ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

Module :

Développement mobile native :

Chapitre 4:
Persistance des Données

Pr. Zakia EL UAHHABI

Persistance des données

Persistance?

- Garder une trace des interactions avec l’utilisateur et conserver des données d’une session à une autre…
- Toute application doit pouvoir charger et enregistrer des données.
- Android propose plusieurs options pour enregistrer les données persistantes d'une application. La solution que vous choisissez dépend de vos besoins spécifiques..

Persistance des données

- Les solutions pour stocker les données dans une application Android sont:
  - ☞ Préférences partagées (Shared Preferences)
  - ☞Stockage dans des fichiers
  - ☞Bases de données SQLite
  - ☞Données distantes

Plan du cours

1. C’est quoi SQLite?
2. SQLite sous Android Studio
   - Création de la base de données
   - SQLiteDatabase
   - Gestion de la table
   - Adaptateur pour les curseurs

C’est quoi SQLite?

- SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions.
- Il utilise très peu de mémoire, ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution.
- SQLite ne nécessite pas de serveur pour fonctionner, ce qui n’est pas le cas de MySQL par exemple.

C’est quoi SQLite?

- SQLite prend en charge les types de données:
  - ☞ TEXT : la donnée est une chaine de caractères (similaire à String en Java);
  - ☞ INTEGER : un nombre entier (similaire à long en Java);
  - ☞ REAL : similaire à double en Java;
  - ☞ NULL: indique l'absence d'information ou une valeur non définie;
  - ☞ NUMERIC: booléen, date et datetime;
  - ☞ BLOB: la donnée est enregistrée comme elle a été donnée. par exemple si vous voulez mettre une image dans votre base de données

Remarque: SQLite ne vérifie pas si les types des données insérées dans les colonnes correspondent au type défini.

SQLite sous Android Studio

- Android offre aussi la possibilité d’utiliser toutes les propriétés d’une base de données relationnelle via la librairie SQLite. Dans ce cas, il stocke la base de données localement à l’application.

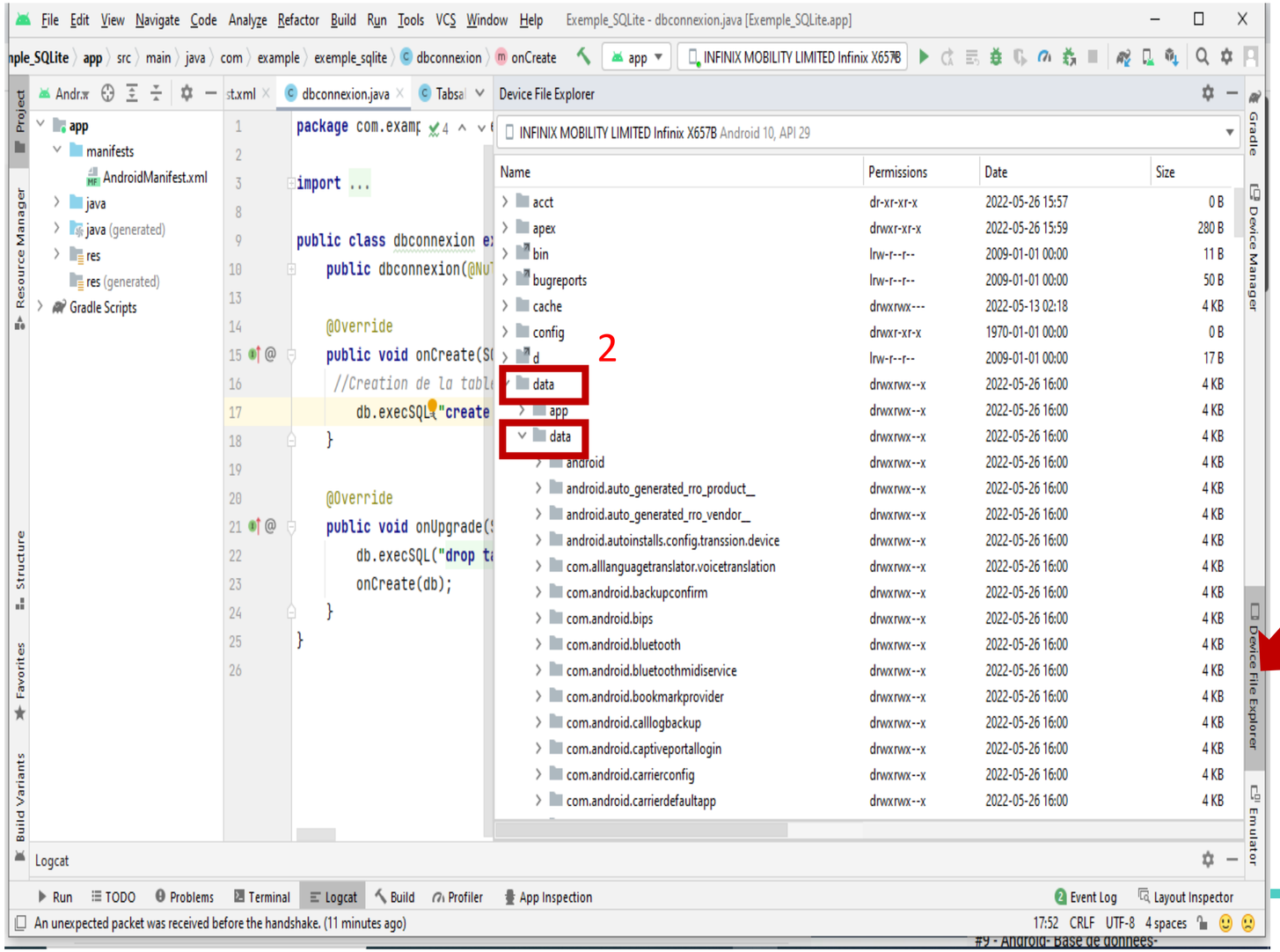
- SQLite ne nécessite pas de serveur pour fonctionner, ce qui n’est pas le cas de MySQL .

