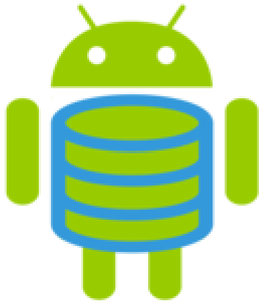


ROOM DB

Réalisé par:

Abir SMAYEN

Siwar OUNI

2DNI

PLAN

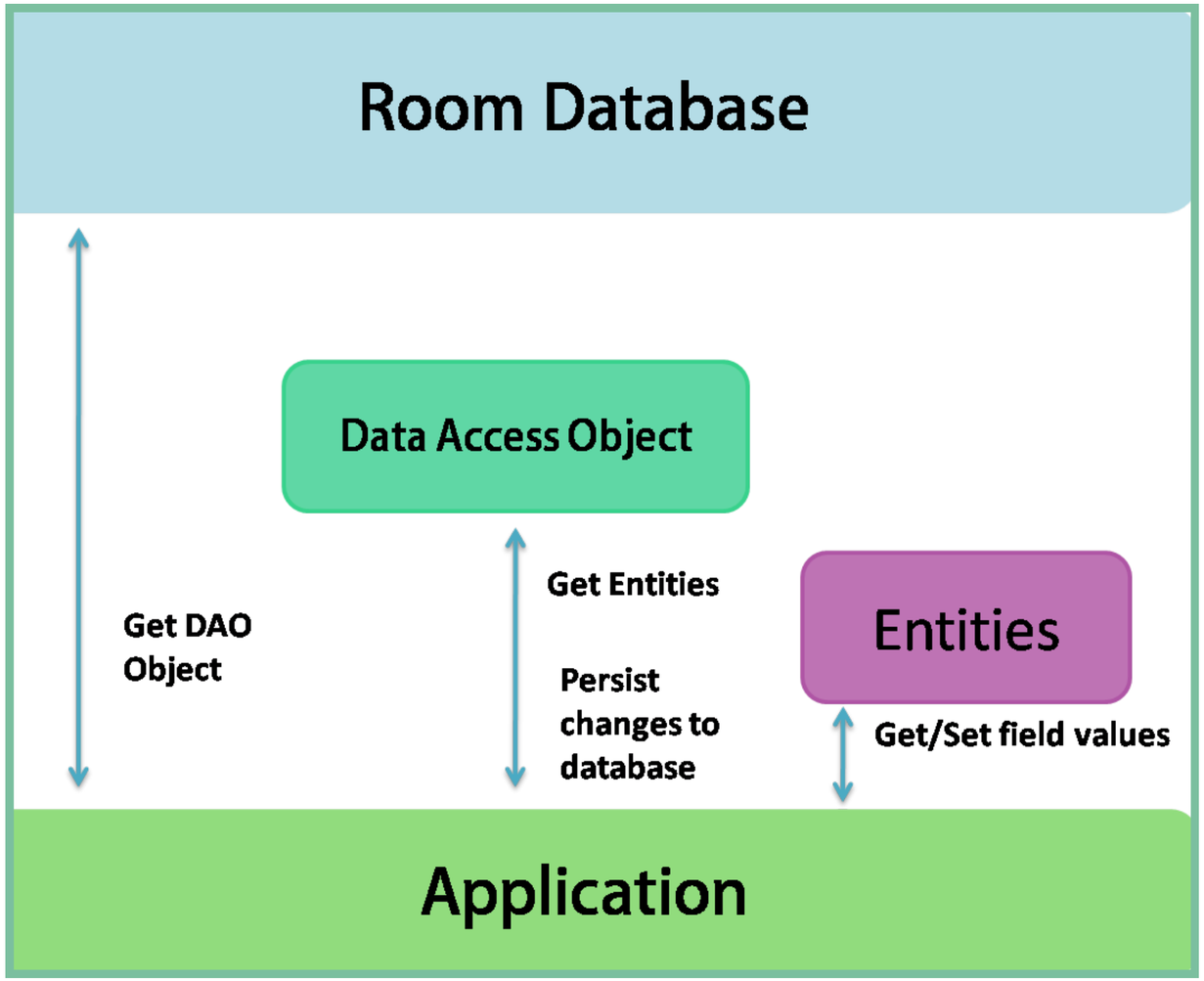
01. CONTEXTE GÉNÉRALE

Room est une bibliothèque de persistance officielle de Google, faisant partie d'Android Jetpack.
Elle permet une abstraction au-dessus de SQLite, rendant plus facile la gestion de bases de données locales.

