Les bases de données

Les bases de données pour Android sont fournies à l'aide de SQLite. L'avantage de SQLite est qu'il s'agit d'un SGBD très compact et par conséquent très efficace pour les applications embarquées, mais pas uniquement puisqu'on le trouve dans Skype, Adobe Reader, Firefox, etc.
SQLite ne nécessite pas de serveur pour fonctionner, son exécution se fait dans le même processus que celui de l'application.

SQLite pour Android

SQLite a été inclus dans le cœur d'Android, ainsi chaque application peut avoir sa propre base de données.
La classe qui utilise une base de données doit hériter de la classe SQLiteOpenHelper.

En général, les bases de données sont stockées dans les répertoires de la forme `/DATA/data/<package>/databases`.

Une base de données **base1** est stockée dans /data/data//NOM_PACKAGE/databases/base1 où NOM_PACKAGE est le nom du package de l'application Android (celui indiqué dans AndroidManifest.xml)

Il est possible d'avoir plusieurs bases de données par application, cependant chaque fichier créé l'est selon le mode `MODE_PRIVATE`. Les bases ne sont donc accessibles qu'au sein de l'application elle-même.

Remarque
Il est préférable d'incrémenter automatiquement la clé de chaque table de la BD.

Le package principal qui contient les classes de gestion des bases de données est `android.database.sqlite`.
Dans le code, une base de données est modélisée par un objet de la classe `android.database.sqlite.SQLiteDatabase`.

Constructeur
SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

- context est le contexte de l'application
- name est le nom du fichier contenant la BD
- factory est utilisé pour créer des Cursor. En général on met null
- version est le numéro de version de la BD (commençant à 1).

`SQLiteDatabase.CursorFactory`: crée des objets cursor ou null par défaut
Sert à créer, ouvrir ou gérer une base de données.

Pour créer une base de données SQLite, il faut créer une sous classe de `SQLiteOpenHelper`, puis redéfinir la méthode de `onCreate(SQLiteDatabase db)` dans laquelle on exécute des commandes SQLite.

C'est dans cette méthode que les instructions seront lancées pour créer les différentes tables et éventuellement les remplir avec des données initiales.

Pour SQLite il n'existe que cinq types de données :

- `NULL` pour les données `NULL`.
- `INTEGER` pour les entiers (sans virgule).
- `REAL` pour les nombres réels (avec virgule).
- `TEXT` pour les chaînes de caractères.
- `BLOB` pour les données brutes, par exemple, une image…

Les étapes nécessaires :

- Créer une classe **MySQLiteHelper** extends **SQLiteOpenHelper**.
- **Le constructeur de MySQLiteHelper doit appeler le constructeur de la super** classe.
- Redéfinir la méthode **onCreate**() pour créer les tables.
- Redéfinir la méthode **onUpgrade**() pour supprimer les anciennes tables et créer les nouvelles.

```
class MaBaseOpenHelper extends SQLiteOpenHelper {
public MaBaseOpenHelper(Context context, String nom, CursorFactory
cursorfactory, int version){
super(context, nom, cursorfactory, version);
}

@Override
public void onCreate(SQLiteDatabase db) {
// TODO Ajoutez votre code de création...
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
// TODO Ajoutez votre code de mise à jour...
}}
```

La BD sera réellement créée lors du lancement de **getWritableDatabase**() (pour une base en lecture et écriture) ou de **getReadableDatabase**() (pour une base en lecture seule).

Pour accéder à une base de données à l'aide de la classe SQLiteOpenHelper, vous pouvez appeler les méthodes getReadableDatabase et getWritableDatabase et ainsi obtenir une instance de la base de données respectivement en lecture seule et en lecture/écriture.

```
private class maBase  extends SQLiteOpenHelper { ... }

maBase   mb = new maBase(this,"base1", null, 1);
SQLiteDatabase db = mb.getWritableDatabase();
...
SQLiteDatabase db = mb.getWReadableDatabase();
```

Création de table

Pour chaque champ de la table, on doit déclarer au moins deux informations le nom et le type Mais aussi des contraintes pour chaque attribut.

- `PRIMARY KEY` pour désigner la clé primaire sur un attribut ;
- `NOT NULL` pour indiquer que cet attribut ne peut valoir `NULL` ;
- `CHECK` afin de vérifier que la valeur de cet attribut est cohérente ;
- `DEFAULT` sert à préciser une valeur par défaut.

