud);

        // -----
        // --- Initialisation de l'interface statique
        // -----
        initInterface();

        /*
         * EN TEST
         */
    }
}
```

```

        */
        editTextCp.setText("75021");
        editTextNomVille.setText("Paris 21");
        editTextIdPays.setText("33");

        // -----
        // --- Initialisation des variables de programme
        // -----
        this.contexte = getBaseContext();
        this.ibd = null;
    } // / onCreate()

    @Override
    // -----
    public void onClick(View vue) {

        String lsMessage = "";

        if (vue == buttonSeConnecter) {
            try {
                // --- GestionnaireOuvertureSQLite(Contexte,
                // Fabrique de curseur);
                this.gos = new
                GestionnaireOuvertureSQLite(this.contexte, null);
                this.ibd = gos.getWritableDatabase();
                this.idao = new VilleDAO(this.ibd);

                lsMessage = "Connexion réussie";
            } catch (Exception e) {
                lsMessage = "Connexion ratée : " + e.getMessage();
            }
        } // / if buttonSeConnecter

        if (vue == buttonSeDeconnecter) {
            try {
                this.gos.close();
                this.ibOK = false;
                this.ibd = null;
                lsMessage = "Vous êtes déconnecté(e) !";
            } catch (Exception e) {
                lsMessage = "Erreur Déconnexion : " +
                e.getMessage();
            }
        } // / if buttonSeDeconnecter

        if (vue == buttonAjouter) {
            try {
                if (this.ibd != null) {
                    Ville ville = new
                    Ville(this.editTextCp.getText().toString(),
                    this.editTextNomVille.getText().toString(),
                    this.editTextIdPays.getText().toString());
                    ibOK = idao.insert(ville);
                    if (ibOK) {
                        lsMessage = "Insertion OK";
                    } else {
                        lsMessage = "Insertion KO";
                    }
                } else {
                    lsMessage = "Vous devez être connecté(e) !";
                }
            }
        }
    }

```



```

        } catch (Exception e) {
            lsMessage = "Erreur Déconnexion : " +
e.getMessage();
        }
    } // / if buttonAjouter

    if (vue == buttonSupprimer) {
        try {
            if (this.ibd != null) {
                ibOK =
idao.delete(editTextCp.getText().toString());
                if (ibOK) {
                    lsMessage = "Suppression OK !";
                } else {
                    lsMessage = "Suppression KO !";
                }
            } else {
                lsMessage = "Vous devez être connecté(e) !";
            }
        } catch (Exception e) {
            lsMessage = "Erreur Delete : " + e.getMessage();
        }
    } // / if buttonSupprimer

    if (vue == buttonVoir) {
        Ville ville = null;

        try {
            if (this.ibd != null) {
                // selectAll
                if
(editTextCp.getText().toString().equals("")) {
                    lsMessage = idao.selectAll();
                    if (lsMessage.equals("")) {
                        lsMessage = "Aucun
enregistrement";
                    }
                }
                // selectOne
                else {
                    ville =
idao.selectOne(editTextCp.getText().toString());
                    if (ville.getNomVille().equals("")) {
                        lsMessage = "Aucun
enregistrement";
                    } else {
                        lsMessage = ville.toString();
                    }
                }
            } else {
                lsMessage = "Vous devez être connecté(e) !";
            }
        } catch (Exception e) {
            lsMessage = e.getMessage();
        }

    } // / if buttonVoir
    textViewMessage.setText(lsMessage);
} // / onClick()

// -----

```

```
private void initInterface() {
    // --- Liaison "variables" et composants du layout
    buttonSeConnecter = (ImageButton)
findViewById(R.id.buttonSeConnecter);
    buttonSeDeconnecter = (ImageButton)
findViewById(R.id.buttonSeDeconnecter);
    buttonAjouter = (ImageButton) findViewById(R.id.buttonAjouter);
    buttonSupprimer = (ImageButton)
findViewById(R.id.buttonSupprimer);
    buttonVoir = (ImageButton) findViewById(R.id.buttonVoir);

    editTextCp = findViewById(R.id.editTextCp);
    editTextNomVille = findViewById(R.id.editTextNomVille);
    editTextIdPays = findViewById(R.id.editTextIdPays);

    textViewMessage = findViewById(R.id.textViewMessage);

    /*
     * Les liaisons événementielles
     */
    buttonSeConnecter.setOnClickListener(this);
    buttonSeDeconnecter.setOnClickListener(this);
    buttonAjouter.setOnClickListener(this);
    buttonSupprimer.setOnClickListener(this);
    buttonVoir.setOnClickListener(this);