Room simplifie :

- La création des tables,
- L'exécution de requêtes SQL,
- L'interaction entre les données et l'interface utilisateur.

02. ARCHITECTURE ROOM DATABASE



Composant 1 : @Entity

@Entity: Utilisée pour définir une classe comme une table dans la base de données, chaque attribut devient une colonne.

Exemple: Création d'une entité

```
@Entity(tableName = "contact")
public class Contact {

    2 usages
    @PrimaryKey(autoGenerate = true)
    @NonNull
    private int _id;

    2 usages
    @ColumnInfo(name = "contact_name")
    private String contactName;

    2 usages
    @ColumnInfo(name = "contact_phone")
    private String contactPhone;
```

Composant 2 : @Dao

@Dao (Data Access Object) : C'est l'interface qui contient les méthodes d'accès aux données (insertion, requêtes, suppression, etc.).

```java
@Dao
public interface ContactDAO {
    1 usage   1 implementation
    @Insert
    void insertContact(Contact contact);

    no usages   1 implementation
    @Update
    void updateContact(Contact contact);

    1 usage   1 implementation
    @Delete
    void deleteContact(Contact contact);

    1 usage   1 implementation
    @Query("SELECT * FROM contact")
    List<Contact> getAllContacts();
}
```

Composant 3 : @Database

@Database : Représente la base de données elle-même et fait le lien entre les entités et les DAO.

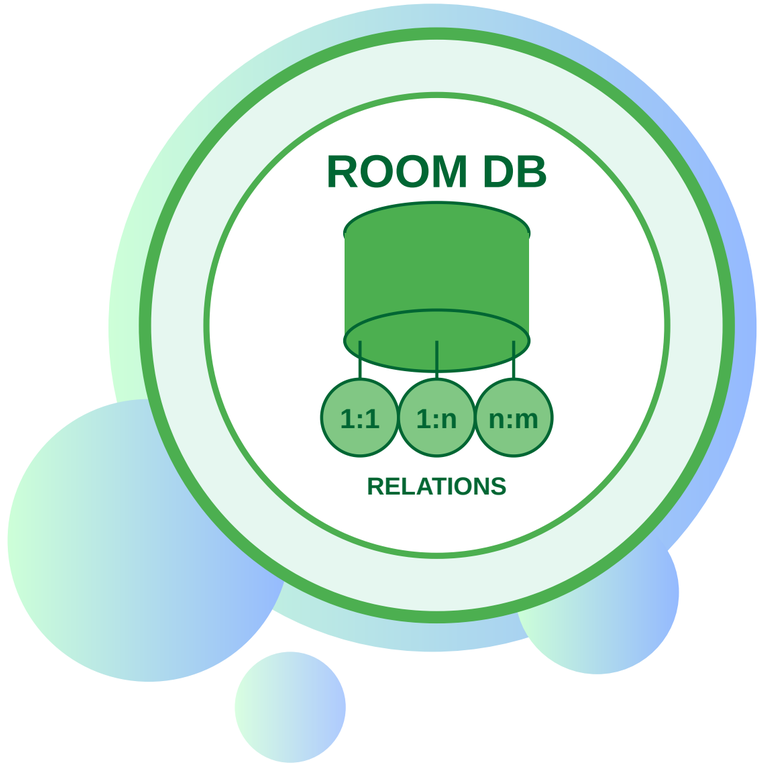
Exemple: Classe AppDatabase

```
@Database(entities = {Contact.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {

    3 usages
    private static AppDatabase INSTANCE;

    3 usages  1 implementation
    public abstract ContactDAO contactDAO();
```

03. GESTION DES RELATIONS ENTRE ENTITÉS



ONE-TO-ONE

Une relation **One-to-One** associe exactement un objet à un autre objet.