Syntaxe

```
CREATE TABLE nom_de_la_table (
     champ_1 type {contraintes},
     champ_2 type {contraintes},
  …);
```

Exemple

```
CREATE TABLE nom_de_la_table (
  nom_du_champ_1 INTEGER PRIMARY KEY,
  nom_du_champ_2 TEXT NOT NULL,
  nom_du_champ_3 REAL NOT NULL CHECK (nom_du_champ_3 > 0),
  nom_du_champ_4 INTEGER DEFAULT 10);

CREATE TABLE employe  (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      nom  TEXT,
      salaire   REAL
   );
```

Les requêtes

Il existe deux types de requêtes SQL. Celles qui attendent une réponse, comme la sélection, et celles qui n'attendent pas de réponse.

Pour exécuter une requête SQL du premier type, on utilise `execSQL(String sql).`

Pour la mise à jour de la base de données, il faut implémenter la méthode `void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion).`

Cette méthode est déclenchée à chaque fois que l'utilisateur met à jour son application.

- `oldVersion` est le numéro de l'ancienne version de la base de données que l'application utilisait.
- `newVersion` est le numéro de la nouvelle version.

Android rajoute automatiquement dans la base une table qui contient la dernière valeur connue de la base. À chaque lancement, Android vérifiera la dernière version de la base par rapport à la version actuelle dans le code. Si le numéro de la version actuelle est supérieur à celui de la dernière version, alors cette méthode est lancée.

Exemple

```
public static final String req = "DROP TABLE IF EXISTS " + table1 + ";";
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
 db.execSQL(req);
 onCreate(db);
}
```

Exemple : création de base de données

```
public class MySQLiteHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "BookDB";
    public MySQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

```
     public void onCreate(SQLiteDatabase db) {
        // SQL statement to create book table
        String CREATE_BOOK_TABLE = "CREATE TABLE books (" +
                "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "title TEXT, "+
                "author TEXT )";

        // create books table
        db.execSQL(CREATE_BOOK_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older books table if existed
        db.execSQL("DROP TABLE IF EXISTS books");

        // create fresh books table
        this.onCreate(db);
    }
}
```

Remarque

Une pratique courante avec la manipulation des bases de données est d'enregistrer les attributs, tables et requêtes dans des constantes. Ainsi on peut les retrouver et les modifier plus facilement. Tous ces attributs seront `publics` puisqu'il est possible qu'on manipule la base en dehors de cette classe.

```
public class MainActivity extends SQLiteOpenHelper {
  public static final String METIER_KEY = "id";
  public static final String METIER_NOM= "intitule";
  public static final String METIER_SALAIRE = "salaire";

  public static final String METIER_TABLE_NAME = "Metier";
  public static final String METIER_TABLE_CREATE =
    "CREATE TABLE " + METIER_TABLE_NAME + " (" +
      METIER_KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
      METIER_NOM+ " TEXT, " +
      METIER_SALAIRE + " REAL);";
}
```

La base de données sur un périphérique externe nécessite la permission `WRITE_EXTERNAL_STORAGE`, sinon la base de données sera en lecture seule.

Après la création de la base de données nous passons à la manipulation de ces données (insertion, sélection, suppression, update, ..).

Récupérer la base

La classe SQLiteOpenHelper fournit la méthode getReadableDatabase() pour accéder à un objet SQLiteDatabase en lecture.

public SQLiteDatabase getReadableDatabase ()

Cette méthode permet de récupérer un identifiant qui sera utilisé pour accéder à la base de données en lecture seul.
Cet identifiant sera valide jusqu'au prochain appel aux méthodes getWritableDatabase () ou close().

public SQLiteDatabase getWritableDatabase ()

Cette méthode permet de récupérer un identifiant qui sera utilisé pour accéder à la base de données en lecture ou en écriture.
Ces méthodes peuvent retourner une exception SQLiteException en cas de problème.

La classe Métier

Classe correspondante à la table:

```
public class Metier {
  private long id;
  private String nom;
  private float salaire;

  public Metier(long id, String intitule, float salaire) {
    this.id = id;
    this.Nom= nom;
    this.salaire = salaire;
  }

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  public String getnom() {
    return intitule;
  }

  public void setnom(String intitule) {
    this.Nom= intitule;
  }

  public float getSalaire() {
    return salaire;
  }

  public void setSalaire(float salaire) {
    this.salaire = salaire;
  }
}
```

La classe DAO

Elle contient au moins les méthodes CRUD (“**C**reate, **R**ead, **U**pdate and **D**elete”).

