Android: Base de données

Assane SECK
Ingenieur-Informaticien

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

INTRODUCTION À SQLITE

- SQLite propose un moteur de base de données relationnelle accessible par Sql
- SQL propre
- Petite empreinte
- A fait ses preuves: Adobe, Apple, Sun, Symbian,... l'utilisent
- Est intégré à Android
- Utilise une API native et pas JDBC
- SQLite a: select, update, insert, delete, create table, drop table,...
- SQLite n'a pas: foreign keys, outer joins et certaines parties d'alter table

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

SQLITE OPENHELPER ET LA CREATION D'UNE BASE DE DONNEES

- Aucune base de données vous est fournies, vous devez créer la vôtre.
- Meilleur moyen: en créant une sous-classe de SQLiteOpenHelper:

  contient la logique de création et mise à jour de base de données
- SQLiteOpenHelper devra surcharger 3 fonctions:

  Le constructeur SQLiteOpenHelper() qui prend le contexte, le nom de la base de

  données et un cursor factory optionnel (null si pas utilisé) ainsi qu'une version pour votre base de données.

  onCreate() qui recoit en paramètre l'objet SQLiteDatabase que vous devez
- populer avec des tables et des données initiales.

  onUpgrade() qui recoit en paramètre l'objet SQLiteDatabase ainsi que l'ancienne et la nouvelle version de la base de données, afin que vous puissiez faire le nécessaire pour mettre à jour la base de données

EXEMPLE DE SQLITEOPENHELPER

```
class BDUser extends SQLiteOpenHelper {

public BDUser(Context context) {
super(context, "bduniv.db", null, 1);
}
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE user (id INTEGER PRIMARY KEY AUTOINCREMENT,
login VARCHAR (50), password VARCHAR(50) );");
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
//Efface l'ancienne table user
db.execSQL("DROP TABLE IF EXISTS user;");
onCreate(db);
}
}
```

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

OUVRIR ET FERMER UNE BASE DE DONNÉES

- Afin d'utiliser votre sous-classe de SQLiteOpenHelper subclass, instanciez la:
- SQLiteOpenHelper dbHelper = new BDUser(context);
- Demandez ensuite d'avoir une base de données en lecture ou écriture:

  getReadableDatabase()

  getWriteableDatabase()

  SQLiteDatabase dbHelper = this. getWriteableDatabase();
- Ceci vous renverra une instance de SQLiteDatabase ou null si un problème est survenu avec la base de données.
- Vous pouvez ensuite utiliser la SQLiteDatabase pour faire des requêtes et des mises à jour
- Finalement, refermez votre instance de la SQLiteDatabase

- **Localisation de la base de données:** File Explorer/data/data/nompackage/databases/nomDeLaBase

OUVRIR ET FERMER UNE BASE DE DONNÉES

- Ouverture d'une base de données (par exemple dans onCreate)
- Fermeture d'une base de données (par exemple dans onDestroy)
- SQLiteDatabase db=(new BDUser(this)).getWritableDatabase();
- db.close();

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

UTILISER DES CURSORS

- Lorsque vous effectuez une requête vers une base de données, vous obtiendrez toujours un Cursor en retour
- Avec ce Cursor, vous pouvez:

  1) Trouver combien de résultats il y a, avec getCount()

  2) Iterer sur les différentes entrées via moveToFirst(), moveToNext() et

  3) isAfterLast()

  Trouver les noms des colonnes via getColumnNames(), convertir ceci en numéros de colonnes via getColumnIndex() et obtenir les valeurs sur l'entrée en cours via des fonctions comme getString() et getInt()

  Réexecuter la requête en utilisant requery()

  Relâcher les ressources du Cursor via close()

EXEMPLE D'UTILISATION DE CURSOR

```
Cursor result = db.rawQuery("select id, login, password from user",null);
result.moveToFirst();
 while(result.isAfterLast()){
  int id = result.getInt(0);
  String login = result.getString(result.getColumnIndex("login"));
  String password= result.getString(2);
  result.moveToNext();
 }
result.close();
```

EXEMPLE D'UTILISATION DE CURSOR

```
Cursor result = db.rawQuery("select id, login, password from user",null);
result.moveToFirst();
 while(result.isAfterLast()){
  int id = result.getInt(0);
  String login = result.getString(result.getColumnIndex("login"));
  String password= result.getString(2);
  result.moveToNext();
 }
result.close();
```

EXEMPLE D'UTILISATION DE CURSOR

```
Cursor result = db.query("user",null,null,null,null,null,null);
result.moveToFirst();
 while(result.isAfterLast()){
  int id = result.getInt(0);
  String name = result.getString(result.getColumnIndex("login"));
  String inventory = result.getString(2);
  result.moveToNext();
 }
result.close();
```

FONCTIONS DE REQUETES

- db.rawQuery(query,selectionArgs);

  OK pour une requête SQL fixe, mais quoi si nous avons besoin de plus de flexibilité?

- db.query() construit la requête SQL depuis des paramètres
  1. Le nom de la table sur laquelle vous voulez effectuer la requête
  2. La liste des colonnes a récupérer
  3. Une clause where avec un paramètre de positionnement optionnel
  4. La liste des valeurs à remplacer pour ces paramètres de positionnement
  5. Une clause GROUP BY, si existante
  6. Une clause ORDER BY, si existante
  7. Une clause HAVING, si existante

- Dans cette fonction, tout sauf le nom de la table peut être null.

EXEMPLE DE REQUETE

```
String[] columns = new String[]{"login","password"};
String[] substitutes = new String[]{"143"}
Cursor c = db.query("user", columns, "id=?", substitutes, null, null, null);
```

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

INSERT, UPDATES ET DELETES

- Tout comme la fonction query, il y a également des fonctions similaires pour l'exécution d'inserts, deletes et updates.

  Si db est une instance de SQLiteDatabase:

  db.insert()
  db.update()
  db.delete()

- Insert et updates fonctionnent avec une instance de ContentValues

  ContentValues sont paires clé-valeurs, similaire aux maps en Java.

```
SQLiteDatabase db = this.getReadableDatabase();
String[] columns = new String[]{"login","password"};
String[] substitutes = new String[]{“seck"}
Cursor c = db.query("user", columns, "login=?",substitutes,null,null,null);
if (cursor != null && cursor.getCount()!=0) {
while (cursor.moveToNext()) {
    cursor.getString(cursor.getColumnIndex("login"));
}}
```

Exemple d’insert: (login, password sont des constantes qui représentent les noms de colonnes)

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues cv = new ContentValues();
cv.put("login", login);
cv.put("password", password);
db.insert("user", null, cv);
```

Exemple d’update:

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues cv = new ContentValues();
cv.put("password", password);
db.update("user", cv, "login="+login, null);
```

Exemple de delete:
fonctionne comme un update, mais sans ContentValues

```
SQLiteDatabase db = this.getWritableDatabase();
db.delete("user", "login="+login, null);
```

Sommaire

Base de données

Introduction à SQLite
SQLiteOpenHelper et la création d'une base de données
Ouvrir et fermer une base de données
Utiliser des cursors
Inserts, updates et deletes
Transactions

TRANSACTIONS

- Multiples opérations sont par défaut considérées comme des multiples transactions
- Chaque transaction peut être lourde
- Il est possible de regrouper ces opérations dans une transaction

TRANSACTIONS : EXEMPLE

```
try{
db.beginTransaction();
    for(ContentValues cv:values){
            db.insert("table", "", cv);
    }
db.setTransactionSuccessful();
    }
catch(SQLException e){
    e.printStackTrace();
    }
finally{
    db.endTransaction();
    }
```

FIN