Développement Android : Activité n°3 (Base de données)

Thierry Vaira <tvaira@free.fr> http://tvaira.free.fr/

25/08/2016 (rev. 1)

Table des matières

Activité n°3 : Base de données SQLite

Objectif

L'objectif est d'écrire une application Android qui manipule une base de données SQLite :

Pré-requis

Vous devez avoir réalisé :

1. l'activité n°2 ou l'activité n°1

Puis :

- Copier et renommer le dossier de la première application en `MyApplication3`
- Renommer l'attribut *package* dans `$HOME/AndroidStudioProjects/MyApplication3/app/src/main/AndroidManifest.xml`
- Renommer la valeur d'*applicationId* dans `/home/tv/AndroidStudioProjects/MyApplication3/app/build.gradle`
- Démarrer `Android Studio` et ouvrir le projet `MyApplication3`
- Faire 'Synchronize'
- Refactoriser le nom du *package* puis renommer la valeur d'*app_name* dans `strings.xml`

Les bases de données

De nombreuses applications manipulent des données et il est souvent nécessaire des les stocker dans des base de données. Ces base de données peuvent être exploités par des requêtes SQL. L'application à développer devra utilisée la technique dite *embedded SQL* : les instructions en langage SQL seront incorporées dans le code source d'un programme écrit dans un autre langage (ici le Java sous Android).

Notions de base

Une base de données (*database*) est un « conteneur » permettant de stocker et de retrouver l'intégralité de données brutes ou d'informations. Dans la très grande majorité des cas, ces informations sont très structurées, et la base est localisée dans un même lieu et sur un même support.

Le dispositif comporte un système de gestion de base de données (SGBD) : un logiciel moteur qui manipule la base de données et dirige l'accès à son contenu. De tels dispositifs comportent également des logiciels applicatifs, et un ensemble de règles relatives à l'accès et l'utilisation des informations.

Une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des relations ou tables. Les lignes de ces relations sont appelées des *nuplets* (tuples) ou enregistrements. Les noms des colonnes (ou champs) sont appelées des attributs.

Les logiciels qui permettent de créer, utiliser et maintenir des bases de données relationnelles sont des système de gestion de base de données relationnels (SGBDR).

Pratiquement tous les systèmes relationnels utilisent le langage SQL (*Structured Query Language*) pour interroger les bases de données. Ce langage permet permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

Terminologie

Modèle de données Le schéma ou modèle de données est la description de l'organisation des données. Il renseigne sur les caractéristiques de chaque type de donnée et les relations entre les différentes données qui se trouvent dans la base de données. Il existe plusieurs types de modèles de données (relationnel, entité-association, objet, hiérarchique et réseau).

Entité Une entité est un objet, un sujet, une notion en rapport avec le domaine d'activité pour lequel la base de données est utilisée, et concernant celui pour lequel des données sont enregistrées (exemple : des personnes, des produits, des commandes, des réservations, . . .).

Attribut Un attribut est une caractéristique d'une entité susceptible d'être enregistrée dans la base de données. Par exemple une personne (entité), son nom et son adresse (des attributs). Les attributs sont également appelés des champs ou des colonnes.

Enregistrement Un enregistrement est une donnée qui comporte plusieurs champs dans chacun desquels est enregistrée une donnée.

Association Les associations désignent les liens qui existent entre différentes entités, par exemple entre un vendeur, un client et un magasin.

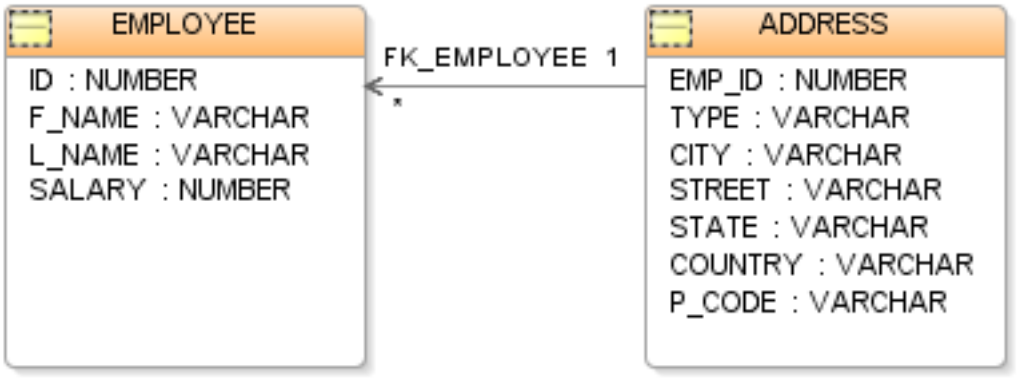
Cardinalité La cardinalité d'une association (d'un lien entre deux entités A et B par exemple) est le nombre de A pour lesquelles il existe un B et inversement. Celle-ci peut être un-a-un, un-a-plusieurs ou plusieurs-à-plusieurs. Par exemple un compte bancaire appartient à un seul client, et un client peut avoir plusieurs comptes bancaires (cardinalité un-a-plusieurs).

