Université Mohamed Boudiaf de M'Sila
Faculté des Mathématique et de l'Informatique
Département d'Informatique

Les Bases de Données

Samir Akhrouf

Année universitaire
2019-2020

Introduction

Une application Android n'est rien de plus qu'une série d'instructions que le processeur doit exécuter. Les instructions peuvent servir à entrer des données ou afficher des résultats.

Nous avons deux opérations importantes:

- La lecture, c’est-à-dire le chargement en mémoire centrale et leur stockage sur un support interne
- L’affichage des résultats après un certain traitement

Exemple de stockage en mémoire centrale:

- durant son cycle de vie, une application Android stocke l'état d'une activité, **paused**, **stopped** pour les restituer lors de son passage à l'état **resumed.**

Exemple de stockage définitif:

- stockage de quelques paramètres
- stockage de données dans des fichiers ou bases de données.

Stockage de données

- Stockage de données simples indexées par clés.
- Stockage de données dans des fichiers non structurés
- Stockage de données structurées dans une base de données

I.Stockage de données simples

- Un ensemble de chaînes indexées par des clés
- Utilisable généralement pour communiquer des informations entre deux processus, activités, …

Utilisation de SharedPreferences

SharedPreferences: Un ensemble d'API Android permettant de manipuler des collections basées clés dans une application.

SharedPreferences

- C'est un objet lié à l'application pointant vers un fichier contenant des collections de données basées clés (key-value pairs).
- Il fournit un ensemble de méthodes permettant de lire et écrire leurs contenus
- Il peut être **privé** (private) : propre à l'application uniquement ou **partagé** (shared) avec d'autres applications.

SharedPreferences

- Création ou accès à un Sharedprefrence

Exemple1 : Plusieurs Sharedprefrence file

```
Context  context  =  getActivity();
SharedPreferences  sharedPref  =
  context.getSharedPreferences(
  getString(R.string.preference_file_key),
  Context.MODE_PRIVATE);
```

Remarque : Le nom doit identifier le Sharedpreference d'une manière unique (l'associer au package) :

"com.example.myapp.PREFERENCE_FILE_KEY"

SharedPreferences

- Création ou accès à un sharedpreference

Exemple2 : Sharedprefernce file par défaut

```
SharedPreferences  sharedPref  =
getActivity().getPreferences(Context.MODE_PRIVATE);
```

- Lecture d'un sharedpreference

Utiliser les méthodes getInt et getString en donnant comme paramètres : la clé et éventuellement une valeur par défaut.

```
int  rang  =  sharedPref.getInt("CLE1",  -1);
```

SharedPreferences

- Ecriture dans un sharedpreference
- Créer un éditeur SharedPreferences.Editor en appellant la méthode edit().
- Passer les clés et les valeurs associées aux méthodes d'écriture : putInt(), putString(), …
- Appeler la method commit() pour sauvegarder ces valeurs

```
int  val  =  ClaculerScore();
SharedPreferences.Editor editor  =  sharedPref.edit();
editor.putInt("CLE1", val);
editor.commit();
```

II. Manipulation des fichiers

- Un ensemble de données stockées d'une manière permanente sur un support de stockage.
- Comme tout système d'exploitation, Android fournit des API permettant de manipuler les fichiers:

Actions *:*

- Creation
- Lecture
- Ecriture
- Suppression
- …

L'API File

- Un objet File permet de manipuler un fichier.
- Il fournit un ensemble de méthodes permettant de lire et écrire son contenu.
- Il peut être **privé** (private) : propre à l'application uniquement ou **partagé** (shared) avec d'autre apps.
- Il peut être stocké sur Mémoire Interne ou externe.

Lecture/écriture d'un fichier de la mémoire externe

Autorisations :

```
<manifest  ...>
   <uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE"  />
   ...
</manifest>
```

```
<manifest  ...>
   <uses-permission
   android:name="android.permission.READ_EXTERNAL_STORAGE"  />
   ...
</manifest>
```

Sauvegarde d'un fichier dans la mémoire interne

- Utiliser le répertoire adéquat :
- getFilesDir() : retourne un objet File représentant le répertoire interne de l'application
- getCacheDir() : retourne un objet File représentant le répertoire interne de l'application pour les fichiers temporaire.
- Les fichiers cache doivent être courts et être supprimés.

