
# DEVELOPPEMENT MOBILE ANDROID

## Bases de données SQLite

INSA BADJI Doctorant à l'Université de Thiès / Tuteur à

# Séquence 6: Bases de données

- Objectif: Manipuler une base de données
  - Afficher
  - Insérer
  - Supprimer
  - Mise à jour
- Plan:
  - Création Base de données avec SQLite
  - Manipulation Cursor Objet
  - Manipuler la Base de données
  - Exemple Lecture / Ecriture avec SQLite

# Bases de données SQLite

- Basé sur le cours de Jean-Marc Farinone
- Et du Pr. Ousmane SALL

# Une BD Android = une SQLiteDatabase

- Dans le code, une base de données est modélisée par un objet de la classe `android.database.sqlite.SQLiteDatabase`
- Cette classe permet donc d'insérer des données (`insert()`), de les modifier (`update()`), de les enlever (`delete()`), de lancer des requêtes SELECT (par `query()`) ou des requêtes qui ne retournent pas de données par `execSQL()`
- Les requêtes Create, Read, Update, Delete (~ INSERT, SELECT, UPDATE, DELETE de SQL) sont dites des requêtes CRUD

```java
package com.example.bdandroidapp;

public class Etudiant {
    private int id;
    private String nom;
    private String prenom;
    private String filiere;

    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 getPrenom() { return prenom; }

    public void setPrenom(String prenom) { this.prenom = prenom; }

    public String getFiliere() { return filiere; }

    public void setFiliere(String filiere) { this.filiere = filiere; }

    @Override
    public String toString() {
        return "Etudiant{" +
                "id=" + id +
                ", nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", filiere='" + filiere + '\'' +
                '}';
    }
}
```

# Une classe d'aide (helper)

- Pour créer et/ou mettre à jour une BD, on écrit une classe qui hérite de la classe abstraite `android.database.sqlite.SQLiteOpenHelper`
- Cela nécessite de créer un contructeur qui appelera un des constructeurs avec argument de `SQLiteOpenHelper` : il n'y a pas de constructeur sans argument dans `SQLiteOpenHelper`
- Le constructeur de `SQLiteOpenHelper` utilisé est `public 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)

# Du helper à SQLiteDatabse

- = de la classe d'aide à la base de données
- Le constructeur précédent est un proxy qui est exécuté rapidement
- La BD sera réellement créée au lancement de `getWritableDatabase()` (pour une base en lecture et écriture) ou de `getReadableDatabase()` (pour une base en lecture seule) sur cet objet de la classe d'aide
- 
```
public class EtudiantOpenHelper extends SQLiteOpenHelper {…}
baseHelper = new EtudiantOpenHelper(context, nomBD, null, 1);
maBaseDonnees = baseHelper.getXXXDatabase();
```

- et un helper est (évidemment) associé à une base de données. XXX est soit Write soit Read

# Création, mise à jour d'une BD : code du helper

- Sur un objet d'une classe héritant de `SQLiteOpenHelper` (classe d'aide), certaines méthodes sont appelées automatiquement :
  - `public void onCreate(SQLiteDatabase db)` est appelée automatiquement par l'environnement d'exécution quand la BD n'existe pas. On met le code de création des tables et leurs contenus dans cette méthode
  - `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` quand le numéro de version de la BD a été incrémentée
- Ces deux méthodes sont abstraites dans la classe de base et doivent donc être implémentées dans la classe d'aide
- `maBaseDonnees` de class `SQLiteDatabase` sera obtenu comme retour de `getWritableDatabase()` Ou `getReadableDatabase()`

# Code de la classe d'aide EtudiantOpenHelper

```
public class EtudiantOpenHelper extends SQLiteOpenHelper {
    String requeteCreationTable = "Create table Etudiants(id integer primary key autoincrement, " +
            "nom text, prenom text, filiere text);";

    public EtudiantOpenHelper(Context context, String name,
            CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(requeteCreationTable);
        Log.i("BD Etudiants", "Table Etudiants crée avec Succés");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Ici nous supprimons la base et les données pour en créer une nouvelle
        // ensuite. Vous pouvez créer une logique de mise à jour propre à votre base permettant
        // de garder les données à la place.
        db.execSQL("drop table Etudiants;");
        // Création de la nouvelle structure.
        onCreate(db);
    }
}
```

