



android

2 - SQLite & Room



Jordan Hiertz

Contact

hiertzjordan@gmail.com

jordan.hiertz@al-enterprise.com

Contenu

android 

- Stockage des données
- Bibliothèque logicielle SQLite
- Bibliothèque de persistance Room
- Programmation asynchrone

Stockage des données

Comment stocker des données dans une application Android

- Stockage spécifique à l'application
- Stockage partagé (fichiers à partager avec d'autres applications)
- Préférences
- Bases de données

Qu'est-ce qu'une base de données ?

Collection de données structurées qui peuvent être facilement consultées, recherchées et organisées et qui se composent de

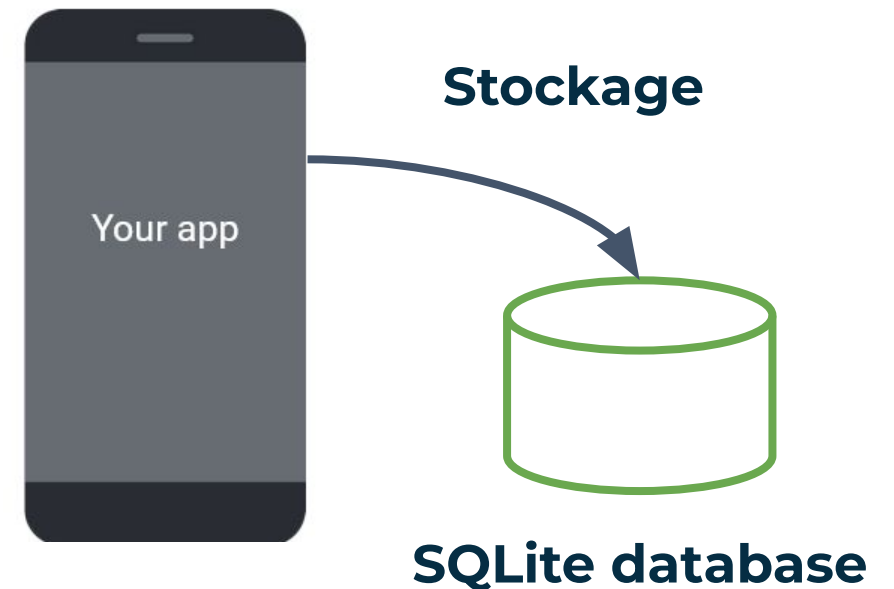
- Tableaux
- Lignes
- Colonnes

WORD_LIST_TABLE		
_id	word	definition
1	"alpha"	"first letter"
2	"beta"	"second letter"
3	"alpha"	"particle"

Structured Query Language (SQL)

Utiliser SQL pour accéder à une base de données relationnelle et la modifier.

- Créer de nouvelles tables
- Rechercher des données
- Insérer de nouvelles données
- Mettre à jour des données
- Supprimer des données



Exemple de commandes SQLite

Create `INSERT INTO colors VALUES ("red", "#FF0000");`

Read `SELECT * from colors;`

Update `UPDATE colors SET hex="#DD0000" WHERE name="red";`

Delete `DELETE FROM colors WHERE name = "red";`

Interagir directement avec une base de données

Pas de vérification à la compilation des requêtes SQL brutes

Nécessité d'écrire beaucoup de "*boilerplate code*" pour convertir des requêtes SQL en objets.

Quelques exemples d'utilisation

```
val queryAll = "SELECT * FROM WORD_LIST_TABLE"  
rawQuery(query, null)
```

```
val queryId = "SELECT word, definition FROM WORD_LIST_TABLE  
WHERE _id> ?"
```

