

ROOMDB



PLAN

1. contexte général
2. Avantages RoomDB
3. les Limites de RoomDB
4. Architecture de RoomDB
5. SÉNARIO D'IMPLÉMENTATION
6. Meilleures pratiques et optimisations
7. QUIZ
8. Mots croisés

1.INTRODUCTION

DÉFINITION

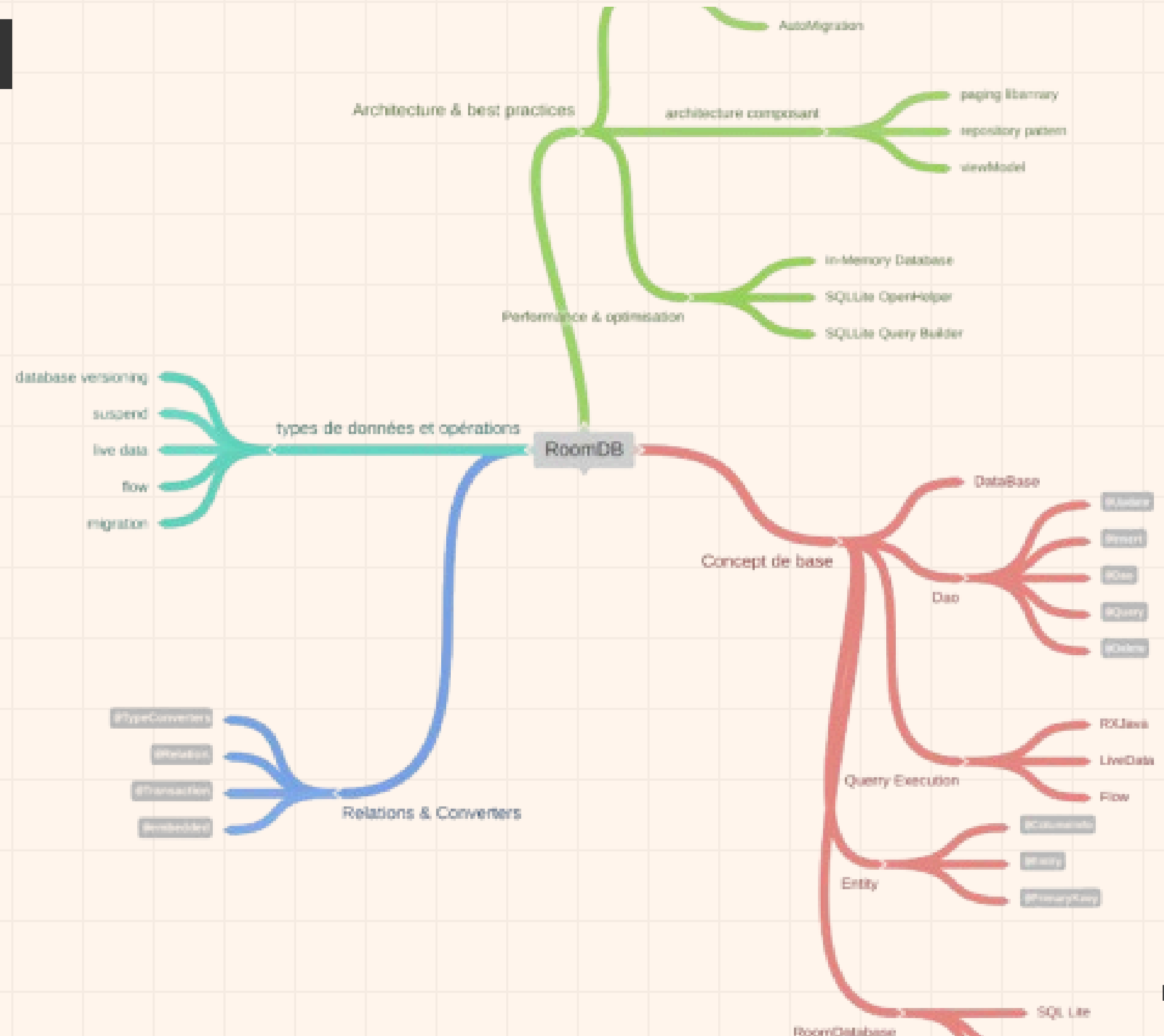
RoomDB est une bibliothèque de persistance pour Android qui simplifie l'utilisation de SQLite en offrant une abstraction plus propre et plus sûre.

Elle fait partie de Android Jetpack et permet de gérer facilement une base de données locale

CEST QUOI ROOMDB ?



