



# Data Storage: SQLite, Room, ViewModel, LiveData and Observer

## Sensor Based Mobile Applications

Patrick Ausderau, Ulla Sederlöf, Jarkko Vuori

Helsinki Metropolia University of Applied Science

2022

# Outline

Room SQLite Database

LiveData, ViewModel, Observer

Lab

For more info:

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

<https://www.sqlite.org/>

[https:](https://developer.android.com/topic/libraries/architecture/livedata)

[//developer.android.com/topic/libraries/architecture/livedata](https://developer.android.com/topic/libraries/architecture/livedata)

<https://developer.android.com/jetpack/guide>

<https://developer.android.com/jetpack/compose/lists>

# Room SQLite Database

- ▶ Support for creating and using local SQLite databases on the device
  - ▶ A database is private to its application
  - ▶ Databases are stored on the device (or emulator) in `/data/data/<package>/databases`
- ▶ Room provides an abstraction layer over SQLite and consist of 3 major components
  - ▶ `@Database` extends `RoomDatabase`, provide access to the database
  - ▶ `@Dao` contains the methods used for accessing the database (CRUD operations)
  - ▶ `@Entity` represents a table (and relations) within the database

# Dependencies

- ▶ in project build gradle, add `google()` in the list of repositories (should already be there by default)
- ▶ in module app build gradle, add in dependencies:

```
1  apply plugin: "kotlin-kapt"
2  dependencies {
3      def roomVersion = "2.4.3"
4      implementation "androidx.room:room-runtime:$roomVersion"
5      annotationProcessor
6          "androidx.room:room-compiler:$roomVersion"
7
8      // To use Kotlin annotation processing tool (kapt)
9      kapt "androidx.room:room-compiler:$roomVersion"
10     // Kotlin Extensions and Coroutines support for Room
11     implementation "androidx.room:room-ktx:$roomVersion"
12     //...
13 }
```

## @Entity

- ▶ data class with @Entity annotation

```
1 @Entity
2 data class User(
3     @PrimaryKey(autoGenerate = true)
4     val uid: Long,
5     val firstname: String,
6     val lastname: String) {
7     //constructor, getter and setter are implicit :)
8     override fun toString() = "$firstname $lastname
9         ($uid)"
10 }
```

## @Entity with relation

```
1  @Entity(foreignKeys = [ForeignKey(
2      entity = User::class,
3      onDelete = CASCADE,
4      parentColumns = ["uid"],
5      childColumns = ["user"])]))
6  data class ContactInfo(
7      val user: Long,
8      val type: String, //e.g. phone, email, fb, twitter,...
9      @PrimaryKey
10     val value: String)
```

```
1  class UserContact {
2      @Embedded
3      var user: User? = null
4      @Relation(parentColumn = "uid", entityColumn = "user")
5      var contacts: List<ContactInfo>? = null
6  }
```

# @Dao

```
1  @Dao
2  interface UserDao {
3      @Query("SELECT * FROM user")
4      fun getAll(): LiveData<List<User>>
5
6      @Query("SELECT * FROM user WHERE user.uid = :userid")
7      // the @Relation do the INNER JOIN for you ;)
8      fun getUserWithContacts(userid: Long): LiveData<UserContact>
9
10     @Insert(onConflict = OnConflictStrategy.REPLACE)
11     suspend fun insert(user: User): Long
12
13     @Update
14     suspend fun update(user: User)
15
16     @Delete
17     suspend fun delete(user: User)
18 }
19
20 @Dao
21 interface ContactInfoDao { /* ... */ }
```

# @Database

```
1  @Database(entities = [(User::class), (ContactInfo::class)],
    version = 1)
2  abstract class UserDB: RoomDatabase() {
3      abstract fun userDao(): UserDao
4      abstract fun contactDao(): ContactInfoDao
5
6      companion object{
7          private var sInstance: UserDB? = null
8          @Synchronized
9          fun get(context: Context): UserDB {
10              if (sInstance == null) {
11                  sInstance =
12                      Room.databaseBuilder(context.applicationContext,
13                          UserDB::class.java, "users.db").build()
14              }
15              return sInstance!!
16          }
17      }
18  }
```



# LiveData and ViewModel

- ▶ LiveData is an observable data holder class
- ▶ LiveData is lifecycle-aware, it will only updates app component observers that are in an active lifecycle state
- ▶ ViewModel is a class that is responsible for preparing and managing the data for a Composable function or an Activity or a Fragment.