```
public class MetierDAO {
  public static final String TABLE_NAME = "metier";
  public static final String KEY = "id";
  public static final String NOM= "nom";
  public static final String SALAIRE = "salaire";

public static final String TABLE_CREATE = "CREATE TABLE " + TABLE_NAME + " (" +
KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " + NOM+ " TEXT, " + SALAIRE + "
REAL);";

public static final String TABLE_DROP = "DROP TABLE IF EXISTS " + TABLE_NAME +";";

  public void ajouter(Metier m) {
    // CODE
  }

  public void supprimer(long id) {
    // CODE
  }

  public void modifier(Metier m) {
    // CODE
  }

  public Metier selectionner(long id) {
    // CODE
  }
}
```

Ajout d’un nouvel enregistrement dans une table

- L'ajout des enregistrements se fait par la méthode insert :

insert (String table, String nullColumnHack, ContentValues values)

Cette méthode permet d'insérer les valeurs indiquées par values dans la table et retourne le numéro de la ligne insérée ou -1 en cas d'erreur.

- Values (classe ContentValues), ensemble de couples (clé, valeur) où la clé, de type String, représente le nom de la colonne et valeur, sa valeur
- Le second argument nullColumnHack est le nom de colonne qui aura la valeur NULL si values est vide. Cela est dû au fait que SQLite ne permet pas de lignes vides. Ainsi avec cette colonne, au moins un champ dans la ligne aura une valeur (= NULL).

`ContentValues`

Pour ajouter une entrée dans la table, on utilise la classe ContentValues qui implémente une interface de type Map comprenant des méthodes supplémentaires pour les types de SQLite :

Parmi les méthodes de cette classe SQLiteDatabase qui permettent de faire des opérations sur la base de données on trouve la méthode put.

La méthode void put()

Cette méthode permet de stocker une valeur. Cette valeur pouvant être de tous types.
put(String key, *type* value)

- type pourra être Byte, Integer, Long, String...byte[].

- key est la clef à utiliser pour le mapping

Exemple

```
ContentValues value = new ContentValues();
value.put(MetierDAO.INTITULE, m.getIntitule()); //m objet metier
value.put(MetierDAO.SALAIRE, m.getSalaire());
mDb.insert(MetierDAO.TABLE_NAME, null, value);
```

Supprimer

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

Permet de supprimer des enregistrements d'une table

- **table** : La table qui possède les enregistrements à effacer.
- **whereClause** Clause where pour indiquer les enregistrements à effacer. null effacera la totalité des enregistrements de la table (n'efface pas la structure de la table !)

La requête retourne le nombre d'enregistrements supprimés si whereClause possède une clause where. 0 sinon.

- **whereArgs** est un tableau des valeurs qui remplaceront les « ? » dans `whereClause`.

  Ainsi, si `whereClause` vaut "nom `LIKE ? AND salaire > ?`" et qu'on cherche les métiers qui ressemblent à « ingénieur avec un salaire supérieur à 1000 € », il suffit d'insérer dans `whereArgs` un `String[]` du genre `{"ingenieur", "1000"}`.

Exemple

```
public void supprimer(long id) {
 SQLiteDatabase mDb = this.getWritableDatabase();
 mDb.delete(TABLE_NAME, KEY + " = ?", new String[] {String.valueOf(id)});
}
```

Mise à jour : `update`

Mettre à jour un ou plusieurs enregistrements d'une table.

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

On ajoute le paramètre `values` pour représenter les changements à effectuer dans le ou les enregistrements cibles.
Donc, si je veux mettre à jour le salaire d'un métier, il suffit de mettre à jour l'objet associé et d'insérer la nouvelle valeur dans un `ContentValues` :

Exemple

```
ContentValues value = new ContentValues();
value.put(SALAIRE, 10000);
mDb.update(TABLE_NAME, value, KEY  + " = ?", new String[] {String.valueOf(m.getId())});
```

Sélection

La méthode de sélection est plus complexe et revêt trois formes différentes en fonction des paramètres qu'on veut lui passer. La méthode la plus simple pour récupérer des données est :