1.INTRODUCTION



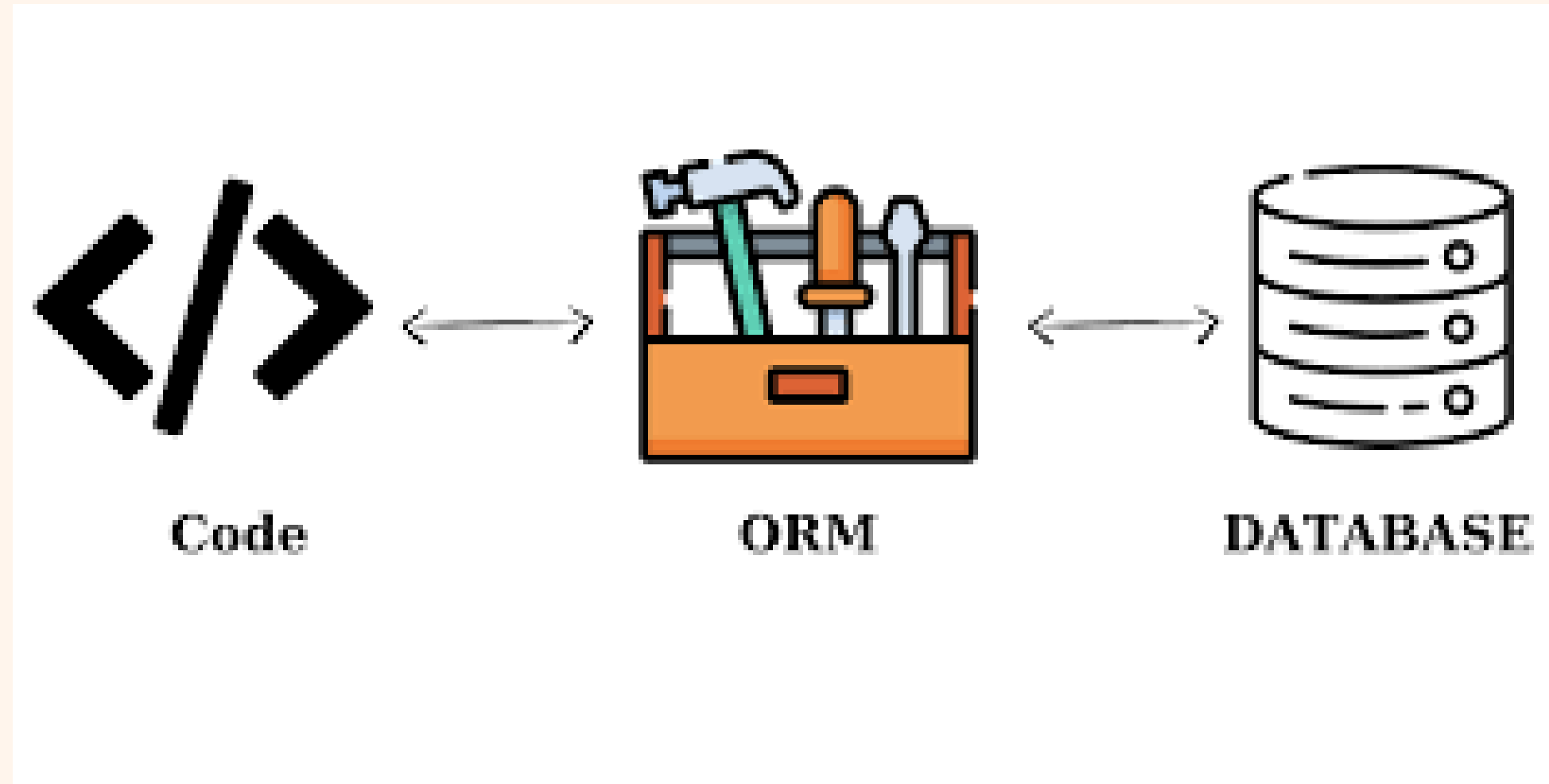
C'EST QUOI ORM ?



ROOMDB EST UN ORM : IL MAPPE DES OBJETS
KOTLIN/JAVA À DES TABLES SQLITE.

IL PERMET D'EXÉCUTER DES OPÉRATIONS CRUD SANS
ÉCRIRE DE REQUÊTES SQL COMPLEXES.

- ORM (Object-Relational Mapping) est une technique permettant d'interagir avec une base de données relationnelle via des objets plutôt qu'en écrivant directement des requêtes SQL.
- Il simplifie la gestion des bases de données en mappant des tables aux classes et des colonnes aux attributs.