# Insérer des données dans une SQLiteDatabase (1/2)

- Ayant obtenu une SQLiteDatabase, on utilise les méthodes de cette classe pour faire des opérations sur la base de données
- public long insert (String table, String nullColumnHack, ContentValues values) insert dans la table table les valeurs indiquées par values
- values, de classe ContentValues, est une suite de couples (clé, valeur) où la clé, de classe String, est le nom de la colonne et valeur, sa valeur
- Bref on prépare tout, la ligne à insérer en remplissant values par des put() su  cessifs puis on lance insert()
- Exemple :

```
public long insertEtudiant(Etudiant etudiant) {
    ContentValues valeurs = new ContentValues();
    valeurs.put("nom", etudiant.getNom());
    valeurs.put("prenom", etudiant.getPrenom());
    valeurs.put("filiere", etudiant.getFiliere());
    Log.i("BD Etudiants", "Etudiant " + etudiant.toString()
            + " inséré avec Succés");
    return maBaseDonnees.insert("Etudiants", null, valeurs);
}
```

# Insérer des données dans une SQLiteDatabase (2/2)

- Le second argument nullColumnHack est le nom de colonne qui aura la valeur NULL si values est vide. Cela est du 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). Bref cela sert seulement lorsque values est vide !
- Cette méthode insert() retourne le numéro de la ligne insérée ou -1 en cas d'erreur
- En fait cette insertion est le Create (C) de CRUD

# Récupérer des données dans une SQLiteDatabase

- La méthode la plus simple (!) pour récupérer des données (~ SELECT) est : public Cursor query (String table, String[] columns, String whereClause, String[] selectionArgs, String groupBy, String having, String orderBy)
  - columns est la liste des colonnes à retourner. Mettre null si on veut toutes les colonnes
  - whereClause est la clause WHERE d'un SELECT (sans le mot WHERE). Mettre null si on veut toutes les lignes
  - selectionArgs est utile si dans whereClause (~ WHERE), il y a des paramêtres notés ?. Les valeurs de ces paramêtres sont indiqués par selectionArgs. Bref en général on met null
  - groupBy est la clause GROUP BY d'un SELECT (sans les mots GROUP BY). Utile pour des SELECT COUT(*). Bref on général on met null
  - having indique les groupes de lignes à retourner (comme HAVING de SQL = un WHERE sur résultat d'un calcul, pas sur les données)
  - orderBy est la clause ORDER BY d'un SELECT (sans les mots ORDER BY). Mettre null si on ne tient pas compte de l'ordre

# Exemple de requête SELECT pour Android

- Euh, la classe android.database.sqlite.SQLiteQueryBuilder est (semble) faite pour cela
- Voir à http://sqlite.org/lang.html
- Sinon, quelques exemples
- Rappel : SELECT peut être obtenu avec : public Cursor query (String table, String[] columns, String whereClause, String[] selectionArgs, String groupBy, String having, String orderBy)
- 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 + '"'

# rawQuery() : requête SELECT pour Android

- La méthode rawQuery() de SQLiteDatabase permet de lancer des simples requêtes SELECT comme SELECT * FROM '' + TABLE_CONTACTS WHERE *condition paramétrée*
- Sa signature est public Cursor rawQuery (String sql, String[] selectionArgs) où sql est une requête SELECT (qui ne doit pas être terminée par ;) et selectionArgs est le tableau qui fixe les valeurs des paramètres (noté ?) dans la clause WHERE
- Par exemple :

```
String countQuery = "SELECT * FROM Etudiants" ;
SQLiteDatabase db = baseHelper.getReadableDatabase();
Cursor cursor = db.rawQuery(countQuery, null);
```