- Si la base de données a pour but de centraliser une grande masse de données et de les fournir à un grand nombre de clients, il est préférable d'utiliser des SGBD basés sur le paradigme client-serveur.

SQLite sous Android Studio

- Chaque application peut avoir donc ses propres bases. Ces bases sont stockées dans le répertoire « **databases** » associé à l'application (/data/data/APP_NAME/databases/nom_base). Nous pouvons les stocker aussi sur une unité externe (sdcard).

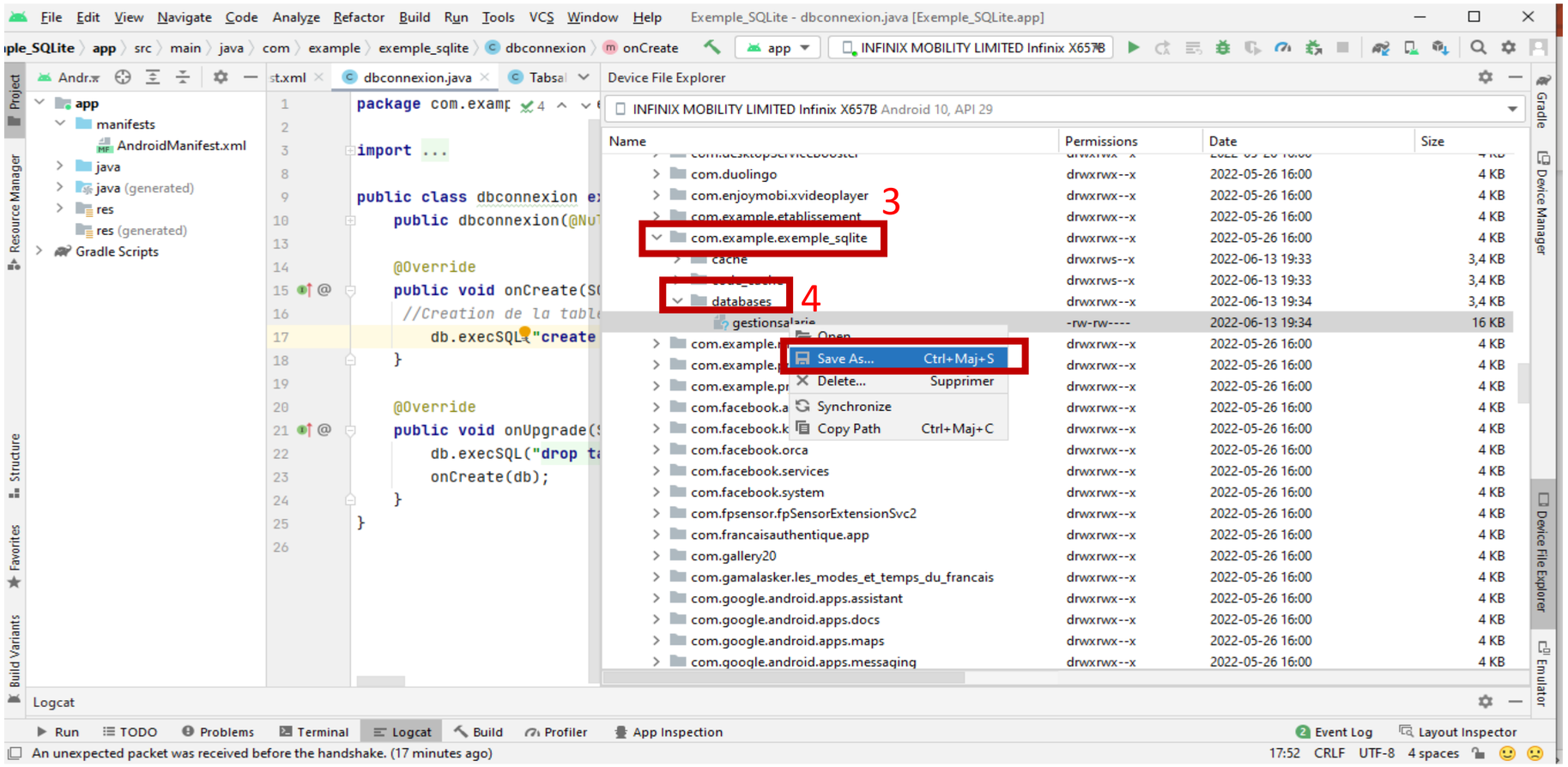
- Chaque base créée, elle le sera en mode « MODE_PRIVATE ». Aucune autre application ne peut y accéder que l'application qui l'a créée.



SQLite sous Android Studio

- Pour visualiser une base de données crée avec une application sous Android Studio, il faut suivre les étapes suivantes:

Etape1: Enregistrer la base de données sur votre ordinateur.

