# 03

# Base de données SQLite

# Base de données SQLite

- Les bases de données **SQLite** sont une option puissante pour stocker des données **structurées** dans un format **relationnel**.
- SQLite est une bibliothèque logicielle qui implémente un moteur de base de données SQL avec **zéro-configuration**, **léger** et **sans dépendances** externes.
- Les données sont stockées dans **des fichiers de base de données locaux** qui peuvent être créés, mis à jour et supprimés à partir de l'application.
- SQLite utilise **SQL** (Structured Query Language) pour interagir avec les données dans la base.

# Base de données SQLite

- Les **données** sont **stockées** dans des fichiers de base de données **locaux** qui peuvent être créés, mis à jour et supprimés à partir de l'application.
- Toutes les **bases de données** créées dans une application seront **accessibles** par nom à travers **seule l'application** qui en est à l'origine.
- Afin **d'exposer** les données d'une base de données particulière à d'autres applications, on utilise un fournisseur de contenu **ContentProvider**.

# Base de données SQLite: Etapes

1. **Création de la base de données:** Créer une classe héritant de **SQLiteOpenHelper**, puis implémenter les méthodes suivantes:
    1. Le **constructeur**.
    2. **onCreate**: contient les opérations réalisées à la création de la base de données, cad, création des tables.
    3. **onUpgrade**: opérations réalisées quand la base fait un upgrade.
2. **Insertion de données.**
3. **Lecture de données.**
4. **Mise à jour de données.**

01

# CRÉATION DE LA BASE DE DONNÉES

# Création de la base de données

- La base de données SQLite est stockée dans le répertoire de données de l'application.
- Chaque application a son propre répertoire de données dans le système de fichiers Android, accessible uniquement par cette application.
- Le chemin vers le répertoire de données de l'application est /data/data/<nom_de_package_de_l'application>/databases/. Les fichiers de base de données sont stockés dans ce répertoire.
- Lorsqu'une application est **désinstallée**, le **répertoire** de données de l'application est **supprimé**, y compris la base de données **SQLite** stockée dans ce répertoire.

# Création de la base de données

- La première étape pour utiliser SQLite dans une application Android est de créer une classe **DBHandler** pour gérer la base de données.

- Cette classe doit étendre la classe **SQLiteOpenHelper**, qui fournit des méthodes pour gérer la **création**, la **mise à jour** et **l'ouverture** de la base de données.

- Le fichier de bases de données est automatiquement créé en **MODE_PRIVATE**, d'où le fait que seule l'application l'ayant créé peut y accéder.

# Création de la base de données

- La première étape consiste à créer une classe qui étend **SQLiteOpenHelper**.

```
class dbHandler extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE = "ContactsDB.db";
    private static final String TABLE = "Contacts";

    public static final String COL_ID = "_id";
    public static final String COL_NOM = "nom";
    public static final String COL_NUMERO = "numero";
```

# Création de la base de données

- La première étape consiste à créer une classe qui étend **SQLiteOpenHelper**.

```
public dbHandler(Context context,  SQLiteDatabase.CursorFactory factory) {
    super(context, DATABASE, factory, VERSION);
}
```

- Les paramètres sont les suivants :
  - context : le contexte de l'application.
  - name : le nom de la base de données.
  - factory : un objet qui sera utilisé pour créer des curseurs pour la base de données.
  - version : la version de la base de données.

# Création de la base de données

- La première étape consiste à créer une classe qui étend **SQLiteOpenHelper**.

```
public dbHandler(Context context,  SQLiteDatabase.CursorFactory factory) {
    super(context, DATABASE, factory, VERSION);
}
```

- Le paramètre factory est généralement utilisé pour spécifier un objet de type CursorFactory. Cet objet est utilisé pour créer des curseurs qui seront utilisés pour parcourir les enregistrements de la base de données. Si cet objet n'est pas spécifié, le curseur par défaut sera utilisé.