Sans ORM (SQL brut) :

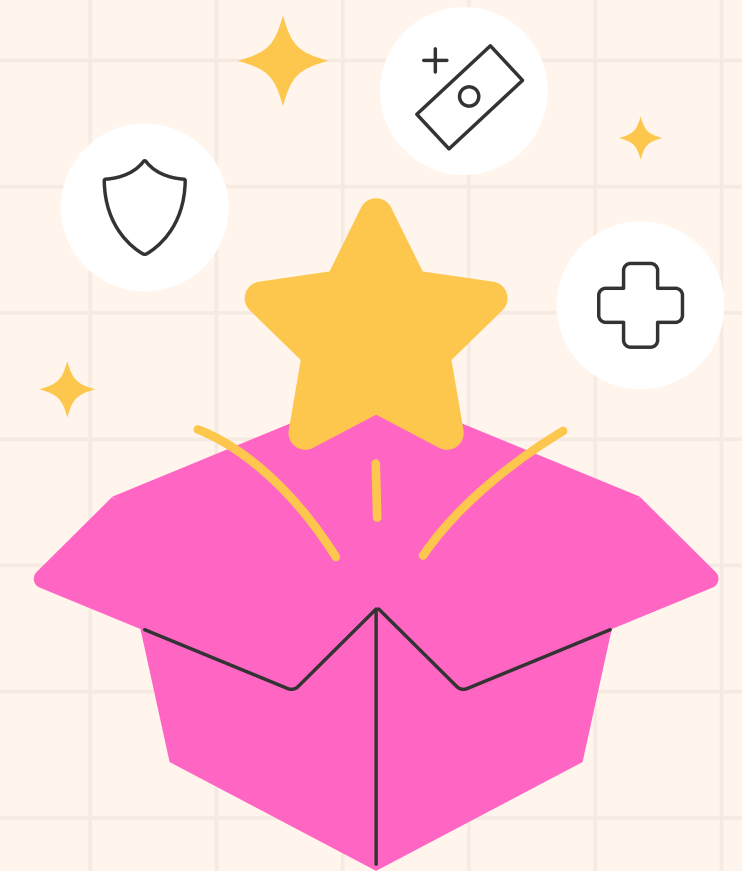
```
SELECT * FROM users WHERE age > 18;
```

Avec RoomDB :

```
@Query("SELECT * FROM users WHERE age > 18")  
fun getAdultUsers(): List<User>
```

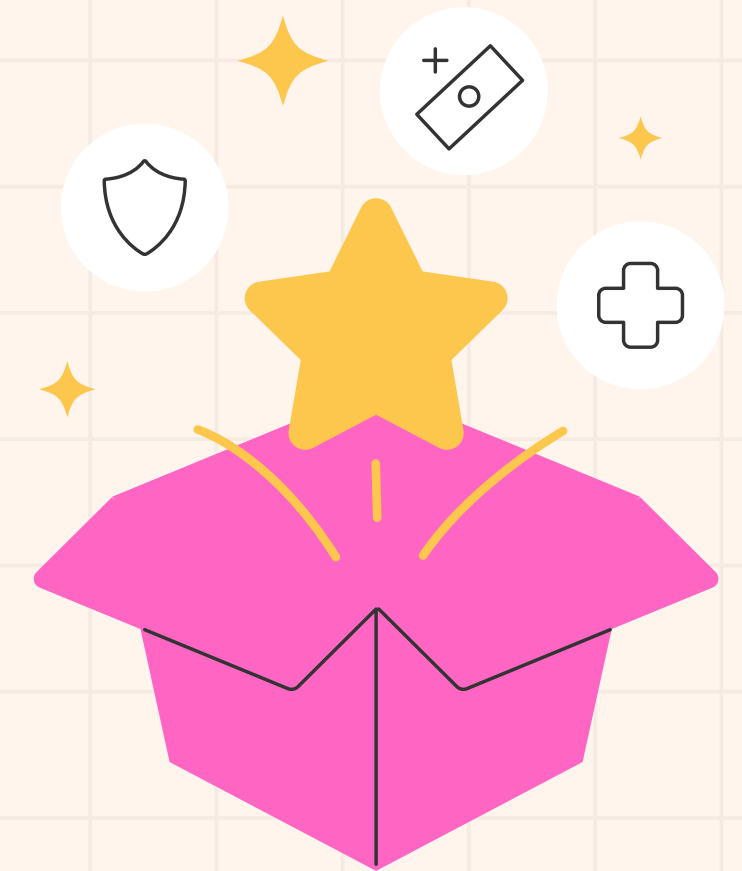
2. AVANTAGES

- **1. Abstraction de SQLite:**
 - ✓ Simplifie l'utilisation de SQLite en évitant l'écriture manuelle de requêtes SQL complexes.
 - ✓ Réduit le boilerplate code et facilite la maintenance.
- **2. Sécurité et Fiabilité:**
 - ✓ Utilise des annotations (@Entity, @Dao, @Database), ce qui réduit les erreurs humaines.
 - ✓ Validation des requêtes SQL à la compilation → erreurs détectées avant l'exécution.



2. AVANTAGES

- **3. Compatibilité avec l'Architecture MVVM:**
 - ✓ Fonctionne parfaitement avec LiveData et Flow pour des mises à jour automatiques des UI.
 - ✓ Prise en charge des Coroutines Kotlin, améliorant les performances.
- **4. Intégration avec d'autres APIs Android:**
 - ✓ Compatible avec ViewModel, LiveData, WorkManager et Paging 3.
 - ✓ Peut être combiné avec Jetpack Compose pour des applications modernes.



