Alex LABUZ

Gérer ses bases SQLite avec Android Room

Introduction

Android Room est une librairie proposé par Google depuis 2017, elle permet d'ajouter une couche d'abstraction sur la manière de gérer les bases de données SQLite de l'application.

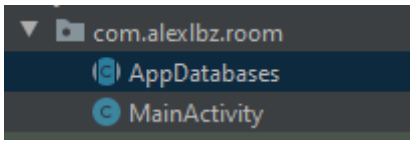
Préparation

Pour commencer il faut importer la librairie suivante :

```
implementation "androidx.room:room-runtime:2.2.5"
annotationProcessor "androidx.room:room-compiler:2.2.5"
```

A ajouter dans build.gradle :app

Maintenant créez une classe abstraite **AppDatabases**



Nous allons instancier cette classe **AppDatabases** pour accéder et manipuler notre DB SQLite.

Cependant il faut absolument que cette classe sois instanciée qu'une seule fois dans toute notre application. Nous allons pour cela mettre en place un singleton.

Mettez le contenu suivant :

```
@Database(entities = {}, version = 1)
public abstract class AppDatabases extends RoomDatabase {

    private static final String DB_NAME = "nom_de_db";
    private static AppDatabases instance;

    public static synchronized AppDatabases getInstance(Context context){
        if(instance == null){
            instance = Room.databaseBuilder(context.getApplicationContext(),
AppDatabases.class, DB_NAME)
                    .allowMainThreadQueries()
                    .build();
        }

        return instance;
    }

}
```

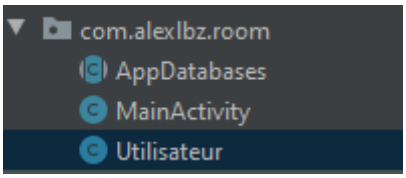
Au-dessus de la classe on met une annotation @Database, celle-ci fait partie de Android Room et permet d'indiquer que cette classe permet le paramétrage et l'accès de la DB.

- Le paramètre **entities** est un tableau qui contiendra toutes nos entités (tables).
- Le paramètre **version** et la version de l'état de notre DB.

Pour instancier qu'une seule fois **AppDatabases** on utilise une fonction asynchrone *getInstance()* et on vérifie avec cette fonction si **AppDatabses** a déjà était instanciée. Si ce n'est le cas on l'instancie avec **Room.databaseBuilder** sinon on ne fait rien.

Créer une entité

Créez une classe Utilisateur



Et mettez à l'intérieur le contenu suivant :

```java
@Entity
public class Utilisateur {

    @PrimaryKey
    private Integer id;

    @ColumnInfo(name = "prenom")
    private String prenom;

    @ColumnInfo(name = "age")
    private Integer age;

    public Utilisateur(String prenom, Integer age) {
        this.prenom = prenom;
        this.age = age;
    }

}
```

On met cette fois ci une annotation @Entity permettant d'indiquer à Android Room que cette classe correspond à une table de notre base de données.

/!\ N'oublier de générer les accesseurs et mutateurs !

En-dessous chaque attribut est également annoté.

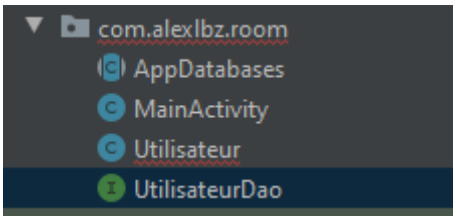
- @PrimaryKey permet d'indiquer la clé primaire (Et oui !)
- @ColumnInfo permet d'indiquer les autres champs, elle prend comme paramètre son nom. Il y a aussi d'autres paramètres comme defaultValue pour définir une valeur par défaut.

Ensuite on créer le constructeur. *Il n'y a pas d'id dans le constructeur car celui-ci est auto-incrémenté par SQLite, mais on peut le mettre quand même pour le faire sois même.*

Créer un DAO

Le dao est une interface qui va nous fournir les outils pour manipuler l'entité qu'on vient créer.

Créez une interface **UtilisateurDao**.




```java
@Dao
public interface UtilisateurDao {

    @Query("SELECT * FROM Utilisateur")
    List<Utilisateur> getAll();

    @Query("SELECT * FROM Utilisateur WHERE age = :age")
    List<Utilisateur> getByAge(Integer age);

    @Insert
    void insert(Utilisateur utilisateur);

    @Delete
    void delete(Utilisateur utilisateur);

}
```

Notre interface **UtilisateurDao** doit être annotée avec @Dao.

On crée à l'intérieur toutes les méthodes (SELECT, INSERT, DELETE …) pour manipuler la DB.

Avec les annotations (@Insert, @Delete …) on a même pas besoin de rédiger de requête SQL, il suffit de créer une méthode avec en paramètre la donnée à manipuler.