# Création de la base de données

- Cette classe doit implémenter deux méthodes importantes : **onCreate** et **onUpgrade**.

```
@Override
public void onCreate(SQLiteDatabase db) {
    Log.i("dbHandler", "db created");

    String requete = "CREATE TABLE "+TABLE+
            " ("+COL_ID+" INTEGER PRIMARY KEY AUTOINCREMENT,"+
            COL_NOM+" TEXT,"+
            COL_NUMERO+" TEXT"+")";
    db.execSQL(requete);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.i("dbHandler", "db updated");

    db.execSQL("DROP TABLE IF EXISTS " + TABLE);
    onCreate(db);
}
```

# Création de la base de données

- Cette classe doit implémenter deux méthodes importantes : **onCreate** et **onUpgrade**.
- La méthode **onCreate** est appelée la première fois que la base de données est créée. Cette méthode est utilisée pour créer les tables et les schémas de la base de données.
- La méthode **onUpgrade** est appelée lorsqu'une mise à jour de la base de données est nécessaire. Cette méthode est utilisée pour effectuer des modifications sur les tables existantes ou pour créer de nouvelles tables.

02

# INSERTION DES DONNÉES

# Insertion de données

- Pour insérer des données dans une base de données SQLite dans une application Android, la méthode **insert()** de la classe SQLiteDatabase est utilisée.
- Voici les étapes pour insérer des données dans une table :
    1. Ouvrir la base de données en mode écriture.
    2. Créer un objet **ContentValues** qui contient les valeurs à insérer dans la table.
    3. Utiliser la méthode **insert()** de la classe **SQLiteDatabase** pour insérer les valeurs dans la table.
    4. Fermer la base de données.

# 1.Ouvrir la base de données en mode écriture

```
public void addContact(Contact contact){
    Log.i("dbHandler", "add contact");

    ContentValues contentValues = new ContentValues();
    contentValues.put(COL_NOM,contact.getNom());
    contentValues.put(COL_NUMERO, contact.getNumero());
    SQLiteDatabase db = this.getWritableDatabase();
    db.insert(TABLE, null, contentValues);
    db.close();
}
```

- **getWritableDatabase** retourne une instance de la base de données en en lecture /écriture.
- **getReadableDatabase** retourne une instance de la base de données en **lecture seule**.

# 2. Créer un objet ContentValues

```
public void addContact(Contact contact){
    Log.i("dbHandler", "add contact");

    ContentValues contentValues = new ContentValues();
    contentValues.put(COL_NOM,contact.getNom());
    contentValues.put(COL_NUMERO, contact.getNumero());
    SQLiteDatabase db = this.getWritableDatabase();
    db.insert(TABLE, null, contentValues);
    db.close();
}
```

- Pour spécifier les valeurs de la ligne à insérer, la méthode accepte un objet de type **ContentValues**.
- Cet objet stocke les valeurs de chaque colonne de la ligne à insérer sous la forme d'une collection d'associations entre le **nom de la colonne (clé)** et la **valeur**.

# 2. Créer un objet ContentValues

```
class Contact {
    private int _id;
    private String nom;
    private String numero;

    public Contact(String nom, String numero) {
        this.nom = nom;
        this.numero = numero;
    }

    public Contact(int _id,String nom, String numero) {
        this._id = _id;
        this.nom = nom;
        this.numero = numero;
    }

    public int get_id() {
        return _id;
    }
```

Dans l'exemple, un objet de la classe Contact est caractérisé par:

- **_id:** clé primaire de la table;
- **nom:** une chaine de caractères;
- **numero:** une chaine de caractères.

# 3. Utiliser la méthode insert()

```
public void addContact(Contact contact){
    Log.i("dbHandler", "add contact");

    ContentValues contentValues = new ContentValues();
    contentValues.put(COL_NOM,contact.getNom());
    contentValues.put(COL_NUMERO, contact.getNumero());
    SQLiteDatabase db = this.getWritableDatabase();
    db.insert(TABLE, null, contentValues);
    db.close();
}
```

- La méthode insert() est appelée avec le **nom de la table**, un paramètre **nul** et **l'objet ContentValues** contenant les valeurs à insérer.
- Cette méthode renvoie l'ID de la nouvelle ligne insérée.