POURQUOI UTILISER ROOMDB ?

- Fournit une couche d'abstraction au-dessus de SQLite.
- Vérifie les requêtes SQL à la compilation pour éviter les erreurs.
- Compatible avec LiveData et Flow pour une mise à jour réactive des données.
- Prend en charge la migration de bases de données.

WHY?



3. LIMITES DE ROOMMDB

1. Pas conçu pour les grandes bases de données

Room est basé sur SQLite, qui est une base de données locale et non conçue pour gérer des millions de lignes efficacement.

Problèmes possibles :

- Requêtes lentes avec des grandes bases de données.
- Manque d'optimisation pour les gros volumes de données.
- Impossible de faire du sharding (répartition des données sur plusieurs bases).



3. LIMITES DE ROOMMDB

2. Pas de support natif pour les bases de données distantes

Room est conçu pour fonctionner localement sur l'appareil. Il ne supporte pas directement les bases de données en ligne comme Firebase ou MySQL.

Problèmes possibles :

- Impossible de partager les données entre plusieurs utilisateurs.
- Pas de synchronisation automatique des données.
- Pas de gestion des conflits entre les mises à jour de plusieurs utilisateurs.



4.ARCHITECTURE DE ROOMMDB

- **Activity**

Interface utilisateur (UI)

