Utilisation de base de données SQLite avec Android

Les classes utiles

- **SQLiteOpenHelper** : classe d'assistance pour gérer la création de la base de données et la gestion des versions
- **ContentValue** : classe utilisée pour stocker un ensemble de valeurs que le **ContentResolver** peut traiter
- **Cursor** : est une interface qui donne accès en lecture-écriture à l'ensemble des résultats retournés par une requête de base de données

Types de données SQLite

- **INTEGER** : nombres entiers (sans virgule)
- **REAL** : nombres réels (avec virgule)
- **TEXT** : chaînes de caractères
- **BLOB** : données brutes (par exemple si l'on veut mettre une image dans la base de données)

Création d'une table

Syntaxe SQL
CREATE TABLE nom_de_la_table (nom_du_champ_1 type {contraintes}, nom_du_champ_2 type {contraintes}, …);

Pour chaque attribut d'une table, on doit déclarer au moins deux informations :

- Son nom, afin de pouvoir l'identifier
- Son type de données

Il est aussi possible d'ajouter des contraintes pour chaque attribut à l'emplacement de {contraintes}. On trouve les contraintes suivantes :

- **PRIMARY KEY** : pour désigner la clé primaire
- **NOT NULL** : pour indiquer que cet attribut de peut pas avoir la valeur NULL
- **CHECK** : afin de vérifier que la valeur de cet attribut est cohérente
- **DEFAULT** : sert à préciser une valeur par défaut

Création de la base

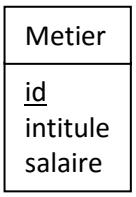
Il existe deux types de requêtes SQL . celles qui envoie une réponse (comme la sélection), et celles qui n'envoient aucune réponse. Afin d'exécuter une requête SQL du second type, nous allons utiliser la méthode suivante :

void execSQL(String sql)

Nous utiliserons cette méthode dès lors qu'il ne s'agira pas de faire un **SELECT**, **UPDATE**, **INSERT** ou **DELETE**.

Exemple de création d'une table:

Schéma	Java

	public class DatabaseHandler extends SQLiteOpenHelper {
	
	public static final String METIER_KEY = "id";
	public static final String METIER_INTITULE = "intitule";
	public static final String METIER_SALAIRE = "salaire";
	public static final String METIER_TABLE_NAME = "Metier";
	
	public static final String METIER_TABLE_CREATE =
	"CREATE TABLE " + METIER_TABLE_NAME + " (" +
	METIER_KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
	METIER_INTITULE + " TEXT, " +
	METIER_SALAIRE + " REAL);";
	
	public DatabaseHandler(Context context, String name, CursorFactory factory, int version) {
	super(context, name, factory, version);
	}
	
	@Override
	public void onCreate(SQLiteDatabase db) {
	db.**execSQL**(METIER_TABLE_CREATE);
	}
	}

Accès à la base

Pour accéder à la base de données depuis n'importe où dans votre code, il vous suffit de construire une instance de votre **SQLiteOpenHelper** (créé précédemment et contenant l'ensemble de la structure de vos tables) avec le constructeur suivant :

SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

- **context** : le contexte de votre application
- **name** : le nom de votre base de données
- **factory** : permet de redéfinir des Cursor personalisés, on se contentera d'indiquer la valeur null pour ce paramètre
- **version** : le numéro de version voulu de votre base de données

On utilise la méthode getWritableDatabase() de la classe SQLiteDatabase pour récupérer (ou créer) une base sur laquelle vous voulez lire/écrire . L'appel de cette méthode déclenche coté système à une vérification de l'état de votre base (via son numéro de version) et appelle une méthode de SQLiteOpenHelper en fonction de son état :

- S'il s'agit de la première fois que vous appelez la base, ce sera la méthode **onCreate(SQLiteDatabase)** qui sera appelée
- Si la version fournie est plus récente que la dernière version fournie on fait alors appel à **onUpgrade(SQLiteDatabase, int, int)**

```
public class MetiersBDD {

	private static final int VERSION_BDD = 1;
	private static final String NOM_BDD = "mabase.db";

	private SQLiteDatabase bdd;

	private DatabaseHandler maBaseSQLite;

	public MetiersBDD (Context context){
		//On créer la BDD et sa table
		maBaseSQLite = new DatabaseHandler (context, NOM_BDD, null,
VERSION_BDD);
	}

	public void open(){
		//on ouvre la BDD en écriture
		bdd = maBaseSQLite.getWritableDatabase();
	}

	public void close(){
		//on ferme l'accès à la BDD
		bdd.close();
	}

	public SQLiteDatabase getBDD(){
		return bdd;
	}
}
```

Manipuler le contenu de la base de données

Pour manipuler le contenu de notre base de données, nous allons implémenter nos propres méthodes à l'aide de méthodes déjà existantes que nous allons détailler par la suite.