# L'objet Cursor (1/2)

- La méthode `query()` retourne un `android.database.Cursor` (~ `java.sql.ResultSet`).`android.database.Cursor` est une interface. Ce qui est retourné est un objet d'une classe qui implémente cette interface
- C'est similaire à JDBC. Le `Cursor` représente un ensemble de "lignes" contenant le résultat de la requête SELECT
- `public int getCount()` retourne le nombre de lignes contenues dans le `Cursor`
- On se positionne au début du `Cursor` (= avant la première ligne) par la méthode `public boolean moveToFirst()` (qui retourne `false` si le `Cursor` est vide
- On teste si on a une nouvelle ligne à lire par la méthode `public boolean moveToNext()` (qui retourne `false` si on était positionné après la dernière ligne)

# L'objet Cursor (2/2)

- On récupère la `columnIndex` cellule de la ligne par la méthode : `public XXX getXXX(int columnIndex).columnIndex` est (évidemment) le numéro de la cellule dans la requête. `XXX` est le type retourné (String, short, int, long, float, double)
- Il n'y a pas de getXXX(String nomDeColonne) contrairement à JDBC
- On referme le `Cursor` (et libère ainsi les ressources) par `public void close ()`
- On peut avoir des renseignements sur le résultat de la requête SELECT (* FROM ...) (Méta données) à l'aide du `Cursor` comme :
  - `public int getColumnCount()` qui retourne le nombre de colonnes contenues dans le `Cursor`
  - `public String getColumnName(int columnIndex)` qui retourne le nom de la `columnIndex` ième colonne

# L'accès aux données : un DAO

- Manipuler le `Cursor`, c'est bien. C'est "un peu" de la programmation "bas niveau"
- Bref un DAO (= Data Access Object), voire une façade s'impose !
- Pour accéder aux données, on masque les bidouilles sous jacentes (requête SQL, etc.) par un objet d'une classe DAO : un bon design pattern !
- Les bidouilles SQL masquées par le DAO sont :
  - insérer des données dans la BD par la méthode `insert()` de la classe `SQLiteDatabase`
  - retourner toutes les données d'une table par la méthode `query()` de la classe `SQLiteDatabase`

# A propos de Supprimer des lignes dans une table :DELETE

- Pour supprimer des lignes dans une table, la méthode utilisée (de la classe `SQLiteDatabase`) est `public int delete (String table, String whereClause, String[] whereArgs)`
  - `table` est la table à manipuler
  - `whereClause` est la clause WHERE filtrant les lignes à supprimer. Si la valeur est `null`, toutes les lignes sont détruites
  - `whereArgs` indiquent les valeurs à passer aux différents arguments de la clause WHERE qui sont notés `?` dans `whereClause`
- Cette méthode retourne le nombre de ligne qui ont été supprimées
- Exemple : Voir code du DAO

# Modifier une table : UPDATE de SQL pour Android

- Pour mettre à jour des lignes dans une table, la méthode utilisée (de la classe `SQLiteDatabase`) est `public int update (String table, ContentValues values, String whereClause, String[] whereArgs)` où :
  - `table` est la table qui doit être mise à jour
  - `values` est une suite de couples (clé, valeur) où la clé, de classe `String`, est le nom de la colonne et valeur, sa valeur
  - `whereClause` est la clause WHERE filtrant les lignes à mettre à jour. Si la valeur est `null`, toutes les lignes sont mises à jour
  - `whereArgs` indiquent les valeurs à passer aux différents arguments de la clause WHERE qui sont notés `?` dans `whereClause`
  - Cette méthode retourne le nombre de ligne qui ont été affectées
- Exemple : Voir DAO

# Bases de données SQLite

- Android dispose d'une base de donnée relationelle basée sur SQLite.
- Attention: la base doit être utilisée avec parcimonie, cela fournit un moyen efficace de gérer une petite quantité de données.