Récupère et affiche les données du Repository

- **Repository**

Intermédiaire entre UI et base de données

Fournit une source unique de vérité

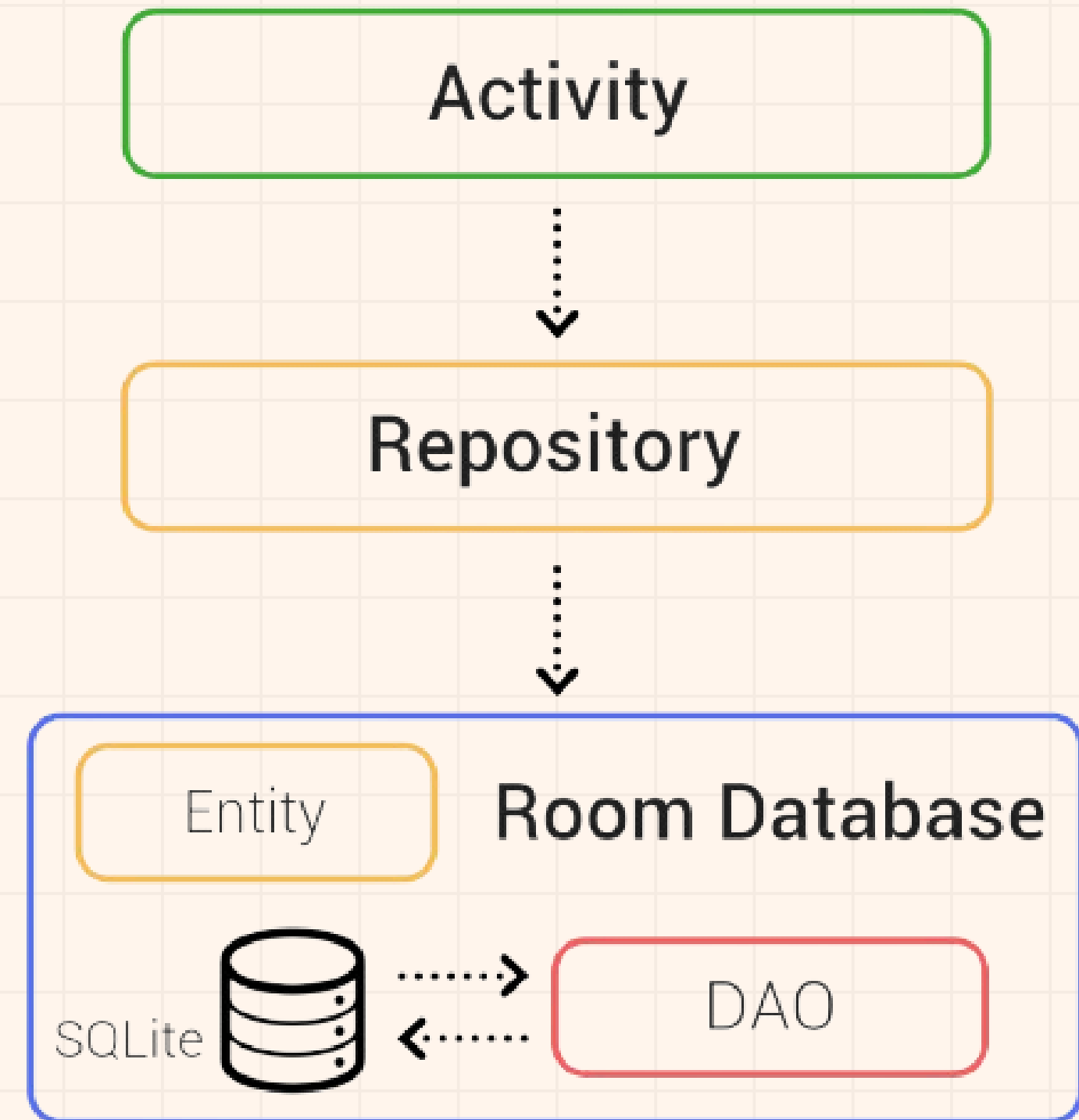
Gère les appels vers DAO

- **Room Database**

Entity : Représente une table SQLite

DAO : Contient les requêtes SQL

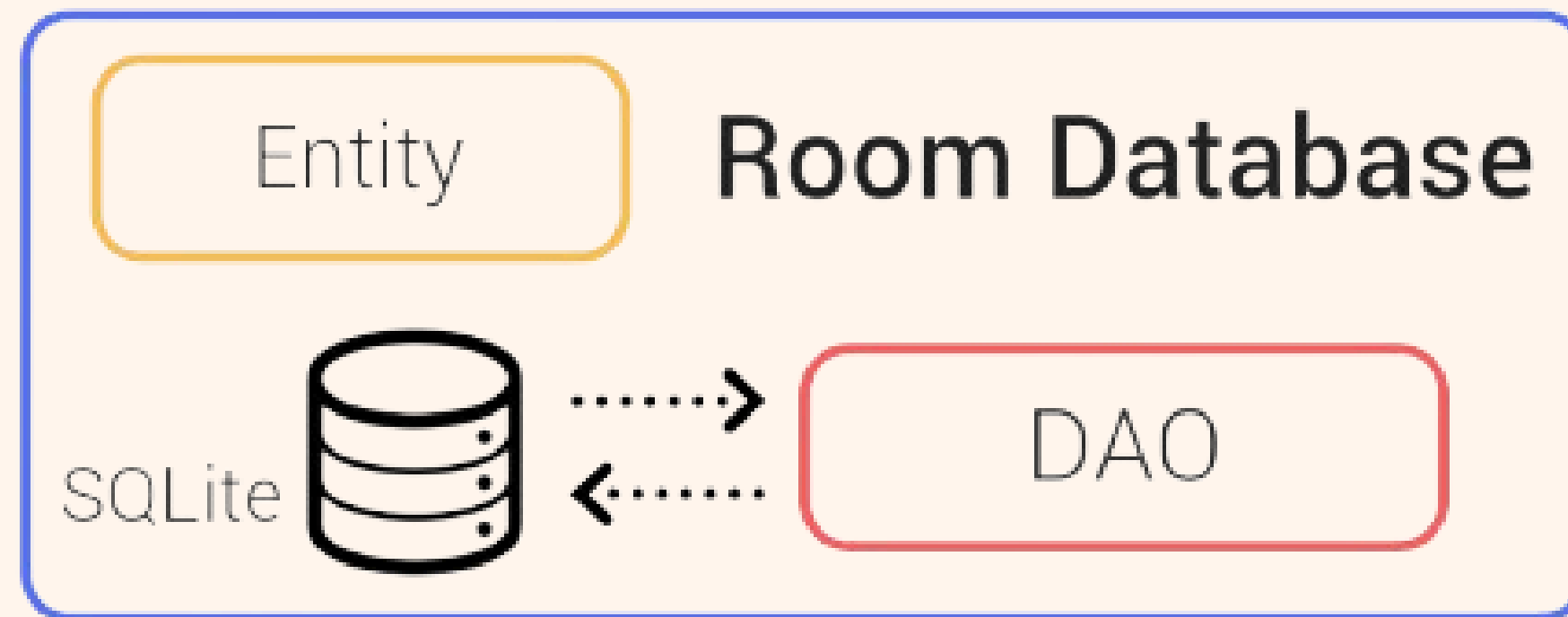
SQLite : Stocke les données



4.ARCHITECTURE DE ROOMMDB

DAO

Un DAO (Data Access Object) dans Room est une interface qui définit comment interagir avec la base de données SQLite d'une application Android. Il sert d'intermédiaire entre l'application et la base de données en permettant l'exécution de requêtes de manière structurée.



PRINCIPALES FONCTIONNALITÉS D'UN DAO

1.Insertion de données :

Ajouter de nouvelles entrées dans la base de données.

2.Mise à jour :

Modifier des enregistrements existants.