} // / initInterface

} // / classe SQLiteExemple
```

### 1.2.18 - Exercice

#### **Objectif**

Corrigez les erreurs de conception du DAO Villes.

Codez le UPDATE dans le DAO et dans l'IHM.

Codez des jointures entre Pays et Ville (une ville et toutes les villes).

## 1.3 - LE CONTENT PROVIDER : FOURNIR DES DONNÉES À D'AUTRES APPLICATIONS

### 1.3.1 - Définition

<http://developer.android.com/guide/topics/providers/content-providers.html>

<http://developer.android.com/reference/android/content/ContentProvider.html>

Un Content Provider stocke et gère des données et les rend accessibles à toutes les applications. On peut traduire Content Provider par Fournisseur de données.

Les données peuvent être un ou des fichiers ou une BD gérée par SQLite ou autre.

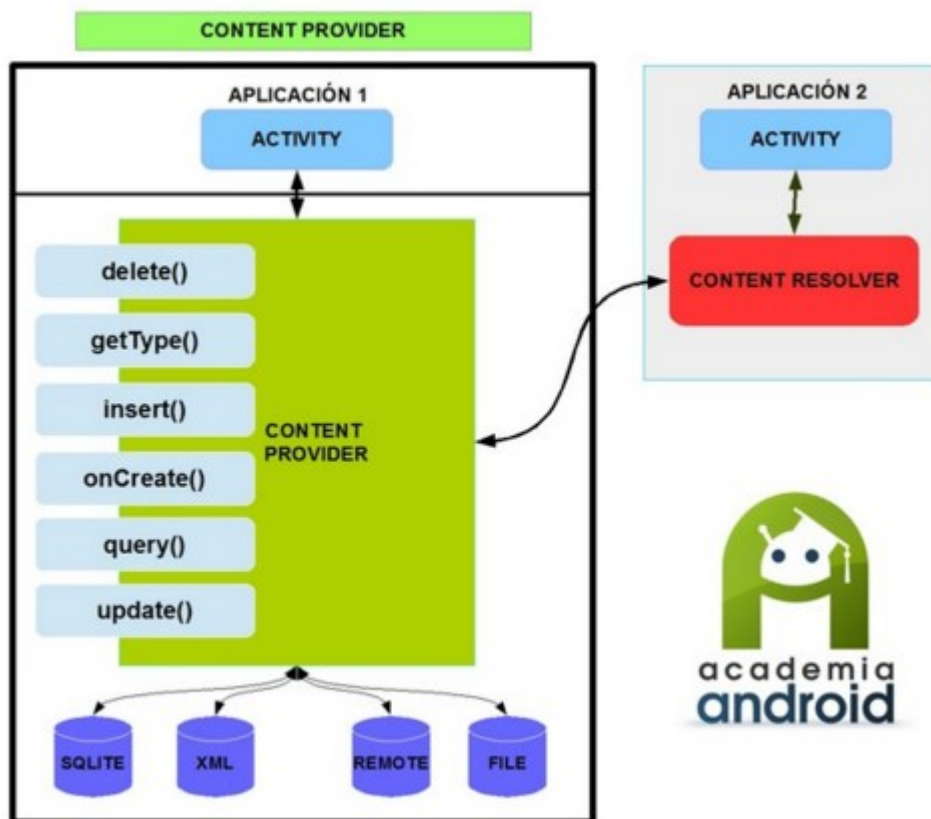
Android fournit plusieurs ContentProviders : entre autres le Contacts Provider (cf plus loin) et le Calendar Provider.

Une BD SQLite sera stockée dans le dossier suivant :

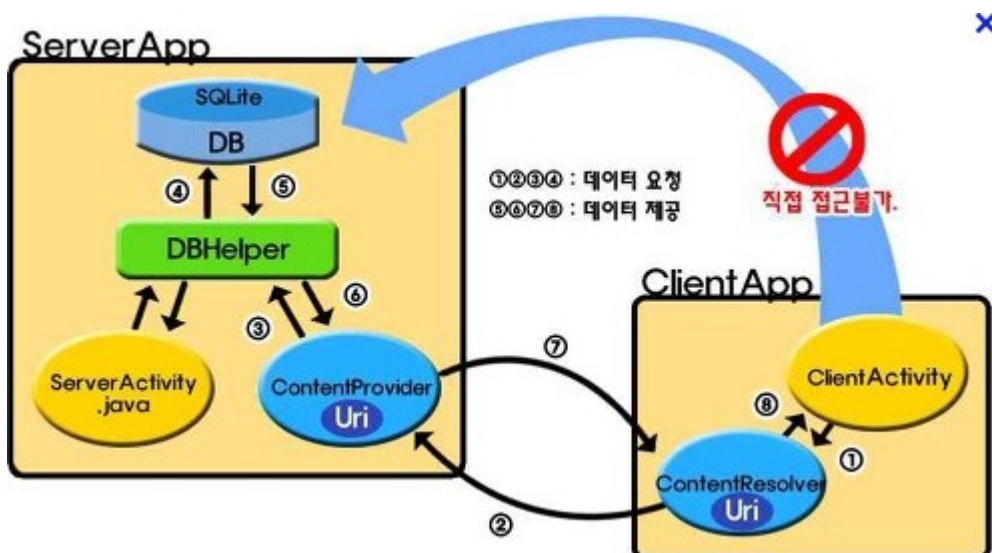
/data/data/nom.du.package/databases/

Par exemple : /data/data/fr.pb.sql/databases/cours.db

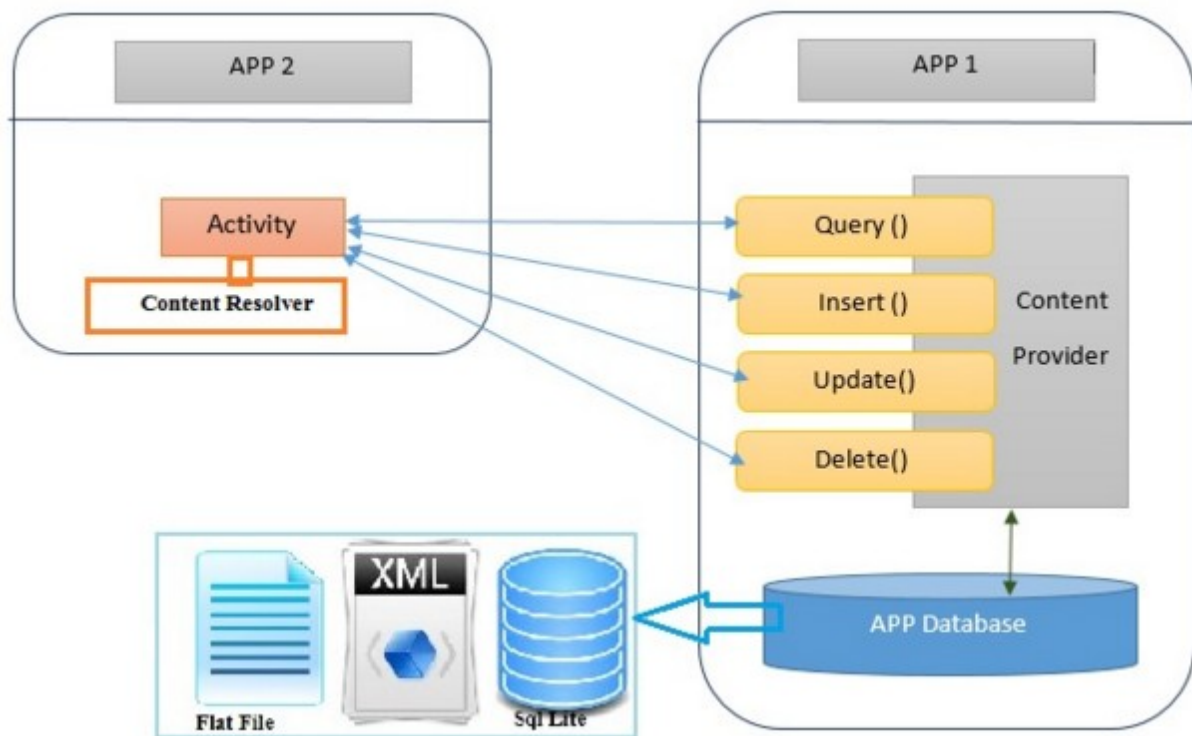
## 1.3.2 - Schémas



source : <http://academiaandroid.com/uso-de-un-content-provider-personalizado/>

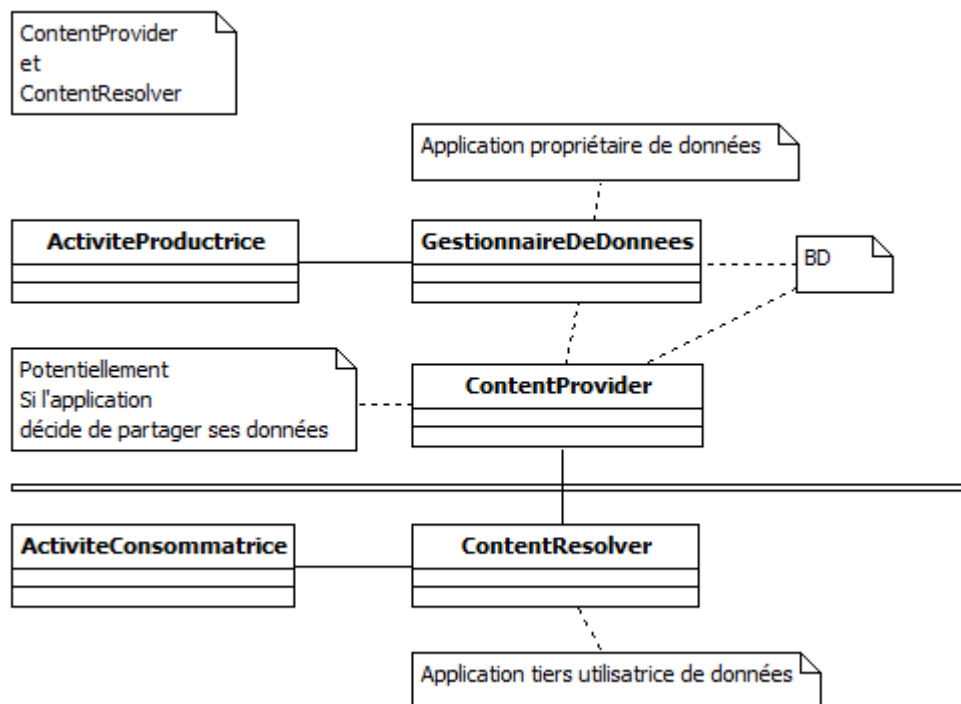


Source : <http://sgro.tistory.com/4>



source : <http://www.codeproject.com/Articles/818578/An-Absolute-Beginner-s-Guide-to-Building-and-Access>

---

1.3.3 - Diagramme

### 1.3.4 - Création

Pour créer un Content Provider il faut :

- ✓ Paramétrer votre système pour stocker les données,
- ✓ Étendre la classe `ContentProvider` pour accéder aux données,
- ✓ Déclarer le Content Provider dans le fichier manifeste de l'application (`AndroidManifest.xml`).

#### La classe qui fournit les données

C'est une classe qui hérite de la classe `ContentProvider`, qui permet de gérer vos données, et qui doit implémenter 6 méthodes :

<code>boolean onCreate()</code>	Appelée pour initialiser le fournisseur
<code>Cursor query()</code>	Exécute un ordre <code>SELECT</code> et renvoie un curseur ou null
<code>URI insert()</code>	Exécute un ordre <code>INSERT</code> et renvoie l'URI du nouvel enregistrement
<code>int update()</code>	Exécute un ordre <code>UPDATE</code> et renvoie le nombre de lignes modifiées
<code>int delete()</code>	Exécute un ordre <code>DELETE</code> et renvoie le nombre de lignes supprimées
<code>String getType()</code>	Renvoie le type MIME des données

Elle est dans le projet de gestion de la BD SQLite (cf section précédente).

#### Le fichier `AndroidManifest.xml`

