

# Bases De Données, SQLite



UVCI

# Table des matières



<b>Objectifs</b>	3
<b>Introduction</b>	4
<b>I - Bases De Données, SQL-lite</b>	6
1. SQLite .....	6
2. Exercice .....	9
3. Les curseurs .....	10
4. Exercice .....	12

# Objectifs

À la fin de cette leçon, l'apprenant sera capable de :

- Stocker les informations dans *une* base de données;

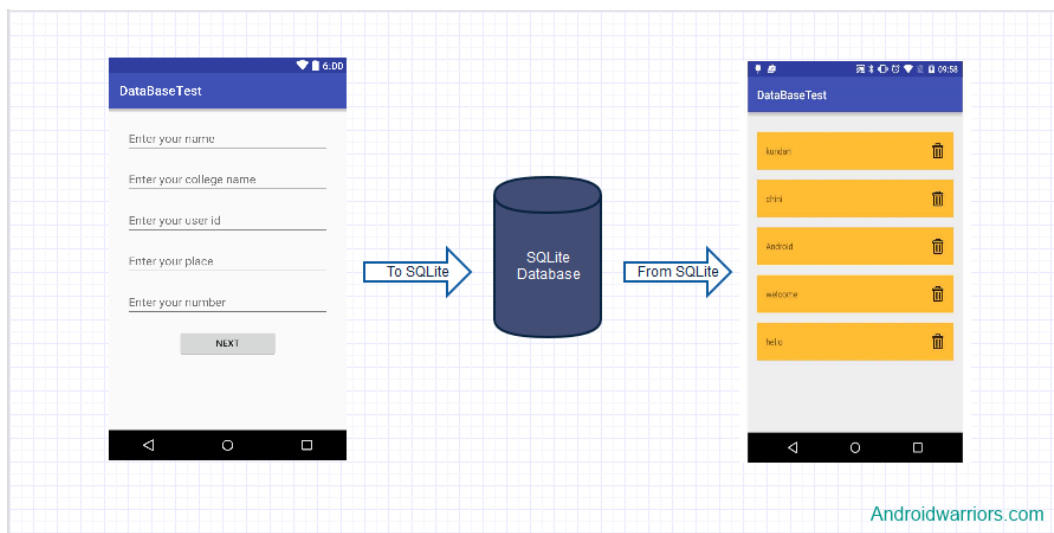
# Introduction

- Une base de données est un dispositif permettant de stocker, et de retrouver l'intégrité de données brutes en rapport à un thème ou une activité.
- L'agencement adopté pour organiser les informations s'appelle le schéma.
- L'unité de base de cette structure s'appelle *la table*.

Une table regroupe des ensembles d'informations qui sont composés de manière similaire.

Une entrée dans une table s'appelle *un enregistrement*, ou un tuple.

Chaque entrée est caractérisée par plusieurs renseignements distincts, appelés des *champs* ou *attributs*.



Par exemple, une table peut contenir le prénom, le nom et l'âge de plusieurs utilisateurs.

Il est possible de représenter une table par un tableau, où les champs seront symbolisés par les colonnes du tableau et pour lequel chaque ligne représentera une entrée différente.

Id	Nom	Prénom	Age
1	Gates	Bill	62
2	Zuckerberg	Mark	33
3	Musk	Elon	46

Une manière simple d'identifier les éléments dans une table est de leur attribuer *une clé*.

Cette clé doit être unique, on dit qu'il s'agit d'une *clé primaire*.

Généralement on utilise le champs *id*.

Cette clé peut aussi servir à faire le lien entre tables.

Lorsque l'on utilise cette clé dans une autre table, on dit que le champ est une clé étrangère.

Elles garantissent la cohérence des données entre tables.

# Bases De Données, SQL-lite



## 1. SQLite

### Définition

---

*SQLite* est une bibliothèque écrite en langage C qui propose un moteur de base de données relationnelle accessible par le langage SQL.