L'implémentation de vos propres méthodes doit se faire dans votre classe permettant l'accès à la base afin de pouvoir utiliser votre objet **SQLiteDatabase.**

Insérer des données

Pour insérer des données, on utilise la méthode :

long insert(String table, String nullColumnHack, ContentValues values)

Cette méthode renvoie le numéro de la ligne ajoutée. Les paramètres de la fonction insert sont :

- **table** : le nom de la table dans laquelle on souhaite insérer des données
- **nullColumnHack** : le nom d'une colonne à utiliser au cas où vous souhaiteriez insérer une valeur vide
- **values** : l'objet contenant les valeurs à insérer

Les **ContentValues** sont utilisés pour insérer des données dans la base. On peut dire qu'ils fonctionnent comme des Bundle puisqu'il s'agit de coupes identifiant-valeur dont l'identifiant représente les attributs des objets à insérer dans la base. L'identifiant doit obligatoirement être une chaine de caractères représentant une des colonnes de la table visée.

```
ContentValues value = new ContentValues();
value.put (« intitule »,  « Informaticien » );
value.put (« salaire »,  2400 );
bdd.insert( « Metier », null, value);

//Pas besoin de préciser de valeur pour la colonne id puisqu'elle s'incrémente tout seul
```

Supprimer des données

Pour supprimer des données, on utilise la méthode :

int delete(String table, String whereClause, String[] whereArgs)

Cette méthode renvoie le nombre de lignes supprimées. Les paramètres de la fonction delete sont :

- **table** : le nom de la table dans laquelle on souhaite supprimer des données
- **whereClause** : correspond au WHERE en SQL. Par exemple, pour sélectionner la première valeur dans la table Metier, on mettra pour whereClause la chaîne « id = 1 ». En pratique, on préférera utiliser la chaîne « id = ? » et ainsi utiliser le 3e paramètre whereArgs.
- **whereArgs** : tableau des valeurs qui remplaceront les « ? » dans whereClause. Ainsi, si whereClause vaut « LIKE ? AND salaire > ? » et qu'on cherche les métiers qui ressemblent à « ingénieur avec un salaire supérieur à 1000 € », il suffit d'insérer dans whereArgs un String[] du comme {"ingenieur", "1000"}.

```
public  void  supprimer(long  id) {
  bdd.delete(«Metier », " id  = ?", new String[] {String.valueOf(id)});
}
```

Mettre à jour des données

Pour mettre à jour des données on utilise la méthode :

int update(String table, ContentValues values, String whereClause, String[] whereArgs)

Les paramètres sont les mêmes que pour la méthode utilisée pour supprimer des données (voir ci-dessus) auxquels ont ajoute le paramètre values pour représenter les changements à effectuer dans le (ou les) enregistrements ciblés.

```
public  void  mettre_a_jour(long  id) {
    ContentValues value = new ContentValues();
    value.put(« salaire », 2635);
    bdd.update(« Metier », value, "id  = ?", new String[] {String.valueOf(id)});
}
```

Sélectionner des données

Pour sélectionner des données on utilisera la méthode :

Cursor rawQuery(String sql, String[] selectionArgs)

Les paramètres sont les suivants :

- **sql** : Contient la requête de sélection, les valeurs des paramètres de cette requête doivent être représentés par des ?
- **selectionArgs** : valeurs des arguments, ce qui viendra remplacer les ? de la requête du premier paramètre. Indiquer null si aucun paramètre dans la requête

```
Cursor c = bdd.rawQuery("select  intitule from Metier  where salaire > ?", new String[]{"1"});
```

Le résultat renvoyé est un **Cursor**, nous allons voir de quoi il s'agit et comment l'utiliser dans la partie suivante

Les curseurs

Les curseurs (**Cursor**) sont des objets qui contiennent les résultats d'une recherche dans une base de données. Ce sont des objets qui fonctionnent comme des tableaux, ils contiennent les colonnes et les lignes renvoyées par la requête.

Pour parcourir les résultats d'une requête il faut procéder ligne par ligne. Pour naviguer parmi les lignes on dispose des méthodes suivantes :

- **boolean moveToFirst()** : pour se rendre à la première ligne
- **boolean moveToLast()** : pour se rendre à la dernière ligne
- **boolean moveToPosition(int position)** : pour se rendre à la position voulue, sachant que vous pouvez savoir le nombre de lignes que contient votre Cursor à l'aide de la méthode int getCount()

Pour naviguer entre les lignes d'un curseur on utilisera :

```
while (cursor.moveToNext()) {
  // Faire quelque chose
}
cursor.close();
```

A chaque ligne de notre **Cursor** correspond un nombre de colonnes désigné par la requête effectuée sur la base de données. L'ordre des colonnes dans votre **Cursor** correspond a l'ordre renvoyé par votre requête (il est donc préférable de le mentionner dans votre requête plutôt que de mettre une '*', cela vous évitera des potentiels problèmes).

Pour récupérer les valeurs, il suffit d'utiliser la méthode :

X getX(int columnIndex)

X correspond au typage de la valeur à récupérer et columnIndex à l'index de la colonne dans laquelle se trouve la valeur.

Exemple :

```
long id = cursor.getLong(0);
String intitule = cursor.getString(1);
double salaire = cursor.getDouble(2);
```

Exemple de parcours complet d'un **Cursor** :

```
while (cursor.moveToNext()) {
    long id = cursor.getLong(0);
    String intitule = cursor.getString(1);
    double salaire = cursor.getDouble(2);
}
cursor.close();
```