3.Suppression :

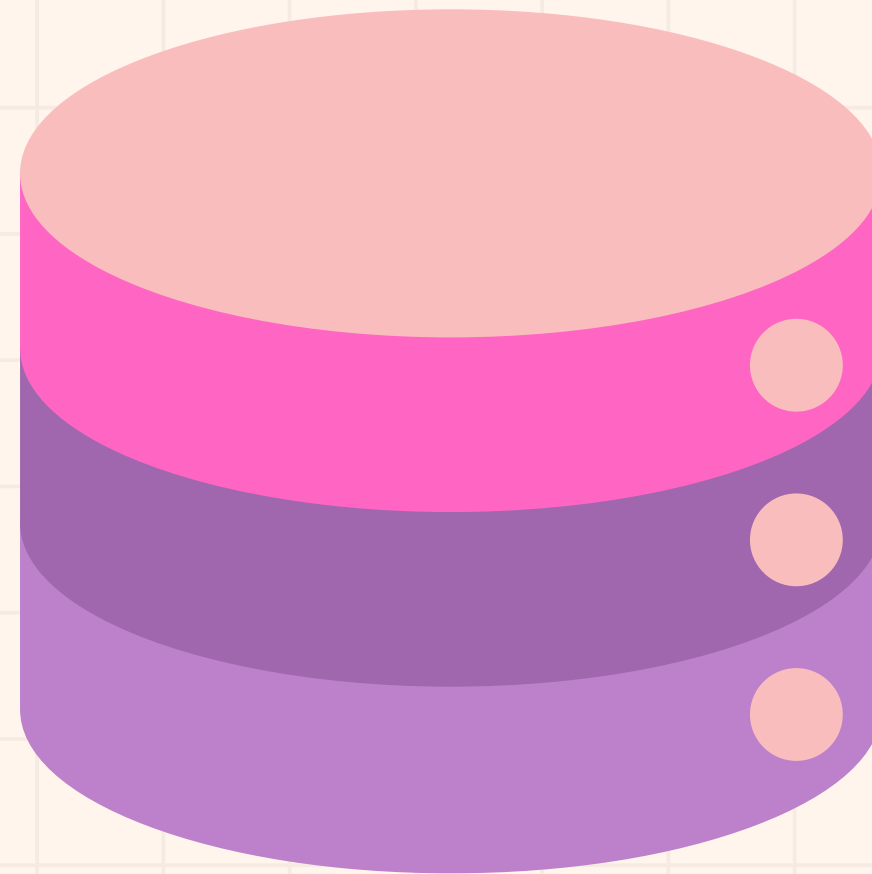
Supprimer une ou plusieurs entrées.

4.Requêtes de sélection :

Récupérer des données en fonction de certains critères

MEILLEURES PRATIQUES ET OPTIMISATIONS

Exécuter des requêtes en
arrière-plan avec Coroutines



Utilisation du
RoomDatabase.

Callback Eviter les accès à la
DB sur le Main Thread

5.SÉNARIO D'IMPLÉMENTATION

1. AJOUTER LA BIBLIOTHÈQUE ROOM À VOTRE FICHIER BUILD.GRADLE

```
implementation "androidx.room:room-runtime:$room_version"  
annotationProcessor "androidx.room:room-compiler:$room_version"
```

2. CRÉER UNE CLASSE D'ENTITÉ POUR DÉFINIR LA STRUCTURE DE LA TABLE

```
@Entity(tableName = "notes")  
data class Note(  
    @PrimaryKey(autoGenerate = true)  
    val id: Int,  
    val title: String,  
    val description: String  
)
```

3. CRÉEZ UNE INTERFACE DAO (DATA ACCESS OBJECT) POUR DÉFINIR LES OPÉRATIONS DE BASE DE DONNÉES :

```
@Dao
interface NoteDao {
    @Insert
    fun insertNote(note: Note)

    @Update
    fun updateNote(note: Note)

    @Delete
    fun deleteNote(note: Note)

    @Query("SELECT * FROM notes")
    fun getAllNotes(): LiveData<List<Note>>
}
```