*SQLite* est le moteur de base de données le plus utilisé au monde. *SQLite* est intégré à tous les téléphones mobiles et à la plupart des ordinateurs et est intégré à plusieurs autres applications utilisées quotidiennement.



*SQLite* est un moteur de bases de données libre qui implémente la plus part des fonctionnalités du SQL92 (standard). *SQLite* utilise donc la plus part des commandes SQL (CREATE, INSERT, UPDATE, DELETE, et SELECT )

### *Types de données*

SQLite n'accepte que 5 types de données:

Type	Définition
<i>NULL</i>	Valeur vide
<i>INTEGER</i>	Entier signé
<i>REAL</i>	Nombre réel
<i>TEXT</i> <i>VARCHAR</i>	Champ texte
<i>BLOB</i>	Champ binaire(image)

Pour utiliser *SQLite* avec *Android*, vous n'avez pas besoins de télécharger et d'exécuter quoi que ce soit puisque ce moteur de bases de données vient avec *Android*. Il vous suffit d'utiliser les classes nécessaires d'*Android* pour créer et manipuler une base de données *SQLite*.

Pour utiliser des bases de données sous *Android*, il est fortement conseillé d'utiliser *SQL-lite*.

Contrairement à MySQL par exemple, *SQLite* ne nécessite pas de serveur pour fonctionner, ce qui signifie que son exécution se fait dans le même processus que celui de l'application.

Par conséquent, une opération massive lancée dans la base de données aura des conséquences visibles sur les performances de votre application.

Il est donc important de maîtriser son implémentation afin de ne pas pénaliser le reste de l'application.

*SQLite* a été inclus dans le cœur même d'*Android*, c'est pourquoi chaque application peut avoir sa propre base.

Les bases de données sont stockées dans le répertoire DATA /data/APP\_NAME/databases/FILENAME.

Il est possible d'avoir plusieurs bases de données par application, cependant elles ne sont accessibles qu'au sein de l'application elle-même. Chaque fichier créé l'est selon le mode *MODE\_PRIVATE*, par conséquent les bases ne sont accessibles qu'au sein de l'application elle-même. Notez que ce n'est pas pour autant qu'une base de données ne peut pas être partagée avec d'autres applications.

### Structuration

Afin de structurer le code Java, il est fortement conseillé de créer plusieurs classes Java, chacune ayant un rôle différent et de les placer dans un dossier différent du reste du code.

Le fait de fragmenter son code en plusieurs classes permet une meilleure maintenabilité du code dans le temps.

Structure conseillée :

- Une classe contenant simplement des interfaces qui définissent les tables.
- Une classe permettant la connexion avec la base de donnée.
- Une classe par table, permettant d'effectuer des opérations sur la table.

## Exemple

- Exemple de classe contenant les interfaces de chaque tables.
- Aussi appelé définition du modèle.
- Hériter de *BaseColumns* permet de ne pas redéfinir le champs id.

```
public class DatabaseAplis {
    public interface Person extends BaseColumns {
        String tableName = "person";
        String columnLastName = "last_name";
        String columnLastNameType = "TEXT";
        String columnFirstName = "first_name";
        String columnFirstNameType = "TEXT";
        String columnAge = "age";
        String columnAgeType = "INTEGER";
        String columnImg = "img";
        String columnImgType = "INTEGER";
        int num_columnID = 0;
        int num_columnLastName = 1;
        int num_columnFirstName = 2;
        int num_columnAge = 3;
        int num_columnImg = 4;
    }
}
```

## Les méthodes

Deux méthodes populaires pour accéder et exploiter à une base de données SQLite avec Android. Une qui utilise surtout la classe *SQLiteDatabase* (proche de la Base de donnée) et celle qui utilise la classe *SQLiteOpenHelper* et *SQLiteDatabase* (proche de la programmation)

Pour pouvoir accéder à la base de données, il faut créer une classe permettant la connexion.