# La méthode insert()

- La signature de la méthode **insert()** est la suivante :

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

- Les paramètres sont les suivants :
  - **table** : le nom de la table.
  - **nullColumnHack** : le nom de la colonne qui peut contenir des valeurs nulles. Si on ne souhaite pas insérer de valeurs nulles, on passe null pour ce paramètre.
  - **values** : un objet ContentValues qui contient les valeurs à insérer dans la ligne. Les clés de l'objet ContentValues doivent correspondre aux noms des colonnes de la table, et les valeurs doivent être des objets du type approprié.

# La méthode insert()

- La signature de la méthode **insert()** est la suivante :

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

- La méthode **insert()** renvoie **l'ID** de la **nouvelle ligne** insérée, ou **-1** en cas **d'erreur**.

# 4. Fermer la base de données

```
public void addContact(Contact contact){
    Log.i("dbHandler", "add contact");

    ContentValues contentValues = new ContentValues();
    contentValues.put(COL_NOM,contact.getNom());
    contentValues.put(COL_NUMERO, contact.getNumero());
    SQLiteDatabase db = this.getWritableDatabase();
    db.insert(TABLE, null, contentValues);
    db.close();
}
```

- Il est important de fermer la base de données après avoir terminé les opérations de lecture/écriture pour **éviter les fuites de mémoire**.

03

# LECTURE DE DONNÉES

# Lecture de données

- Les étapes à suivre pour lire des données à partir d'une base de données SQLite dans une application Android sont:
    1. Ouvrir une base de données en mode lecture.
    2. Exécuter une requête de sélection.
    3. Récupérer les résultats de la requête.
    4. Fermer le curseur et la base de données.

# 1.Ouvrir une base de données en mode lecture

```
public ArrayList<Contact> getContacts(){
    Log.i("dbHandler","get contacts");

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE,null);

    ArrayList<Contact> contactList = new ArrayList<Contact>();
    while(cursor.moveToNext())
        contactList.add(new Contact(cursor.getInt(0),
                cursor.getString(1), cursor.getString(2)));

    cursor.close();
    db.close();
    return contactList;
}
```

# 1.Ouvrir la base de données en mode lecture

- **getWritableDatabase** retourne une instance de la base de données en en lecture /écriture.
- **getReadableDatabase** retourne une instance de la base de données en **lecture seule**.

# 2. Exécuter une requête de sélection

```
public ArrayList<Contact> getContacts(){
    Log.i("dbHandler","get contacts");

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE,null);

    ArrayList<Contact> contactList = new ArrayList<Contact>();
    while(cursor.moveToNext())
        contactList.add(new Contact(cursor.getInt(0),
                cursor.getString(1), cursor.getString(2)));

    cursor.close();
    db.close();
    return contactList;
}
```

# La méthode rawQuery

- La méthode rawQuery() de la classe SQLiteDatabase est utilisée pour exécuter une requête SQL arbitraire sur la base de données. Voici la syntaxe de la méthode rawQuery() :

```
Cursor rawQuery(String sql, String[] selectionArgs)
```

- **sql** : une chaîne de caractères contenant la requête SQL à exécuter.
- **selectionArgs** : un tableau de chaînes de caractères optionnel contenant les arguments de sélection de la requête. Si la requête ne contient pas de paramètres de sélection, ce paramètre peut être null.

# La méthode rawQuery

- La méthode **rawQuery**() de la classe **SQLiteDatabase** est utilisée pour exécuter une requête SQL arbitraire sur la base de données. Voici la syntaxe de la méthode rawQuery() :

```
Cursor rawQuery(String sql, String[] selectionArgs)
```

- La méthode **rawQuery**() retourne un objet **Cursor** qui peut être utilisé pour parcourir les résultats de la requête.
- Les méthodes **Cursor** les plus utilisées pour extraire les données du curseur sont **moveToFirst(), moveToNext(), getString(), getInt(),** etc..