```
Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
```

Ces paramètres sont explicites puisqu'ils représentent à chaque fois des mots-clés de SQL :

- `distinct`, si vous ne voulez pas de résultats en double.
- `table` est l'identifiant de la table.
- `columns` est la liste des colonnes à retourner. Mettre null si on veut toutes les colonnes.
- `selection` est la clause WHERE d'un SELECT (sans le mot WHERE comme "salaire=?"). Mettre null si on veut toutes les lignes.
- `selectionArgs` : est utile si dans whereClause, il y a des paramêtres notés ?. Les valeurs de ces paramètres sont indiquées par selectionArgs. En général on met null.

Vous pouvez inclure des ? dans la "whereClause". Ces caractères génériques seront remplacés par les valeurs du tableau selectionArgs.

- `group by` permet de grouper les résultats.
- `having` est utile pour filtrer parmi les groupes.
- `order by` permet de trier les résultats. Mettre `ASC` pour trier dans l'ordre croissant et `DESC` pour l'ordre décroissant.
- `limit` pour fixer un nombre maximal de résultats voulus.

Exemple

db.query(TABLE_CONTACTS, new String[] { KEY_ID, KEY_NAME, KEY_PH_NO }, KEY_ID + "=?", new String[] { String.valueOf(id) }, null, null, null, null);

est l'équivalent de

"SELECT KEY_ID, KEY_NAME, KEY_PH_NO FROM TABLE_CONTACTS WHERE KEY_ID=' "+ id+" ' "

Remarque

Il vaut mieux utiliser un curseur,

`Cursor rawQuery(String sql, String[] selectionArgs)`

- sql : requête SQL comme select (ne pas terminer par ;) avec éventuellement une clause where
- selectionArgs : clause where déclarée dans la requête SQL pourra contenir des ?. Ces ? seront remplacés automatiquement avec les différentes valeurs de selectionArgs.

Dans ce cas on peut écrire la requête normale dans `sql` et remplacer les « ? » dans `selectionArgs`.

Exemple

Afficher tous les métiers qui rapportent en moyenne plus de 1€ :

```
Cursor c = mDb.rawQuery("select " + NOM+ " from " + TABLE_NAME + " where salaire > ?", new String[]{"1"});
```

Les curseurs

Les curseurs sont des objets qui contiennent les résultats renvoyées par la requête.

Les lignes

Pour parcourir les résultats d'une requête, il faut procéder ligner par ligne. Pour naviguer parmi les lignes, on peut utiliser les méthodes suivantes :

- `boolean moveToFirst()` pour aller à la première ligne.
- `boolean moveToLast()` pour aller à la dernière.
- `boolean moveToPosition(int position)` pour aller à la `position` voulue, sachant que vous pouvez savoir le nombre de lignes avec la méthode `int getCount()`.
- `boolean moveToNext()` pour aller à la ligne suivante. Par défaut on commence à la ligne -1, donc, en utilisant un `moveToNext()` sur un tout nouveau `Cursor`, on passe à la première ligne. On aurait aussi pu accéder à la première ligne avec `moveToFirst()`, bien entendu.
- `boolean moveToPrevious()` pour aller à l'entrée précédente.

Pour récupérer la position actuelle, on utilise `int getPosition()`. Vous pouvez aussi savoir si vous êtes après la dernière ligne avec `boolean isAfterLast()`.

Par exemple, pour naviguer entre toutes les lignes d'un curseur, on fait :

```
while (cursor.moveToNext()) {
  // Faire quelque chose
}
cursor.close();
```

```
for(cursor.moveToFirst();!cursor.isAfterLast();
cursor.moveToNext()) {
  // Votre code
}
cursor.close();
```

Les colonnes

On sait déjà à l'avance qu'on a trois colonnes, dont la première contient un entier, la deuxième, une chaîne de caractères, et la troisième, un réel.

Pour récupérer le contenu d'une de ces colonnes, il suffit d'utiliser une méthode du style `X getX(int columnIndex)` avec `X` le type de la valeur à récupérer et `columnIndex` la colonne dans laquelle se trouve cette valeur.

Exemple

```
long id = cursor.getLong(0);
String Nom= cursor.getString(1);
double salaire = cursor.getDouble(2);
Metier m = new Metier (id, intitule, salaire);
```

Création de la classe de gestion de base de données

```
public class gestion_DB extends SQLiteOpenHelper {
public static final String req = "CREATE TABLE table1 (id INTEGER PRIMARY KEY AUTOINCREMENT, nom TEXT,
salaire REAL);";

public gestion_DB(Context context)  {
 super(context, "DB1", null, 1);
}

public void onCreate(SQLiteDatabase db) {
   db.execSQL(req);
}
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                    // TODO Auto-generated method stub
}
//requête instertion
public void addelement() {
   SQLiteDatabase db = this.getWritableDatabase();
   ContentValues values = new ContentValues();
   values.put("id", 100);
   values.put("intitule", "ali");
   values.put("salaire", 50000);
db.insert("table1", null, values);
db.close();
}