4. CRÉEZ UNE CLASSE DE BASE DE DONNÉES POUR INSTANCIER LA BASE DE DONNÉES ET FOURNIR UN ACCÈS AU DAO :

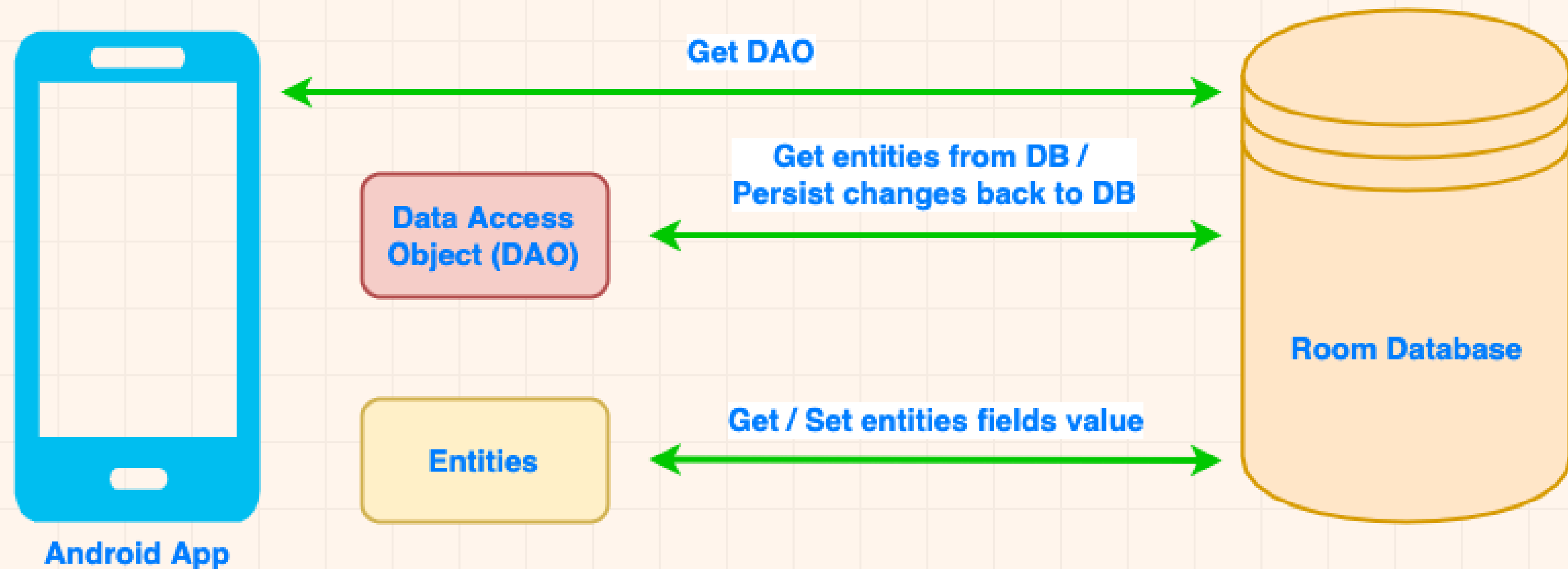
```
@Database(entities = [Note::class], version = 1)
abstract class NoteDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao
}
```

5. DANS VOTRE ACTIVITY OU FRAGMENT, INSTANCIEZ LA BASE DE DONNÉES ET UTILISEZ LE DAO POUR EFFECTUER DES OPÉRATIONS :

```
val database = Room.databaseBuilder(  
    context.applicationContext,  
    NoteDatabase::class.java,  
    "note_database"  
).build()  
  
val noteDao = database.noteDao()  
  
val notes = noteDao.getAllNotes()  
  
...  
  
noteDao.insertNote(note)  
  
...  
  
noteDao.updateNote(note)  
  
...  
  
noteDao.deleteNote(note)
```

PARCOURS D'ACCÈS AU DONNÉES

Room Architecture





Sql lite	ROOM db
Beaucoup de code (cree des tables , insérer , maj , supprimer les donnees)	Utilise des notations (@entity,@dao, @query)
Execute les requete sans verifier leur validité avant l'execution à peuvent provoquer des erreurs au runtime	Verifier les requetes a la compilation à reduire les erreurs
Mise a jour manuellement	Fonctionne avec live data et flow(kotlin) à mise a jour automatique
Ecrire des scripts SQL pour chaque modification de la structure	Gerer les migration
Cela doit modifier ses requêtes chaque fois que le schéma de base de données subit des modifications	Il n'est pas nécessaire de modifier le code lorsque le schéma de la base de données subit des modifications.

LiveData



C'est une classe du Jetpack (Android Architecture Components) conçue pour les interfaces utilisateur (UI). Elle permet d'observer les changements de données seulement sur le thread principal (UI thread).