```
<provider android:name="la classe fournisseur"
  android:authorities="Fournisseur"
  android:enabled="true"
  android:exported="true">
</provider>
```

Exemple de déclaration dans le fichier `AndroidManifest.xml` dans l'élément `<application>` :

```
<provider
  android:name=".FournisseurSQLite"
  android:authorities="fr.pb.sql.FournisseurSQLite"
  android:enabled="true"
  android:exported="true">
</provider>
```

L'exemple qui suit est volontairement simplifié. Seules les méthodes `onCreate()`, `query()` et `delete()` sont réellement codées.

#### Remarque :

pour observer les résultats il faut attendre la prochaine section sur le `ContentResolver`.



---

### 1.3.5 - Syntaxes

**Publication de l'URI pour ce fournisseur. Elle sert au consommateur pour accéder au fournisseur. C'est une constante publique de la classe qui étend le ContentProvider.**

```
public static final Uri CONTENT_URI = Uri.parse("content://fr.pb.sql.FournisseurSQLite");
```

**Indispensable** pour que les applications utilisatrices trouvent le fournisseur.

#### La méthode onCreate()

```
boolean onCreate()
```

Utilisée pour se connecter à la BD.

#### La méthode getType()

```
String getType(Uri uri)
```

Renvoie le type MIME d'un résultat.

#### La méthode query()

```
Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

Renvoie un Cursor

Dans cet exemple le nom de la table est codé "en dur".

```
curseur = this.ibd.query("villes", null, null, null, null, null, null);
```

Dans cet exemple le nom de la table sera envoyé au moment de l'appel à la méthode.

```
curseur = this.ibd.query(uri.getLastPathSegment(), null, null, null, null, null, null);
```

parce l'URI est ainsi aussi bien dans le Provider que dans le Resolver : Uri uri = Uri.parse("content://FournisseurSQLite/villes");  
cf l'exemple page 52 :

```
Uri uri = Uri.parse("content://FournisseurSQLite/villes");
```

### La méthode insert()

Uri insert(Uri uri, ContentValues values)

This method inserts a new row into the provider and returns a content URI for that row.

The content URI returned in newUri identifies the newly-added row, with the following format:  
content://user\_dictionary/words/<id\_value>

The <id\_value> is the contents of \_ID for the new row. Most providers can detect this form of content URI automatically and then perform the requested operation on that particular row.

To get the value of \_ID from the returned Uri, call ContentUris.parseId().

Ajoute une ligne avec les clés/valeurs passées dans values sur l'uri.

Renvoie une Uri.

Notes :

URI (Uniform Resource Identifier) est une chaîne de caractères identifiant une ressource sur un réseau.

URL (Uniform Resource Locator), désigne une chaîne de caractères utilisée pour adresser les ressources du World Wide Web : document HTML, image, son, boîte aux lettres électronique, etc.

Les URL constituent un sous-ensemble des URI. La syntaxe d'une URL est décrite dans la RFC 3986.

### La méthode delete()

int delete(Uri uri, String selection, String[] selectionArgs)

Supprime une ou plusieurs lignes en fonction d'un critère de sélection simple ou complexe.

Renvoie le nombre de lignes supprimées.

### La méthode update()

int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)

Modifie une ou plusieurs lignes en fonction d'un critère de sélection simple ou complexe.

Renvoie le nombre de lignes modifiées.

