Room
SQLite DB for Android

room-compose (on GitHub)

Step 1: Add Dependencies

```
plugins {         // In top-level build.gradle.kts
  id("androidx.room") version "2.8.1" apply false
  id("com.google.devtools.ksp") version "2.0.21-1.0.25 apply false
}
```

KSP (2.0.21) version should match Kotlin version

```
plugins {                          // In App build.gradle.kts
  id("androidx.room")
  id("com.google.devtools.ksp")
}
room {
  schemaDirectory("$projectDir/cs357-schemas")
}
dependencies {
  val room_version = "2.8.1"
  implementation("androidx.room:room-runtime:$room_version")
  ksp("androidx.room:room-compiler:$room_version")
  implementation("androidx.room:room-ktx:$room_version")
}
```


Potential Build Error (1)

Possible build error when updating the libraries to a newer version

```
java.lang.AbstractMethodError: Receiver class androidx.room.migration.bundle.FieldBundle$$serializer does
    not define or inherit an implementation of the resolved method
    'abstract kotlinx.serialization.KSerializer[]typeParametersSerializers()' of interface
    kotlinx.serialization.internal.GeneratedSerializer.

    more error messages here
```

Solution:

1. Delete the `schemas` and `build` directories and rebuild
2. Wipe device data (**end of the slides**)
3. Build ⇒ Clean Project
4. File ⇒ Invalidate Caches

Potential Build Error (2)

Possible build error caused by RoomDB library

```
Using kotlin.sourceSets DSL to add Kotlin sources is not allowed with built-in Kotlin.
```

Solution: Add the following flag in `gradle.properties`

```
android.disallowKotlinSourceSets=false
```


Step 2: Declare Data Class(es)

- *One class per SQL Table*
- *Name of SQL tables and columns are inferred from your class declaration*

```
@Entity(tableName = "guests")
data class Guest(
  @PrimaryKey(autoGenerate = true) val gid: Int,
  val firstName: String,
  val lastName: String)
```

***guests**: table name in SQL table*
***Guest**: class name in Kotlin code*

```
@Entity
data class PartyTable(
  @PrimaryKey(autoGenerate = true) val tid: Int,
  @ColumnInfo(name = "seats") val seatingCapacity: Int)
```

***seats**: column name in SQL table*
***seatingCapacity**: field name in Kotlin code*

Step 5a: Create App

```
import android.app.Application
import androidx.room.Room
import your_package_name_here.MyDatabase

class MyRoomApplication: Application() {
  lateinit var myDB: MyDatabase
  override fun onCreate() {
    super.onCreate()
    myDB = Room.databaseBuilder(applicationContext,
                                MyDatabase::class.java, name = “my-db”).build()
  }
}
```

```
package your_package_name_here

@Database
abstract class MyDatabase: RoomDatabase() {
  // more code here
}
```


Step 3: Define Data Access Object (SQL operators)

```
@Dao // Data Access Objects
interface PartyDao {
  @Insert
  suspend fun addGuest(g: Guest)

  @Insert
  suspend fun youCanNameThisFunctionAnything(_: PartyTable)

  @Update // Update a record with matching primary key
  suspend fun modifyTable(_: PartyTable)

  @Delete // Delete a record with matching primary key
  suspend fun removeIt(g: Guest)
}
```

Step 3b: Define Interface for SQL Operations

```
@Dao // Data Access Objects
interface PartyDao {
  @Query("DROP * FROM guests")
  suspend fun guestCleanup()

  @Query("SELECT * FROM partytable seats >= :seats")
  suspend fun searchTable(seats: Int): List<PartyTable>


  @Query("SELECT * FROM guests")
  suspend fun getAllGuests(): List<Guest>
  // fun getAllGuests(): Flow<List<Guest>>   // Without suspend
}
```