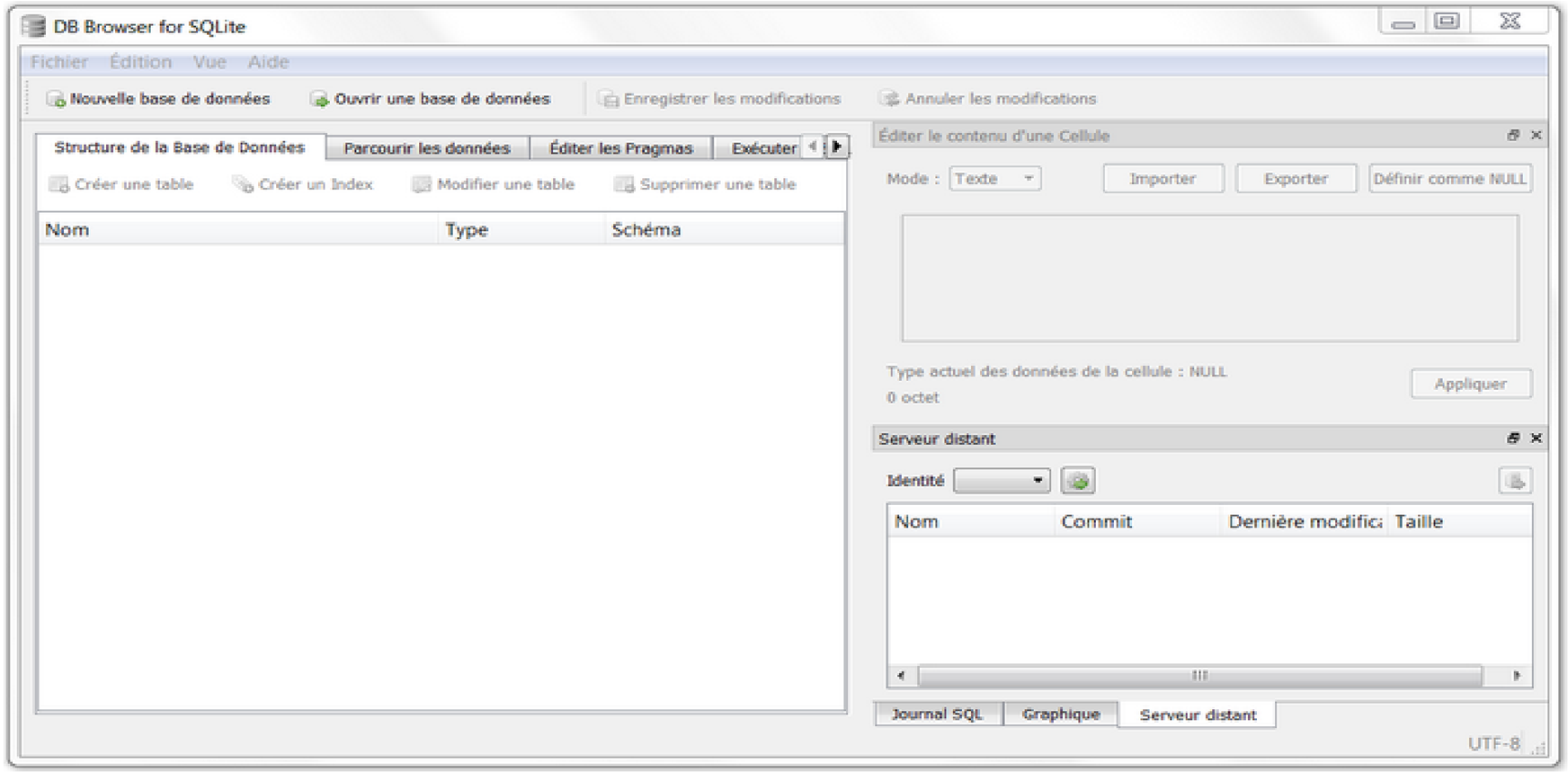


SQLite sous Android Studio

- **Etape 2:**Pour visualiser une base de données SQLite, il est possible d'utiliser l'un de ces deux logiciels:
  - ☞ https://sqlitebrowser.org
  - ☞ https://sqlitestudio.pl/index.rvt

SQLite sous Android Studio

- Exemple SQLite Browser



SQLite sous Android Studio

Création de la base de données

- Pour créer une base de donnée SQLite dans une application Android, on crée une classe qui hérite de SQLiteOpenHelper.

```
public class MyClass extends SQLiteOpenHelper { …}
```

SQLite sous Android Studio

Création de la base de données

- La classe fille qui hérite SQLiteOpenHelper nécessite de :

1.Créer un constructeur:

Dans le constructeur de votre sous-classe, vous appelez la méthode super() de SQLiteOpenHelper, en précisant le contexte de l'application, le nom de la base de données, le factory et sa version actuelle.

```
public MyClass(Context context) {
super(context, "commments.db", null, 1);
}
```

Contexte de l'application

Version: est le numéro de version de la BD (commençant à 1)

Nom de la BD

le factory: est utilisé pour créer des Cursor. En général=null

SQLite sous Android Studio

Création de la base de données

- La classe fille qui hérite SQLiteOpenHelper nécessite de :

2. Redéfinir la méthode onCreate() :

onCreate(): est appelée pour créer la base si elle n'existe pas, et créer des tables et leurs contenus.

```
public void onCreate(SQLiteDatabase database) {
database.execSQL(Table_CREATE);
}
```

La variable « TABLE_CREATE » va contenir la requête « **SQL** » qui permet de créer la table.

Création de la base de données

- La classe fille qui hérite SQLiteOpenHelper nécessite de :

2. Redéfinir la méthode onUpgrade() :

onUpgrade(): est appelée pour mettre à jour la version de votre base. Elle est déclenchée à chaque fois que l'utilisateur met à jour son application.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
db.execSQL("DROP TABLE IF EXISTS personnes");
onCreate(db);
}
```

Remarque: Il est préférable de créer une classe par table. Cette classe va définir les méthodes « onCreate » et « onUpgrade ».

SQLite sous Android Studio

SQLiteDatabase

- **SQLiteDatabase**: Classe permettant de manipuler une base de données SQLite sous Android. Elle dispose d'un ensemble de méthodes pour ouvrir, effectuer des requêtes, mettre à jour et fermer la base de données:

  ☞ Plus précisément, SQLiteDatabase fournit les méthodes pour les requêtes CRUD: insert(), read(), update() et delete().

  ☞ Elle fournit également la méthode execSQL(), qui permet d'exécuter une instruction SQL directement.

SQLite sous Android Studio

Gestion de la table

- Après la création d'une classe SQLiteOpenHelper qui permet de créer la BD (tables...) et la mise à jour, on peut créer une deuxième classe qui sera une classe de **gestion d'une table (insert, delete, update, select...).**
- Pour pouvoir accéder à la table créée, on doit utiliser un objet de la première classe dans cette classe de gestion:

Exemple:

```
//Constructeur de deuxième classe
public class2( Context c){
   MyClass c; //  MyClass est la première classe créée qui hérite de la classe SQLiteOpenHelper
   c=new MyClass(context);}
```

- La classe SQLiteOpenHelper fournit les méthodes getReadableDatabase() et getWriteableDatabase() pour accéder à un objet SQLiteDatabase en lecture ou en écriture.

SQLite sous Android Studio

Gestion de la table

☞ **Méthode 1: utilisation des requêtes SQL**

On utilise la méthode execSQL(Requête SQL)

☞ **Méthode 2: Utiliser les méthodes génériques de la classe SQLiteDatabase**

On utilise les méthodes insert(), update(), delete() qui admettent un paramètre de type **ContentValues**.

ContentValues: une classe qui décrit sous forme de clé / valeur la donnée à insérer à la table.

SQLite sous Android Studio

Gestion de la table (**Insertion**)

Pour ajouter une entrée dans la table, on utilise l'une des méthodes suivantes:

☞ **Méthode 1: utilisation de la requêtes SQL « insert into »**

INSERT INTO nom_de_la_table(colonne1,colonne2,...) VALUES (valeur1,valeur2,...)

Exemple

```
SQLiteDatabase db=context.getWritableDatabase();
 db.execSQL('insert_SQL_Query');
```

SQLite sous Android Studio

Gestion de la table (**Insertion**)

Pour ajouter une entrée dans la table, on utilise l'une des méthodes suivantes:

☞ **Méthode 2: Utiliser la méthode insert(...)**

Exemple:

```
//context c'est l'instance de la première classe
SQLiteDatabse db=context.getWritableDatabase();
ContentValues values=new ContentValues();
values.put("champ1","val1");
values.put("champ2","val2");
values.put("champ3","val3");
values.put(...);
db.insert("nomTable",null,values);
```

SQLite sous Android Studio

Gestion de la table (**Suppression**)

☞ **Méthode 1: utilisation de la requêtes SQL «DELETE »**

```
DELETE FROM nomTable WHERE Condition
```

Exemple

```
SQLiteDatabase db=context.getWritableDatabase();

db.execSQL("delete from Personne where _id='5'");
```

Remarque: Les tables de base de données doivent utiliser l'identifiant _id comme clé primaire de la table. Plusieurs fonctions Android s'appuient sur cette norme.

SQLite sous Android Studio

Gestion de la table (**Suppression**)

☞ **Méthode 2: Utiliser la méthode delete(...)**

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

Exemple:

```
SQLiteDatabse db=context.getWritableDatabase();
String[] s=new String[] {"cond"};
db.delete("nomTable","id=?",s);
```

SQLite sous Android Studio

Gestion de la table (**Mise à jour**)

☞ **Méthode 1: utilisation de la requêtes SQL «UPDATE »**

UPDATE table SET nomColonne1='nouvelle valeur' WHERE condition

Exemple

```
SQLiteDatabase db=context.getWritableDatabase();//on interroge la BD
db.execSQL("update Personne set adresse='"+adresse+"'where
id='+String.valueOf(id)));
```

SQLite sous Android Studio

Gestion de la table (**Mise à jour**)

☞ **Méthode 2: Utiliser la méthode update(...)**

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

Exemple:

```
SQLiteDatabse db=context.getWritableDatabase();
ContentValues contentValue=new ContentValues();
contentValue.put("Nom",newName);//stocker la valeur dans contentValue
String[] Args={oldName};// C est une tableau qui va contenir les arguments (les valeurs qu'on veut les modifier)
db.update("Personne", contentValue, "Nom=?", Args);
```

SQLite sous Android Studio

Gestion de la table (**Selection**)

☞ Méthode 1:

```
Cursor curseur=db.query(boolean distinct, String table, String[] columns,
String selection, String[] selectionArgs, String groupBy, String
having, String orderBy, String limit);
```

- ☞ String table: Nom de la table
- ☞ String[] columns: Les champs (colonnes) à récupérer
- ☞ String selection: pour le where (null si cette clause n'est pas utilisée)
- ☞ String[] selectionArgs : les arguments du where
- ☞ String groupBy: Group by qui permet de grouper les résultats.
- ☞ String having: Having est utile pour filtrer parmi les groupes
- ☞ String orderBy: Order by permet de trier les résultats. Mettre ASC pour trier dans l'ordre croissant et DESC pour l'ordre décroissant.
- ☞ String limit: limit pour fixer un nombre maximal de résultats voulus.

SQLite sous Android Studio

Gestion de la table (**Selection**)

Cursor curseur=db.query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit);

Exemple:

```
SQLiteDatabase  db=this.getReadableDatabase();//on se connecte à la BD en mode écriture
String table="Salarie";
String[] columns={"matricule","nom"};
String selection="nom=?";
String[] selectionArgs ={"wardi"};
String groupBy=null;
String having = null;
String orderBy="matricule DESC";
String limit="5";