Exemple:

```
data class UserWithProfile(
    @Embedded val user: User,
    @Relation(
        parentColumn = "id",
        entityColumn = "userId"
    )
    val profile: UserProfile
)
```

Cette classe représente une relation où chaque utilisateur possède exactement un profil.

ONE-TO-MANY

Une relation **One-to-Many** permet à une entité d'être associée à plusieurs instances d'une autre entité.

Par exemple, un utilisateur peut avoir plusieurs publications, mais chaque publication appartient à un seul utilisateur.

```
data class UserWithPosts(
    @Embedded val user: User,
    @Relation(
        parentColumn = "id",
        entityColumn = "userId"
    )
    val posts: List<Post>
)
```

MANY-TO-MANY

Une relation **Many-to-Many** permet à plusieurs instances d'une entité d'être associées à plusieurs instances d'une autre entité.

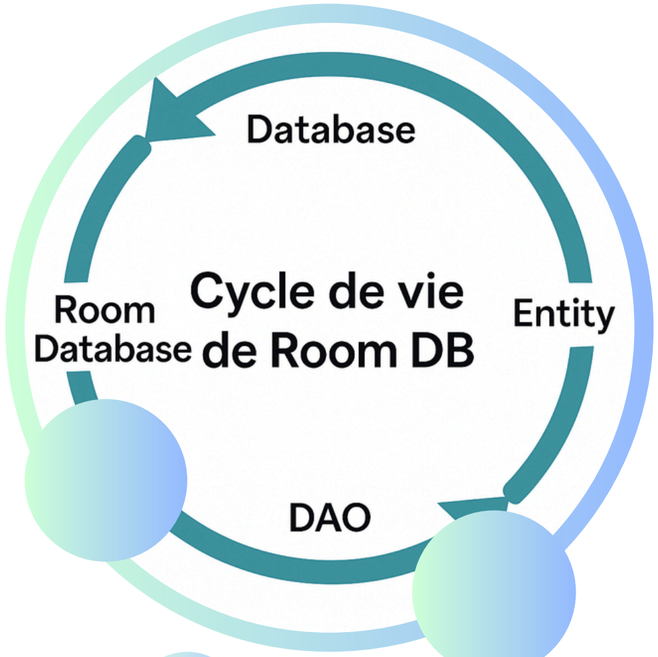
Par exemple, un étudiant peut suivre plusieurs cours et un cours peut être associé à plusieurs étudiants via une table de jointure.

```
@Entity(primaryKeys = ["studentId", "courseId"])
data class StudentCourseCrossRef(
    val studentId: Int,
    val courseId: Int
)

data class StudentWithCourses(
    @Embedded val student: Student,
    @Relation(
        parentColumn = "id",
        entityColumn = "id",
        associateBy = Junction(StudentCourseCrossRef::class)
    )
    val courses: List<Course>
)
```

04. CYCLE DE VIE DE ROOM DB

1. Définir les entités (tables).
2. Créer les DAO (accès aux données).
3. Créer la classe RoomDatabase.
4. Initialiser la base dans l’application.
5. Appeler les DAO pour insérer, modifier ou lire des données.
6. Observer les données avec LiveData ou Flow (Optionnel) .