### 1.3.6 - Codes

```
package fr.pb.sqllocal;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.net.Uri;

/*
 * Classe qui sert de fournisseur de données SQL (SQLite en l'occurrence)
 *
 * Methodes : onCreate, query, insert, update, delete, getType
 */
public class FournisseurSQLite extends ContentProvider {

    private SQLiteDatabase db;

    public static final Uri CONTENT_URI =
Uri.parse("content://fr.pb.sql.FournisseurSQLite");

    @Override
    public boolean onCreate() {
        Context context = getContext();
        // Connexion à la BD
        GestionnaireOuvertureSQLite dbHelper = new
GestionnaireOuvertureSQLite(context, null);
        this.db = dbHelper.getWritableDatabase();
        return (this.db == null) ? false : true;
    } /// onCreate

    @Override
    public Cursor query(Uri uri, String[] projection, String restriction,
String[] restrictionArgs, String sortOrder) {
        Cursor curseur = null;
        // --- Tous les enregistrements
        try {
            // --- query(table, tColonnes, sWhere, tParamsWhere, sGroupBy,
sHaving, sOrderBy)
            curseur = this.db.query("ville", projection, restriction,
restrictionArgs, null, null, sortOrder);
        }
        catch(SQLiteException e) {
            curseur = null;
        }
        return curseur;
    } /// query

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // A modifier lors de l'exercice
        return null;
    } /// insert
```

```
@Override
public int update(Uri uri, ContentValues values, String restriction,
String[] restrictionArgs) {
    // A modifier lors de l'exercice
    return 0;
} /// update

@Override
public int delete(Uri uri, String restriction, String[]
restrictionArgs) {

    // --- delete(table, sWhere, tWhere)
    int count = this.db.delete("villes", "cp=?", restrictionArgs);
    return count;
} /// delete

@Override
public String getType(Uri uri) {
    // Renvoie le type MIME d'un résultat
    return null;
} /// getType

} /// FournisseurSQLite
```

---

### 1.3.7 - Exercices : ContentProvider

Complétez les méthodes insert() et update().

Créez un ContentProvider pour un fichier CSV. Nommez-le FournisseurDeContenuCSV.

## 1.4 - EXPLOITATION D'UN CONTENT PROVIDER : LE CONTENT RESOLVER

### 1.4.1 - Objectif

Afficher le contenu de la table Villes.



### 1.4.2 - Démarche

Création d'une nouvelle application nommée **AndroidSQLiteContentResolver**.  
Dans un package nommé **fr.pb.androidsqlitecontentresolver**.  
Création d'un layout nommé **content\_resolver\_sqlite.xml** avec une ListView et deux TextView.  
Création d'une classe Activity nommée **ContentResolverSQLite**.  
Codage de la méthode onCreate() pour afficher la liste des villes dans la ListView.

### 1.4.3 - Syntaxes

#### Obtenir un ContentResolver

```
ContentResolver cr = getContentResolver();
```

#### Composer une URI

```
Uri uri = Uri.parse("content://chemin");
```

chemin correspond à la valeur de android:authorities dans le Manifest du fournisseur.

Par exemple : <provider android:name="fr.pb.sql.FournisseurSQLite"  
android:authorities="FournisseurSQLite">

```
Uri uri = Uri.parse("content://FournisseurSQLite");
```

si le nom de la table est "en dur" dans le ContentProvider.

ou avec un nom de table en plus

```
Uri uri = Uri.parse("content://FournisseurSQLite/villes");
```

si le nom de la table est paramétré dans le ContentProvider.

#### Récupérer un curseur

```
Cursor curseur = cr.query(uri, projection, restriction, arguments de la  
restriction, orderBy);
```

Equivalent de SELECT \* FROM table :

```
Cursor curseur = cr.query(uri, null, null, null, null);
```

Note : cf aussi CursorLoader pour charger un curseur dans un thread à part.

#### Supprimer un ligne

```
int liSupprime = cr.delete(uri, lsWhere, tArgsWhere);
```

---

### 1.4.4 - Codes

#### Le layout : content\_resolver\_sqlite.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:paddingTop="5dp"
        android:text="La liste de villes"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="#7FFF00"
        android:textSize="30sp"
        android:textStyle="bold" />

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message"
        android:textColor="#ED7F10"
        android:textSize="20sp" />

</LinearLayout>
```

**ContentResolverSQLite.java**

```
package fr.pb.androidsqlitecontentresolver;

import java.util.ArrayList;
import java.util.List;

import android.net.Uri;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.TextView;
import android.app.Activity;
import android.content.ContentResolver;
import android.database.Cursor;

/*
 * ContentResolverSQLite : un ContentResolver
 * Utilisation des donnees du ContentProvider suivant :
 * fr.pb.sql.FournisseurSQLite
 */
public class ContentResolverSQLite extends Activity {

    private ListView listView;
    private TextView textViewMessage;
    private List<String> liste;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.content_resolver_sqlite);

        listView = (ListView) findViewById(R.id.listView);
        textViewMessage = findViewById(R.id.textViewMessage);

        StringBuilder lsbContenu = new StringBuilder();
        liste = new ArrayList<String>();

        try {
            /*
             * Exemple avec le ContentResolver SQL
             */
            ContentResolver cr = getContentResolver();

            /*
             * Dans le ContentProvider
             * public static final Uri CONTENT_URI =
             * Uri.parse("content://fr.pb.sql.FournisseurSQLite");
             */

            String lsURI = "content://fr.pb.sql.FournisseurSQLite";

            Uri uri = Uri.parse(lsURI);

            String[] tCols = {"cp", "nom_ville"};

            // --- Utilisation de la methode query()
            // --- query(uri, tCols, restriction,
            tValeursRestriction, ORDER BY);
            // --- selectAll tries sur le CP
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```
        Cursor curseur = cr.query(uri, tCols, null, null, "cp");

        while (curseur.moveToNext()) {
            liste.add(curseur.getString(0) + " - " +
curseur.getString(1));
        }
        curseur.close();

        // --- Utilisation de la methode delete()
        String lsWhere = "cp=?";
        String[] tArgsWhere = {"75021"};
        int liSupprime = cr.delete(uri, lsWhere, tArgsWhere);
        lsbContenu.append("\nSupprimé(s) : " +
Integer.toString(liSupprime));

        ArrayAdapter<String> aaListe = new
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, liste);

        listView.setAdapter(aaListe);
    } catch (Exception e) {
        lsbContenu.append("Erreur : " + e.getMessage());
    }

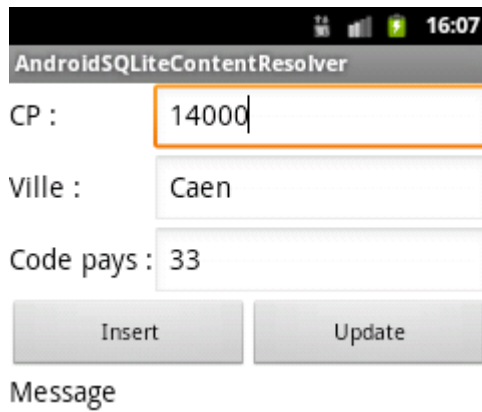
    textViewMessage.setText(lsbContenu.toString());

