Chapitre VII : Bases de Données SQLite

Introduction à SQLite

- SQLite est un moteur de base de données relationnelle léger, autonome et sans serveur.
- Créé en 2000, il est conçu pour être embarqué directement dans les applications.
- SQLite est écrit en C et est open source, ce qui le rend facilement accessible et largement adopté.

Introduction à SQLite

Caractéristiques principales de SQLite :

- **Léger et intégré :** SQLite est très compact (moins de 500 Ko), ce qui le rend idéal pour les appareils mobiles où les ressources sont limitées.
- **Aucune configuration de serveur :** contrairement aux bases de données classiques (MySQL, PostgreSQL), SQLite ne nécessite aucun processus serveur distinct ; les opérations de base de données sont directement intégrées dans l'application.
- **Basé sur des fichiers :** toutes les données sont stockées dans un seul fichier plat, ce qui rend la gestion et le déploiement très simples (fichier .sqlite ou .db).

Introduction à SQLite

Importance de SQLite dans les applications mobiles

- **Gestion efficace des données :** nécessité de stocker, organiser et accéder à de grandes quantités de données de manière structurée.
- **Performance :** accélère les opérations de recherche, d'ajout, de suppression et de mise à jour des données.
- **Persistance des données :** permet de conserver les informations même après la fermeture de l'application.
- **Exemples d'applications :** Systèmes de gestion de contacts, applications de notes, historiques de navigation.

Base de donnée

- Une base de données permet de **stocker un ensemble d'informations** de manière structurée et organisée.
- Les données sont organisées **suivant un schéma** qui définit comment les informations sont structurées et interconnectées.
- Ce modèle structuré facilite la récupération, la gestion et la manipulation des données.
- La base de données est constituée d'**entités** appelées **Tables**.
- Chaque **Table** regroupe des ensembles d'informations similaires, facilitant leur organisation et leur gestion.

Base de donnée

- Une table est composée de :
  - **Tuples** ou **Enregistrements** : Représentent les **lignes** de la table. Chaque tuple correspond à un enregistrement unique dans la table.
  - **Attributs** : Caractérisent chaque tuple ; ils sont représentés par les **colonnes** de la table. Chaque attribut représente un type spécifique d'information pour chaque enregistrement.

Nom	**Âge**	**Métier**
Ahmed	45	Auteur
Khadidja	36	Étudiante

Type de donnée

1. **INTEGER** : Représente des nombres entiers. Exemples : **1**, **42**, **-5**.
2. **REAL** : Représente des nombres à virgule flottante pour les valeurs décimales. Exemples : **3.14**, **-0.5**, **100.0**.
3. **TEXT** : représente des chaînes de caractères, idéal pour stocker des noms, des adresses, des descriptions. Exemples : **"Alice"**, **"Paris"**.
4. **BLOB** : représente des données binaires brutes, pouvant inclure des images, des fichiers audio, des documents, etc.
5. **NULL** : représente l'absence de valeur dans une colonne.

Type de donnée

- Contrairement à d'autres bases de données, SQLite utilise un système de **typage dynamique** :
  - Cela signifie que les valeurs stockées ne sont pas strictement limitées au type déclaré pour chaque colonne.
  - Par exemple, une colonne déclarée **INTEGER** peut contenir une valeur **TEXT** si nécessaire, bien que ce soit déconseillé pour des raisons de cohérence.

Création d'une table dans SQLite

- Une table est l'élément de base dans lequel les données sont organisées dans une base de données.
- Avant d'ajouter des données, il est essentiel de définir la structure de la table, spécifiant les attributs (colonnes) et leurs types de données.
- La syntaxe générale est la suivante :

```
CREATE TABLE nom_de_la_table (
  nom_colonne1 TYPE_DONNEE,
  nom_colonne2 TYPE_DONNEE,
  ...
);
```

Création d'une table dans SQLite

- **Exemple de commande :**

```
CREATE TABLE Utilisateur (
  ID INTEGER PRIMARY KEY AUTOINCREMENT,
  Nom TEXT NOT NULL,
  Age INTEGER,
  Metier TEXT
);
```

Opérations CRUD avec SQLite

- Les opérations CRUD (Create, Read, Update, Delete) sont les quatre opérations fondamentales pour manipuler des données dans une base de données.
- Elles permettent de gérer le cycle de vie des données : de leur insertion à leur suppression.
- **Create (Créer)** :
  - Utilisée pour ajouter de nouvelles données dans une table.
  - Commande SQL : **INSERT INTO**.

INSERT INTO Utilisateur (Nom, Age, Metier) VALUES ('Rachid', 29, 'Développeur');

Opérations CRUD avec SQLite

- **Read (Lire)** :
  - Utilisée pour lire ou récupérer des données de la base.
  - Commande SQL : **SELECT**.