Cette classe doit hériter de *SQLiteOpenHelper* et redéfinir les méthodes :

- *onCreate(SQLiteDatabase db\_)*, créait la BDD si elle n'existe pas.
- *onUpgrade(SQLiteDatabase db\_, int oldVersion\_, int newVersion\_)*, permet de gérer les mises à jour de la base de données.

Généralement, elle contient aussi le code SQL permettant la création et la mise à jour de la BDD sous forme de constantes.

## Attention

ces méthodes n'ont pas de lien avec le cycle de vie d'une activité.

## Exemple

SQLiteOpenHelper



```

public class DatabaseOpenHelper extends SQLiteOpenHelper {

    public static final int     databaseVersion     = 2;
    public static final String  databaseName        = "dataBase.db";

    private static final String SQLCreateTablePerson =
        "CREATE TABLE "+ DatabaseAplis.Person.tableName + " (" +
        DatabaseAplis.Person._ID + " INTEGER PRIMARY KEY," +
        DatabaseAplis.Person.columnLastName + " " +
        DatabaseAplis.Person.columnLastNameType + "," +
        DatabaseAplis.Person.columnFirstName + " " +
        DatabaseAplis.Person.columnFirstNameType + "," +
        DatabaseAplis.Person.columnAge + " " +
        DatabaseAplis.Person.columnAgeType + "," +
        DatabaseAplis.Person.columnImg + " " +
        DatabaseAplis.Person.columnImgType +
        " )";

    private static final String SQLDeleteTablePerson =
        "DROP TABLE IF EXIST " + DatabaseAplis.Person.tableName;

    public DatabaseOpenHelper(Context context_)
    {
        super(context_, databaseName, factory: null, databaseVersion);
    }

    @Override
    public void onCreate(SQLiteDatabase db_) {
        db_.execSQL(SQLCreateTablePerson);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db_, int oldVersion_, int newVersion_) {
        db_.execSQL(SQLDeleteTablePerson);
        onCreate(db_);
    }
}

```

## 2. Exercice

### Exercice

---

Une base de données est un dispositif permettant de stocker un ensemble d'informations de manière structurée.

- ☐ Vrai
- ☐ Faux

### Exercice

---

L'unité de base de la structure d'une base de données s'appelle la

- ☐ Classe

- ☐ Table
- ☐ Occurrence
- ☐ Aucune réponse juste

## Exercise

*SQLite* est inclus dans le cœur d'Android.

Vrai ou Faux ?

- ☐ Vrai
- ☐ Faux

### 3. Les curseurs

Pour pouvoir manipuler les données retournées par les *DAO*, Android intègre une classe du nom de *Cursor*.

Ce sont des objets qui contiennent les résultats d'une recherche dans une base de données.

Ils fonctionnent comme des tableaux, ils contiennent les colonnes et lignes qui ont été renvoyées par la requête.

👉 Exemple : code permettant de retourner le contenu d'une table :

Retourne un `Cursor` contenant la totalité de la table.

```
private Cursor getAllPersons()
{
    SQLiteDatabase db = m_sqlLiteHelper.getReadableDatabase();
    return db.query(DatabaseAplis.Person.tableName,
        columns: null, selection: null, new String[] {},
        groupBy: null, having: null, orderBy: null);
}
```

*quelques méthodes de l'objet Curseur:*

Pour parcourir les résultats d'une requête, il faut procéder ligne par ligne.

- boolean *moveToFirst()* : pour aller à la première ligne.
- boolean *moveToLast()* : pour aller à la dernière ligne.
- boolean *moveToPosition(int position)* : pour aller à la position voulue.
- boolean *moveToNext()* : pour aller à la ligne suivante.
- boolean *moveToPrevious()* : pour aller à la ligne précédente.
- *getCount()* permet d'obtenir le nombre d'enregistrements de la requête.
- *isAfterLast()* permet de vérifier si la fin du résultat de la requête a été atteint.

Les retours sont des booléens, *true* si l'opération a réussi, *false* si elle a échoué.