//requête selection
String getelement(int id) {
SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = db.query("table1", new String[] { "id", "intitule", "salaire" },"id=?", new String[] {
String.valueOf(id) }, null, null, null, null);
if (cursor != null)
cursor.moveToFirst();
String s= Integer.parseInt(cursor.getString(0))+cursor.getString(1)+ cursor.getString(2);
     return s;
}
}
```

Activité principale

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

 gestion_DB db = new gestion_DB(this);

 db.addelement();

 String s=db.getelement(100);

TextView t1=(TextView)findViewById(R.id.textView2);
 t1.setText(s);
      }
```

Exemple

```
public class BD extends SQLiteOpenHelper{
    public static final String DATABASE_NAME = "Student.db";
    public static final String TABLE_NAME = "student_table";
    public static final String COL_1 = "ID";
    public static final String COL_2 = "NAME";
    public static final String COL_3 = "SURNAME";
    public static final String COL_4 = "MARK";

    public BD(Context context) {
        super(context, DATABASE_NAME, null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table "+TABLE_NAME+"(ID INTEGER PRIMARY KEY
AUTOINCREMENT,NAME TEXT,SURNAME TEXT,MARK INTEGER)");
    }

    @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
      db.execSQL("DROP TABLE IF EXIST "+TABLE_NAME);
        onCreate(db);
    }

    public long insertData(String name, String secondname, String mark){
        long resultat;
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValue = new ContentValues();
        contentValue.put(COL_2, name);
        contentValue.put(COL_3, secondname);
        contentValue.put(COL_4, mark);
        resultat = db.insert(TABLE_NAME, null, contentValue);
        return resultat;
    }

    public Cursor getAllData(){
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor resultat = db.rawQuery("select * from "+TABLE_NAME, null);
        return resultat;
    }

public boolean updateData(String id, String name, String prename, String mark){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValue = new ContentValues();
        contentValue.put(COL_1, id);
        contentValue.put(COL_2, name);
        contentValue.put(COL_3, prename);
        contentValue.put(COL_4, mark);
int resultat = db.update(TABLE_NAME, contentValue, "ID = ?",new String[] { id });
        if(resultat == 1)
            return true;
        else
            return false;
    }}
public class MainActivity extends AppCompatActivity {
    private BD databasehelper;
    EditText name,prename,mark,id;
    Button add,viewAll,update;
    Cursor cursor;
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        databasehelper = new BD(this);
        name = (EditText) findViewById(R.id.name);
        prename = (EditText) findViewById(R.id.secondname);
        mark = (EditText) findViewById(R.id.mark);
        add = (Button) findViewById(R.id.add);
        viewAll = (Button) findViewById(R.id.show);
        update = (Button) findViewById(R.id.update);
        id = (EditText) findViewById(R.id.idpersone);

        add.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
 long resultat = databasehelper.insertData(name.getText().toString(),
prename.getText().toString(), mark.getText().toString());

 if(resultat != -1)
Toast.makeText(MainActivity.this,"Added",Toast.LENGTH_SHORT).show();
 else
Toast.makeText(MainActivity.this,"no pb, on refait",Toast.LENGTH_SHORT).Show();
            }});
viewAll.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                cursor = databasehelper.getAllData();
                if(cursor.getCount() == 0){
Toast.makeText(MainActivity.this,"No Data :(",Toast.LENGTH_SHORT).show();
                    showMessage("Error","No data found!!!");
                }
                StringBuffer buffer = new StringBuffer();
                while(cursor.moveToNext()){
                    buffer.append("ID : "+cursor.getString(0)+"\n");
                    buffer.append("NAME : "+cursor.getString(1)+"\n");
                    buffer.append("PRENAME : "+cursor.getString(2)+"\n");
                    buffer.append("MARK :"+cursor.getString(3)+"\n");
                    buffer.append(" ----------------- \n");
                }
                showMessage("Your data",buffer.toString());
            }});
        update.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
boolean isUpdated = databasehelper.updateData(id.getText().toString(),
name.getText().toString(), prename.getText().toString(),
mark.getText().toString());
                if(isUpdated == true)
      showMessage("Your data is updated",id.getText().toString());
                else
     showMessage("Your data not is updated",id.getText().toString());
            }});
    }

    public void showMessage(String title,String message){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(true);
        builder.setTitle(title);
        builder.setMessage(message);
        builder.show();
    }}
```

```java
public class DBHandler extends SQLiteOpenHelper {
    // la version de notre BD
    private static final int DATABASE_VERSION = 1;
    // Database Name
    private static final String DATABASE_NAME = "personnesInfo";

    // le nom de la table personne
    private static final String TABLE_PERSONNES = "personnes";

    // les noms des colonnes de la table personnes
    private static final String KEY_ID = "cin";
    private static final String KEY_NOM = "nom";
    private static final String KEY_PRENOM= "prenom";

    public DBHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_PERSONNES_TABLE = "CREATE TABLE " + TABLE_PERSONNES + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NOM + " TEXT,"
                + KEY_PRENOM + " TEXT" + ")";
        db.execSQL(CREATE_PERSONNES_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // supprimer la table si elle existe
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_PERSONNES);
        // creer la table de nouveau
        onCreate(db);
    }

    // Ajout d'une personne
    public void addPersonne(Personne personne) {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();

        // cin de la personne
        values.put(KEY_ID, personne.getCin());
        // nom de la personne
        values.put(KEY_NOM, personne.getNom());
        // prenom de la personne
        values.put(KEY_PRENOM, personne.getPrenom());
        // Insertion des lignes
        db.insert(TABLE_PERSONNES, null, values);
       //Fermeture de la connexion à la BD
        db.close();
    }
```

```
    // recuperer une personne
    public Personne getPersonne(int id) {
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.query(TABLE_PERSONNES, new String[]{KEY_ID,
                        KEY_NOM, KEY_PRENOM}, KEY_ID + "=?",
                new String[]{String.valueOf(id)}, null, null, null, null);
        if (cursor != null)
            cursor.moveToFirst();

        Personne personne = new Personne(cursor.getString(2),cursor.getInt(0),
cursor.getString(1));
        // returner personne
        return personne;
    }

    // recuperer toutes les personnes
    public List<Personne> getAllPersonnes() {
        List<Personne> personnesList = new ArrayList<Personne>();
        // Select All Query
        String selectQuery = "SELECT  * FROM " + TABLE_PERSONNES;

        SQLiteDatabase db = this.getWritableDatabase();
        Cursor cursor = db.rawQuery(selectQuery, null);

        // on fait une boucle pour remplir la liste
        if (cursor.moveToFirst()) {
            do {
                Personne personne = new Personne();
                personne.setCin(cursor.getInt(0));
                personne.setNom(cursor.getString(1));
                personne.setPrenom(cursor.getString(2));
                // ajout de la personne dans la liste
                personnesList.add(personne);
            } while (cursor.moveToNext());
        }

        // retourner la liste remplit
        return personnesList;
    }

    // Recuperer le nombre denregistrements
    public int getPersonneCount() {
        String countQuery = "SELECT  * FROM " + TABLE_PERSONNES;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