} // / onCreate
} // / classe ContentResolverSQLite
```

---

### 1.4.5 - Exercices : ContentResolver

- 1) Modifiez l'ordre query() du ContentResolver afin de faire un SelectOne.
- 2) Modifiez le ContentResolver pour tester le code des méthodes ajoutées dans l'exercice précédent.



AndroidSQLiteContentResolver

CP : 14000

Ville : Caen

Code pays : 33

Insert Update

Message

## 1.5 - TP : SMS GROUPÉS

Envoyer des SMS à partir d'une table d'une BD SQLite en fonction d'une catégorie.

**Note** : il faut savoir envoyer des SMSs !!!

Le formulaire pour gérer les contacts avec la liste déroulante pour choisir la catégorie.

(+) pour ajouter un contact,

(-) pour supprimer un contact,

(Voir) pour visualiser les contacts,

(Envoi) pour envoyer des SMS en rafale.

La zone de texte multi-lignes pour l'affichage ...



La BD SQLite : repertoire.db.

La table [correspondant] avec les colonnes id\_correspondant, nom, telephone, id\_categorie.

La table [categorie] avec les colonnes id\_categorie, categorie.

Les catégories sont : famille, ami, professionnel, autre.

## CHAPITRE 2 - ANNEXES

## **2.1 - LE CONTENT RESOLVER DES CONTACTS**

---

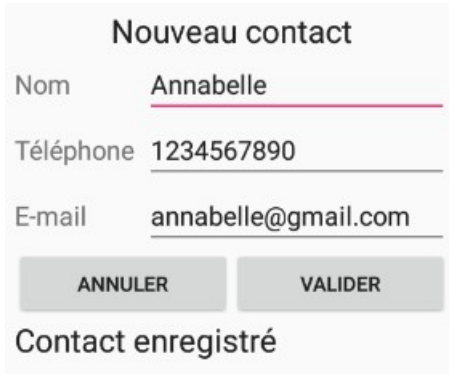
### **2.1.1 - Présentation**

---

## 2.1.2 - Insérer un contact

### 2.1.2.1 - Objectif

Insérer un contact.



Nouveau contact

Nom Annabelle

Téléphone 1234567890

E-mail annabelle@gmail.com

ANNULER VALIDER

Contact enregistré

A ajouter dans le fichier AndroidManifest :

```
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

L'activité sera nommée ainsi :  
ContactsInsert.java

Et le layout sera nommé ainsi :  
contacts\_insert.xml

#### 2.1.2.2 - Syntaxes

--

### 2.1.2.3 - Le layout : contacts\_insert.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:orientation="horizontal">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nouveau contact"
            android:textAppearance="?android:attr/textAppearanceLarge" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="30"
            android:text="Nom"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/editTextNom"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="70"
            android:ems="10">

            <requestFocus />
        </EditText>
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="30"
            android:text="Téléphone"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <EditText
            android:id="@+id/editTextTelephone"
            android:layout_width="0dp"
```



```
        android:layout_height="wrap_content"
        android:layout_weight="70"
        android:ems="10" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="30"
        android:text="E-mail"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editTextEmail"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="70"
        android:ems="10" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
        android:id="@+id/buttonAnnuler"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="50"
        android:text="Annuler" />

    <Button
        android:id="@+id/buttonValider"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="50"
        android:text="Valider" />
</LinearLayout>

<TextView
    android:id="@+id/textViewMessage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Message"
    android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>
```

## 2.1.2.4 - L'activité : ContactsInsert.java

```
import android.os.Bundle;
import android.app.Activity;
import android.provider.ContactsContract;
import android.content.ContentProviderOperation;
import android.widget.*;
import android.view.View;
import android.view.View.*;
import java.util.ArrayList;

/**
 *
 */
public class ContactsInsert extends Activity implements OnClickListener {

    // Attributs pour les widgets
    private EditText editTextNom;
    private EditText editTextTelephone;
    private EditText editTextEmail;
    private Button buttonValider;
    private Button buttonAnnuler;
    private TextView textViewMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.contacts_insert);
        initInterface();
        initEvents();

        /*
        En test
        */
        editTextNom.setText("Annabelle");
        editTextTelephone.setText("1234567890");
        editTextEmail.setText("annabelle@gmail.com");

    } // / onCreate

    /**
     *
     */
    private void initInterface() {
        // Liaison widget <--> Attribut
        editTextNom = findViewById(R.id.editTextNom);
        editTextTelephone = findViewById(R.id.editTextTelephone);
        editTextEmail = findViewById(R.id.editTextEmail);
        buttonValider = findViewById(R.id.buttonValider);
        buttonAnnuler = findViewById(R.id.buttonAnnuler);
        textViewMessage = findViewById(R.id.textViewMessage);
    } // / initInterface

    /**
     *
     */
}
```

```

private void initEvents() {
    // Liaison widget <--> Events
    buttonValider.setOnClickListener(this);
    buttonAnnuler.setOnClickListener(this);
} // / initEvents

@Override
public void onClick(View v) {

    if (v == buttonValider) {
        contactInsert();
    }

} // / onClick

/**
 *
 */
private void contactInsert() {

    String nom = editTextNom.getText().toString();
    String telephoneMobile = editTextTelephone.getText().toString();
    String email = editTextEmail.getText().toString();

    boolean lbOK = true;

    if(nom.trim().equals("") || telephoneMobile.trim().equals("") ||
email.trim().equals("")){
        lbOK = false;
    }

    if(lbOK) {
        // Une ArrayList surtout pas une List
        // Represents a single operation to be performed as part of a
batch of operations.
        //
http://developer.android.com/reference/android/content/ContentProviderOperation.html
        ArrayList<ContentProviderOperation> listeCPO = new
ArrayList<ContentProviderOperation>();

        // Un nouveau contact
        listeCPO.add(ContentProviderOperation.newInsert(
            ContactsContract.RawContacts.CONTENT_URI)
            .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE,
null)
            .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
null).build());

        // Nom
        // Note the use of withValueBackReference(String, int) to
refer to the as-yet-unknown index value of the raw contact inserted in the
first operation.
        listeCPO.add(ContentProviderOperation.newInsert(
            ContactsContract.Data.CONTENT_URI)
            .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
            .withValue(ContactsContract.Data.MIMETYPE,

```

```

ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
    .withValue (

ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
nom).build());

    // Telephone Mobile
    listeCPO.add(ContentProviderOperation.
        newInsert(ContactsContract.Data.CONTENT_URI)
        .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
        .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
        .withValue(ContactsContract.CommonDataKinds.Phone.NUMB
ER, telephoneMobile)
        .withValue(ContactsContract.CommonDataKinds.Phone.TYPE
,
ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE).build());

    // Email

    listeCPO.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONT
ENT_URI)
        .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
        .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
        .withValue(ContactsContract.CommonDataKinds.Email.DATA
, email)
        .withValue(ContactsContract.CommonDataKinds.Email.TYPE
, ContactsContract.CommonDataKinds.Email.TYPE_HOME)
        .build());

    // Asking the Contact provider to create a new contact
    try {
        getContentResolver().applyBatch(ContactsContract.AUTHORITY,
listeCPO);

