SQLITE SOUS ANDROID STUDIOS

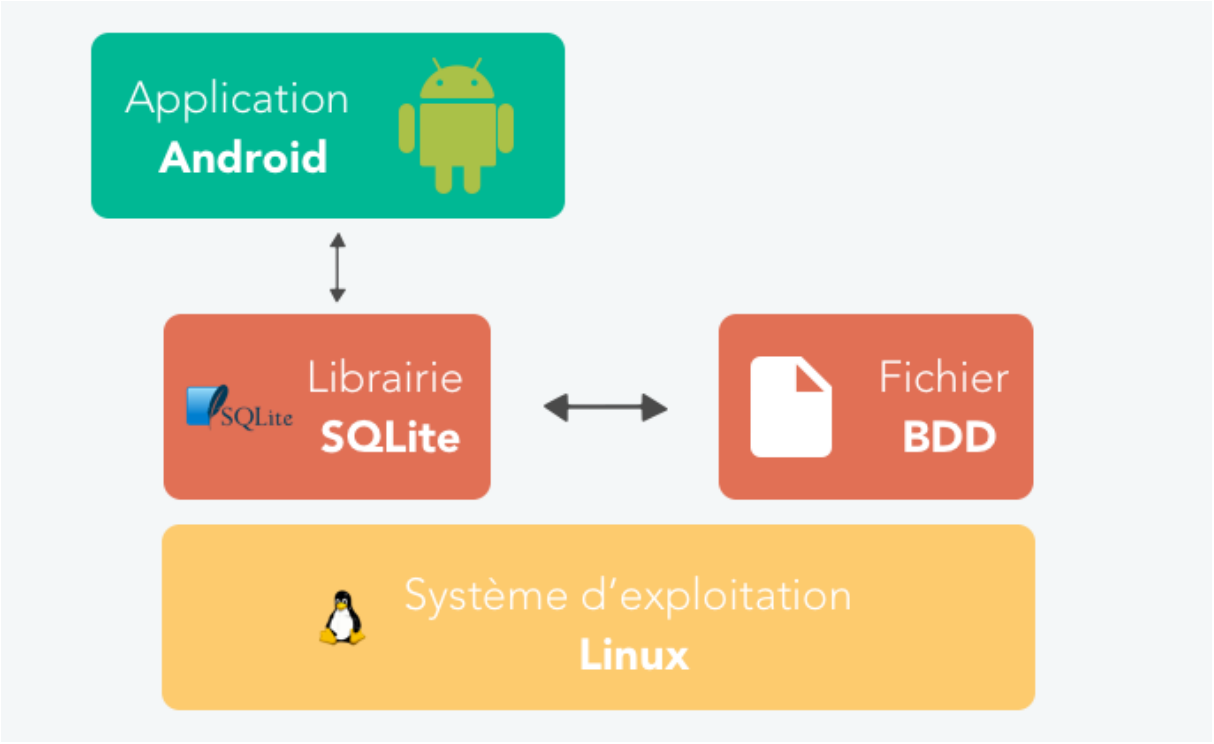
Introduction

SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions et les prepared statement. La base de données nécessite peu de mémoire lors de l'exécution (env. 250 ko), ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution.

SQLite prend en charge les types de données TEXT (similaire à String en Java), INTEGER (similaire à long en Java) et REAL (similaire à double en Java). Tous les autres types doivent être convertis en l'un de ces types avant d'être enregistrés dans la base de données. SQLite ne vérifie pas si les types des données insérées dans les colonnes correspondent au type défini, par exemple, vous pouvez écrire un nombre entier dans une colonne de type chaîne de caractères et vice versa.

Architecture de SQLite

SQLite est un **`moteur de base de données relationnelle`** entièrement manipulable grâce au langage SQL. Contrairement aux serveurs de base de données traditionnelles comme `MySQL`, `PostgreSQL` ou encore `Microsoft SQL Server`, l'intégralité de la base de données est **`stockée dans un fichier`** (et non sur un serveur distant).



Paquetage

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

Création et mise à jour de la base avec SQLiteOpenHelper

Pour créer et mettre à jour une base de données dans votre application Android, vous créez une classe qui hérite de SQLiteOpenHelper. Dans le constructeur de votre sous-classe, vous appelez la méthode super() de SQLiteOpenHelper, en précisant le nom de la base de données et sa version actuelle.

Dans cette classe, vous devez redéfinir les méthodes suivantes pour créer et mettre à jour votre base de données.