    SELECT * FROM Utilisateur;

  - **Filtrer les résultats** : On peut ajouter des conditions pour filtrer les résultats, par exemple :

    SELECT * FROM Utilisateur WHERE Age > 30;

Opérations CRUD avec SQLite

- **Update (Mettre à jour)** :
  - Utilisée pour modifier les données existantes dans une table.
  - Commande SQL : **UPDATE**.
  - **Exemple** : Modifier le métier de **Karim** dans la table **Utilisateur**

```
UPDATE Utilisateur SET Metier = 'Chirurgien' WHERE Nom = 'Karim';
```

- **Delete (Supprimer)** :
  - Utilisée pour supprimer des enregistrements de la base de données.
  - Commande SQL : **DELETE**.

```
DELETE FROM Utilisateur WHERE Nom = 'Asma';
```

Utilisation avancée des requêtes SQL

- Calcul de la moyenne d'âge
  - `SELECT AVG(age) FROM Person GROUP BY Profession ;`
- Comptage des personnes par nom
  - `SELECT COUNT(*) FROM Person WHERE name = 'M';`
- Suppression de livres par titre
  - `DELETE FROM Book WHERE Title NOT LIKE '%DB%';`
- Sélection et tri de produits par prix
  - `SELECT * FROM Product WHERE Price > 5 ORDER BY Price ASC;`

Gestion de base de données avec SQLiteDatabase

- **La classe SQLiteDatabase** est la classe principale en Android pour **créer**, **supprimer**, **exécuter** des requêtes SQL et **effectuer** d'autres tâches de gestion de base de données.
  - **Méthodes clés :** permettent d'exécuter une requête.
    - **void execSQL(String sql)**
    - **Cursor rawQuery(String sql, String[] selectionArgs)**
- **La classe Cursor**
  - **Permet de parcourir le résultat d'une requête SELECT.**

Méthodes de la classe SQLiteDatabase

- **La méthode execSQL**
  - Pour exécuter une requête SQL pour laquelle on ne souhaite pas de réponse: telle que **CREATE**, **DELETE**, **INSERT**, **UPDATE**.
  - **Void execSQL (String sql) :** exécute une requête SQL sans paramètres supplémentaires.
    - **Exemple: bdd.execSQL("DROP TABLE Person");**
  - **Void execSQL (String sql, String[] args) :** exécute une requête SQL avec des arguments qui remplacent les placeholders.
    - **Exemple: bdd.execSQL ("DELETE FROM Person WHERE FirstName=? AND SecondName=?", new String[] { Ali, Ilyes });**

Méthodes de la classe SQLiteDatabase

- La méthode *rawQuery*
  - ***rawQuery*** est utilisée pour exécuter des requêtes SQL de type **SELECT** qui retournent un ensemble de résultats.
  - ***rawQuery*** retourne un objet de type **Cursor** qui permet de parcourir les **n-uplets** un à un :
  - **Exemple :**
    - `Cursor cursor = db.rawQuery("SELECT * FROM Person WHERE FirstName = ?", new String[]{"John"});`

La classe Cursor

- Un **Cursor** est une interface qui fournit un accès en lecture aux résultats retournés par ***rawQuery***.

Fonctions Principales :

- ***moveToFirst()***, ***moveToNext()*** : se déplacer à travers les lignes du résultat.
- ***getInt(i), getLong(i), getFloat(i), getString(i)*** : extraire les données de la colonne actuelle.
- **getColumnCount()** : retourne le nombre total de colonnes de la table.
- **getCount() :** retourne le nombre total de lignes de la table.
- **getPosition() :** retourne la position actuelle du curseur dans la table
- **isAfterLast() :** retourne vrai si le parcours est fini.
- ***close()*** : Fermer le cursor pour libérer les ressources.

La classe Cursor

- **Exemple de navigation**

```
if (cursor.moveToFirst()) {
    do {
        String name = cursor.getString(cursor.getColumnIndex("FirstName"));
        // Utiliser la variable 'name'
    } while (cursor.moveToNext());
}
cursor.close();
```

