

PERSISTANCE DES DONNEES:BDD

CRÉATION D'APPLICATIONS: CRÉATION D'INTERFACES UTILISATEUR AVANCÉES

DÉVELOPPEMENT D'APPLICATION MOBILE (ANDROID)

E. Djebbar
Département de Génie des systèmes
Ecole Nationale Polytechnique d'Oran

STOCKAGE DE DONNÉES

- Préférences : données clé-valeur
- Mémoire interne : système de fichiers
- Mémoire externe : système de fichiers éjectable
- Base de données : SQLite
- Réseau : serveur dédié, Web Service, Cloud, etc

STOCKAGE DANS UNE BASE DE DONNÉES SQLITE

- L'avantage d'une base de données est qu'elle permet de manipuler et de stocker des données complexes et structurées, ce qui serait impossible, ou du moins difficile à faire, avec les autres moyens de persistance décrits précédemment.

STOCKAGE DANS UNE BASE DE DONNÉES SQLITE

- À la différence de bon nombre d'autres bases de données, SQLite s'exécute sans nécessiter de serveur,
- Ce qui implique que l'exécution des requêtes sur la base de données s'effectue dans le même processus que l'application.

CHEMIN DE STOCKAGE DES BASES DE DONNÉES PAR DÉFAUT

- Toutes les bases de données sont stockées par défaut dans /data/data/<espace de noms>/ databases sur votre appareil.
- 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.

CONCEVOIR UNE BASE DE DONNÉES SQLITE POUR UNE APPLICATION ANDROID

- SQLite est dédiée pour les appareils mobiles
 - Peu d'espace de stockage, de mémoire vive et de puissance
- Éviter donc de mettre des volumes de données importants dans vos bases ou d'effectuer des requêtes fréquentes
- De structures simples résultent des requêtes simples et des données facilement identifiables.
- SQLite étant une base de données légère,
 - Eviter d'y enregistrer des données binaires (images, par exemple).

CRÉER ET METTRE À JOUR LA BASE DE DONNÉES SQLITE

- Le SDK Android offre une classe d'aide : `SQLiteOpenHelper`.
- Celle-ci doit être dérivée de façon à personnaliser les méthodes nécessaires à l'application. Parmi ces méthodes:
 - une méthode de création `onCreate`,
 - une méthode de mise à jour `onUpgrade`
 - une méthode pour ouvrir la base de données en toute simplicité.

CRÉER ET METTRE À JOUR LA BASE DE DONNÉES SQLITE

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
class MaBaseOpenHelper extends SQLiteOpenHelper {
public MaBaseOpenHelper(Context context, String nom, CursorFactory
cursorfactory, int version)
{
super(context, nom, cursorfactory, version);
}
```


SQUELETTE DE LA CLASSE D'AIDE D'UNE BASE DE DONNÉES SQLITE

@Override

```
public void onCreate(SQLiteDatabase db) {  
    // TODO Ajoutez votre code de création ici ...  
}
```

@Override

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    // TODO Ajoutez votre code de mise à jour ici ...  
}  
}
```

CRÉATION D'UNE CLASSE D'AIDE À LA CRÉATION/MISE À JOUR D'UNE BASE DE DONNÉES

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
class MaBaseOpenHelper extends SQLiteOpenHelper {
private static final String TABLE_PLANETES = "table_planetes";
private static final String COLONNE_ID = "id";
private static final String COLONNE_NOM = "nom";
private static final String COLONNE_RAYON = "rayon";
private static final String REQUETE_CREATION_BD = "create table "
+ TABLE_PLANETES + " (" + COLONNE_ID
+ " integer primary key autoincrement, " + COLONNE_NOM
+ " text not null, " + COLONNE_RAYON + " text not null);";
```

CRÉATION D'UNE CLASSE D'AIDE À LA CRÉATION/MISE À JOUR D'UNE BASE DE DONNÉES

```
public MaBaseOpenHelper(Context context, String nom,  
    CursorFactory cursorfactory, int version) {  
    super(context, nom, cursorfactory, version);  
}  
  