05. MIGRATIONS DE BASES DE DONNÉES

Les applications évoluent et nos schémas de base de données aussi. Les migrations permettent de mettre à jour notre base sans perdre les données des utilisateurs.

```
@Database(
    entities = [User::class],
    version = 2
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao

    companion object {
        val MIGRATION_1_2 = object : Migration(1, 2) {
            override fun migrate(database: SupportSQLiteDatabase) {
                database.execSQL("ALTER TABLE users ADD COLUMN age INTEGER")
            }
        }
    }
}

Room.databaseBuilder(context, AppDatabase::class.java, "database-name")
    .addMigrations(AppDatabase.MIGRATION_1_2)
    .build()
```

06. GESTION DES THREADS AVEC ROOM

Le Main Thread est responsable de l'affichage et des interactions de l'utilisateur.
Effectuer des tâches longues comme des accès à la base de données sur ce thread peut bloquer l'interface et nuire à l'expérience utilisateur.
Pour éviter cela, Room interdit les requêtes directes sur le thread principal.

En cas de non-respect, tu risques une erreur comme :
android.os.NetworkOnMainThreadException

Room propose plusieurs moyens pour exécuter les requêtes en arrière-plan, de manière sûre et moderne.

06. GESTION DES THREADS AVEC ROOM

Coroutines (recommandé)

Utilise le mot-clé suspend pour marquer une fonction comme exécutable de façon asynchrone.

Flow (réactif et moderne)

Permet d’écouter les changements en base de façon continue .

LiveData (observateur d’ancienne génération)

Recommandé si tu utilises encore l’architecture ViewModel + LiveData.

Executors (option manuelle)

On peut aussi gérer les threads manuellement (option moins pratique).

CONFIGURATION

Pour utiliser Room dans une application, on ajoute les dépendances suivantes au fichier **build.gradle**.

```
// Dépendances Room
implementation(libs.room.runtime)
annotationProcessor(libs.room.compiler)
```

07. AVANTAGES ET INCONVÉNIENTS

Avantages

- Évite d’écrire du SQL pur (grâce à l’annotation @Query)
- Génère automatiquement du code pour accéder à la base.
- Vérification des requêtes SQL à la compilation.
- Intégration facile avec LiveData, Flow, ViewModel, etc.

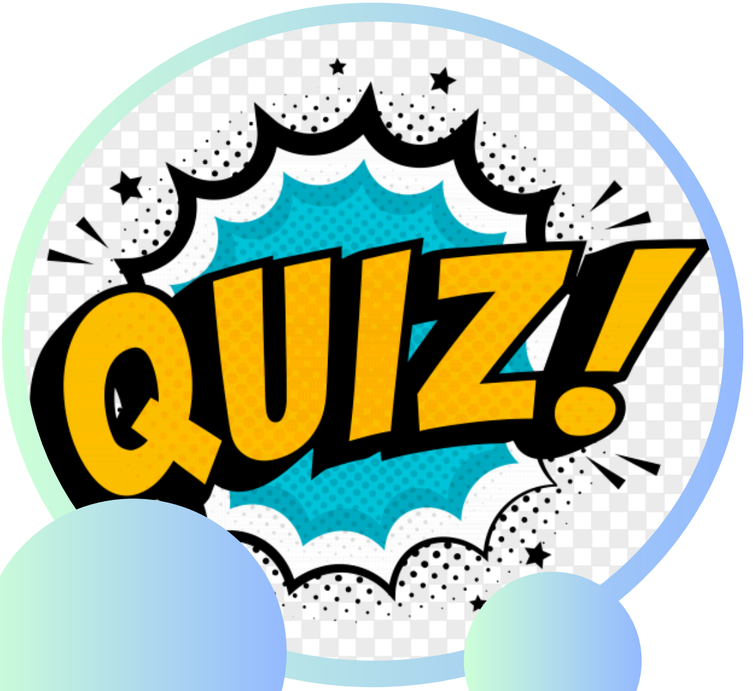
Inconvénients

- Moins souple que du SQL brut
- Complexe pour les relations complexes
- Nécessite une bonne organisation (DAO)

09. CONCLUSION

Room Database est un moyen moderne et efficace de gérer les données locales dans les applications Android. En utilisant des entités, des DAO et une classe de base de données, nous pouvons simplifier le processus de gestion des bases de données tout en garantissant la fiabilité et la performance.

10. QUIZ



MERCI POUR VOTRE ATTENTION