 *Attention*

Un curseur doit être fermé avec la méthode *close()*.

Pour récupérer la position actuelle, il faut utiliser la méthode *int getPosition()* qui retourne la position courante.

Lorsque la ligne est sélectionnée, il faut récupérer le contenu de chaque colonnes.

Pour cela, il suffit d'utiliser une méthode du style *X getX( int columnIndex)*.

- long getLong(int columnIndex).
- String getString(int columnIndex).

```
public ArrayList<Person> getPersons()
{
    ArrayList<Person> persons = new ArrayList<>();
    //---
    Cursor cursor = getAllPersons();
    //---
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        persons.add(new Person( cursor.getString(DatabaseAplis.Person.num_columnLastName),
                                cursor.getString(DatabaseAplis.Person.num_columnFirstName),
                                cursor.getInt(DatabaseAplis.Person.num_columnAge),
                                cursor.getInt(DatabaseAplis.Person.num_columnID),
                                cursor.getInt(DatabaseAplis.Person.num_columnImg)));
        cursor.moveToNext();
    }
    //---
    cursor.close();
    return persons;
}
```

Une fois la classe permettant d'établir la connexion écrite, il faut écrire les classes (une par table) permettant de manipuler les tables.

Chacune de ces classes doit avoir un accès à la classe gérant la connexion à la BDD.

On nomme ces classes *DAO (Data Access Objects)*.

Elles font le lien entre des classes représentant des objets et les tables correspondant dans la BDD.

```
public class DBPerson {

    private SQLiteOpenHelper m_sqlLiteHelper;

}

public DBPerson(SQLiteOpenHelper sqlLiteOpenHelper) { m_sqlLiteHelper = sqlLiteOpenHelper; }

public void insert(Person person_)
{
    SQLiteDatabase db = m_sqlLiteHelper.getWritableDatabase();
    //---
    ContentValues values = new ContentValues();
    values.put(DatabaseAplis.Person.columnLastName, person_.getLastName());
    values.put(DatabaseAplis.Person.columnFirstName, person_.getFirstName());
    values.put(DatabaseAplis.Person.columnAge, person_.getAge());
    values.put(DatabaseAplis.Person.columnImg, person_.getImg());
    //---
    db.insert(DatabaseAplis.Person.tableName, nullColumnHack: null, values);
}
```

## Exemple

Mise à jour

```

public void update(Person person_)
{
    SQLiteDatabase db = m_sqliteHelper.getWritableDatabase();
    //---
    ContentValues values = new ContentValues();
    values.put(DatabaseAplis.Person.columnLastName, person_.getLastName());
    values.put(DatabaseAplis.Person.columnFirstName, person_.getFirstName());
    values.put(DatabaseAplis.Person.columnAge, person_.getAge());
    values.put(DatabaseAplis.Person.columnImg, person_.getImg());
    //---
    String selection = DatabaseAplis.Person._ID + " = ?";
    String[] selectionArgs = {String.valueOf(person_.getId())};
    //---
    db.update(DatabaseAplis.Person.tableName, values,
        selection, selectionArgs);
}

```

## Exemple

### Suppression

```

public void delete(Person person_)
{
    SQLiteDatabase db = m_sqliteHelper.getWritableDatabase();
    //---
    String selection = DatabaseAplis.Person._ID + " = ?";
    String[] selectionArgs = {String.valueOf(person_.getId())};
    //---
    db.delete(DatabaseAplis.Person.tableName,
        selection,
        selectionArgs);
}

```

## 4. Exercice

### Exercice

La méthode getCount() permet de :

- ☐ obtenir le nombre d'enregistrements de la requête
- ☐ obtenir l'index de colonne pour un nom de colonne de la table.
- ☐ vérifier si la fin du résultat de la requête a été atteint.

### Exercice

Un curseur doit être fermé avec la méthode :

- ☐ Close()
- ☐ getClose()
- ☐ Return()

☐ Je ne sais pas