Modèle de données relationnel C'est le type de modèle de données le plus couramment utilisé pour la réalisation d'une base de données. Selon ce type de modèle, la base de données est composée d'un ensemble de tables (les relations) dans lesquelles sont placées les données ainsi que les liens. Chaque ligne d'une table est un enregistrement.

Base de données relationnelle C'est une base de données organisée selon un modèle de données de type relationnel, à l'aide d'un SGBD permettant ce type de modèle.

Clé primaire Dans les modèles de données relationnels, la clé primaire est un attribut dont le contenu est différent pour chaque enregistrement de la table, ce qui permet de retrouver un et un seul enregistrement (unicité). Dans les modèles de données relationnels, une clé étrangère est un attribut qui contient une référence à une donnée connexe (la valeur de la clé primaire de la donnée connexe).

SQL (*Structured Query Language*) C'est un langage de requête structurée et normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.



Exemples SQL

Recherche des lignes (aussi appelés tuples) dans une table existante :

```
SELECT nom, service
FROM   employe
WHERE  statut = 'stagiaire'
```



```
ORDER  BY nom;
```

Insère une ligne (aussi appelés tuple) dans une table existante :

```
INSERT INTO employe (nom, service, statut)
VALUES ('toto', 'rd', 'developpeur');
```

Modifie un tuple existant dans une table :

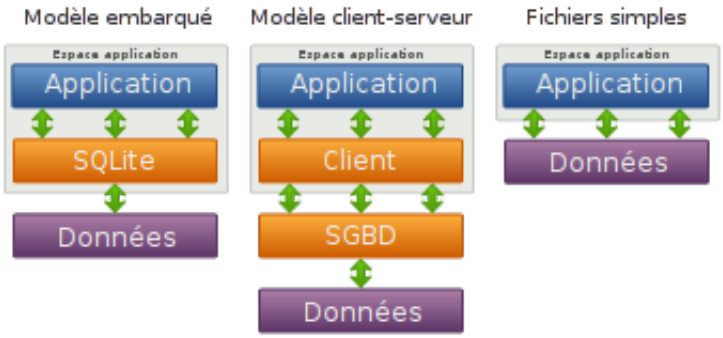
```
UPDATE employe
SET statut = 'chef'
WHERE nom = 'toto';
```

Supprime un tuple existant dans une table :

```
DELETE FROM employe
WHERE nom = 'toto';
```

SQLite

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL. Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme. SQLite est le moteur de base de données le plus distribué au monde, grâce à son utilisation dans de nombreux logiciels grand public comme Firefox, Skype, Google Gears, dans certains produits d'Apple, d'Adobe et de McAfee et dans les bibliothèques standards de nombreux langages comme PHP ou Python. De par son extrême légèreté (moins de 300 Kio), il est également très populaire sur les systèmes embarqués, notamment sur la plupart des smartphones modernes : l'iPhone ainsi que les systèmes d'exploitation mobiles Symbian et Android l'utilisent comme base de données embarquée.



SQLite est intégrée dans chaque appareil Android.

Remarque : L'accès à une base de données SQLite implique l'accès au système de fichiers. Cela peut être lent. Par conséquent, il est recommandé d'effectuer les opérations de base de données de manière asynchrone.

Si l'application crée une base de données, celle-ci est par défaut enregistrée dans le répertoire : `/data/APP_NAME/databases/DATABASE_NAME`.

Le *package* `android.database` contient toutes les classes nécessaires pour travailler avec des bases de données. Le *package* `android.database.sqlite` contient les classes spécifiques à SQLite.

Activité n°3

L'objectif est de pouvoir gérer une liste de serveurs (nom, adresse IP et numéro de port) à partir d'une base de données et l'intégrer à l'application existante.

La classe Serveur

2. On va commencer par créer une nouvelle classe `Serveur` pour les données à gérer. Cette classe est très simple puisque elle est définie par les attributs `id`, `nom`, `adresseIP` et `port`. On ajoute le constructeur ainsi que les accesseurs (*getter* et *setter*) :

```
package com.example.tv.myapplication3;

public class Serveur
{
    private int id;
    private String nom;
    private String adresseIP;
    private int port;

    public Serveur() {}

    public Serveur(String nom, String adresseIP, int port)
    {
        this.nom = nom;
        this.adresseIP = adresseIP;
        this.port = port;
    }

    public int getId()
    {
        return id;
    }

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

    public String getNom()
    {
        return nom;
    }

    public void setNom(String nom)
    {
        this.nom = nom;
    }

    public String getAdresseIP()
    {
        return adresseIP;
    }

    public void setAdresseIP(String adresseIP)
    {
        this.adresseIP = adresseIP;
    }

    public int getPort()
    {
        return port;
    }

    public void setPort(int port)
    {
        this.port = port;
    }

    public String toString()
    {
```



```
        return "id : " + id + "\nnom : " + nom + "\nadresse IP : " + adresseIP + "\nport : " + port;
    }
}
```

La classe ServeurSQLite

Pour créer et mettre à jour une base de données dans une application Android, on doit créer une classe qui hérite de `SQLiteOpenHelper`. Dans le constructeur de cette sous-classe, on appelera la méthode `super()` de `SQLiteOpenHelper`, en précisant le nom de la base de données et sa version actuelle.

Dans cette classe, on doit redéfinir les méthodes suivantes pour créer et mettre à jour votre base de données :

- `onCreate()` : pour accéder à une base de données qui n'est pas encore créée.
- `onUpgrade()` : si la version de la base de données évolue, cette méthode permettra de mettre à jour le schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode `onCreate()`.

Les deux méthodes reçoivent en paramètre un objet `SQLiteDatabase` qui est la représentation Java de la base de données.

La classe `SQLiteOpenHelper` fournit les méthodes `getReadableDatabase()` et `getWriteableDatabase()` pour accéder à un objet `SQLiteDatabase` en lecture/écriture.

Les tables de base de données doivent utiliser l'identifiant ___id_ comme **clé primaire** de la table. Plusieurs fonctions Android s'appuient sur cette norme.

Remarque : C'est une bonne pratique de créer une classe par table. Cette classe définit des méthodes statiques `onCreate()` et `onUpgrade()` qui sont appelées dans les méthodes correspondantes de la superclasse `SQLiteOpenHelper`.

3. On va donc créer la classe `ServeurSQLite` qui hérite donc de `SQLiteOpenHelper`. Cette classe va permettre de définir la table qui sera produite lors de l'instanciation.

```
package com.example.tv.myapplication3;

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

public class ServeurSQLite extends SQLiteOpenHelper
{
    public static final String DATABASE_NAME = "serveurs.db";
    public static final int    DATABASE_VERSION = 1;
    public static final String TABLE_SERVEURS = "table_serveurs";
    public static final String COL_ID = "ID";
    public static final String COL_NOM = "NOM";
    public static final String COL_ADRESSE_IP = "ADRESSE_IP";
    public static final String COL_PORT = "PORT";
    public static final int    NUM_COL_ID = 0;
    public static final int    NUM_COL_NOM = 1;
    public static final int    NUM_COL_ADRESSE_IP = 2;
    public static final int    NUM_COL_PORT = 3;

    private static final String CREATE_BDD = "CREATE TABLE " + TABLE_SERVEURS + " ("
            + COL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + COL_NOM + " TEXT NOT NULL, "
            + COL_ADRESSE_IP + " TEXT NOT NULL,"
            + COL_PORT + " INTEGER NOT NULL);";

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

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        // on crée la table
        db.execSQL(CREATE_BDD);
    }
```



```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        // au choix : on supprime la table puis on la recrée
        db.execSQL("DROP TABLE " + TABLE_SERVEURS + ";");
        onCreate(db);
    }
}
```

La classe ServeurBDD

`SQLiteDatabase` est la classe de base pour travailler avec une base de données **SQLite** sous Android et fournit des méthodes pour ouvrir, effectuer des requêtes, mettre à jour et fermer la base de données :

– les méthodes `insert()`, `update()` et `delete()`

```
ContentValues values = new ContentValues();

values.put(ServeurSQLite.COL_NOM, "bigbrother");
values.put(ServeurSQLite.COL_ADRESSE_IP, "192.168.52.83");
values.put(ServeurSQLite.COL_PORT, "5000");

getWritableDatabase().insert(ServeurSQLite.TABLE_SERVEURS, null, values);
```

– la méthode `execSQL()` qui permet d'exécuter une instruction SQL

```
getWritableDatabase().execSQL("DROP TABLE " + TABLE_SERVEURS + ";");
```

– la méthode `rawQuery()` qui exécute une requête SQL SELECT

```
Cursor c = bdd.rawQuery("SELECT * FROM" + ServeurSQLite.TABLE_SERVEURS + " WHERE " + ServeurSQLite.
    COL_ADRESSE_IP + " = ?", new String[] { adresseIP });
```

– la méthode `query()` qui fournit une interface structurée pour une requête SQL

```
Cursor c = bdd.query(ServeurSQLite.TABLE_SERVEURS, new String[] {ServeurSQLite.COL_ID, ServeurSQLite.COL_NOM,
    ServeurSQLite.COL_ADRESSE_IP, ServeurSQLite.COL_PORT}, ServeurSQLite.COL_NOM + " LIKE \"" + nom +"\"", null
    , null, null, null);
```

Remarques :

– L'objet `ContentValues` permet de définir des clés/valeurs. La clé représente l'identifiant de la colonne de la table et la valeur représente le contenu de l'enregistrement dans cette colonne. `ContentValues` peut être utilisé pour les insertions et les mises à jour des enregistrements de la base de données.
– Une requête retourne un objet `Cursor`. Un **curseur** représente le résultat d'une requête et pointe généralement vers une ligne de ce résultat. La classe `Cursor` fournit des méthodes `getXXX()` par type de données : `getLong(columnIndex)`, `getString(columnIndex)`, ... pour accéder aux données d'une colonne de la position courante du résultat. Le paramètre `columnIndex` est l'index numérique de la colonne.

Il faut maintenant créer une classe qui va faire le lien entre les classes `Serveur` et `ServeurSQLite` afin de gérer l'insertion, la suppression, la modification et d'exécuter des requêtes dans la base de données.

4. On va créer la classe `ServeurBDD` avec quelques méthodes de service :

```
package com.example.tv.myapplication3;

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

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

public class ServeurBDD
```



```
{
    private SQLiteDatabase bdd = null;
    private ServeurSQLite serveurSQLite = null;

    public ServeurBDD(Context context)
    {
        // cn crée la BDD et sa table
        serveurSQLite = new ServeurSQLite(context);
    }

    public void open()
    {
        // on ouvre la BDD en écriture
        if (bdd == null)
                bdd = serveurSQLite.getWritableDatabase();
    }

    public void close()
    {
        if (bdd != null)
            if (bdd.isOpen())
                bdd.close();
    }

    public SQLiteDatabase getBDD()
    {
        return bdd;
    }

    public long insererServeur(Serveur serveur)
    {
        ContentValues values = new ContentValues();

        values.put(ServeurSQLite.COL_NOM, serveur.getNom());
        values.put(ServeurSQLite.COL_ADRESSE_IP, serveur.getAdresseIP());
        values.put(ServeurSQLite.COL_PORT, serveur.getPort());

        return bdd.insert(ServeurSQLite.TABLE_SERVEURS, null, values);
    }

    public int modifierServeur(int id, Serveur serveur)
    {
        ContentValues values = new ContentValues();
        values.put(ServeurSQLite.COL_NOM, serveur.getNom());
        values.put(ServeurSQLite.COL_ADRESSE_IP, serveur.getAdresseIP());
        values.put(ServeurSQLite.COL_PORT, serveur.getPort());

        return bdd.update(ServeurSQLite.TABLE_SERVEURS, values, ServeurSQLite.COL_ID + " = " + id, null);
    }

    public int supprimerServeur(int id)
    {
        return bdd.delete(ServeurSQLite.TABLE_SERVEURS, ServeurSQLite.COL_ID + " = " + id, null);
    }

    public Serveur getServeur(String nom)
    {
        Cursor c = bdd.query(ServeurSQLite.TABLE_SERVEURS, new String[] {ServeurSQLite.COL_ID, ServeurSQLite.
    COL_NOM, ServeurSQLite.COL_ADRESSE_IP, ServeurSQLite.COL_PORT}, ServeurSQLite.COL_NOM + " LIKE \"" + nom +"
    \"", null, null, null, null);

        return cursorToServeur(c, true);
    }

    public Serveur getServeurByAdresseIP(String adresseIP)
    {
```

```
        Cursor c = bdd.rawQuery("SELECT * FROM" + ServeurSQLite.TABLE_SERVEURS + " WHERE " + ServeurSQLite.
    COL_ADRESSE_IP + " = ?", new String[] { adresseIP });

        return cursorToServeur(c, true);
    }

    public List<Serveur> getServeurs()
    {
        List<Serveur> serveurs = new ArrayList<Serveur>();

        Cursor cursor = bdd.query(ServeurSQLite.TABLE_SERVEURS, new String[] {ServeurSQLite.COL_ID,
    ServeurSQLite.COL_NOM, ServeurSQLite.COL_ADRESSE_IP, ServeurSQLite.COL_PORT}, null, null, null, null, null)
    ;

        cursor.moveToFirst();
        while (!cursor.isAfterLast())
        {
            Serveur serveur = cursorToServeur(cursor, false);
            serveurs.add(serveur);
            cursor.moveToNext();
        }

        cursor.close();

        return serveurs;
    }

    // Cette méthode permet de convertir un cursor en un objet de type Serveur
    private Serveur cursorToServeur(Cursor c, boolean one)
    {
        if (c.getCount() == 0)
            return null;

        if(one == true)
            c.moveToFirst();

        Serveur serveur = new Serveur();

        serveur.setId(c.getInt(ServeurSQLite.NUM_COL_ID));
        serveur.setNom(c.getString(ServeurSQLite.NUM_COL_NOM));
        serveur.setAdresseIP(c.getString(ServeurSQLite.NUM_COL_ADRESSE_IP));
        serveur.setPort(c.getInt(ServeurSQLite.NUM_COL_PORT));

        if(one == true)
            c.close();

        return serveur;
    }
}
```

L'interface utilisateur

On va d'abord créer une activité `ParametresActivity` pour assurer l'insertion et la suppression de serveurs dans la base de données.

Elle sera lancée à partir du menu 'Paramètres'.

5. Créer une nouvelle activité vide `ParametresActivity` qui hérite de `ListActivity` :

```
package com.example.tv.myapplication3;
```

```
import android.app.ListActivity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class ParametresActivity extends ListActivity
{
}
```

6. Modifier `MainActivity` pour y ajouter le lancement d'un `Intent` :

```
public boolean onOptionsItemSelected(MenuItem item)
{
    Intent intent;
    int id = item.getItemId();

    switch (id)
    {
        case R.id.action_settings:
            intent = new Intent(MainActivity.this, ParametresActivity.class);
            startActivity(intent);
            return true;
        case R.id.action_test:
            intent = new Intent(MainActivity.this, MyActivity.class);
            startActivity(intent);
            return true;
        case R.id.quitter:
            //Pour fermer l'application il suffit de faire finish()
            finish();
            return true;
    }

    return super.onOptionsItemSelected(item);
}
```

7. Créer un nouveau *layout* `parametres.xml` pour cette activité :

L'affichage des données de la base se fera dans un `ListView`.

Attention, celui-ci nécessite un *id* particulier (`@android:id/list`) et obligatoire :

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@android:id/list"/>
```

Ensuite, il faut créer un nouveau *layout* `list.xml` pour les éléments de la `ListView`. Android "veut" un `TextView` pour cela :

```
<?xml version="1.0" encoding="utf-8"?>
    <TextView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@android:id/text1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceListItemSmall"
        android:gravity="center_vertical"
        android:paddingStart="?android:attr/listPreferredItemPaddingStart"
        android:paddingEnd="?android:attr/listPreferredItemPaddingEnd"
        android:minHeight="?android:attr/listPreferredItemHeightSmall" />
```


8. Compléter l'activité `ParametresActivity` pour y ajouter la gestion de la base de données :

```
package com.example.tv.myapplication3;

import android.app.ListActivity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;
import java.util.List;

public class ParametresActivity extends ListActivity
{
    private ServeurBDD serveurBDD;
    private EditText edtNom;
    private EditText edtAdresseIP;
    private EditText edtPort;
    private Button btnAjouter;
    private Button btnSupprimer;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.parametres);

        edtNom = (EditText)findViewById(R.id.editTextNom);
        edtNom.setContentDescription("Le nom du serveur");
        edtAdresseIP = (EditText)findViewById(R.id.editTextAdresseIP);
        edtAdresseIP.setContentDescription("L'adresse IP du serveur");
        edtPort = (EditText)findViewById(R.id.editTextPort);
        edtPort.setContentDescription("Le numéro de port du serveur");

        btnAjouter = (Button)findViewById(R.id.buttonAjouter);
        btnAjouter.requestFocus();
        btnSupprimer = (Button)findViewById(R.id.buttonSupprimer);

        serveurBDD = new ServeurBDD(this);
        serveurBDD.open();

        List<Serveur> values = serveurBDD.getServeurs();

        ArrayAdapter<Serveur> adapter = new ArrayAdapter<Serveur>(this, R.layout.list, values);
        setListAdapter(adapter);
        adapter.setNotifyOnChange(true);
    }

    public void onClick(View v)
    {
        ArrayAdapter<Serveur> adapter = (ArrayAdapter<Serveur>) getListAdapter();
        Serveur serveur = null;

        if (v.getId() == R.id.buttonAjouter)
        {
            Toast toast = Toast.makeText(getApplicationContext(), "Ajout ...", Toast.LENGTH_SHORT );
            toast.show();

            serveur = new Serveur(edtNom.getText().toString(), edtAdresseIP.getText().toString(), Integer.
    parseInt(edtPort.getText().toString()));
            long id = serveurBDD.insererServeur(serveur);
```



```
            if(id != -1)
            {
                serveur.setId((int) id);
                adapter.add(serveur);
                //adapter.notifyDataSetChanged();
            }
        }

        if (v.getId() == R.id.buttonSupprimer)
        {
            if (getListAdapter().getCount() > 0)
            {
                serveur = serveurBDD.getServeur(edtNom.getText().toString());
                if (serveur == null)
                    return;

                int pos = 0;
                boolean trouve = false;
                for(pos = 0; pos < getListAdapter().getCount(); pos++)
                {
                    Serveur s = (Serveur)getListAdapter().getItem(pos);
                    if(s.getId() == serveur.getId())
                    {
                        trouve = true;
                        break;
                    }
                }

                if(trouve == true)
                {
                    Toast toast = Toast.makeText(getApplicationContext(), "Suppression Id " + Integer.toString(
    serveur.getId()), Toast.LENGTH_SHORT );
                    toast.show();
                    serveur = (Serveur)getListAdapter().getItem(pos);
                    serveurBDD.supprimerServeur(serveur.getId());
                    adapter.remove(serveur);
                    //adapter.notifyDataSetChanged();
                }
            }
        }

        if (v.getId() == R.id.buttonFermer)
        {
            serveurBDD.close();
            finish();
        }
    }

    protected void onListItemClick(ListView l, View v, int position, long id)
    {
        super.onListItemClick(l, v, position, id);

        Toast toast = Toast.makeText(getApplicationContext(), "Suppression ...", Toast.LENGTH_SHORT );
        toast.show();

        ArrayAdapter<Serveur> adapter = (ArrayAdapter<Serveur>) getListAdapter();
        Serveur serveur = null;

        serveur = (Serveur)getListAdapter().getItem(position);
        serveurBDD.supprimerServeur(serveur.getId());
        adapter.remove(serveur);
        //adapter.notifyDataSetChanged();
    }

    @Override
    protected void onResume()
```

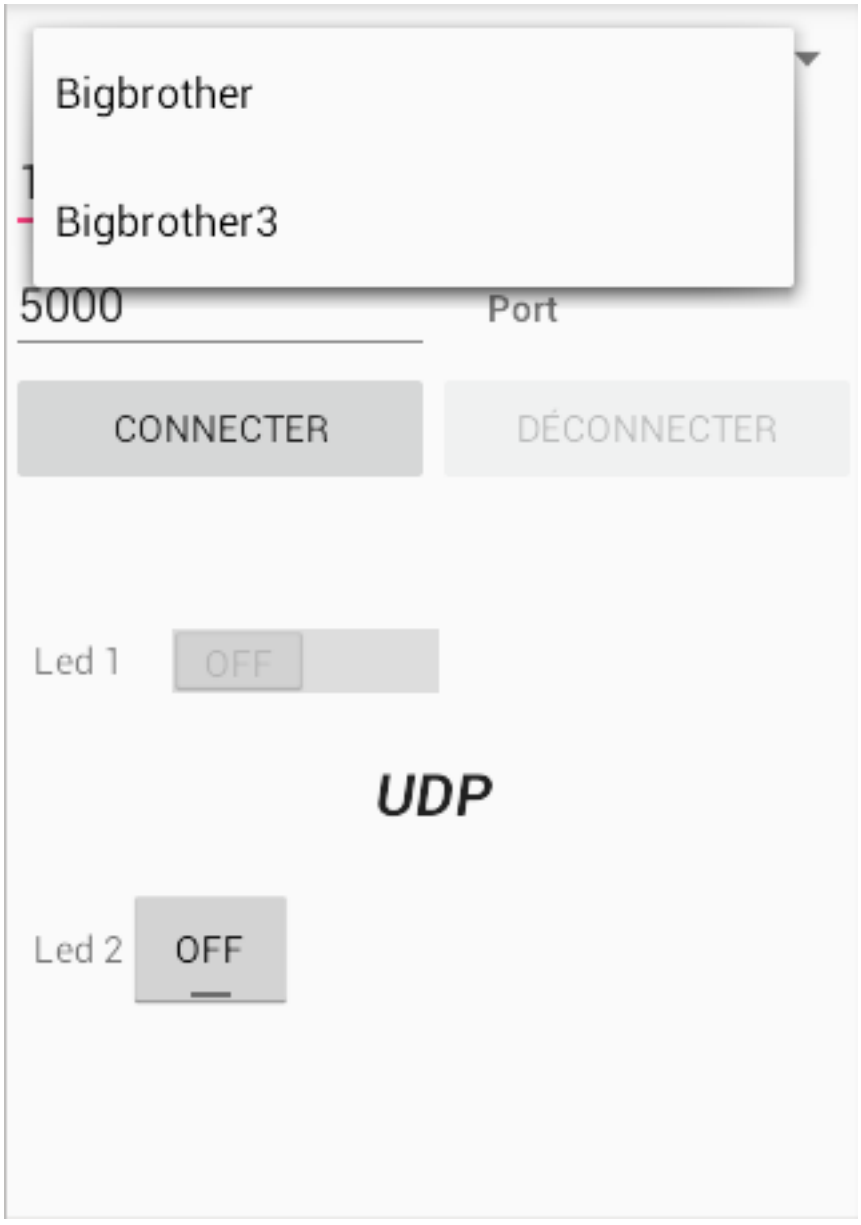


```
    {
        serveurBDD.open();
        super.onResume();
    }

    @Override
    protected void onPause()
    {
        serveurBDD.close();
        super.onPause();
    }
}
```

Remarque : lorsque la vue est affichée, le clavier s'affiche automatiquement car un `EditText` possède le focus. On peut retirer ce comportement pour cette activité en modifiant l'`AndroidManifest.xml`

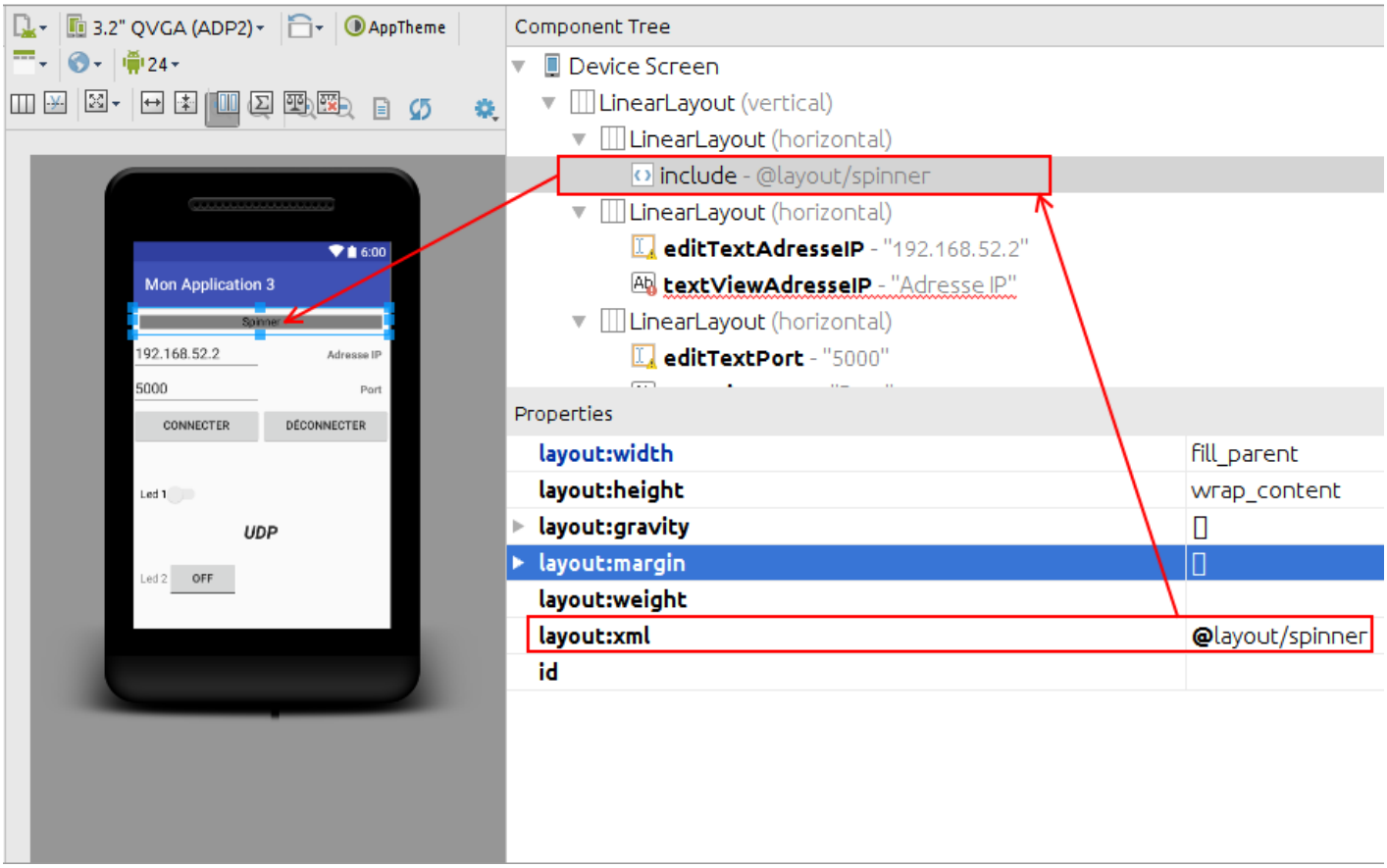
```
<activity android:name=".ParametresActivity" android:windowSoftInputMode="stateHidden"></activity>
```

On va maintenant intégrer une utilisation de la liste des serveurs disponibles dans l'activité `MyActivity`. On affichera les noms des serveurs dans un `Spinner` (une liste déroulante). Évidemment, un clic mettra à jour les champs **Adresse IP** et **Port** à partir de l'enregistrement stocké dans la base de données.



9. Créer un nouveau *layout* `spinner.xml` puis modifier le *layout* existant `commande.xml` en y ajoutant un `include` vers le nouveau *layout* :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <Spinner
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/spinnerListeServeurs"
        android:layout_margin="10dp"/>
</LinearLayout>
```



10. Modifier l'activité `MyActivity` pour y intégrer l'utilisation de la liste des serveurs disponibles :

```
package com.example.tv.myapplication3;

public class MyActivity extends AppCompatActivity implements View.OnClickListener
{
    private Spinner spinnerListeServeurs;
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.commande);

        spinnerListeServeurs = (Spinner)findViewById(R.id.spinnerListeServeurs);
```

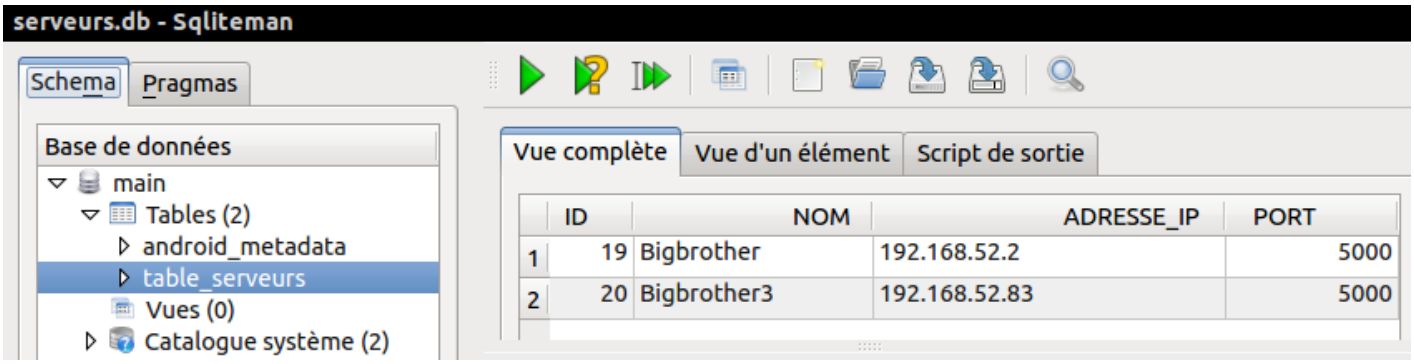


```
        spinnerListeServeurs.setContentDescription("La liste des serveurs");
        creerListeServeurs();

        ...
    }

    private void creerListeServeurs()
    {
        ServeurBDD serveurBDD = new ServeurBDD(this);
        serveurBDD.open();

        final List<Serveur> serveurs = serveurBDD.getServeurs();
        final List<String> noms = new ArrayList<String>();

        for(int i = 0; i < serveurs.size(); i++)
        {
            Serveur serveur = serveurs.get(i);
            noms.add(serveur.getNom());
        }

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, noms
    );
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinnerListeServeurs.setAdapter(adapter);

        spinnerListeServeurs.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long id)
            {
                Serveur serveur = serveurs.get(position);
                edtAdresseIP.setText(serveur.getAdresseIP());
                edtPort.setText(Integer.toString(serveur.getPort()));
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
                // TODO Auto-generated method stub
            }
        });

        serveurBDD.close();
    }
}
```

Pour finir, il est possible de récupérer la base de données créée à partir de l'émulateur `adb` :

```
$ adb -d shell "run-as com.example.tv.myapplication3 cat /data/data/com.example.tv.myapplication3/databases/
    serveurs.db" > serveurs.db
```

On peut l'ouvrir avec le *plugin* **SQLite Manager de Firefox** ou avec `sqliteman` :



Sur certaines versions d'Android, la commande `run-as` retourne une erreur "package unknown" qui est un bug connu (https://issuetracker.google.com/issues/36976703). Pour corriger cela, on va fixer le droit de lecture à tous sur le fichier de base de données.

On ajoute dans la méthode `onCreate()` de la classe `ServeurSQLite` le code suivant :

```
String path = db.getPath(); // path = /data/data/com.example.tv.myapplication3/databases/serveurs.db
File f = new File(path);
boolean r = f.setReadable(true, false); // -rw-rw-r--
if(r)
{
    Log.d("BD", "Ajout droit lecture " + path); // d = debug
}
else
{
    Log.e("BD", "Erreur ajout droit lecture " + path); // e = erreur
}
```

Remarque : il faut desinstaller l'application pour rappeler la méthode `onCreate()` (ou changer la version dans `DATABASE_VERSION`) au moins une fois pour fixer les droits.

On vérifie :

```
$ adb -d shell "ls -l /data/data/com.example.tv.myapplication3/databases/serveurs.db"
-rw-rw-r-- u0_a199  u0_a199     53248 2017-04-29 08:15 serveurs.db
```

On récupère ensuite la base de données dans le dossier local :

```
$ adb pull serveurs.db
```

On peut vérifier avec `sqlite3` ou le *plugin* **SQLite Manager de Firefox** ou avec `sqliteman` :

```
$ sqlite3 serveurs.db
sqlite> .tables
table_serveurs
sqlite> select * from table_serveurs;
19|Bigbrother|192.168.52.2|5000
20|Bigbrother3|192.168.52.83|5000
```

Documentation

- Guide de référence d'Android SDK
- Cours et tutoriels pour Android
- FAQ Android
- Utilisation de base de données SQLite sous Android

Retour