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

  private static final int DATABASE_VERSION = 2;
  private static final String DICTIONARY_TABLE_NAME = "dictionary";
  private static final String DICTIONARY_TABLE_CREATE =
              "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
              KEY_WORD + " TEXT, " +
              KEY_DEFINITION + " TEXT);";

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

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

# *Lecture / Ecriture dans la BDD*

- Pour réaliser des écritures ou lectures, on utilise les méthodes **getWritableDatabase()** et **getReadableDatabase()** qui renvoient une instance de **SQLiteDatabase**. Sur cet objet, une requête peut être exécutée au travers de la méthode **query()**:

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

- L'objet de type **Cursor** permet de traiter la réponse (en lecture ou écriture), par exemple:
  - **getCount()**: nombre de lignes de la réponse
  - **moveToFirst()**: déplace le curseur de réponse à la première ligne
  - **getInt(int columnIndex)**: retourne la valeur (int) de la colonne passée en paramètre
  - **getString(int columnIndex)**: retourne la valeur (String) de la colonne passée en paramètre
  - **moveToNext()**: avance à la ligne suivante
  - **getColumnName(int)**: donne le nom de la colonne désignée par l'index
  - ...

# Le code du DAO DataBaseHelper.class

```
package com.example.sqliteapp;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import android.widget.Toast;

import androidx.annotation.Nullable;

public class DatabaseHelper extends SQLiteOpenHelper {
    public static final String DATABASE_NAME = "Etudiants.db";
    public static final String TABLE_NAME = "etudiant_table";
    public static final String COL_1 = "ID";
    public static final String COL_2 = "PRENOM";
    public static final String COL_3 = "NOM";
    public static final String COL_4 = "NOTES";


    public DatabaseHelper(@Nullable Context context) {
        super(context, DATABASE_NAME, null, 1);
        SQLiteDatabase db = this.getWritableDatabase();
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table "+ TABLE_NAME + "( ID INTEGER PRIMARY KEY AUTOINCREMENT, PRENOM TEXT, NOM
TEXT, NOTES TEXT)");
    }
```

# Le code du DAO DataBaseHelper.class (suite)

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS "+TABLE_NAME);
    onCreate(db);
}
public boolean insertData(String prenom, String nom, String note){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COL_2, prenom);
    values.put(COL_3, nom);
    values.put(COL_4, note);
    long resultat = db.insert(TABLE_NAME, null, values);
    Log.i("##### check insert", " result "+resultat);
    if (resultat == -1)
        return false;
    else
        return true;
}

public Cursor getAllData(){
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor res = db.rawQuery("SELECT * FROM "+TABLE_NAME, null);
    return res;
}
```

# Le code du DAO DataBaseHelper.class (suite)

```
    // Mise a jour d'un enregistrement
    public boolean updateData(String id, String nom, String prenom,
String note){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues valeurs = new ContentValues();
        valeurs.put(COL_1, id);
        valeurs.put(COL_2, prenom);
        valeurs.put(COL_1, nom);
        valeurs.put(COL_1, note);
        db.update(TABLE_NAME, valeurs, "ID = ?", new String[]{id});
        return true;
    }

    public Integer deleteData (String id){
        SQLiteDatabase db = this.getWritableDatabase();
        return db.delete(TABLE_NAME, "ID = ?", new String[]{id});
    }
}
```

# Le code du DAO MainActivity.class

```
public class MainActivity extends AppCompatActivity {
    DatabaseHelper myDb;
    Button button ;
    Button buttonAfficher ;
    Button buttonViewUpdate ;
    Button buttonDelete;
    EditText editTextPrenom, editTextNom, editTextNote, editTextId;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        myDb = new DatabaseHelper(this);

        editTextNom = (EditText)findViewById(R.id.editTextNom);
        editTextPrenom = (EditText)findViewById(R.id.editTextPrenom);
        editTextId =(EditText)findViewById(R.id.editTextId);
        editTextNote = (EditText)findViewById(R.id.editTextNote);
        button = (Button)findViewById(R.id.btnSoumettre);

       button.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               boolean isInserted = myDb.insertData(editTextPrenom.getText().toString(),
                       editTextNom.getText().toString(),
                       editTextNote.getText().toString());
               if (isInserted == true)
                   Toast.makeText(getApplicationContext(),
                           "Insertion des donnees avec avec succes", Toast.LENGTH_LONG).show();
               else
                   Toast.makeText(getApplicationContext(),
                           "IEchec de l'insertion des donnees ", Toast.LENGTH_LONG).show();
           }
       });
```

# Le code du DAO MainActivity.class (suite)