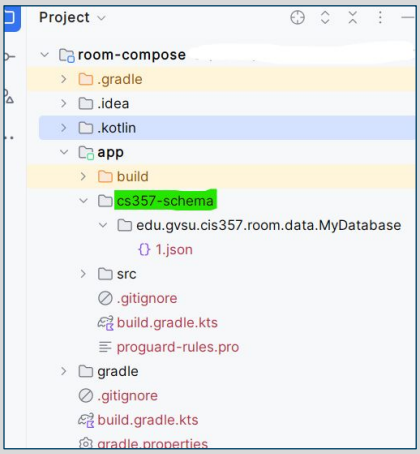

List<Guest> vs Flow<List<Guest>>

`@Query("SELECT * FROM guests")` `suspend fun getAll(): List<Guest>`	Query result is NOT updated on table updates
`@Query("SELECT * FROM guests")` `fun getAll(): Flow<List<Guest>>`	Query result is *automatically updated* when the table updates

Step 4: Define The DataBase Class

```
@Dao // Data Access Objects
interface PartyDao { /* your access functions here */}
```

```
@Database(entities = [Guest::class, PartyTable::class],
          version = 1,
          exportSchema = true)
abstract class MyDatabase: RoomDatabase() {
  abstract fun getInstance(): PartyDao
}
```



Enable `exportSchema` *if you plan for future DB migration*

Step 5b: Update AndroidManifest.xml

```
<?xml version="1.0" encodiing="utf-8">
<manifest _______/>
  <application
     android:name="MyRoomApplication"
     _____>
    <activity android:name=".MainActivity">
      <!-- more contents here --->
    </activity>
  </application>
</manifest>
```

Step 6: Access DB in ViewModel Coroutine

```
class MyViewModel (app:Application): AndroidViewModel(app) {
  private val myDB: PartyDao = (app as MyRoomApplication).myDB.getInstance();
  lateinit var guests: Flow<List<Guest>>
    private set   // prevent view from directly altering guests

  init {
    viewModelScope.launch(Dispatchers.IO) {
      guests = myDB.getAllGuests()
    }
  }

  fun addNewGuest(firstName: String, /* args */) {
    viewModelScope.launch(Dispatchers.IO) {
      val g = Guest(firstName = firstName, lastName = "__")
      myDB.addGuest(g) // this automatically updates the Flow<xxxxx>
    }
  }
}
```

```
@Dao // Data Access Objects
interface PartyDao {
  @Insert
  suspend fun addGuest(_: Guest)

  @Query("_____")
  fun getAllGuests(): Flow<List<Guest>>
}
```


Alt 6: Altering Query Results in ViewModel

```
class MyViewModel (app:Application): AndroidViewModel(app) {
  private val myDB: PartyDao = (app as MyRoomApplication).myDB.getInstance();
  private var _guests = MutableStateFlow<List<Guest>>(emptyList())
  val guests = _guests.asStateFlow()
  init {
    viewModelScope.launch(Dispatchers.IO) {
      myDB.getAllGuests().collect { gx in
        withContext(Dispatchers.Main) { _guests.value = gx }
      }
    }
  }

  fun sortGuestList() {
      _guest.update { it.sortedBy { it.firstName } }
  }
}
```

```
@Dao // Data Access Objects
interface PartyDao {
  @Insert
  suspend fun addGuest(_: Guest)

  @Query("_____")
  fun getAllGuests(): Flow<List<Guest>>
}
```