La Classe SQLiteOpenHelper

- **SQLiteOpenHelper** est une classe abstraite fournie par Android pour simplifier la gestion des bases de données.
- Gère la création de la base de données si elle n'existe pas.
- Gère la mise à jour de la base de données lors des changements de version.
- Sa syntaxe est donnée ci-dessous:

```
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(){
        super(context,DATABASE_NAME,null,1);
    }
    public void onCreate(SQLiteDatabase db) {}
    public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

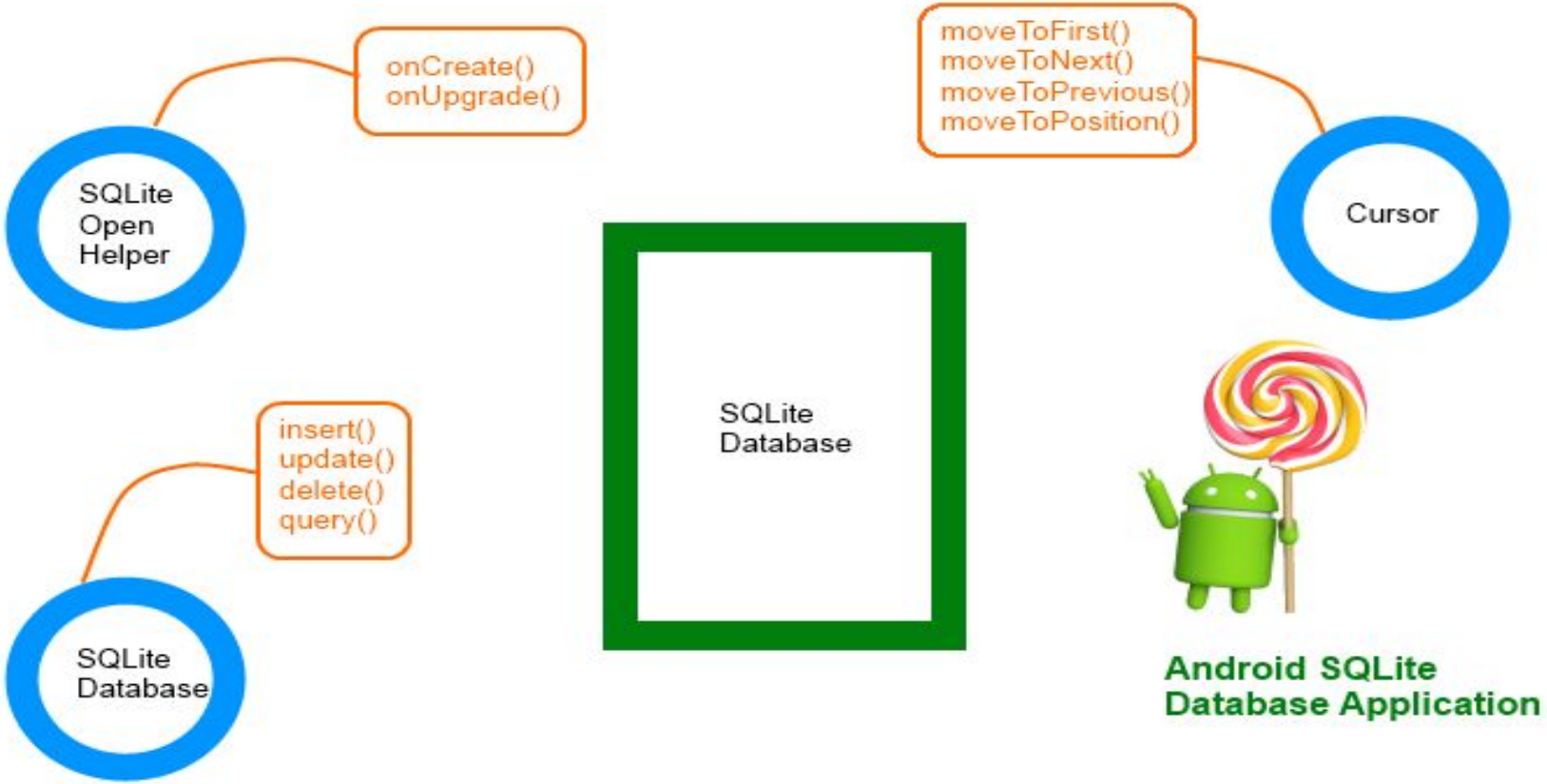
Les méthodes de la Classe SQLiteOpenHelper

- **La méthode onCreate()**
  - La méthode de callback **void onCreate (SQLiteDatabase db)** est automatiquement appelée au moment de la création de la base de données.
  - Le paramètre db représente la base.
  - Cette méthode est appelée quand la base de données n'existe pas encore. C'est dans cette méthode que vous devrez lancer les instructions pour créer les différentes tables et éventuellement les remplir avec des données initiales.

```
public void onCreate(SQLiteDatabase db) { db.execSQL("CREATE TABLE Contacts (ID INTEGER PRIMARY KEY, Name TEXT, Phone TEXT)"); }
```

Les méthodes de la Classe SQLiteOpenHelper

- **La méthode onUpgrade()**
  - La méthode de mise à jour, qui est déclenchée à chaque fois que l'utilisateur met à jour son application.
  - **Void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**
  - Cette méthode est appelée quand la version de l'application est supérieure à celle de la base.
  - Son rôle est de mettre à jour les tables de la base de données.
  - **Paramètre :**
    - **db :** la base de données manipulée
    - **oldVersion :** la version précédente de la base
    - **newVersion :** la nouvelle version de la base


onCreate()
onUpgrade()
SQLite
Open
Helper
moveToFirst()
moveToNext()
moveToPrevious()
moveToPosition()
Cursor
SQLite
Database
insert()
update()
delete()
query()
SQLite
Database
Android SQLite
Database Application