# La méthode rawQuery: selectionArgs

```
Cursor cursor = db.rawQuery( sql: "SELECT * FROM " + TABLE +
        " WHERE " + COL_NOM + " =?1", new String[]{nom});
```

- la requête SQL sélectionne toutes les colonnes de la table spécifiée par la constante TABLE, mais seulement pour les lignes où la valeur de la colonne COL_NOM est égale à la valeur de la variable nom. La valeur de nom est passée à la requête SQL sous forme de paramètre à l'aide du tableau de chaînes de caractères {nom}.
- La requête SQL contient un paramètre marqué par ?1, qui est remplacé par la valeur de nom lors de l'exécution de la requête. Le caractère ? est un paramètre de liaison de SQLite qui permet **d'éviter les injections SQL et améliorer les performances de la requête**.

# 3. Récupérer les résultats de la requête

```
public ArrayList<Contact> getContacts(){
    Log.i("dbHandler","get contacts");

    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE,null);

    ArrayList<Contact> contactList = new ArrayList<Contact>();
    while(cursor.moveToNext())
        contactList.add(new Contact(cursor.getInt(0),
                cursor.getString(1), cursor.getString(2)));

    cursor.close();
    db.close();
    return contactList;
}
```

# La méthode moveToNext

- La méthode **moveToNext**() de la classe **Cursor** est utilisée pour déplacer le curseur vers la ligne suivante des résultats de la requête. Voici la syntaxe de la méthode moveToNext() :

```
boolean moveToNext()
```

- La méthode moveToNext() retourne **true** si le curseur a été déplacé avec succès vers la ligne suivante, ou **false s'il n'y a plus de lignes à parcourir**.

# Les méthodes usuelles de Cursor

Voici quelques-unes des méthodes les plus utilisées de la classe Cursor :

- **moveToFirst()** : déplace le curseur sur la première ligne des résultats.
- **moveToNext()** : déplace le curseur sur la ligne suivante des résultats.
- **moveToPrevious()** : déplace le curseur sur la ligne précédente des résultats.
- **moveToPosition(int position)** : déplace le curseur sur la ligne spécifiée par la position.
- **getColumnIndex(String columnName)** : retourne l'index de la colonne spécifiée par son nom.

# Les méthodes usuelles de Cursor

Voici quelques-unes des méthodes les plus utilisées de la classe Cursor :

- **getInt(int columnIndex)** : retourne la valeur de la colonne spécifiée par son index sous forme d'un entier.
- **getLong(int columnIndex)** : retourne la valeur de la colonne spécifiée par son index sous forme d'un long.
- **getString(int columnIndex)** : retourne la valeur de la colonne spécifiée par son index sous forme d'une chaîne de caractères.
- **getCount()** : retourne le nombre de lignes des résultats.
- **close()** : ferme le curseur et libère les ressources.

04

# MISE À JOUR DE DONNÉES

# Mise à jour des données

- La mise à jour de données dans une base de données SQLite sous Android se fait en utilisant des méthodes fournies par la classe **SQLiteDatabase**.
- Deux méthodes existent:
  - **update();**
  - **delete().**

# update()

- La syntaxe générale de la méthode update() est la suivante :

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

- **table** : le nom de la table à mettre à jour.
- **values** : un objet ContentValues contenant les nouvelles valeurs à mettre à jour. Les clés de cet objet doivent correspondre aux noms des colonnes de la table.
- **whereClause** : une chaîne de caractères qui spécifie les lignes à mettre à jour. Cette chaîne doit inclure un ou plusieurs marqueurs de position (?) pour les arguments fournis dans le paramètre whereArgs.