Pour récupérer les données on utilise @Query et cette fois ci, on rédige la requête SQL pour récupérer ce qu'il nous faut. Ensuite on stocke les données dans une List<> similaire aux ArrayList<>.

Mise à jour AppDatabases

Il faut ensuite rajouter notre entité **Utilisateur** dans AppDatabases.

```
@Database(entities = {Utilisateur.class}, version = 1)
public abstract class AppDatabases extends RoomDatabase {

    public static synchronized AppDatabases getInstance(Context context){
        ...
    }

    public abstract UtilisateurDao utilisateurDao();

}
```

Manipuler et afficher les données

Dans votre activité instanciez **AppDatabases** de cette manière.

```
private AppDatabases db;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    this.db = AppDatabases.getInstance(this);
}
```

Faisons nos tests, essayez ce code :

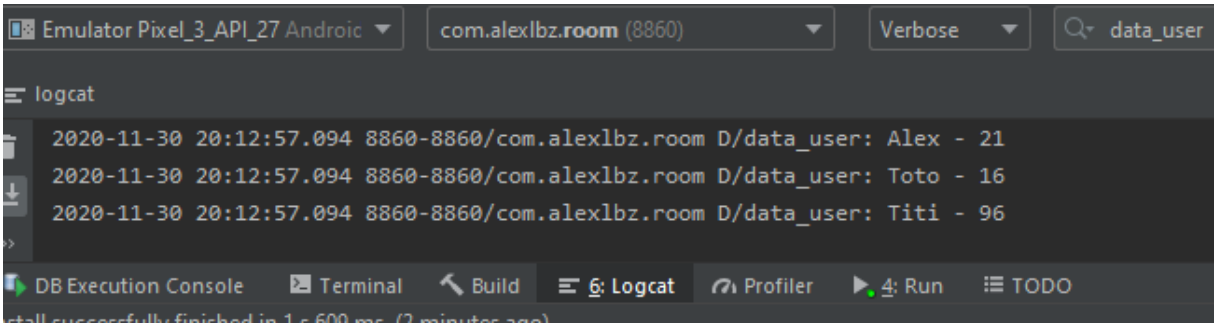
```
this.db = AppDatabases.getInstance(this);

this.db.utilisateurDao().insert(new Utilisateur("Alex", 21));
this.db.utilisateurDao().insert(new Utilisateur("Toto", 16));
this.db.utilisateurDao().insert(new Utilisateur("Titi", 96));
```

Nous faisons 3 insertions dans la table utilisateur. Lancer l'application et ensuite remplacer le code par celui-ci :

```
this.db = AppDatabases.getInstance(this);

List<Utilisateur> list = this.db.utilisateurDao().getAll();
for(Utilisateur u : list){
    Log.d("data_user", u.getPrenom() + " - " + u.getAge());
}
```

Ce code récupère les 3 utilisateurs enregistrés précédemment et les affiches en log.

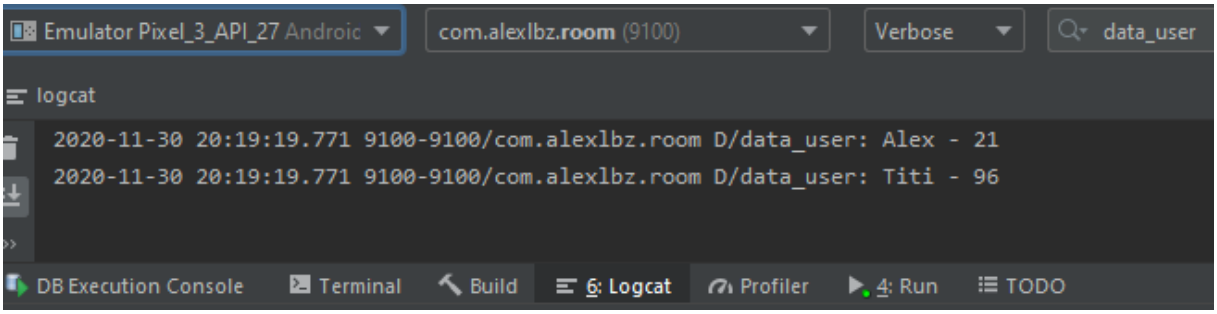
Relancez l'application avec le nouveau code. Dans Android Studio ouvrez le menu Logcat et tapez « data_user » dans le champ de recherche et vous devez voir les 3 utilisateurs



Ajouter maintenant cette ligne :

```
this.db = AppDatabases.getInstance(this);

List<Utilisateur> list = this.db.utilisateurDao().getAll();
this.db.utilisateurDao().delete(list.get(1));
for(Utilisateur u : list){
    Log.d("data_user", u.getPrenom() + " - " + u.getAge());
}
```