Flow

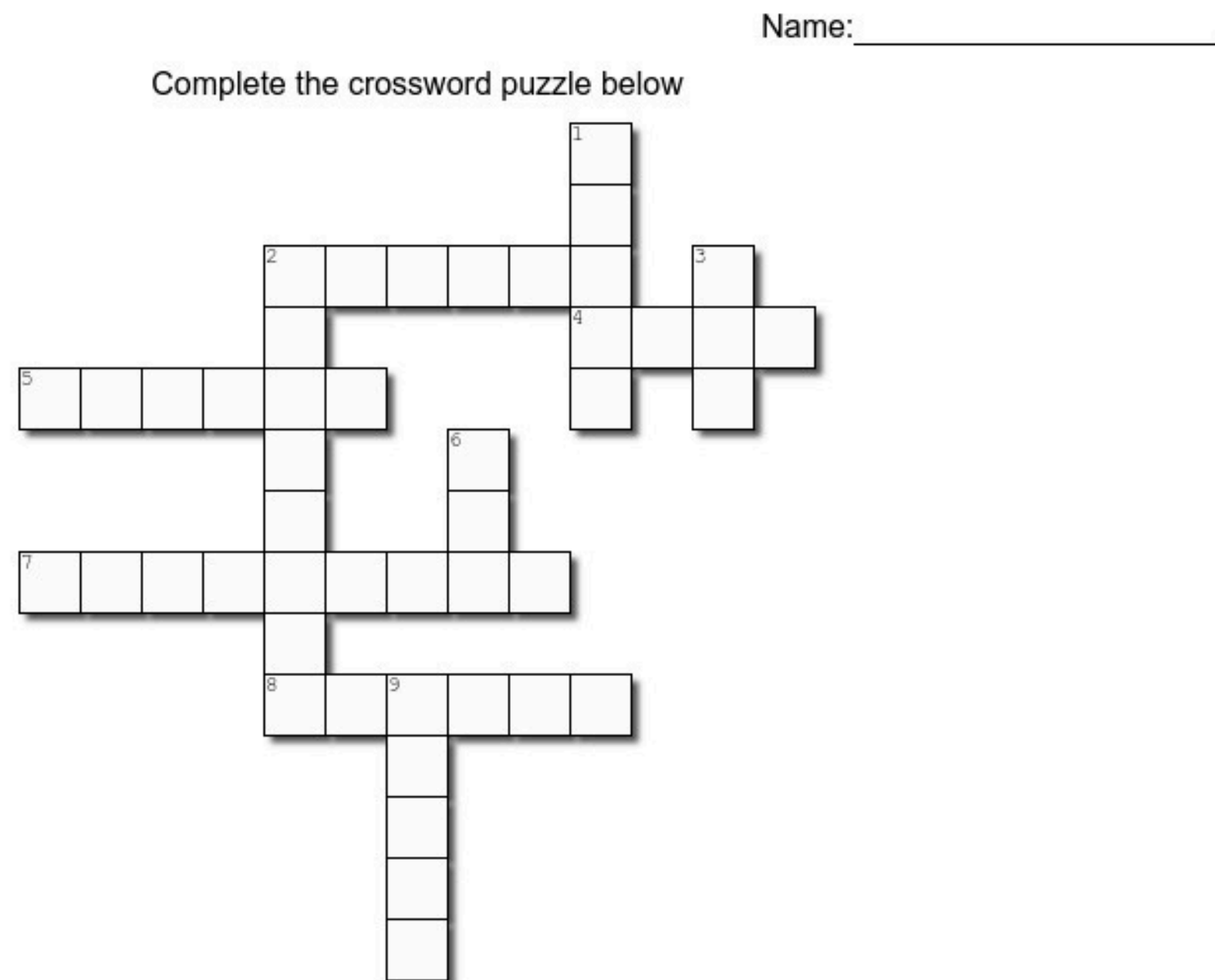
C'est une alternative moderne et plus puissante à LiveData basée sur Kotlin Coroutines. Flow est conçu pour gérer des flux de données asynchrones.

 Room est donc une bibliothèque qui :

- ✓ Utilise SQLite en arrière-plan.
- ✓ Fournit des DAO pour accéder aux données.
- ✓ Supporte LiveData et Flow pour des mises à jour automatiques.
- ✓ Permet une gestion simplifiée des migrations.

 **Conclusion** : Room n'est pas une base de données, mais bien une bibliothèque qui facilite l'utilisation de SQLite en Android. 

8.MOTS CROISÉS :

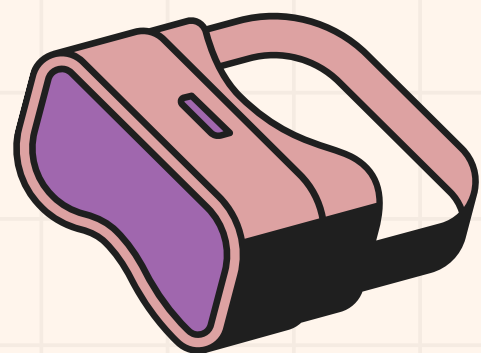


Across

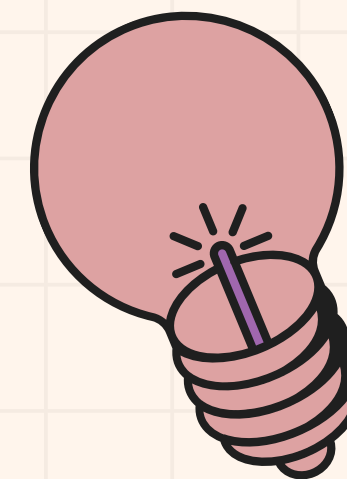
2. Action de supprimer des données
4. Bibliothèque de persistance pour Android.
5. Modifier une valeur dans la base
7. Processus pour mettre à jour la structure de la base de données.
8. Classe représentant une table en base de données.

Down

1. Commande pour récupérer des données.
2. Conteneur pour les données, contient les tables.
3. Une ligne de données dans une table.
6. Interface qui permet d'accéder aux données.
9. Structure qui stocke les données en colonnes et lignes.



MERCI POUR VOTRE



ATTENTION