@Override  
public void onCreate(SQLiteDatabase db) {  
    db.execSQL(REQUETE_CREATION_BD);  
}
```

CRÉATION D'UNE CLASSE D'AIDE À LA CRÉATION/MISE À JOUR D'UNE BASE DE DONNÉES

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Dans notre cas, nous supprimons la base et les données pour en créer
    // une nouvelle ensuite. Vous pouvez créer une logique de mise à jour
    // propre à votre base permettant de garder les données à la place.
    db.execSQL("DROP TABLE" + TABLE_PLANETES + ";");
    // Création de la nouvelle structure.
    onCreate(db);
}
```

OUVRIR UNE BASE DE DONNÉES EN LECTURE ET LECTURE/ÉCRITURE

```
MaBaseOpenHelper monHelper = new MaBaseOpenHelper(context,  
NOM_BASE_DONNEES, null, VERSION_BASE_DONNEES);  
SQLiteDatabase db = monHelper.getWritableDatabase();  
...  
SQLiteDatabase db = monHelper.getWReadableDatabase();
```

CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
public class PlanetesDBAdaptateur {  
    private static final int BASE_VERSION = 1;  
    private static final String BASE_NOM = "planetes.db";  
    private static final String TABLE_PLANETES = "table_planetes";  
    private static final String COLONNE_ID = "id";  
    private static final int COLONNE_ID_ID = 0;  
    private static final String COLONNE_NOM = "nom";  
    private static final int COLONNE_NOM_ID = 1;  
    private static final String COLONNE_RAYON = "rayon";  
    private static final int COLONNE_RAYON_ID = 2;  
    // La requête de création de la structure de la base de données.  
    private static final String REQUETE_CREATION_BD = "create table "  
    + TABLE_PLANETES + " (" + COLONNE_ID  
    + " integer primary key autoincrement, " + COLONNE_NOM  
    + " text not null, " + COLONNE_RAYON + " text not null);";
```

CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
// L'instance de la base qui sera manipulée au travers de cette classe.  
private SQLiteDatabase maBaseDonnees;  
private MaBaseOpenHelper baseHelper;  
public PlanetesDBAdaptateur(Context ctx) {  
    baseHelper = new MaBaseOpenHelper(ctx, BASE_NOM, null, BASE_VERSION);  
}  
/**  
 * Ouvre la base de données en écriture.  
 */  
public SQLiteDatabase open() {  
    maBaseDonnees = baseHelper.getWritableDatabase();  
    return maBaseDonnees;  
}
```

CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
/**  
 * Ferme la base de données.  
 */  
public void close() {  
    maBaseDonnees.close();  
}  
public SQLiteDatabase getBaseDonnees() {  
    return maBaseDonnees;  
}
```


CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
/**
 * Récupère une planète en fonction de son nom.
 */
public Planete getPlanete(String nom) {
    // Insérer le code de requête d'une planète.
}

/**
 * Récupère une planète en fonction de son id.
 */
public Planete getPlanete(int id) {
    // Insérer le code de requête d'une planète.
}

/**
 * Retourne toutes les planètes de la base de données.
 */
public ArrayList<Planete> getAllPlanetes() {
    // Insérer le code de requête de l'ensemble des planètes de la base.
}
```

CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
/**
 * Insère une planète dans la table des planètes.
 */
public long insertPlanete(Planete planete) {
    // Insérer le code d'insertion.
}

/**
 * Met à jour une planète dans la table des planètes.
 */
public int updatePlanete(int id, Planete planeteToUpdate) {
    // Insérer le code de mise à jour de la base.
}

/**
 * Supprime une planète à partir de son nom.
 */
public boolean removePlanete(String nom) {
    // Insérer le code de suppression d'une planète.
}
```

CRÉATION DE L'ADAPTATEUR DE LA BASE DE DONNÉES

```
/**  
 * Supprime une planète à partir de son id.  
 */  
public boolean removePlanete(int id) {  
    // Insérer le code de suppression d'une planète.  
}  
private class MaBaseOpenHelper extends SQLiteOpenHelper {  
    ...  
}  
}
```

CRÉATION D'UNE CLASSE TYPÉE POUR MANIPULER LES DONNÉES ISSUES DE LA BASE