```
 buttonAfficher = (Button) findViewById(R.id.btnAfficher);
buttonAfficher.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Cursor res = myDb.getAllData();
        if (res.getCount() == 0){
            AfficherMessage("Erreur","Aucune donnee disponible !");
        }
        StringBuffer buffer = new StringBuffer();
        while (res.moveToNext()){
            buffer.append("Id :"+res.getString(0)+"\n");
            buffer.append("Prenom :"+res.getString(1)+"\n");
            buffer.append("Nom :"+res.getString(2)+"\n");
            buffer.append("Note :"+res.getString(3)+"\n");
        }

        // Affichage des donnees recuperes
        AfficherMessage("Donnees",buffer.toString());
    }
});

buttonViewUpdate = (Button)findViewById(R.id.btnUpdate);
buttonViewUpdate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
         boolean isUpdate = myDb.updateData(editTextId.getText().toString(),
                 editTextPrenom.getText().toString(),
                 editTextNom.getText().toString(),
                 editTextNote.getText().toString()
                 );
         if (isUpdate == true)
             Toast.makeText(getApplicationContext(),
                     "Mise a jour avec succes", Toast.LENGTH_LONG).show();
         else
             Toast.makeText(getApplicationContext(),
                     "Echec de la mise a jour ", Toast.LENGTH_LONG).show();
    }
});
```

# Le code du DAO MainActivity.class (suite)

```
buttonDelete =(Button) findViewById(R.id.btnDelete);
        buttonDelete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Integer deleteRows = myDb.deleteData(editTextId.getText().toString());
                if (deleteRows > 0)
                    Toast.makeText(getApplicationContext(),
                            "Suppression avec succes", Toast.LENGTH_LONG).show();
                else
                    Toast.makeText(getApplicationContext(),
                            "Echec de la suppression ", Toast.LENGTH_LONG).show();
            }
        });
    }

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

# Le code du DAO MainActivity.xml

```
<RelativeLayout
    android:layout_width="409dp"
    android:layout_height="729dp"
    tools:layout_editor_absoluteX="1dp"
    tools:layout_editor_absoluteY="1dp">

    <EditText
        android:id="@+id/editTextNom"
        android:layout_width="226dp"
        android:layout_height="wrap_content"

android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="81dp"
        android:layout_marginTop="45dp"
        android:text="Nom" />
```

```
<EditText
        android:id="@+id/editTextPrenom"
        android:layout_width="225dp"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="82dp"
        android:layout_marginTop="115dp"
        android:text="Prénom" />

    <EditText
        android:id="@+id/editTextNote"
        android:layout_width="225dp"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="71dp"
        android:layout_marginTop="185dp"
        android:text="Note" />
```

# Le code du DAO MainActivity.xml (suite)

```
<EditText
    android:id="@+id/editTextId"
    android:layout_width="225dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="63dp"
    android:layout_marginTop="251dp"
    android:text="Id a editer" />

<Button
    android:id="@+id/btnSoumettre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="313dp"
    android:text="Enregister à la Base de
données" />
```

```
<Button
    android:id="@+id/btnAfficher"
    android:layout_width="285dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="36dp"
    android:layout_marginTop="383dp"
    android:text="Afficher les donees" />

<Button
    android:id="@+id/btnUpdate"
    android:layout_width="285dp"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="34dp"
    android:layout_marginTop="445dp"
    android:text="Mise a jour" />
```

# Le code du DAO MainActivity.xml (suite)

```
<Button
            android:id="@+id/btnDelete"
            android:layout_width="285dp"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginLeft="30dp"
            android:layout_marginTop="507dp"
            android:text="Supprimer" />
    </RelativeLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

# Fin de la séquence 3