        Toast.makeText(this, "Contact enregistré",
Toast.LENGTH_SHORT).show();
        textViewMessage.setText("Contact enregistré");
    } catch (Exception e) {
        Toast.makeText(this, "Exception: " + e.getMessage(),
Toast.LENGTH_SHORT).show();
        textViewMessage.setText("Exception: " + e.getMessage());
    }
    else {
        Toast.makeText(this, "Toutes les zones de saisie sont
obligatoires", Toast.LENGTH_SHORT).show();
        textViewMessage.setText("Toutes les zones de saisie sont
obligatoires");
    }
} /// contactInsert

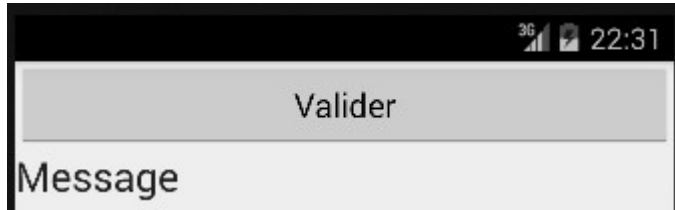
```

```
} // / class
```

---

### 2.1.3 - Insérer des contacts

**Objectif** : insérer des contacts.



Utile pour des tests avec un émulateur qui ne sauvegarde pas les contacts. Et aussi un premier pas pour récupérer des contacts d'une BD MySQL ou d'un fichier CSV.

L'activité sera nommée ainsi :  
ContactsInserts.java

Et le layout sera nommé ainsi :  
contacts\_inserts.xml

Le layout : contacts\_inserts.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/buttonValider"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Valider" />

    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>
```

## L'activité : ContactsInserts.java

```
package pb.fr.contentresolvercontacts;

import android.os.Bundle;
import android.app.Activity;
import android.provider.ContactsContract;
import android.content.ContentProviderOperation;
import android.widget.*;
import android.view.View;
import android.view.View.*;
import java.util.ArrayList;

/**
 *
 */
public class ContactsInserts extends Activity implements OnClickListener {

    // Attributs pour les widgets
    private Button buttonValider;
    private TextView textViewMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.contacts_inserts);
        buttonValider = findViewById(R.id.buttonValider);
        textViewMessage = findViewById(R.id.textViewMessage);
        buttonValider.setOnClickListener(this);

    } // / onCreate

    @Override
    public void onClick(View v) {
        contactInserts();
    } // / onClick

    /**
     *
     */
    private void contactInserts() {
        String nom = "";
        String telephoneMobile = "";
        String email = "";

        for (int i = 1; i <= 10; i++) {
            nom = "Tintin" + Integer.toString(i);
            telephoneMobile = "060000000" + Integer.toString(i);
            email = "tintin" + Integer.toString(i) + "@free.fr";
            // Une ArrayList surtoute pas une List
            // Represents a single operation to be performed as part of a
            batch of operations.
            //
http://developer.android.com/reference/android/content/ContentProviderOperation.html
            ArrayList<ContentProviderOperation> listeCPO = new
            ArrayList<ContentProviderOperation>();
```



```
// Nouveau contact
listeCPO.add(ContentProviderOperation.newInsert(
    ContactsContract.RawContacts.CONTENT_URI)
    .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE,
null)
    .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,
null)
    .build());

// Nom
// Note the use of withValueBackReference(String, int) to
refer to the as-yet-unknown index value of the raw contact inserted in the
first operation.
listeCPO.add(ContentProviderOperation.newInsert(
    ContactsContract.Data.CONTENT_URI)
    .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
    .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
    .withValue(
ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
nom).build());

// Telephone Mobile
listeCPO.add(ContentProviderOperation.
    newInsert(ContactsContract.Data.CONTENT_URI)
    .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
    .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
    .withValue(ContactsContract.CommonDataKinds.Phone.NUMB
ER, telephoneMobile)
    .withValue(ContactsContract.CommonDataKinds.Phone.TYPE
,
ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE)
    .build());

// Email
listeCPO.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONT
ENT_URI)
    .withValueBackReference(ContactsContract.Data.RAW_CONT
ACT_ID, 0)
    .withValue(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
    .withValue(ContactsContract.CommonDataKinds.Email.DATA
, email)
    .withValue(ContactsContract.CommonDataKinds.Email.TYPE
, ContactsContract.CommonDataKinds.Email.TYPE_HOME)
    .build());

// Asking the Contact provider to create a new contact
try {
```

```
getContentResolver().applyBatch(ContactsContract.AUTHORITY, listeCPO);
        Toast.makeText(this, "Contacts enregistrés",
Toast.LENGTH_SHORT).show();
        textViewMessage.setText("Contacts enregistrés");
    } catch (Exception e) {
        //e.printStackTrace();
        Toast.makeText(this, "Exception: " + e.getMessage(),
Toast.LENGTH_SHORT).show();
        textViewMessage.setText("Exception: " + e.getMessage());
    }
    } /// for i
    } /// contactInserts
} // / class
```

---

## 2.1.4 - Visualiser les contacts (les noms)

<http://developer.android.com/guide/topics/providers/contacts-provider.html>

### 2.1.4.1 - Objectif

Utiliser le ContentResolver des contacts.

Dans un premier temps afficher la liste de contacts : l'ID et le NOM



Note :

Créez quelques contacts au préalable si cela est nécessaire.



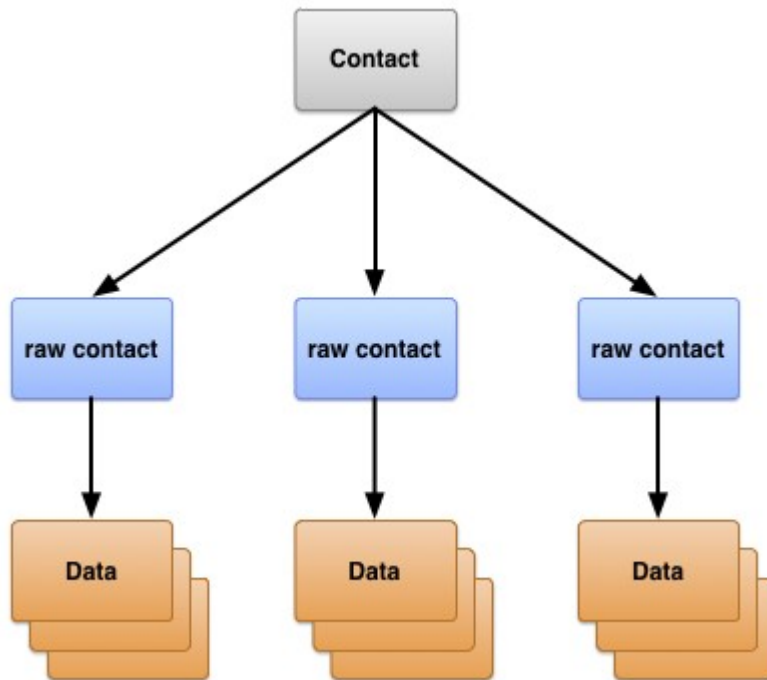
cf aussi le projet `ProjetImportExportContacts`.