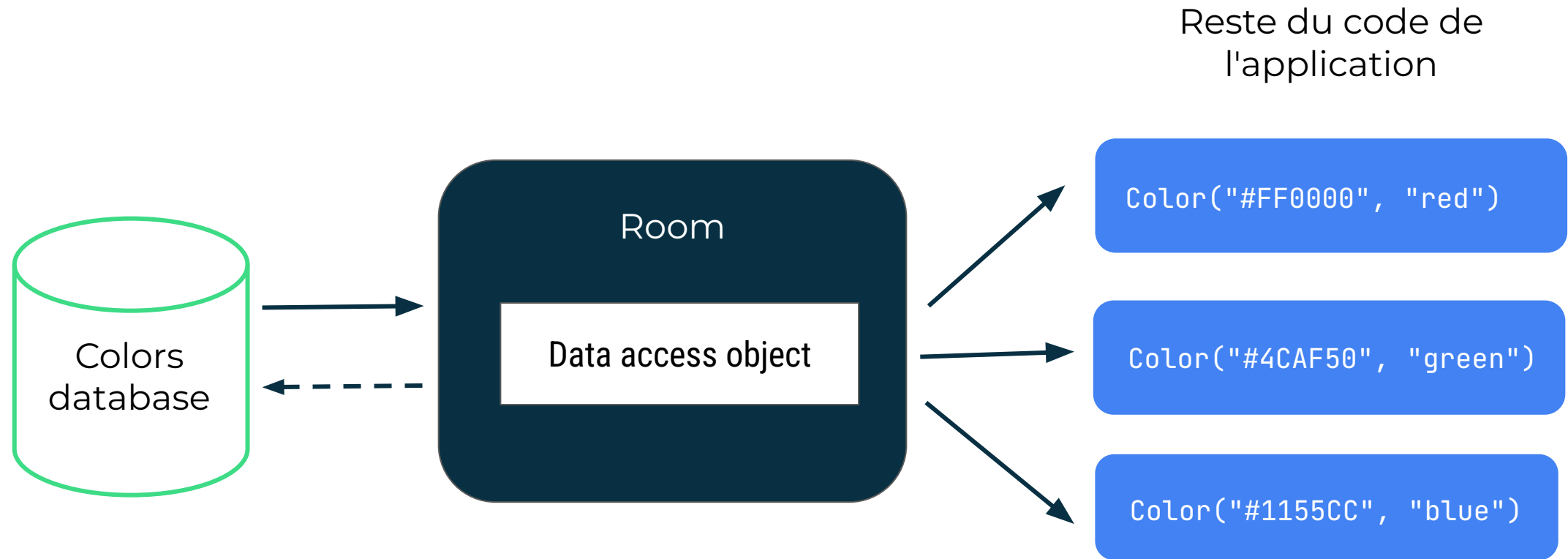
```
val selectionArgs = arrayOf("2")  
  
rawQuery(query, selectionArgs)
```

Bibliothèque de persistance Room

Ajouter les dépendances Gradle

```
dependencies {  
    implementation "androidx.room:room-runtime:$room_version"  
    kapt "androidx.room:room-compiler:$room_version"  
  
    // Kotlin Extensions and Coroutines support for Room  
    implementation "androidx.room:room-ktx:$room_version"  
  
    // Test helpers  
    testImplementation "androidx.room:room-testing:$room_version"  
}
```

Room



Room

- **Entity** Représente une table dans la base de données.
- **DAO** Contient les méthodes utilisées pour accéder à la base de données.
- **Database** Principal point d'accès aux données relationnelles persistantes de l'application.

Color class

```
data class Color {  
    val hex: String,  
    val name: String  
}
```

Annotations

Fournir des informations supplémentaires au compilateur

- **@Entity** marque la classe d'entité
- **@Dao** pour DAO
- **@Database** pour la base de données

Elles peuvent prendre des paramètres :

- **@Entity**(tableName = "colors")

Entity

Classe qui correspond à une table de base de données SQLite

- **@Entity**
- **@PrimaryKey**
- **@ColumnInfo**

Example Entity

```
@Entity(tableName = "colors")
data class Color {
    @PrimaryKey(autoGenerate = true) val _id: Int,
    @ColumnInfo(name = "hex_color") val hex: String,
    val name: String
}
```

colors
_id
hex_color
name

Data access object (DAO)

Travailler avec des classes DAO au lieu d'accéder directement à la base de données:

- Définir les interactions avec la base de données dans le DAO
- Déclarer le DAO comme une interface ou une classe abstraite
- Room créer l'implémentation au moment de la compilation
- Room vérifie toutes les requêtes

Exemple DAO

```
@Dao
interface ColorDao {
    @Query("SELECT * FROM colors")
    fun getAll(): Array<Color>

    @Insert
    fun insert(vararg color: Color)

    @Update
    fun update(color: Color)

    @Delete
    fun delete(color: Color)
}
```