Step 7: Wiring Up with View

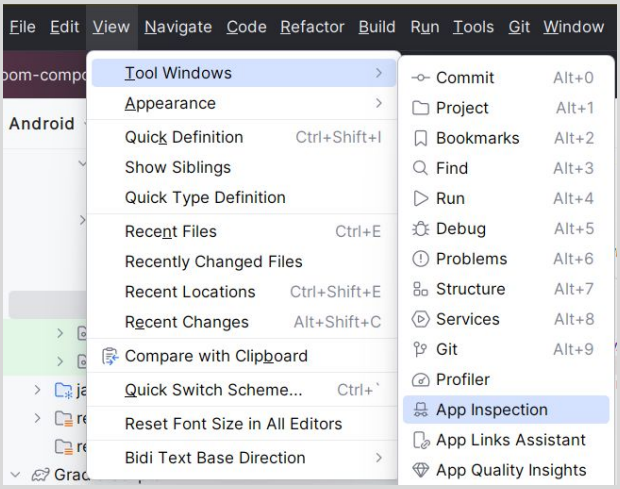
```
@Composable
fun SomeFunction (viewModel: MyViewModel) {
  val guests by viewModel.guests.collectsState(emptyList())
  Column {
    Button(onClick = { viewModel.addNewGuest(“Ray”) }) {
      Text(“Add Guest”)
    }
    LazyColumn {
      items(guests) {
        Text(“${it.firstName} ${it.lastName}”)
      }
    }
  }
}
```

```
class MyViewModel: _____ {
  lateinit var guests: Flow<List<Guest>>
  private val myDB: PartyDao = ______,getInstance()
  fun addNewGuest(firstName: String) {
    val g =
  }
}
```