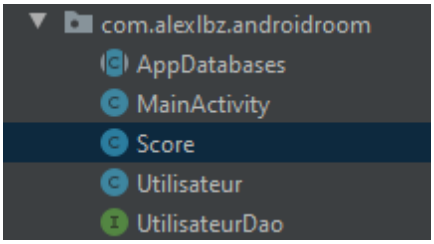
Et relancer l'application. Mettez cette ligne en commentaire, relancez encore une fois l'application. Regarder le logcat et l'utilisateur qui occupé l'emplacement 1 à disparu :



Il nous reste donc 2 utilisateurs

Créer des relations

Nous allons créer une nouvelle entité **Score**



Dans notre exemple on va dire qu'un score appartient à un utilisateur, copier le code ci-dessous :

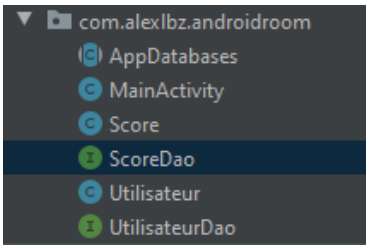
```
@Entity(foreignKeys = @ForeignKey(entity = Utilisateur.class,
        parentColumns = "id",
        childColumns = "idUtilisateur"))
public class Score {

    @PrimaryKey
    private Integer id;

    @ColumnInfo(name = "score")
    private Integer score;

    @ColumnInfo(name = "idUtilisateur")
    private Integer idUtilisateur;

    public Score(Integer score, Integer idUtilisateur) {
        this.score = score;
        this.idUtilisateur = idUtilisateur;
    }

}
```

/!\ N'oublier de générer les accesseurs et mutateurs !

Il s'agit d'une entité très simple, il y a cependant des choses nouvelles :

- Dans l'annotation @Entity nous précisons la table qui doit être jointe (la table Utilisateur), la clé primaire de cette table et la clé étrangère de notre table Score.
- Un attribut **idUtilisateur** qui est notre clé étrangère.

Créez maintenant l'interface **ScoreDao**



Mettez le contenu suivant :

```
public interface ScoreDao {

    @Query("SELECT * FROM Score")
    List<Score> getAll();

    @Insert
    void insert(Score score);

}
```

Alex LABUZ

Modifier la classe **AppDatabases** comme ceci :

```
@Database(entities = {Utilisateur.class, Score.class}, version = 2)
public abstract class AppDatabases extends RoomDatabase {

    public static synchronized AppDatabases getInstance(Context context){
        ...
    }

    public abstract UtilisateurDao utilisateurDao();
    public abstract ScoreDao scoreDao();

}
```

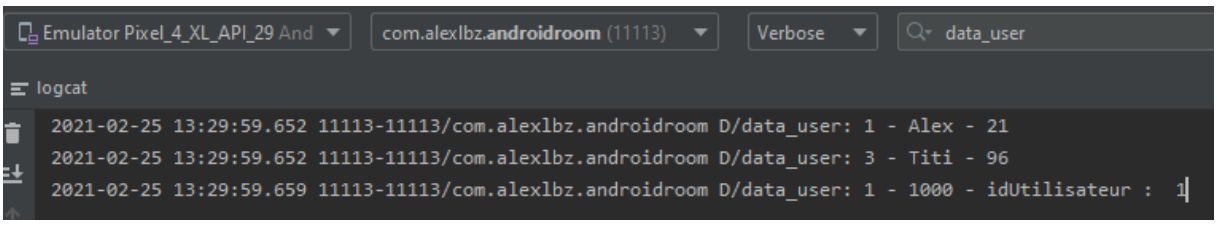
Ajoutons maintenant un score et attribuons-le à notre premier utilisateur :

```
this.db = AppDatabases.getInstance(this);

List<Utilisateur> list = this.db.utilisateurDao().getAll();
this.db.scoreDao().insert(new Score(1000, list.get(0).getId()));

for(Utilisateur u : list){
    Log.d("data_user", u.getId() + " - " + u.getPrenom() + " - " + u.getAge());
}

List<Score> listScore = this.db.scoreDao().getAll();
for(Score s : listScore){
    Log.d("data_user", s.getId() + " - " + s.getScore() + " - idUtilisateur :  " +
s.getIdUtilisateur());
}
```

On fait une insertion dans la table **Score** avec un score de 1000 et on récupère l'id du premier utilisateur qu'on vient indiquer dans le champ idUtilisateur de la table **Score**.

On affiche les id des utilisateurs et on fait une nouvelle boucle qui affiche la liste des **Scores**.

Relanczr l'application et vous devrez avoir un résultat comme celui-là :



https://developer.android.com/training/data-storage/room

https://stackoverflow.com/questions/50103232/using-singleton-within-the-android-room-library/50145026