# update()

- La syntaxe générale de la méthode update() est la suivante :

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

- **whereArgs** : un tableau de chaînes de caractères qui contient les arguments à utiliser pour remplacer les marqueurs de position dans la clause **whereClause**.
- Il est important de noter que la clause **whereClause** est utilisée pour spécifier les lignes à mettre à jour. Si cette clause n'est pas spécifiée, i.e, **nul**, toutes les lignes de la table seront mises à jour.
- Il est donc important de s'assurer que la clause **whereClause** est correctement spécifiée pour éviter de mettre à jour les mauvaises lignes.

# update()

- La syntaxe générale de la méthode update() est la suivante :

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

- **La méthode renvoie un entier qui représente le nombre de lignes mises à jour dans la table.**

# update(): Exemple

```
public int updateContact(int _id,Contact contact){
    Log.i("dbHandler", "Update contact by id");

    ContentValues contentValues = new ContentValues();
    contentValues.put(COL_ID,contact.get_id());
    contentValues.put(COL_NOM, contact.getNom());
    contentValues.put(COL_NUMERO,contact.getNumero());

    SQLiteDatabase db = this.getWritableDatabase();
    int a=db.update(TABLE,contentValues,COL_ID+"=?",
            new String[]{String.valueOf(_id)});

    db.close();
    return a;
}
```

# delete()

- La syntaxe générale de la méthode delete() est la suivante :

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

- **table** : le nom de la table à mettre à jour.
- **whereClause** : une chaîne de caractères qui spécifie les lignes à supprimer. Cette chaîne doit inclure un ou plusieurs marqueurs de position (?) pour les arguments fournis dans le paramètre whereArgs.
- **whereArgs** : un tableau de chaînes de caractères qui contient les arguments à utiliser pour remplacer les marqueurs de position dans la clause **whereClause**.

# delete()

- La syntaxe générale de la méthode delete() est la suivante :

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

- **La méthode renvoie un entier qui représente le nombre de lignes supprimées dans la table.**

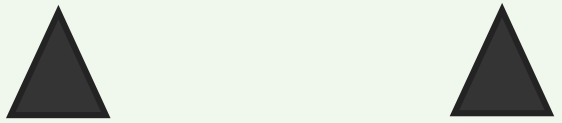
# delete(): Exemple

```
public Boolean deleteContact(String nom, String tel){
    Log.i("dbHandler", "delete contact by name/tel");

    SQLiteDatabase db = this.getWritableDatabase();
    int count = db.delete(TABLE,
            COL_NOM + "=?1 AND "+COL_NUMERO+"=?2",
            new String[]{nom,tel});

    db.close();
    return count>0;
}
```