```
public class Planete {  
...private int id;  
...private String nom;  
...private float rayon;  
...public Planete() { }  
...public Planete(String nom, float rayon)  
...{  
.....this.nom = nom;  
.....this.rayon = rayon;  
...}
```

CRÉATION D'UNE CLASSE TYPÉE POUR MANIPULER LES DONNÉES ISSUES DE LA BASE

```
...public int getId() {  
.....return id;  
...}  
...public void setId(int id) {  
.....this.id = id;  
...}  
...public String getNom() {  
.....return nom;  
}  
...public void setNom(String nom) {  
.....this.nom = nom;  
...}
```

CRÉATION D'UNE CLASSE TYPÉE POUR MANIPULER LES DONNÉES ISSUES DE LA BASE

```
...public float getRayon() {  
.....return rayon;  
...}  
...public void setRayon(float rayon) {  
.....this.rayon = rayon;  
...}  
}
```

EFFECTUER UNE REQUÊTE DANS UNE BASE SQLITE

- Toutes les requêtes de sélection SQLite s'effectuent via la méthode query d'une instance de SQLiteDatabase.
- Cette méthode retourne un curseur permettant ensuite de naviguer dans les résultats
- La requête de sélection SQL sera construite par la méthode, puis compilée pour interroger la base de données.
- Les paramètres de la requête sont donc très proches d'une requête SELECT standard.

REQUÊTE DES DONNÉES DE LA BASE DE DONNÉES

...

```
private static final String TABLE_PLANETES = "table_planetes";
```

```
public static final String COLONNE_ID = "id";
```

```
public static final int COLONNE_ID_ID = 0;
```

```
public static final String COLONNE_NOM = "nom";
```

```
public static final int COLONNE_NOM_ID = 1;
```

```
public static final String COLONNE_RAYON = "rayon";
```

```
public static final int COLONNE_RAYON_ID = 2;
```

```
/**
```

```
 * Récupère une planète en fonction de son nom.
```

```
 */
```

```
public Planete getPlanete(String nom) {
```

```
    Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {  
        COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,
```

```
        COLONNE_NOM + " LIKE " + nom, null);
```

```
    return cursorToPlanete(c);
```

```
}
```


REQUÊTE DES DONNÉES DE LA BASE DE DONNÉES

/**

* Récupère une planète en fonction de son nom.

*/

```
public Planete getPlanete(String nom) {  
    Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {  
        COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,  
        COLONNE_NOM + " LIKE " + nom, null);  
    return cursorToPlanete(c);  
}
```

/**

* Récupère une planète en fonction de son id.

*/

```
public Planete getPlanete(int id) {  
    Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {  
        COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,  
        COLONNE_ID + " = " + id, null);  
    return cursorToPlanete(c);  
}
```

REQUÊTE DES DONNÉES DE LA BASE DE DONNÉES

```
public ArrayList<Planete> getAllPlanetes() {  
    Cursor c = maBaseDonnees.query(TABLE_PLANETES, new String[] {  
        COLONNE_ID, COLONNE_NOM, COLONNE_RAYON }, null, null, null,  
        null, null);  
    return cursorToPlanetes(c);  
}  
...
```

REMARQUE

- Les méthodes `cursorToPlanetes` et `cursorToPlanete` permettent de transformer le curseur de résultat en objet métier

LES MÉTHODES DE NAVIGATION DE LA CLASSE CURSOR

Nom	Description
<code>moveToFirst</code>	Déplace le curseur à la première position pour lire les données de la première ligne.
<code>moveToLast</code>	Déplace le curseur à la dernière position pour lire les données de la dernière ligne de la requête.
<code>moveToNext</code>	Déplace le curseur d'une position pour lire la ligne suivante.
<code>moveToPrevious</code>	Déplace le curseur d'une position pour lire la ligne précédente.
<code>moveToPosition(int)</code>	Déplace le curseur à la position indiquée.

MÉTHODES D'INFORMATION DE LA CLASSE CURSOR

Nom	Description
<code>getCount</code>	Retourne le nombre de lignes qui sont renvoyées par la requête.
<code>getColumnName(int)</code>	Retourne le nom de la colonne spécifiée par son index.
<code>getColumnNames</code>	Retourne un tableau de chaînes de caractères avec le nom de toutes les colonnes retournées.
<code>getColumnCount</code>	Retourne le nombre de colonnes renvoyées par la requête.

PARCOURIR L'INTÉGRALITÉ DES LIGNES RETOURNÉES PAR UNE REQUÊTE SQLITE