# LiveData and ViewModel

- ▶ in app build config, add dependencies:

```
1  def lifecycle_version = "2.5.1"
2  // ViewModel
3  implementation
4      "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
5  // ViewModel utilities for Compose
6  implementation
7      "androidx.lifecycle:lifecycle-viewmodel-compose:$lifecycle_version"
8  // LiveData
9  implementation
10     "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
11  implementation
12     "androidx.compose.runtime:runtime-livedata:$compose_version"
```

# LiveData and ViewModel

```
1  class UserViewModel(application: Application):  
    AndroidViewModel(application) {  
2      private val userDB = UserDB.get(application)  
3  
4      fun getAll(): LiveData<List<User>> =  
        userDB.userDao().getAll()  
5  
6      fun insert(user: User) {  
7          viewModelScope.launch {  
8              userDB.userDao().add(user)  
9          }  
10     }  
11  
12     /* fun update, delete, getDetails,... */  
13 }
```

# LiveData and ViewModel

```
1  class MainActivity : ComponentActivity() {  
2      companion object{  
3          private lateinit var userViewModel: UserViewModel  
4      }  
5  
6      override fun onCreate(savedInstanceState: Bundle?) {  
7          super.onCreate(savedInstanceState)  
8          userViewModel = UserViewModel(application)  
9          setContent {  
10             MainAppNav(userViewModel)  
11         }  
12     }  
13 }
```

# LiveData and ViewModel

```
1  @Composable
2  fun InsertUser(userViewModel: UserViewModel) {
3      // candidate for mutableStateListOf
4      var fname by remember { mutableStateOf("") }
5      var lname by remember { mutableStateOf("") }
6      Column {
7          TextField(value = fname, label = {
8              Text(stringResource(R.string.fname)) },
9              onChange = { fname = it })
10         TextField(value = lname, label = {
11             Text(stringResource(R.string.lname)) },
12             onChange = { lname = it })
13         Button(onClick = {
14             userViewModel.insert(User(0, fname, lname))
15         }) {
16             Text(stringResource(R.string.insert))
17         }
18     }
19 }
```

# LiveData and ViewModel

```
1  @Composable
2  fun ListUsers(userViewModel: UserViewModel, navController:
   NavController) {
3      val userList =
         userViewModel.getAll().observeAsState(listOf())
4      LazyColumn {
5          item {
6              Row {
7                  Text(stringResource(R.string.header))
8              }
9          }
10         items(userList.value) {
11             Text("User: $it", Modifier.clickable {
12                 navController.navigate("details/${it.uid}")
13             })
14         }
15     }
16 }
```

# LiveData and ViewModel

```
1  @Composable
2  fun MainAppNav(userViewModel: UserViewModel) {
3      val navController = rememberNavController()
4      NavHost(navController, startDestination = "main") {
5          composable("main") {
6              Column {
7                  InsertUser(userViewModel)
8                  ListUsers(userViewModel, navController)
9              }
10         }
11         composable("details/{userId}") {
12             val id = it.arguments?.getString("userId")?.toLong() ?: 0
13             DetailView(userViewModel, id, navController)
14         }
15     }
16 }
```

Build gradle and details about navigation: <https://developer.android.com/jetpack/compose/navigation>

## Lab\_w1\_d5 Room and LiveData

- ▶ Create an app that will use a simple Room SQLite database (at least two related tables). E.g.
  - ▶ Ice Hockey teams (name/created year) and their players (name/position)
  - ▶ Movies (title/year/director) and their actors (name/role)
  - ▶ Recipes (name/country of origin) and their ingredients (name/quantity)
  - ▶ Animals Families (English/Latin name) and their species (English/Latin name/area)
  - ▶ ...
- ▶ Create two views to insert data (one for each table) and LazyColumn to list the content (select from the two related tables).

Note: start easy with one table.

Hint: when modifying the database (entities, dao,...), before redeploying your app to emulator/debugging device, go to phone settings ⇒ App & notification, select your app ⇒ Storage & cache and clear both cache and storage (so you avoid database modification exception).



# Lab\_w1\_d5 Room and LiveData