**DELETE FROM Contacts WHERE nom=?1 AND numero=?2**

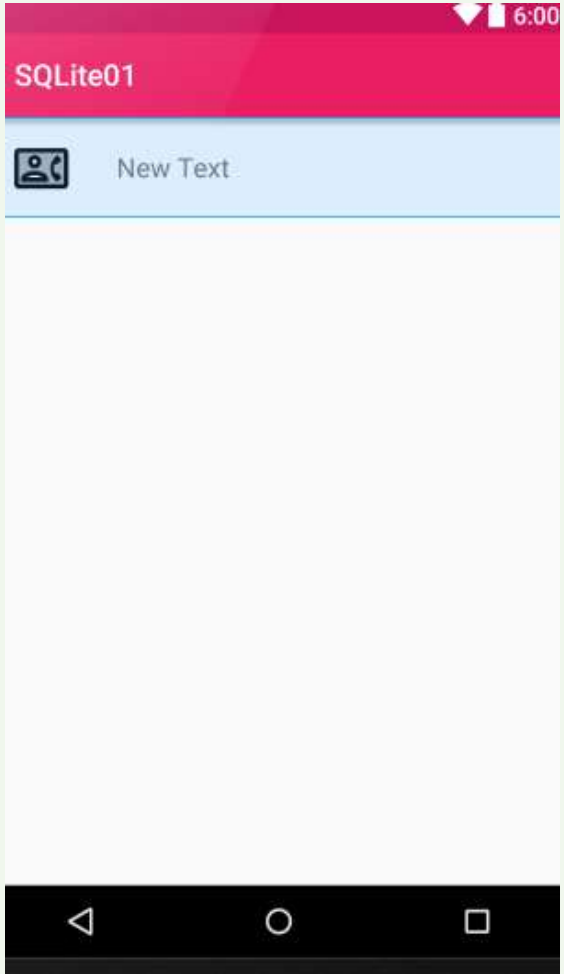


**nom** **tel**

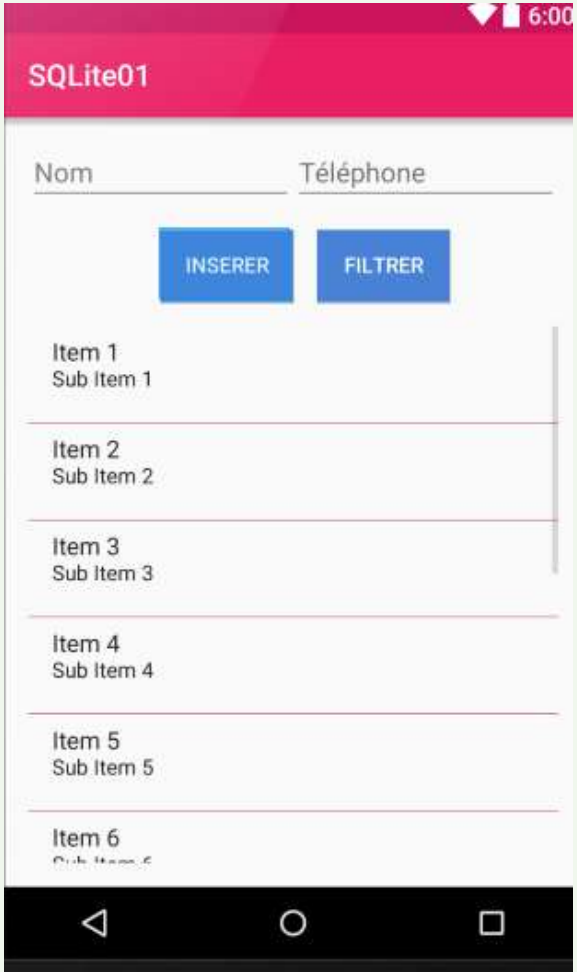
# UTILISATION DE LA CLASSE DBHANDLER

# MainActivity: Layout

list_example_entry.xml




activity_main.xml



# MainActivity: onCreate

```
public class MainActivity extends AppCompatActivity {
    private ArrayAdapter adapter;
    private ArrayList<Contact> contactList;
    private dbHandler db;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        db = new dbHandler(this, null);
        viewData();
    }

    public  void viewData(){
        contactList = db.getContacts();
        adapter = new ArrayAdapter<Contact>(this,
                R.layout.list_example_entry,
                R.id.nomText,
                contactList);
        ListView listView = (ListView)findViewById(R.id.listView);
        listView.setAdapter(adapter);
    }
```

# MainActivity: onClick (Bouton Insérer)

```
public void inserer(View view){
    String nom = ((EditText)findViewById(R.id.nomText))
            .getText().toString();

    String telephone = ((EditText)findViewById(R.id.telText))
            .getText().toString();

    Contact contact = new Contact(nom,telephone);

    db.addContact(contact);
    viewData();
    adapter.notifyDataSetChanged();
}
```

# MainActivity: onCreate (onItemLongClick-delete contact)

```
ListView listView = (ListView)findViewById(R.id.listView);
//Supprimer contacte sélectionné
listView.setOnItemLongClickListener(
        new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent,
                                   View view, int position, long id) {

        db.deleteContact(contactList.get(position).get_id());
        viewData();
        Toast.makeText(MainActivity.this, "Contacte supprimé",
                Toast.LENGTH_SHORT).show();
        return true;
    }
});
```

# Tester L'application

**LA CLASSE CONTACT DOIT IMPLÉMENTER LA MÉTHODE TOSTRING**