Debugging Tools

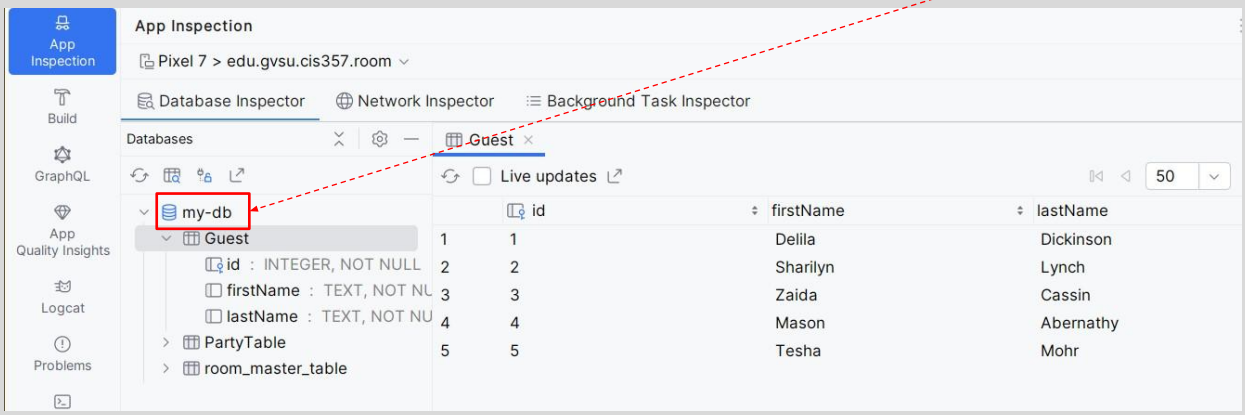
DB Inspection

View ⇒ Tool WIndows ⇒ App Inspection

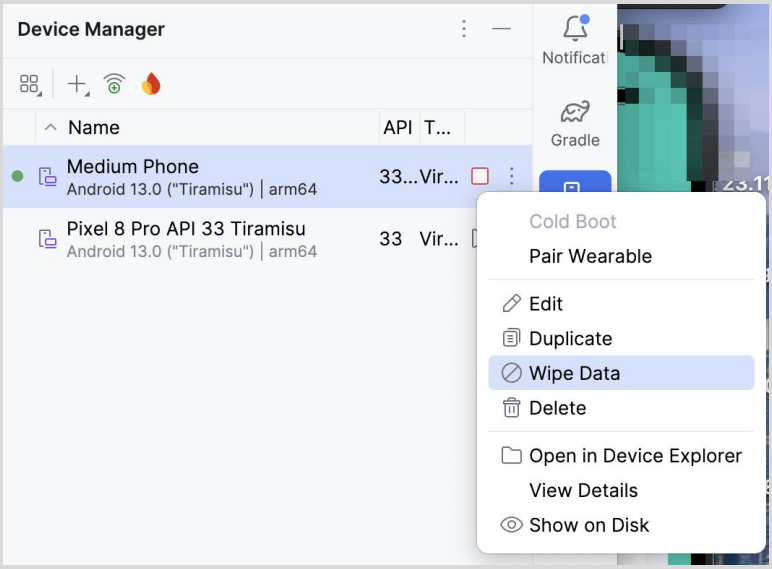


Android Studio: App Inspector

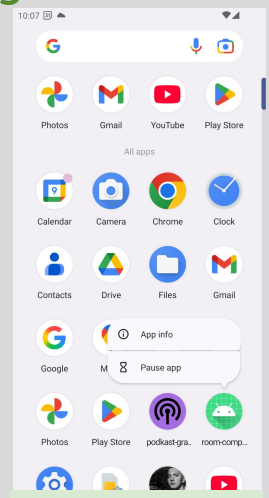
```
Room.databaseBuilder(applicationContext,  MyDatabase::class.java, name = "my-db")
```



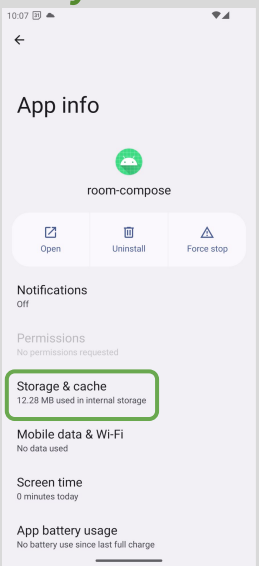
Android Studio: Wipe Data of AVD

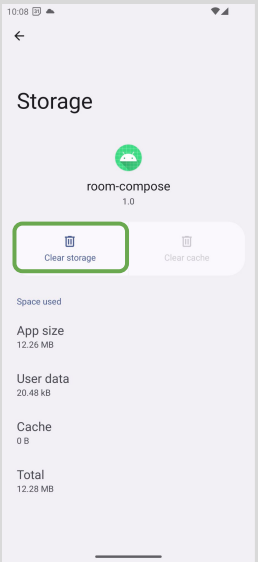


Wiping Data on a Physical Device



#1: Long Press on the app icon ⇒ App Info





Advanced Topics

Composite Primary Keys

```
@Entity(primaryKeys = [“courseCode”, “sectionNumber”])
data class Course(
  val courseCode: String,
  val sectionNumber: String.
  val courseTitle: String,
  val credit: Int)
```

Using ForeignKey

```
@Entity
data class ParentTable(
  @PrimaryKey pk_in_parent: Int,
  // other column definitions
}
```

```
@Entity(foreignKeys = [ForeignKey(
  entity = ParentTable::class,
  parentColumns = [“pk_in_parent”].
  childColumns = [“pk_from_parent”],
  onDelete = CASCADE)]
data class ChildTable(
  @PrimaryKey prima: Int,
  pk_from_parent: Int,    // foreign key
  // other column definitions
}
```

CASCADE: deleting a row in `ParentTable` will also delete matching rows in the `ChildTable`

Schema Update: DB Migration

```
// Data class after the version update.
@Entity data class PartyTable(
  @PrimaryKey(autoGenerate = true) val tid: Int,    // In version 1
  val seatingCapacity: Int,                         // In version 1
  val seatingOccupancy: Int = 0)                    // New in version 2
```

```
class MyApp: Application() {
   lateinit var myDB: MyDatabase
   override fun onCreate() {
       myDB = Room.databaseBuilder(applicationContext, MyDatabase::class.java, "my-db")
           .addMigrations(V1_TO_V2).build()
   }

   val V1_TO_V2 = object : Migration(1,2) {
     override fun migrate(db: SupportSQLiteDatabase) {
       db.execSQL("ALTER TABLE partytable ADD COLUMN seatingoccupancy INTEGER DEFAULT 0")
     }
   }
}
```