Exemple Query

```
@Dao
interface ColorDao {

    @Query("SELECT * FROM colors")
    fun getAll(): Array<Color>

    @Query("SELECT * FROM colors WHERE name = :name")
    fun getColorByName(name: String): Color

    @Query("SELECT * FROM colors WHERE hex_color = :hex")
    fun getColorByHex(hex: String): Color
}
```

Créer la base de données Room

- Annoter la classe avec `@Database` et inclure la liste des entités :
`@Database(entities = [Color::class], version = 1)`
- Déclarez une classe abstraite qui hérite RoomDatabase :
`abstract class ColorDatabase : RoomDatabase() {`
 - Déclarez une méthode abstraite sans argument qui renvoie le DAO :
`abstract fun colorDao(): ColorDao`

Exemple base de données Room

```
@Database(entities = [Color::class], version = 1)
abstract class ColorDatabase : RoomDatabase() {
    abstract fun colorDao(): ColorDao
    companion object {
        @Volatile
        private var INSTANCE: ColorDatabase? = null
        fun getInstance(context: Context): ColorDatabase {
            ...
        }
    }
    ...
}
```

Exemple base de données Room

```
fun getInstance(context: Context): ColorDatabase {  
    return INSTANCE ?: synchronized(this) {  
        INSTANCE ?: Room.databaseBuilder(  
            context.applicationContext,  
            ColorDatabase::class.java, "color_database"  
        )  
        .fallbackToDestructiveMigration()  
        .build()  
        .also { INSTANCE = it }  
    }  
}
```


Récupérer et utiliser un DAO

Récupère le DAO de la base de données :

```
val colorDao = ColorDatabase.getInstance(application).colorDao()
```

Créer une nouvelle couleur et utiliser le DAO pour l'insérer dans la base de données :

```
val newColor = Color(hex = "#6200EE", name = "purple")  
colorDao.insert(newColor)
```

Programmation asynchrone

Tâches de longue durée

- Télécharger des informations
- Synchronisation avec un serveur
- Écrire dans un fichier
- Calculs lourds
- Lire ou écrire dans une base de données

Nécessité d'une programmation asynchrone

- Temps limité pour accomplir les tâches et rester réactif
- Équilibre avec la nécessité d'exécuter des tâches de longue durée
- Contrôle de la manière et du lieu d'exécution des tâches

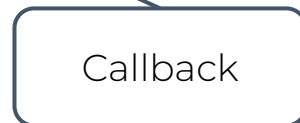
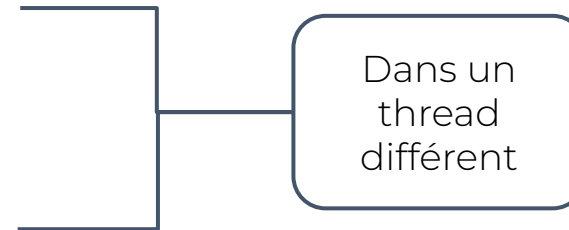
Programmation asynchrone dans Android

- Thread
- Callbacks
- Coroutines

Exemple Room asynchrone

```
val colorDao = ColorDatabase.getInstance(application).colorDao()
val executorService = Executors.newSingleThreadExecutor()
```

```
fun getAllColors(onSuccess: (List<Color>) → Unit) {
    executorService.submit(() → {
        val colors = colorDao.getAll()
        onSuccess(colors)
    })
}
```




Exemple Room asynchrone

```
override fun onCreateView() {  
    super.onCreateView()  
  
    repositoryInstance.getAllColors(onSuccess = { colors →  
        runOnUiThread {  
            // Update view with data  
        }  
    })  
}
```

Exemple Room coroutines

```
val colorDao = ColorDatabase.getInstance(application).colorDao()
```



```
suspend fun getAllColors() : List<Color> {  
    return colorDao.getAll()  
}
```


Exemple Room coroutines

```
override fun onCreateView() {  
    super.onCreateView()  
  
    lifecycleScope.launch {  
        val colors = repositoryInstance.getAllColors()  
        // Update view with data  
    }  
}
```

Conclusion

En résumé

- Créer et configurer une base de données à l'aide de la bibliothèque Room
- Utiliser la programmation asynchrone pour interroger une base de données ou exécuté des opérations lourdes.