Sauvegarder un fichier dans la mémoire interne

- Création d'un Fichier

  Appeler le constructeur File

```
File file = new File(context.getFilesDir(), filename);
```

- Ecriture dans un Fichier

```
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream;
try {
    outputStream = openFileOutput(filename,
    Context.MODE_PRIVATE);
    outputStream.write(string.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Sauvegarder un fichier dans la mémoire interne

- Création d'un Fichier TEMPORAIRE

```
File  file;
try  {
   String  fileName  =  "myfiletemp"
   file  =  File.createTempFile(fileName,  null,
   context.getCacheDir());
catch  (IOException  e)  {
   //  Error  while  creating  file
}
return  file;
```

Sauvegarder un fichier dans la mémoire externe

- Vérifier si la mémoire externe est montée:

getExternalStorageState() : retourne l'état de la mémoire externe.

Etat = Environment.MEDIA_MOUNTED

getExternalStorageState() : retourne l'état de la mémoire externe.

(Environment.MEDIA_MOUNTED)

Vérifier si la mémoire externe est prête à écrire
(Environment.MEDIA_MOUNTED_READ_ONLY)

Sauvegarder un fichier dans la mémoire externe

```
public boolean isExternalStorageWritable() {
String state = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED.equals(state)) {
 return true;
}
 return false;
}

/* Checks if external storage is available to at least read
*/
public boolean isExternalStorageReadable() {
String state = Environment.getExternalStorageState();
 if (Environment.MEDIA_MOUNTED.equals(state) ||
    Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) { return true;
 }
 Return false;
}
```

Un fichier dans la Mem Ext peut être :

- **Private** : supprimés lors de la désinstallation de l'application.
- **Public** : non supprimés lors de la désinstallation de l'application (Ex. rapports, photo, bdd, ..)

Sauvegarder des fichiers publics

```
public File getAlbumStorageDir(String albumName) {
// Get the directory for the user's public pictures
  directory.
File file = new
  File(Environment.getExternalStoragePublicDirectory(
  Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
    Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

- Environment.DIRECTORY_PICTURES : pour permettre à android d'organiser les fichiers.
- Pour un répertoire quelconque; faites passer null.

- Sauvegarder des fichiers privés

```
public File getAlbumStorageDir(String albumName) {
// Get the directory for the user's public pictures
   directory. File file = new
   File(Environment.getExternalFilesDir(
Environment.DIRECTORY_PICTURES), albumName);
if (!file.mkdirs()) {
Log.e(LOG_TAG, "Directory not created");
}
return file;
```

- Environment.DIRECTORY_PICTURES : pour permettre à android d'organiser les fichiers.
- Pour un répertoire quelconque; faites passer null.
- Supprimer un fichier dans la mémoire externe

```
myFile.delete()
```

- Avoir de l'espace libre

`getFreeSpace()` or `getTotalSpace()`

III. Manipulation des Bases de données SQL

- Android utilise SQLite3 comme SGBD pour manipuler les BDDs relationnelles.
- SQLite est un SGBD compact convenables aux apps mobiles, mais aussi utiles pour des BDDs spécifiques dans des apps desktop (Skype, Adobe Reader, …).
- SQLite est fourni dans Android sous forme de bibliothèque d'API utilisable dans le code client.
- Ce n'est pas un serveur auquel on se connecte et on en envoie de requêtes.
- SQL lite API tourne dans le même process que l'application.
- *Chaque application administre et manipule sa propre base indépendamment des autres.*

- Une base de donnée SqlLite est un fichier, ayant généralement l'extension «.db» et créé en mode MODE_PRIVATE
- Le fichier est stocke par défaut dans le répertoire:

/data/data/<packagename>/databases/

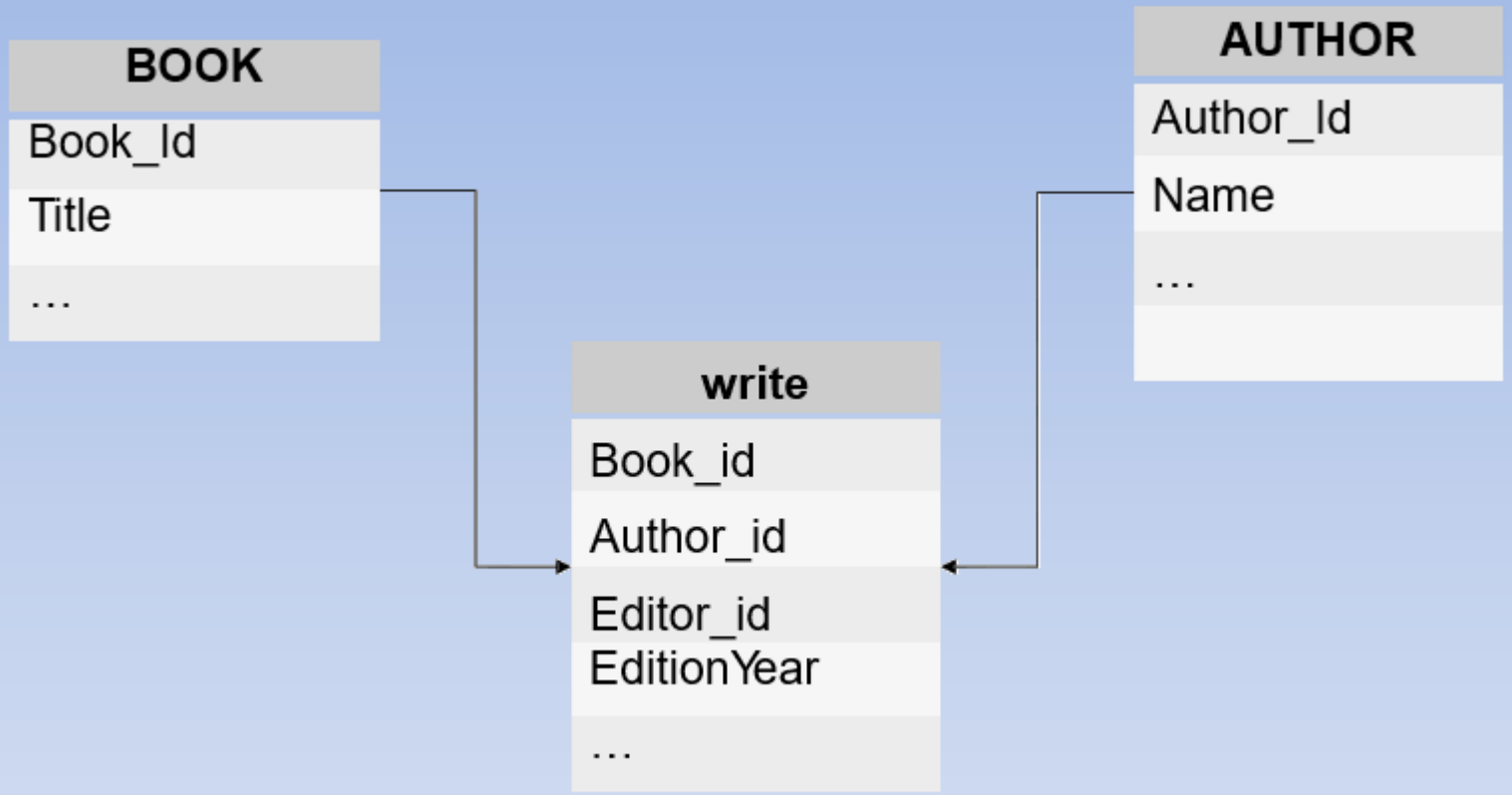
Pour manipuler une BDD, on doit:

- Définir le schéma de notre données
- Créer la BDD
- Mettre à jour la BDD : select, insert, delete ou update

a. Définition du schéma de la BDD

- C'est une déclaration formelle décrivant l'organisation ou la structure interne de la BDD.
- Un schéma de la BDD spécifie sa structure interne en terme de tables et les relation inter-tables.
- Reflétée par l'expression SQL utilisée lors de la création de la BDD.

a. Définition du schéma de la BDD



- Un schéma de BDD est spécifie en utilisant des « contract classes ».
- Un contract class est un conteneur de constantes définissant des Urls, des tables et des colonnes.
- Un contract class doit être crée pour chaque table de la BDD.

```
public final class BookContract {
// To prevent someone from accidentally instantiating the contract
class,
// give it an empty construtor

 public BookContract() {}

/* Inner class that defines the table contents */
public static abstract class BookEntry implements BaseColumns {
public static final String TABLE_NAME = “book";
public static final String COLUMN_NAME_BOOK_ID = “bookid";
public static final String COLUMN_NAME_TITLE = "title";
//...
}
}
```

b. Créer la BDD

- La base de donnée est créée en étendant SqlHelper.
- On fournit les expressions Sql permettant de créer, supprimer et mettre à jour la BDD.

```
Public static final String SQL_CREATE_TABLE = "CREATE TABLE " +
    BookEntry.TABLE_NAME + " (" +
BookEntry._ID + " INTEGER PRIMARY KEY," +
    BookEntry.COLUMN_NAME_BOOK_ID + " TEXT; " +
    BookEntry.COLUMN_NAME_TITLE + " TEXT " +
//... Any other options for the CREATE command
" )";

public static final String SQL_DELETE_TABLE =
"DROP TABLE IF EXISTS " + BookEntry.TABLE_NAME;
```

c. manipuler la BDD

- La classe SQLiteOpenHelper fournit un ensemble d'APIs pour manipuler maj de la BDD.
- Etendre cette classe et surcharger les mcallback methods : onCreate(), onUpgrade() et onOpen()