#### 2.1.4.2 - Prémisses

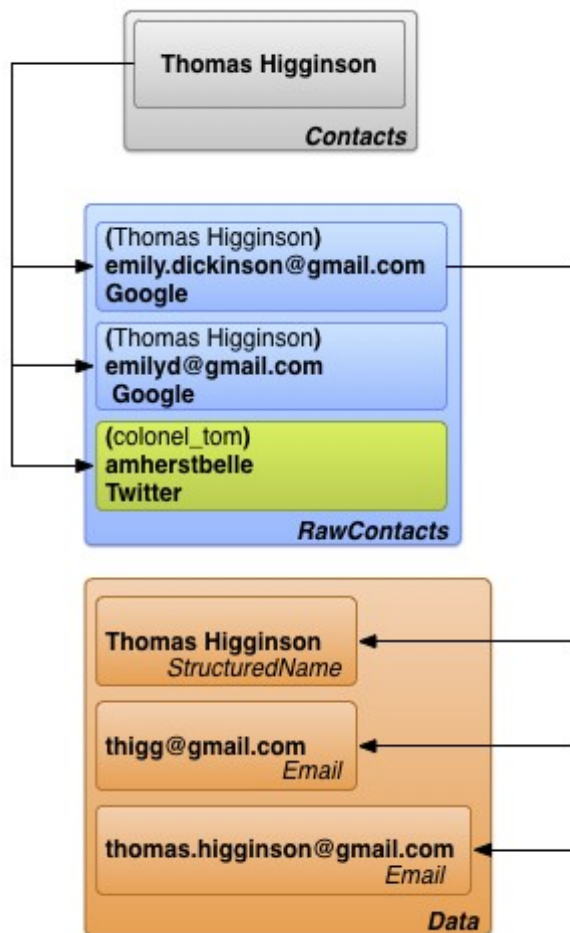
La BD des Contacts du téléphone est partagée en lecture/écriture.  
Vous pouvez donc créer une application pour la gérer.

Considérez qu'elle est composée de plusieurs tables puisque certaines informations sont multiples (téléphone, e-mail, etc) alors que d'autres sont uniques (nom, prénom, etc).

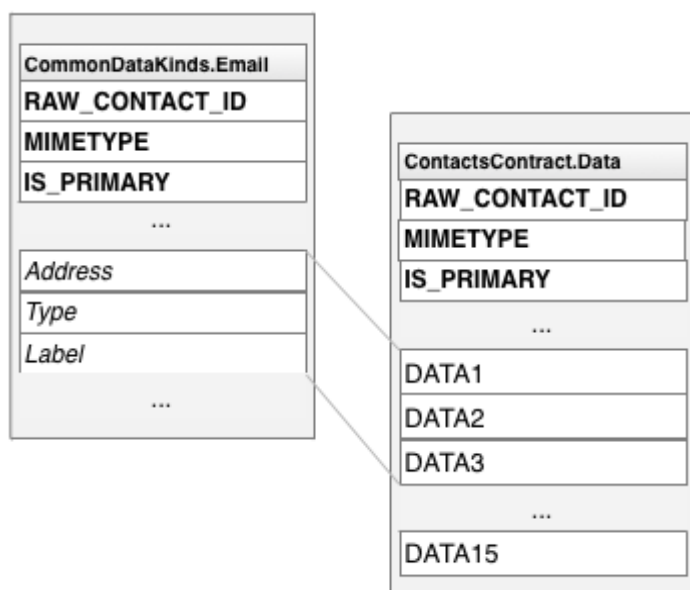
Schémas empruntés à <http://developer.android.com/guide/topics/providers/contacts-provider.html>



## 2.1.4.3 - Table Contacts



## 2.1.4.4 - Table Email et Data



#### 2.1.4.5 - Démarche

Créer un projet nommé **ContentResolverContacts**,  
créer un package nommé **fr.pb.contentresolvercontacts**,

créer un layout nommé **contacts\_select\_all**.  
créer une activité nommée **ContactsSelectAll**.

A ajouter dans le fichier Manifest.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Si vous souhaitez aussi faire du CUD avec les contacts il faudra ajouter cette permission :

```
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

coder.

#### 2.1.4.6 - Quelques syntaxes

##### Création d'un ContentResolver

```
ContentResolver cr = getContentResolver();
```

##### Composition de l'URI du ContentResolver

```
Uri uri = ContactsContract.Contacts.CONTENT_URI;
```

ou

```
Uri uri = Uri.parse("content://contacts/people");
```

##### Exécution d'une requête du type SELECT \* FROM table

```
Cursor cursor = cr.query(uri, null, null, null, null);
```

`query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`

##### Récupération du rang d'une colonne

```
int colonne =  
cursor.getColumnIndexOrThrow(ContactsContract.Contacts.NAME);
```

NAME peut avoir les valeurs suivantes : `_ID`, `DISPLAY_NAME`, `PHOTO_ID`, etc.



#### 2.1.4.7 - Le layout : contacts\_select\_all.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textViewContenu"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="TextView" />
    </ScrollView>

</LinearLayout>
```

Note : le ScrollView pour les autres ContentResolver.

#### 2.1.4.8 - L'activité : ContactsSelectAll.java

Affiche la liste de contacts : l'ID et le NOM

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.content.ContentResolver;
import android.database.Cursor;
import android.net.Uri;
import android.provider.ContactsContract;

/*
 * Utiliser le ContentProvider des Contacts
 */
public class ContactsSelectAll extends Activity {

    private TextView textViewContenu;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.contacts_select_all);

        textViewContenu = findViewById(R.id.textViewContenu);

        StringBuilder lsbContenu = new StringBuilder();
        lsbContenu.append("Les Contacts via un ContentResolver\n\n");

        try {
            // Un content resolver
            ContentResolver cr = getContentResolver();
            // L'URI des contacts
            Uri uri = ContactsContract.Contacts.CONTENT_URI;
            // Le SELECT pour obtenir tous les contacts
            // query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder)
            Cursor curseurContacts = cr.query(uri, null, null, null,
            ContactsContract.Contacts.DISPLAY_NAME + " DESC");

            // Recuperation des rangs de certaines colonnes
            int colonneID =
            curseurContacts.getColumnIndexOrThrow(ContactsContract.Contacts._ID);
            int colonneNom =
            curseurContacts.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_NA
            ME);

            // Balayage du curseur des contacts
            while (curseurContacts.moveToNext()) {
                // Recuperation de l'ID du contact

                lsbContenu.append(curseurContacts.getString(colonneID));
                lsbContenu.append(" : ");
                // Recuperation du nom du contact

                lsbContenu.append(curseurContacts.getString(colonneNom));
                lsbContenu.append("\n");
            }
            // Fermeture du curseur des contacts
        }
```

```
        curseurContacts.close();

