Experiment 4

Experiment 4: Set up a local database (SQLite or Room) to store basic restaurant data.

Lab Objective:
To set up a local database in an Android application (SQLite or Room) to store basic restaurant data.

Prerequisites:

1. Basic Android development knowledge (Android Studio, XML).
2. Kotlin or Java programming familiarity.
3. Basic understanding of SQLite or Room.
4. Android Studio with SDK configured.

Outcome:
• Create a local database (SQLite or Room) to store restaurant data.
• Define schema including fields: name, location/address, cuisine/category, rating.
• Implement basic CRUD operations (create, read, update, delete).
• Use DAO interfaces (Room) for database operations.
• Verify persistence across app launches.

Option A — Room (recommended)

Step A1: Add Gradle dependencies (app-level `build.gradle`)

```
dependencies {
   implementation "androidx.room:room-runtime:2.5.2"
   kapt "androidx.room:room-compiler:2.5.2"
   implementation "androidx.room:room-ktx:2.5.2"
}
```

Apply kapt plugin at top of `build.gradle`:

```
apply plugin: 'kotlin-kapt'
```

Step A2: Entity — `Restaurant.kt`

```
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "restaurants")
data class Restaurant(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val category: String,
    val address: String?,
    val rating: Double?
)
```

Step A3: DAO — `RestaurantDao.kt`

```
import androidx.room.*

@Dao
interface RestaurantDao {

    @Query("SELECT * FROM restaurants ORDER BY name ASC")
    suspend fun getAll(): List<Restaurant>

    @Query("SELECT * FROM restaurants WHERE id = :id")
    suspend fun getById(id: Int): Restaurant?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(vararg restaurants: Restaurant)

    @Update
    suspend fun update(restaurant: Restaurant)

    @Delete
    suspend fun delete(restaurant: Restaurant)
}
```

Step A4: Database — `AppDatabase.kt`

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Restaurant::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun restaurantDao(): RestaurantDao
    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

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

Step A5: Pre-populate (optional) — insert sample data on first run

```
// Example: in Application.onCreate() or inside a coroutine in first activity
val db = AppDatabase.getInstance(context)
CoroutineScope(Dispatchers.IO).launch {
    val dao = db.restaurantDao()
    if (dao.getAll().isEmpty()) {
        dao.insert(
            Restaurant(name = "Spice Garden", category = "Indian", address = "12 MG Road",
rating = 4.2),
            Restaurant(name = "Sushi House", category = "Japanese", address = "18 Lake
Street", rating = 4.6),
            Restaurant(name = "Green Bowl", category = "Vegan", address = "33 Market Lane",
rating = 4.4)
        )
    }
}
```

Step A6: Usage example (MainActivity.kt — read & display in logs)

```
import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import kotlinx.coroutines.*

class MainActivity : AppCompatActivity() {

    private val TAG = "MainActivity"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val db = AppDatabase.getInstance(this)
        val dao = db.restaurantDao()

        CoroutineScope(Dispatchers.IO).launch {
            val list = dao.getAll()
            for (r in list) {
                Log.i(TAG, "Restaurant: ${r.id} | ${r.name} | ${r.category} | ${r.address} |
${r.rating}")
            }
        }
    }
}
```

Option B — SQLite (manual SQLiteOpenHelper)

Step B1: Schema (SQL)

```
CREATE TABLE restaurants (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  category TEXT NOT NULL,
  address TEXT,
  rating REAL
);
```

Step B2: `DatabaseHelper.kt`

```
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, "restaurant.db", null, 1) {

    override fun onCreate(db: SQLiteDatabase) {
        val create = """
            CREATE TABLE restaurants (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                category TEXT NOT NULL,
                address TEXT,
                rating REAL
        );
    """.trimIndent()
    db.execSQL(create)
  }

  override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    db.execSQL("DROP TABLE IF EXISTS restaurants")
    onCreate(db)
  }

  fun insertRestaurant(name: String, category: String, address: String?, rating: Double?) :
Long {
    val db = writableDatabase
    val cv = ContentValues().apply {
      put("name", name)
      put("category", category)
      put("address", address)
      put("rating", rating)
    }
    return db.insert("restaurants", null, cv)
  }

  fun getAllRestaurants(): List<RestaurantRecord> {
    val list = mutableListOf<RestaurantRecord>()
    val db = readableDatabase
    val cursor: Cursor = db.query("restaurants",
arrayOf("id","name","category","address","rating"),
      null,null,null,null,"name ASC")
    cursor.use {
      while (it.moveToNext()) {
        val id = it.getInt(it.getColumnIndexOrThrow("id"))
        val name = it.getString(it.getColumnIndexOrThrow("name"))
        val category = it.getString(it.getColumnIndexOrThrow("category"))
        val address = it.getString(it.getColumnIndexOrThrow("address"))
        val rating = if (!it.isNull(it.getColumnIndexOrThrow("rating")))
it.getDouble(it.getColumnIndexOrThrow("rating")) else null
        list.add(RestaurantRecord(id, name, category, address, rating))
      }
    }
    return list
  }

  data class RestaurantRecord(val id: Int, val name: String, val category: String, val
address: String?, val rating: Double?)
}
```

Step B3: Usage example (MainActivity.kt)

```
import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    private val TAG = "MainActivity"
    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        dbHelper = DatabaseHelper(this)

        // Insert sample row
        val id = dbHelper.insertRestaurant("Spice Garden", "Indian", "12 MG Road", 4.2)
        Log.i(TAG, "Inserted id = $id")

        // Read rows
        val restaurants = dbHelper.getAllRestaurants()
        for (r in restaurants) {
            Log.i(TAG, "Restaurant: ${r.id} | ${r.name} | ${r.category} | ${r.address} | ${r.rating}")
        }
    }
}
```

CRUD Quick Reference

- Room: use `@Insert`, `@Update`, `@Delete`, and `@Query` in DAO. Use coroutines or Rx for background threads.
- SQLiteOpenHelper: use `writableDatabase.insert()`, `update()`, `delete()` and rawQuery or `query()` for reads.

Test Cases

1. Fresh install (Room or SQLite) → DB created → table exists.
2. