Activité : Base de données sous Android

Thierry Vaira <tvaira@free.fr>

19/03/2017 (rev. 1)

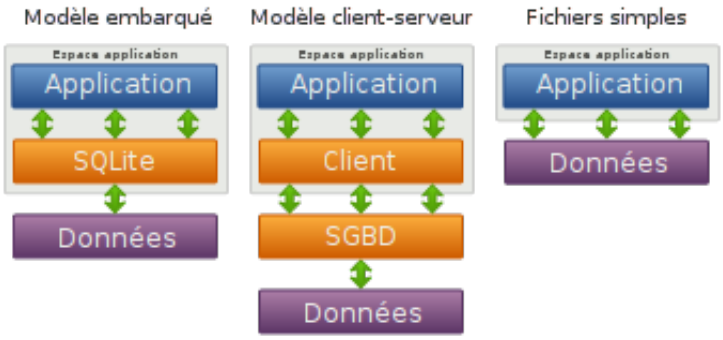
Table des matières

Base de données sous Android

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).

Lire : Activité bases de données

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.

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.

La classe `XXXSQLite`

Pour créer et mettre à jour une base de données SQLite dans une application Android, on doit créer une classe qui hérite de `SQLiteOpenHelper`.

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

- `onCreate()` : pour créer une base de données qui ne l'est pas encore
- `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().

Remarque : si on doit modifier la structure de la base de données, il faudra supprimer la base de données pour pouvoir re-appeler `onCreate()`. Pour cela, le plus simple est de supprimer les données associées à l'application dans le menu Paramètres de la tablette.

On crée une classe `XXXSQLite`, qui hérite donc de `SQLiteOpenHelper`, pour définir l'ensemble des tables de la base de données qui seront produites lors de l'instanciation.

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

public class XXXSQLite extends SQLiteOpenHelper
{
    public static final String  DATABASE_NAME = "xxx.db";
    public static final int     DATABASE_VERSION = 1;
    // Pour création des tables
    private static final String CREATE_BDD_APPAREILS =
     "CREATE TABLE appareils (" +
        "idAppareil INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ," +
        "nom VARCHAR(45) NULL ," +
        "etat TINYINT(1) NULL  );";
    private static final String CREATE_BDD_CAPTEURS =
    "CREATE TABLE capteurs (" +
        "idCapteurs INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ," +
        "typeCapteurs INTEGER NULL ," +
        "nom VARCHAR(45) NULL ," +
        "unite VARCHAR(45) NULL ," +
        "CONSTRAINT fk_capteurs_1 FOREIGN KEY (typeCapteurs ) REFERENCES typeCapteurs (idtypeCapteurs ) );";
    private static final String CREATE_BDD_PARAMETRESVITAUX =
    "CREATE TABLE parametresVitaux (" +
        "idParametresVitaux INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ," +
        "idCapteur INTEGER NULL ," +
        "mesure DOUBLE NULL ," +
        "horodatage DATETIME," +
        "CONSTRAINT fk_parametresVitaux_1 FOREIGN KEY (idCapteur ) REFERENCES capteurs (idCapteurs ));";
    private static final String CREATE_BDD_TYPECAPTEURS =
    "CREATE TABLE typeCapteurs (" +
        "idtypeCapteurs INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL ," +
        "typeCapteurs VARCHAR(45) NULL );";
    // etc ...

    // Pour insertion des données initiales
    private static final String INSERT_BDD_APPAREILS_1 = "INSERT INTO appareils VALUES(1,'chauffage',0);";
    private static final String INSERT_BDD_APPAREILS_2 = "INSERT INTO appareils VALUES(2,'eclairage',0);";
    private static final String INSERT_BDD_CAPTEURS_1 = "INSERT INTO capteurs VALUES(1,1,'temperature eau','C');
    ";
    private static final String INSERT_BDD_CAPTEURS_2 = "INSERT INTO capteurs VALUES(2,1,'temperature air','C');
    ";
    private static final String INSERT_BDD_CAPTEURS_3 = "INSERT INTO capteurs VALUES(3,2,'ph','');";
    private static final String INSERT_BDD_CAPTEURS_4 = "INSERT INTO capteurs VALUES(4,3,'niveau d''eau','cm');"
    ;
    private static final String INSERT_BDD_TYPECAPTEURS_1 = "INSERT INTO typeCapteurs VALUES(1,'temperature');";
    private static final String INSERT_BDD_TYPECAPTEURS_2 = "INSERT INTO typeCapteurs VALUES(2,'ph');";
    private static final String INSERT_BDD_TYPECAPTEURS_3 = "INSERT INTO typeCapteurs VALUES(3,'niveau');";
    // etc ...

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

    @Override
    public void onCreate(SQLiteDatabase db)
```



```
    {
        // on crée la BDD :
        // 1. création des tables
        db.execSQL(CREATE_BDD_APPAREILS);
        db.execSQL(CREATE_BDD_CAPTEURS);
        db.execSQL(CREATE_BDD_PARAMETRESVITAUX);
        db.execSQL(CREATE_BDD_TYPECAPTEURS);
        // 2. insertion des données initiales
        db.execSQL(INSERT_BDD_APPAREILS_1);
        db.execSQL(INSERT_BDD_APPAREILS_2);
        db.execSQL(INSERT_BDD_CAPTEURS_1);
        db.execSQL(INSERT_BDD_CAPTEURS_2);
        db.execSQL(INSERT_BDD_CAPTEURS_3);
        db.execSQL(INSERT_BDD_CAPTEURS_4);
        db.execSQL(INSERT_BDD_TYPECAPTEURS_1);
        db.execSQL(INSERT_BDD_TYPECAPTEURS_2);
        db.execSQL(INSERT_BDD_TYPECAPTEURS_3);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        // on supprime la table puis on la recrée
        db.execSQL("DROP TABLE appareils;");
        db.execSQL("DROP TABLE capteurs;");
        db.execSQL("DROP TABLE parametresVitaux;");
        db.execSQL("DROP TABLE typeCapteurs;");
        onCreate(db);
    }
}
```

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

```
$ adb -d shell "run-as com.example.tv.APP_NAME cat /data/data/com.example.tv.APP_NAME/databases/DATABASE_NAME" >
     bd.sqlite
```

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

La classe `XXXBDD`

`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.

On crée une classe `XXXBDD`, qui hérite donc de `SQLiteDatabase`, pour accéder à la base de données de l'application.

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

public class XXXBDD
{
    private SQLiteDatabase bdd = null;
    private XXXSQLite xxxSQLite = null;

    public XXXBDD(Context context)
    {
        // on crée la BDD
        xxxSQLite = new GesAquaSQLite(context);
    }

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



```
    public void close()
    {
        // on ferme la BDD
        if (bdd != null)
            if (bdd.isOpen())
                bdd.close();
    }

    public SQLiteDatabase getBDD()
    {
        // pas ouverte ?
        if (bdd == null)
            open();
        // on retourne un accès à la BDD
        return bdd;
    }
}
```

Le principe de son utilisation est le suivant :

```
XXXBDD xxxBDD = new XXXBDD(this);

SQLiteDatabase bdd = xxxBDD.getBDD();

xxxBDD.close();
```

Effectuer des requêtes SQL

La classe `SQLiteDatabase` fournit des méthodes pour effectuer des requêtes sur les tables de la base de données :

- les méthodes `insert()`, `update()` et `delete()`
- la méthode `execSQL()` qui permet d'exécuter une instruction SQL
- la méthode `rawQuery()` qui exécute une requête SQL SELECT
- la méthode `query()` qui fournit une interface structurée pour une requête SQL

Les méthodes `insert()`, `update()` et `delete()` utilisent des objets `ContentValues` qui permettent 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.

Les méthodes `rawQuery()` et `query()` retournent 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. On peut déplacer le **curseur** pour se positionner sur un résultat qaund la requête en a fourni plusieurs : `moveToFirst()`, `moveToNext()`, ...

On va créer une classe `Appareils` pour la table "appareils".

```
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 Appareils
{
    private XXXBDD xxxBDD;
    private SQLiteDatabase bdd;

    public Appareils(Context context)
    {
        xxxBDD = new GesAquaBDD(context);
        xxxBDD.open();
        bdd = xxxBDD.getBDD();
    }
```



```
public List<Appareil> getListe()
{
    List<Appareil> appareils = new ArrayList<Appareil>();

    Cursor cursor = bdd.query("appareils", new String[] {"idAppareil", "nom", "etat"}, null, null, null,
null, null);

    cursor.moveToFirst();
    while (!cursor.isAfterLast())
    {
        Appareil appareil = cursorToServeur(cursor, false);
        appareils.add(appareil);
        cursor.moveToNext();
    }

    cursor.close();

    return appareils;
}

public long inserer(Appareil appareil)
{
    ContentValues values = new ContentValues();

    values.put("nom", appareil.getNom());
    values.put("etat", appareil.getEtat());

    return bdd.insert("appareils", null, values);
}

public int modifier(int id, Appareil appareil)
{
    ContentValues values = new ContentValues();
    values.put("nom", appareil.getNom());
    values.put("etat", appareil.getEtat());

    return bdd.update("appareils", values, "idAppareil = " + id, null);
}

public int supprimer(int id)
{
    return bdd.delete("appareils", "idAppareil = " + id, null);
}

public Appareil getAppareil(String nom)
{
    Cursor c = bdd.rawQuery("SELECT * FROM appareils WHERE nom = ?", new String[] { nom });

    return cursorToServeur(c, true);
}

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

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

    Appareil appareil = new Appareil();

    appareil.setId(c.getInt(0));
    appareil.setNom(c.getString(1));
    appareil.setEtat(c.getInt(2));
```



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

        return appareil;
    }
}
```

Cette classe manipule des objets `Appareil` :

```
public class Appareil
{
    private int id;
    private String nom;
    private int etat;

    public Appareil()
    {
        this.nom = "";
        this.etat = 0;
    }

    public Appareil(String nom, int etat)
    {
        this.nom = nom;
        this.etat = etat;
    }

    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 int getEtat()
    {
        return etat;
    }

    public void setEtat(int etat)
    {
        this.etat = etat;
    }

    public String toString()
    {
        return "id : " + id + "\nnom : " + nom + "\netat : " + etat;
    }
}
```


Exemples de requêtes

```
-- Récupère les mesures de température (id = 1)
SELECT * FROM parametresVitaux INNER JOIN capteurs ON parametresVitaux.idCapteur=capteurs.idCapteurs WHERE
    capteurs.idCapteurs=1

-- Récupère les mesures de température (id = 1) avec l'unite et l'horodatage
SELECT parametresVitaux.mesure AS temperature, capteurs.unite, parametresVitaux.horodatage FROM parametresVitaux
     INNER JOIN capteurs ON parametresVitaux.idCapteur=capteurs.idCapteurs WHERE parametresVitaux.idCapteur=1

-- Récupère le nombre de mesures de température (id = 1)
SELECT COUNT(*) FROM parametresVitaux INNER JOIN capteurs ON parametresVitaux.idCapteur=capteurs.idCapteurs
    WHERE capteurs.idCapteurs = 1

-- Récupère le minimum et le maximum des mesures de température (id = 1)
SELECT MIN(parametresVitaux.mesure), MAX(parametresVitaux.mesure) FROM parametresVitaux INNER JOIN capteurs ON
    parametresVitaux.idCapteur=capteurs.idCapteurs WHERE capteurs.idCapteurs=1

-- Récupère les mesures de température (id = 1) avec l'unite et l'horodatage dans une plage horodatée
SELECT parametresVitaux.mesure, capteurs.unite, parametresVitaux.horodatage FROM parametresVitaux INNER JOIN
    capteurs ON parametresVitaux.idCapteur = capteurs.idCapteurs WHERE capteurs.idCapteurs=1 AND
    parametresVitaux.horodatage>='2017-03-16 14:00:00' AND horodatage<='2017-03-16 14:01:00'

-- Récupère les mesures de température (id = 1) avec l'unite et l'horodatage supérieure à un seuil
SELECT parametresVitaux.mesure, capteurs.unite, parametresVitaux.horodatage FROM parametresVitaux INNER JOIN
    capteurs ON parametresVitaux.idCapteur=capteurs.idCapteurs WHERE capteurs.idCapteurs=1 AND parametresVitaux
    .mesure>24

-- Récupère la dernière mesure des 3 capteurs
SELECT mesure FROM parametresVitaux WHERE horodatage IN (SELECT MAX(horodatage) FROM parametresVitaux)

-- Récupère les 5 dernières mesures du capteur de ph
SELECT * FROM (SELECT mesure, horodatage FROM parametresVitaux WHERE parametresVitaux.idCapteur=2 ORDER BY
    horodatage DESC LIMIT 5) tmp ORDER BY horodatage ASC LIMIT 5
```

Par exemple la requête qui récupère les mesures de température avec l'unité et l'horodatage se codera de la manière suivante :

```
String requete = "SELECT parametresVitaux.mesure AS temperature, capteurs.unite, parametresVitaux.horodatage
    FROM parametresVitaux INNER JOIN capteurs ON parametresVitaux.idCapteur=capteurs.idCapteurs WHERE
    parametresVitaux.idCapteur=1";

Cursor c = bdd.rawQuery(requete, null);

//...
```