Cursor cursor=db.query(table,columns,selection,selectionArgs,groupBy,having ,orderBy,limit )
```

SQLite sous Android Studio

Gestion de la table (**Selection**)

☞ Méthode 2:

La méthode Cursor rawQuery(String **sql**, String[] **selectionArgs**) permet de lancer des simples requêtes SELECT comme:

SELECT * FROM **TABLE** WHERE **condition**

-**sql**: une requête SELECT;
-**selectionArgs**: le tableau qui fixe les valeurs des paramètres (note ?) dans la clause WHERE.

Les méthodes query*() et rawQuery() retournent un objet Cursor. Cet objet représente le résultat de la requête exécutée et pointe généralement sur un élément de la liste du résultat.

SQLite sous Android Studio

Gestion de la table (**Selection**)

- Cursor (Méthodes de navigation)

Les curseurs représente un ensemble de "lignes" contenant le résultat de la requête SELECT. En effet, la classe Cursor offre les méthodes nécessaires pour parcourir le résultat d'une requête, et le procéder ligner par ligne.

- curseur.moveToFirst(): positionner le curseur sur le premier enregistrement.
- curseur.moveToLast():positionner le curseur sur le dernier enregistrement.
- curseur.moveToNext():positionner le curseur sur l'enregistrement suivant.
- curseur.moveToPrevious():positionner le curseur sur l'enregistrement précèdent.
- curseur.moveToPosition(int): Déplacer le curseur à la position indiquée.

SQLite sous Android Studio

Gestion de la table (**Selection**)

- Cursor (Méthodes d'information)

- curseur.getCount(): retourne le nombre d'enregistrements qui sont renvoyées par la requête.
- curseur.getColumnName(String nom):récupérer l'index (la position) d'un champ suivant son nom.
- curseur.getColumnCount(): Retourne le nombre de colonnes renvoyées par la requête.
- curseur.getString (index du champ), getInt, getDouble...pour récupérer la valeur d'un champ (suivant son index ou sa position dans l'enregistrement).
- int getPosition(): Récupérer la position actuelle.
- boolean isAfterLast():permet de déterminer si on a atteint la fin de la liste du résultat.
- getColumnIndex() : retourne l'index correspondant au nom de la colonne ou retourne -1 si le nom de la colonne n'existe pas

SQLite sous Android Studio

Gestion de la table (**Selection**)

Exemple:

```
ArrayList ls=new ArrayList();
SQLiteDatabase db=this.getReadableDatabse();
Cursor rq=db.rawQuery("Select * FROM salarie", null);
rq.moveToFirst();// mettre le curseur à la première ligne de la table
While(rq.isAfterLast()==false){//vérifier avec la méthode isAfterLast() si on arrive à la fin
// utiliser la méthode add() pour ajouter les données à la liste
ls.add(rq.getString(rq.getColumnIndex("matricule"))+ " "+rq.getSring(rq.getColumnIndex("nom")));
rq.moveToNext(); // on positionne le curseur sur l'enregistrement suivant avec la méthode
moveToNext.
}
return ls;
```

SQLite sous Android Studio

Adaptateur pour les curseurs

- Comme n'importe quel adaptateur, un CursorAdapter fera la transition entre des données et un **AdapterView** (ex.ListView). Il adapte le contenu qui provient d'une base de données.

- Cependant, comme on trouve rarement une seule information dans un curseur, on préférera utiliser un SimpleCursorAdapter.

SQLite sous Android Studio

Adaptateur pour les curseurs

- SimpleCursorAdapter est utiliséquand les données sont issues d'un **Cursor**. Cette interface fournit un accès aléatoire en lecture/écriture au résultat fourni par une requête sur une base de données.
- Pour construire ce type d'adaptateur, on utilisera le constructeur suivant :

SimpleCursorAdapter (Context context, int layout, Cursor c, String[] from, int[] to)

avec:

-**layout**: est l'identifiant de la mise en page des vues dans l'AdapterView.
-**c** :est le curseur.
-**from** :indique une liste de noms des colonnes afin de lier les données au layout.
-**to**: contient les TextView qui afficheront les colonnes contenues dans from.

Démonstration: SQLite sous Android

SQLite Room

SQLite Room

Mapping Object Relationnel (ORM)

- Le Mapping Object Relationnel (ORM) est une technique de programmation qui permet de mettre à disposition des classes objet permettant de manipuler les bases de données relationnelles.
- Le développeur ne manipule que des objets, la partie persistance est gérée par l'ORM.
- Android nous fournit son ORM appelé Room

Exemple:

Utilisateur	**Objet**
ID	
Nom	
prenom	

SGBD

ID	Nm	prenom

SQLite Room

- SQLITE Room est une couche d'abstraction au-dessus d’une base SQLite, permettant ainsi de créer très facilement une base de données, ses tables et de mettre à jour ses données.

- Il est vivement recommandé d'utiliser Room au lieu d'utiliser directement les API SQLite

SQLite Room

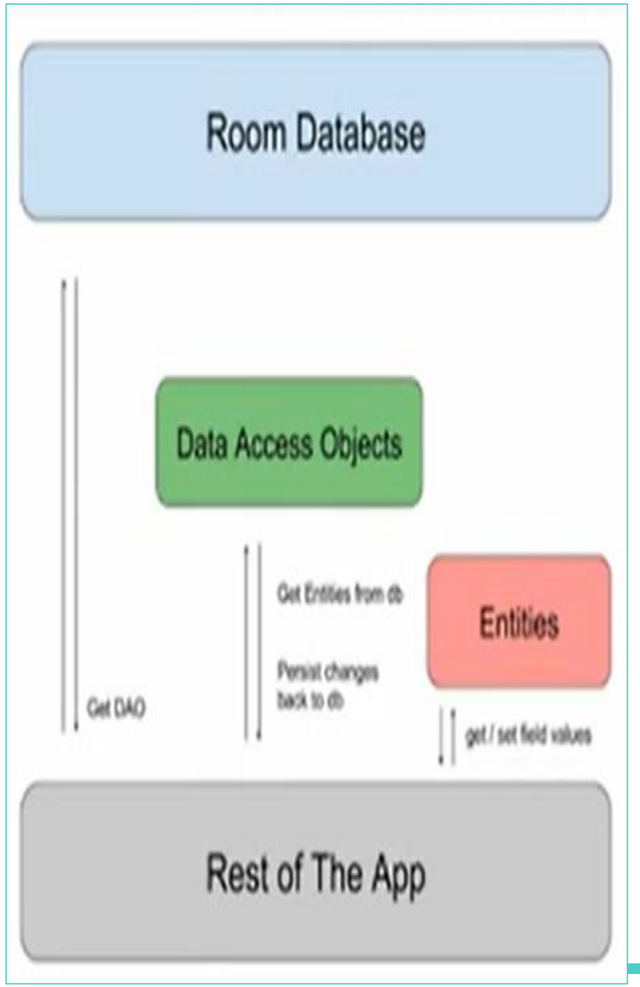
- Parmi des avantages de SQLITE Room, on a:
  - ☞ Vérification des requêtes SQL à la compilation
  - ☞ Utilisation des annotations pratiques qui permet de minimiser le code standard répétitif.
  - ☞ Pas besoin d’écrire de SQL ;
  - ☞ ….

Il est vivement recommandé d'utiliser Room au lieu d'utiliser directement les API SQLite

SQLite Room

Architecture SQLITE Room

- L'architecture de Room est articulée autour de trois composants:
  - Entités de données qui représentent les tables de la base de données de votre application.
  - Objets d'accès aux données (DAO) qui fournissent des méthodes que votre application peut utiliser pour récupérer, mettre à jour, insérer et supprimer des données dans la base de données.
  - La classe de base de données qui contient la base de données et fournit à votre application les instances des DAO associées à cette base de données.



SQLite Room

Implémentation de SQLITE Room

- Il faut ajouter les librairies de composants Room requises dans le fichier **build.gradle** (Module app):

1. Ouvrez le fichier Gradle build.gradle (Module: app) au niveau du module.
2. Dans le bloc dependencies, ajoutez les dépendances de la bibliothèque Room indiquées dans le code suivant:

```
dependencies{
 //ROOM
   implementation "androidx.room:room-runtime:x.x.x“        // x.x.x : version
   annotationProcessor "androidx.room:room-compiler:x.x.x"
   //VIEW MODEL & LIVE DATA
   implementation 'androidx.lifecycle:lifecycle-extensions:x.x.x’}
```

SQLite Room

1.Création de l’entité

- Exemple: Entité « utilisateur »

Chaque instance de la classe utilisateur représente une ligne de la table utilisateur dans la base de données de l'application.

```
@Entity
 public class utilisateur {
@PrimaryKey
    private Int  code;
@ColumnInfo(name="nom_user")
    private String nom;
@ColumnInfo(name="prennom_user")
    private String prenom;
```

SQLite Room

2. Objet d’accès aux données (DAO):

- Exemple: Entité « utilisateur »

Chaque **UserDAO** fournit les méthodes utilisées par le reste de l’application pour interagir avec les données de la table utilisateur.

```
@Dao
 public interface UserDAO {
@Query("SELECT * FROM utilisateur")
List<utilisateur> afficher();
@insert
Void insertAll(utilisateur);
@delete
void delete(utilisateur user);
```

SQLite Room

3. Base de données:

- Exemple: Entité « utilisateur »

Définir une classe **UserDB** utilisée pour la configuration de la base de données. Elle représente le point d'accès principal de l'application aux données persistantes.

La classe de base de données doit :

- ☞ être annotée avec une annotation @Database.
- ☞ être une classe abstraite qui hérite de RoomDatabase.
- ☞ définir une méthode abstraite qui n'a aucun argument et renvoie une instance de la classe DAO.

SQLite Room

2. Base de données:

- Exemple: Entité « utilisateur »

```
@Database (entities={utilisateur.class},version=1)
Public abstract class UserDB extends RoomDatabase {
Public abstract UserDAO userDAO();    //instance de la classe DAO
}
```

SQLite Room

2. Base de données:

- Exemple: Entité « utilisateur »

Pour créer une instance de la basse de données, on utilise le code suivante:

```
UserDB db=Room.databaseBuilder(context.getApplicationContext(), UserDB.class, nom_BD)
        .allowMainThreadQueries()
        .build();
```

Pour interagir avec la base de données, on peut utiliser les méthodes de l'instance DAO:

```
UserDAO userDAO =db. userDAO();
List<utilisateur> utilisateur= userDAO.afficher()
```

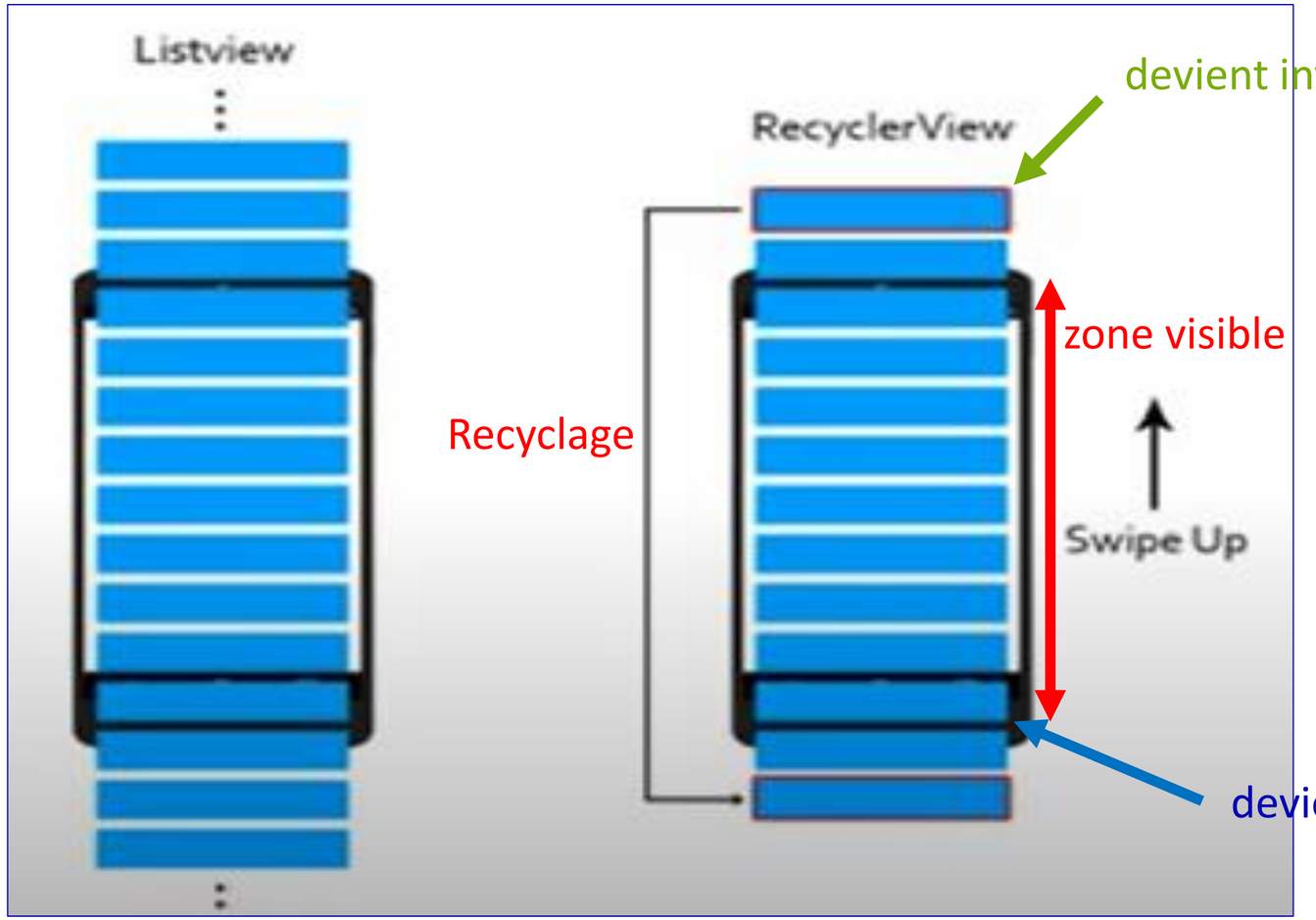
Recycler View

- Le Recycler View est le successeur de listView et GridView. C’est une vue qui intègre un défilement automatique et qui veille à économiser la mémoire pour l’affichage. L'objectif du RecyclerView est de pouvoir gérer l’affichage d’une large base de données de manière plus performante possible.

- Le nom « RecyclerView » vient de l’astuce : les vues qui deviennent invisibles à cause du défilement vertical sont recyclées et renvoyées de l’autre côté mais en changeant seulement le contenu à afficher.

Recycler View

Recycler View vs List View:

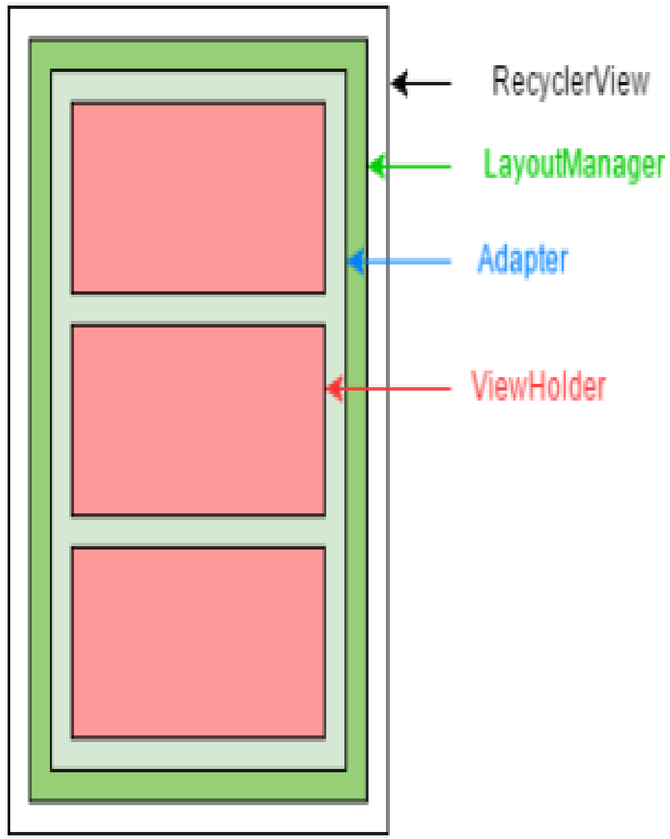


Recycler View

Les composants d'un Recycler View

Pour qu'un Recycler View fonctionne correctement, il nécessite l'utilisation des composants suivants:

1. LayoutManager: le gestionnaire de disposition permet de positionner correctement l'ensemble des données de la liste (LineaarLayoutManager avec une orientation verticale ou horizontale, GridLayoutManager….)
2. RecyclerView.Adapter: permet de faire le lien entre la source de données et les vues affichées dans le RecyclerView. Il va gérer un ensemble de **ViewHolder**.



Recycler View

Les composants d'un Recycler View

Pour créer un adaptateur, on doit:

1.créer une classe adaptateur qui dérive de la classe RecyclerView.Adapter.

2.créer une classe de ViewHolder à l'intérieur de la classe adaptateur. Elle contient les informations d'affichage du View Layout qui représente la mise en page commune entre les éléments de la liste.

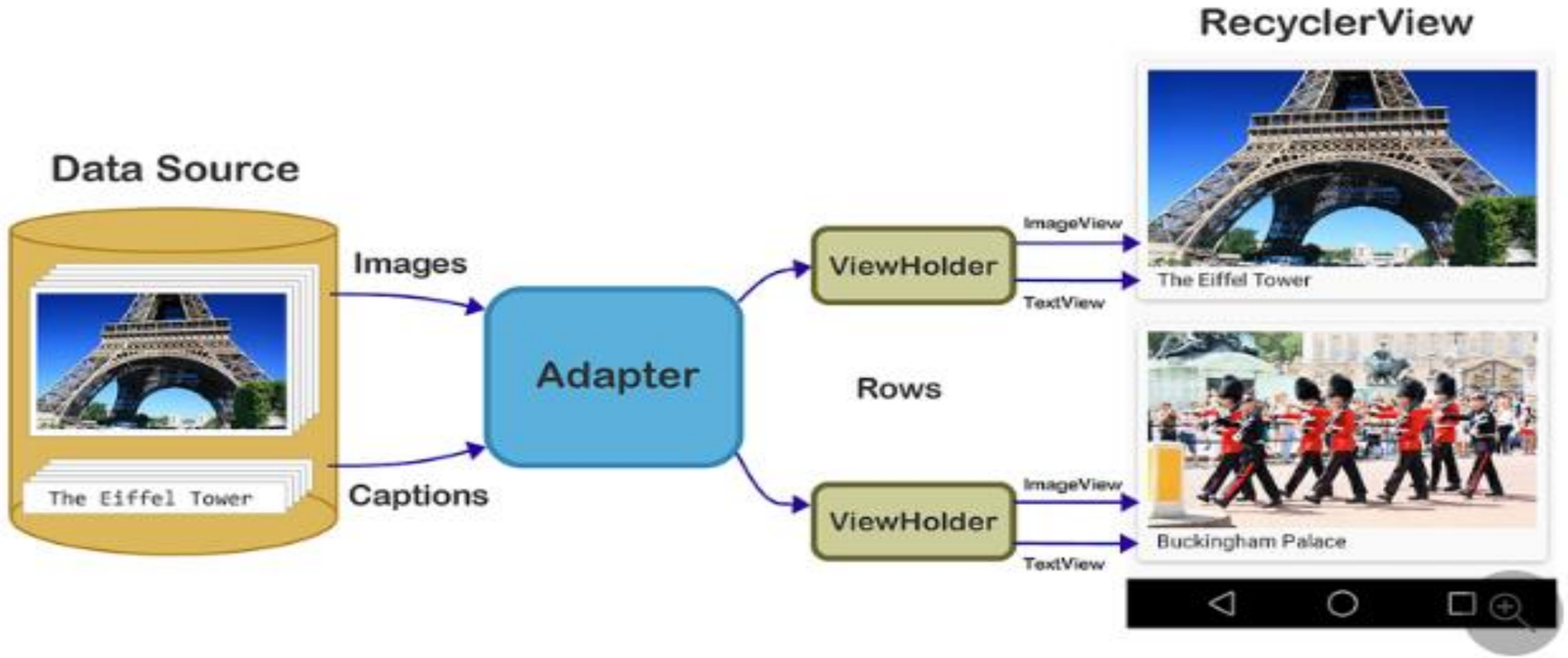
Exemple:

```
public class MainAdapter extends RecyclerView.Adapter<MainAdapter.ViewHolder>
{
        ... constructeur ...
        ... surcharge des méthodes nécessaires...
        public class ViewHolder extends RecyclerView.ViewHolder {
        //cette classe permet de faire une correspondance entre java et les composants graphiques
            TextView textview;
            ImageView btEdit, btDel;
            public ViewHolder(@NonNull View itemView) {
                super(itemView);
                textview=itemView.findViewById((R.id.textView3));
                btEdit=itemView.findViewById((R.id.btEdit));
            }}
```

Recycler View

Les composants d’un Recycler View

Exemple:



Recycler View

Les composants d'un Recycler View

Pour qu'un adaptateur puisse gérer l'affichage des éléments de la liste. Il doit implémenter les méthodes suivantes:

☞ onCreateViewHolder() : permet de créer un ViewHolder. Elle est appelée au début de l'affichage de la liste, pour initialiser ce qu'on voit à l'écran.

Exemple:

```
public MainAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.item_modele,parent,false);
    return new ViewHolder(view); //return le cadre de vue
}
```

inflate = transformer un fichier XML en vues Java.

Recycler View

Les composants d'un Recycler View

Pour qu'un adaptateur puisse gérer l'affichage des éléments de la liste. Il doit implémenter les méthodes suivantes:

- ☞ onBindViewHolder() : permet de lier les éléments de la ViewHolder, à la position spécifiée dans la vue, à leurs valeurs dans la base de données.
- ☞ getItemCount : retourne le nombre d'éléments dans la source de données.
- ☞ …

la méthode onCreateViewHolder() ne sera appelée qu'à l'initialisation, là où la méthode onBindViewHolder sera appelée à chaque fois qu'un nouvel élément doit être affiché.

Recycler View :Récapitulation

Pour utiliser un RecyclerView , on doit suivre les étapes suivants :

- ☞ Définir le RecyclerView dans l'activité
- ☞ Créer le layout utilisé pour afficher un élément de la liste
- ☞ Créer l'Adaptateur qui va gérer les ViewHolder.
- ☞ Attacher l'Adaptateur et le LayoutManager au RecyclerView