- onCreate() *:* est appelée lors de l'ouverture de la BDD. On doit installer la BDD dans cet méthode en créant les tables et initialisant les données.
- onUpgrade(), onDowngrade() *:* est appelée lorsque la version de l'application est mise à jour pour maintenir la compatibilité.
- onOpen() *:* est appelée lorsque la BDD est ouverte ou mise à jour.

```
public class BooksDbHelper extends SQLiteOpenHelper {
// If you change the database schema, you must increment the database
version. public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = “Books.db";

public BooksDbHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
public void onCreate(SQLiteDatabase db) {
db.execSQL(SQL_CREATE_TABLE);
}
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
// This database is only a cache for online data, so its upgrade policy
is
// to simply to discard the data and start over
db.execSQL(SQL_DELETE_TABLE);
onCreate(db);
}
public void onDowngrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
onUpgrade(db, oldVersion, newVersion);
}
}
```

d. Instancier la BDD

```
BooksDbHelper mDbHelper = new BooksDbHelper(getContext());
```

- Insertion de ligne dans une table

L'insertion dans la BDD se fait en passant un objet ContentValues à la méthode Insert.

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(BookEntry.COLUMN_NAME_BOOK_ID , id);
values.put(BookEntry. COLUMN_NAME_BOOK_TITLE , title);

// Insert the new row, returning the primary key value of the new
row long newRowId;
newRowId = db.insert(
                    BookEntry.TABLE_NAME,

                    BookEntry.COLUMN_NAME_NULLABLE, values);
```

Insertion de ligne dans une table

```
newRowId = db.insert(
                        tablename,
Default_Values,// elsewhere provide  null values
);
```

Lire des info depuis une table

La lecture est basée sur la méthode query (une requête select)

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the
database
// you will actually use after this query. String[] projection = {
BookEntry._ID, BookEntry.COLUMN_NAME_TITLE //,...
};

// How you want the results sorted in the resulting Cursor
String sortOrder =          BookEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
BookEntry.TABLE_NAME,   // The table to query
```

```
projection,           //  The  columns  to  return
selection,            //  The  columns  for  the  WHERE  clause
selectionArgs,        //  The  values  for  the  WHERE  clause
null,                 //  don't  group  the  rows
null,                 //  don't  filter  by  row  groups
sortOrder             //  The  sort  order
);
```

Lire des info depuis une table

```
public Cursor query (String table, String[] columns, String selection,
   String[] selectionArgs, String groupBy,  String having, String orderBy,
   String limit)
```

Paramètres	
table	String: The table name to compile the query against
columns	String: A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	String: A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
selectionArgs	String: You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
groupBy	String: A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
having	String: A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
orderBy	String: How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
limit	String: Limits the number of rows returned by the query, formatted as LIMIT

Lire des informations depuis une table

Cursor : un ensemble de lignes structurées selon la projection.

Pour les parcourir on utilise les méthodes 'movXX' du curseur.

```
abstract boolean  moveToFirst()Move the cursor to the first row.
abstract boolean  moveToLast()Move the cursor to the last row.
abstract boolean  moveToNext()Move the cursor to the next row.
abstract boolean  moveToPosition(int position)Move the cursor to an absolute position.
abstract boolean  moveToPrevious()
```

Lire des informations depuis une table

Récupérer les données en appelant getXX(int index)

Index est l'index du colonne que l'on peut avoir en appelant getColumnIndex

public abstract int getColumnIndex (String columnName)

Renvoie l'index de base zéro pour le nom de la colonne donnée, ou -1 si la colonne n'existe pas. Si vous êtes certains que la colonne existe, utilisez getColumnIndexOrThrow(String) instead, which will make the error more clear.

```
public ArrayList<Book> getBooks() {

   ArrayList<Book> books= new ArrayList<Book>();
 SQLiteDatabase db = getReadableDatabase();

   Cursor bookCursor = db.query(BookEntry.TABLE_NAME, null,
            null, null, null, null, null);
    bookCursor.moveToFirst();
   while (bookCursor .moveToNext()) { book
    = new Book();

    int id = bookCursor.getColumnIndex(BookEntry.COLUMN_NAME_ID)

    book.setId(bookCursor.getInt(id);

    id = bookCursor.getColumnIndex(BookEntry.COLUMN_NAME_TITLE);

    book.setTitle(bookCursor.getString(id));

    books.add(book);

 }
 projCursor.close();
 return (books);
 }
```

Supprimer des lignes d'une table

- Faire une sélection basée sur des colonnes en spécifiant leurs arguments

```
// Define 'where' part of query.
String selection = BookEntry.COLUMN_NAME_ENTRY_ID + " LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { String.valueOf(rowId) };
// Issue SQL statement.
db.delete(table_name, selection, selectionArgs);
```

Modifier une BDD

- Combine la syntaxe de Insert et Delete sur une selection

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// New value for one column
ContentValues values = new ContentValues();
values.put(BookEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the ID
String selection = BookEntry.COLUMN_NAME_ENTRY_ID + " LIKE
?"; String[] selectionArgs = { String.valueOf(rowId) };
int count = db.update(
    booksDbHelper.BookEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```