    } catch (Exception e) {
        lsbContenu.append("Erreur : ");
        lsbContenu.append(e.getMessage());
    }

    textViewContenu.setText(lsbContenu.toString());

} // / onCreate

} /// class
```

#### 2.1.4.9 - Les noms et numéros de téléphone avec une projection

```
// Le ContentResolver
ContentResolver cr = getContentResolver();
// Les contacts via PHONE
Uri uriTelephones = Phone.CONTENT_URI;
// Le curseur des telephones
// query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
String[] projection = {Phone.DISPLAY_NAME, Phone.NUMBER};
Cursor curseurTelephones = cr.query(uriTelephones, projection, null, null,
Phone.DISPLAY_NAME);

// Balayage du curseur des telephones du contact courant
while (curseurTelephones.moveToNext()) {
    String nom = curseurTelephones.getString(0);
    String numeroDeTelephone = curseurTelephones.getString(1);
    // Concaténation ...
}
// Fermeture du curseur des téléphones
curseurTelephones.close();
```

---

### 2.1.5 - Exercices : noms, téléphones et emails

1) Récupérer les numéros de téléphone.



cf

## 2) Récupérer les e-mails.



cf

---

### 2.1.6 - Visualiser un contact

Le script est quasiment le même que celui qui affiche tous les contacts, c'est la requête qui change à laquelle il faut ajouter une restriction.

Pour le résultat un while n'est plus nécessaire, un if suffit.

Les valeurs des colonnes seront récupérées via leur rang dans ce script.



The screenshot shows a mobile application interface with a light gray background. At the top, the title "Un contact" is displayed in a dark gray font. Below the title, there is a label "Nom" followed by a text input field containing the text "Tintin1". A pink underline is visible under the input field. Below the input field, there is a gray button with the text "VALIDER" in white. At the bottom of the screen, the text "Tintin1 : 060000001" is displayed in a dark gray font.

```
private String contactRechercher(String nom) {
    StringBuilder lsbContenu = new StringBuilder();
    try {
        /*
        VIA PHONE
        */
        // Le ContentResolver
        ContentResolver cr = getContentResolver();
        // Les contacts via PHONE
        Uri uriTelephones =
ContactsContract.CommonDataKinds.Phone.CONTENT_URI;

        // Le curseur des telephones
        String[] projection =
{ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME,
ContactsContract.CommonDataKinds.Phone.NUMBER};
        String lsWhere = "display_name=?";
        String[] tWhere = {nom};

        // query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
        Cursor curseurTelephones = cr.query(uriTelephones, projection,
lsWhere, tWhere, ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME);

        // Si un seul resultat : la recherche doit etre faite sur des
valeurs uniques
        if (curseurTelephones.moveToNext()) {
            lsbContenu.append(curseurTelephones.getString(0));
            lsbContenu.append(" : ");
            lsbContenu.append(curseurTelephones.getString(1));
            lsbContenu.append("\n");
        } /// IF
        else {
            lsbContenu.append("Contact introuvable");
        }
        // Fermeture du curseur des telephones
        curseurTelephones.close();
    } catch (Exception e) {
        lsbContenu.append("Erreur : ");
        lsbContenu.append(e.getMessage());
    }
    return lsbContenu.toString();
} /// contactRechercher
```

A vous de mettre en place le layout, l'activité et cette méthode !



### 2.1.7 - Supprimer un contact

En principe via le ContentProvider et la méthode delete() comme pour l'insertion mais ... l'URI .phone ne le permet pas.

#### Syntaxe

#### Code

---

### 2.1.8 - TD-TP : contacts to CSV

Contacts to CSV

et

CSV to Contacts

cf projet ContactsToCSV.

## 2.2 - AUTRES CONTENTRESOLVER SYSTÈME

Il est possible de récupérer l'historique de navigation, l'historique des appels, les paramètres système, le dictionnaire, la liste des media, etc.

## 2.2.1 - L'historique du navigateur



La demande de permission à ajouter :

```
<uses-permission
android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
```

L'URI :

```
Uri uri = Uri.parse("content://browser/bookmarks");
```

ou

Quelques index de colonnes :

0 = index  
1 = alias + url  
2 = url

La boucle sur le curseur :

```
while (cursor.moveToNext()) {
    lsbContenu.append(cursor.getString(2));
    lsbContenu.append("\n\n");
}
```

Note : dans certains cas vous pourriez exécuter cette double boucle :

```
int liColonnes = cursor.getColumnCount();
while (cursor.moveToNext()) {
    for (int i = 0; i < liColonnes; i++) {
        lsbContenu.append(cursor.getString(i));
        lsbContenu.append("-");
    }
    lsbContenu.append("\n\n");
}
```

Mais dans le cas présent ça plante parce qu'il y a une colonne de type BLOB dans le curseur et il n'y a pas d'autoboxing BLOB/String.

---

### 2.2.2 - L'historique des appels téléphoniques

```
Uri uri = Uri.parse("content://call_log/calls");
```

ou

```
Uri uri = Calls.CONTENT_URI;
```

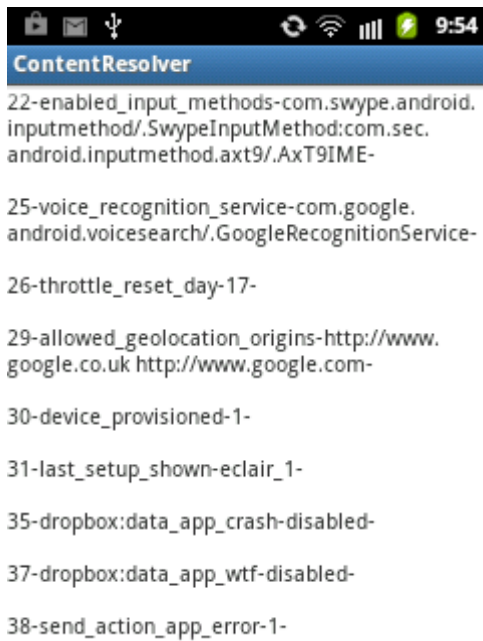
### 2.2.3 - Les media

```
Uri uri = Media.INTERNAL_CONTENT_URI;
```

ou

```
Uri uri = Media.EXTERNAL_CONTENT_URI;
```

## 2.2.4 - Les paramètres de sécurité



```
Uri uri = Uri.parse("content://settings/secure");
```

et la double boucle sur le curseur ...



### 2.2.5 - Données CSV, ContentProvider et ContentResolver

**TODO** : données CSV et ContentProvider et ContentResolver.

## 2.3 - LES TYPES DE CURSORS ANDROID

AbstractCursor,  
AbstractWindowedCursor,  
CrossProcessCursor,  
CrossProcessCursorWrapper,  
CursorWrapper,  
MatrixCursor,  
MergeCursor,  
MockCursor,  
SQLiteCursor.

## 2.4 - LE CURSORLOADER D'ANDROID

<http://developer.android.com/reference/android/content/CursorLoader.html>