• onCreate() : est appelée par le framework lors de l'ouverture de la base de données pour la première fois. Dans ce cas il va créer le fichier « *.db »

• onUpgrade() : est appelée si la version de la base de données est augmentée dans le code de votre application. Cette méthode vous permet de mettre à jour un schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode onCreate().

Exemple MyPositionHelper :

```
public class MyPositionHelper  extends SQLiteOpenHelper {
// declaration de nom de la table, titre des champs
    public static final String table_pos="Position";
    public static final String col_id="Identifiant";
    public static final String col_long="Longitude";
    public static final String col_lat="Latitude";

    String req="create table "+table_pos+"("
            +col_id+" Integer primary Key autoincrement,"
            +col_long+" Text not null,"
            +col_lat+" Text not null"
            +")";

    public MyPositionHelper(@Nullable Context context, @Nullable String name,
@Nullable SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(req);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

        db.execSQL(" drop table "+table_pos);
        onCreate(db);
    }
}
```

Créez la classe DAO Data Access Object. Elle maintient la connexion avec la base de données et prend en charge l'ajout, la modification, la suppression des données

Exemple PositionManager:

```
public class PositionManager {
    Context con;
    SQLiteDatabase mabase;

    // Constructeur
    public PositionManager(Context con) {
        this.con = con;
    }
La classe SQLiteOpenHelper fournit les méthodes getReadableDatabase() et getWriteableDatabase() pour accéder à un objet SQLiteDatabase en lecture, respectif en écriture.
    public void ouvrir(String fichier)
    {
        MyPositionHelper helper=new MyPositionHelper(con,fichier,null,1);
        // declaration d'une base
        // si on modifie la version == appel implicite à onUpgrade

        mabase= helper.getWritableDatabase();
        // permet de ouvrir la base si elle existe
        // si n'existe pas, elle cree  le fichier
       //  et appel oncreate ==> creation   des tables
    }

    public void inserer(String ch1,String ch2)
    {
        // insertion dans la base
        ContentValues v=new ContentValues();
        // v est un hashmap
        v.put(MyPositionHelper.col_long,ch1);
        v.put(MyPositionHelper.col_lat,ch2);
        mabase.insert(MyPositionHelper.table_position,null,v);
    }

    ArrayList<Position> selectionnertout()
    {
        // initialisation de la valeur de retour
        ArrayList<Position> data=new ArrayList<Position>();
        // selection depuis la base
        Cursor cr=mabase.query(MyPositionHelper.table_position
                ,new String[]{MyPositionHelper.col_id,MyPositionHelper.col_lat,
                        MyPositionHelper.col_long}
                ,null,null,null,null,null
        );
        // conversion d'un cursor à une arraylist data
        cr.moveToFirst();
        while (!cr.isAfterLast()) {
            int i1 = cr.getInt(0);
            String i2 = cr.getString(1);
            String i3 = cr.getString(2);
            data.add(new Position(i1,i2,i3));
            cr.moveToNext();
        }
        return data;
    }
```

```
    long modifier ( int id,String lon,String lat)
    {
        int a=-1;
        // initialisation nouvelle valeur
        ContentValues v=new ContentValues();
        v.put(MyPositionHelper.col_id,id);
        v.put(MyPositionHelper.col_long,lon);
        v.put(MyPositionHelper.col_lat,lat);
        a=mabase.update(MyPositionHelper.table_position,
                v,
                MyPositionHelper.col_id+"="+id,null);
        return a;
    }
    long supprimer(int id)
    {
        int a=-1;
        a=mabase.delete(MyPositionHelper.table_position,
                MyPositionHelper.col_id+"="+id,null);
         return a;
    }
}
```

Tableau 1 Paramètres de la méthode query()

Paramètre	Commentaire
String dbName	Le nom de la table pour laquelle la requête est compilée.
String[] columnNames	Une liste des colonnes à retourner. En passant « null », toutes les colonnes seront retournées.
String whereClause	Clause « where », filtre pour la sélection des données, « null » permet de sélectionner toutes les données.
String[] selectionArgs	Vous pouvez inclure des ? dans la « whereClause ». Ces caractères génériques (placeholders) seront remplacés par les valeurs du tableau selectionArgs.
String[] groupBy	Un filtre qui déclare comment regrouper les lignes, avec « null » les lignes ne seront pas groupées.
String[] having	Filtre pour les groupes, null signifie pas de filtrage.
String[] orderBy	Colonnes de la table utilisées pour ordonner les données, avec null les données ne seront pas ordonnées.

Si une condition n'est pas requise, vous pouvez transmettre la valeur null, par exemple, pour une clause group by.