```
    // Mise a jour d'une personne
    public int updatePersonne(Personne personne) {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(KEY_NOM, personne.getNom());
        values.put(KEY_PRENOM, personne.getPrenom());

        // mise a jour
        return db.update(TABLE_PERSONNES, values, KEY_ID + " =
?",new String[]{String.valueOf(personne.getCin())});
    }

    // suppression des peronnes
    public void deletePersonne() {
        SQLiteDatabase db = this.getWritableDatabase();
        db.execSQL("delete from "+ TABLE_PERSONNES);
        db.close();
    }
}

**************************************************

public class MainActivity extends ActionBarActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        DBHandler db = new DBHandler(this);
        db.deletePersonne();
        // Insertion des données
        Log.d("Insert: ", "Insertion en cours ..");
        db.addPersonne(new Personne("crytex", 11111, "crytex"));
        db.addPersonne(new Personne("user1",22222,"user1"));
        db.addPersonne(new Personne("user2",33333,"user2"));
        db.addPersonne(new Personne("user3",44444,"user3"));

        // lecture des personnes
        Log.d("Lecture: ", "Lecture des personnes..");
        List<Personne> personnes = db.getAllPersonnes();

        for (Personne personne : personnes) {
            String log = "Id: " + personne.getCin() + " ,Nom:
" + personne.getNom() + " ,Prenom: " + personne.getPrenom();
            // ecriture dans la fenetre log
            Log.d("Personnes: ", log);
        }
```