```
Cursor c = maBaseDeDonnees.query(..);  
...  
// On place le curseur au début en vérifiant qu'il contient des résultats.  
if (c.moveToFirst() {  
do {  
...  
int index = c.getInt(COLONNE_ID_ID);  
String nom = c.getString(COLONNE_NOM_ID);  
...  
} while (c.moveToNext());  
}
```

TRANSFORMATION DES DONNÉES DE LA BASE VERS UNE CLASSE TYPÉE

```
private Planete cursorToPlanete(Cursor c) {  
    // Si la requête ne renvoie pas de résultat.  
    if (c.getCount() == 0)  
        return null;  
    Planete retPlanete = new Planete();  
    // Extraction des valeurs depuis le curseur.  
    retPlanete.setId(c.getInt(COLONNE_ID_ID));  
    retPlanete.setNom(c.getString(COLONNE_NOM_ID));  
    retPlanete.setRayon(c.getFloat(COLONNE_RAYON_ID));  
    // Ferme le curseur pour libérer les ressources.  
    c.close();  
    return retPlanete;  
}
```

TRANSFORMATION DES DONNÉES DE LA BASE VERS UNE CLASSE TYPÉE

```
private ArrayList<Planete> cursorToPlanetes(Cursor c) {  
    // Si la requête ne renvoie pas de résultat.  
    if (c.getCount() == 0)  
        return new ArrayList<Planete>(0);  
    ArrayList<Planete> retPlanetes = new ArrayList<Planete>(c.getCount());  
    c.moveToFirst();  
    do {  
        Planete planete = new Planete();  
        planete.setId(c.getInt(COLONNE_ID_ID));  
        planete.setNom(c.getString(COLONNE_NOM_ID));  
        planete.setRayon(c.getFloat(COLONNE_RAYON_ID));  
        retPlanetes.add(planete);  
    } while (c.moveToNext());  
    // Ferme le curseur pour libérer les ressources.  
    c.close();  
    return retPlanetes;  
}
```


INSÉRER DES DONNÉES

- Insérer ou mettre à jour des données dans une base SQLite repose sur l'utilisation de la méthode insert de la classe SQLiteDatabase.
- 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 et la valeur.

INSERTION DE DONNÉES DANS UNE BASE DE DONNÉES SQLITE

...

```
public long insertPlanete(Planete planete) {  
    ContentValues valeurs = new ContentValues();  
    valeurs.put(COLONNE_NOM, planete.getNom());  
    valeurs.put(COLONNE_RAYON, planete.getRayon());  
    return maBaseDonnees.insert(TABLE_PLANETES, null, valeurs);  
}
```

```
public long insertPlanete(ContentValues valeurs) {  
    return maBaseDonnees.insert(TABLE_PLANETES, null, valeurs);  
}
```

...

MISE À JOUR DE DONNÉES DANS UNE BASE DE DONNÉES SQLITE

...

```
public int updatePlanete(int id, Planete planeteToUpdate) {  
    ContentValues valeurs = new ContentValues();  
    valeurs.put(COLONNE_NOM, planeteToUpdate.getNom());  
    valeurs.put(COLONNE_RAYON, planeteToUpdate.getRayon());  
    return maBaseDonnees.update(TABLE_PLANETES, valeurs, COLONNE_ID + " = "  
    + id, null);  
}
```

SUPPRIMER DES DONNÉES

...

```
public int removePlanete(String nom) {  
    return maBaseDonnees.delete(TABLE_PLANETES, COLONNE_NOM + " LIKE "  
    + nom, null);  
}  
  
public int removePlanete(int id) {  
    return maBaseDonnees.delete(TABLE_PLANETES, COLONNE_ID + " = " + id,  
    null);  
}
```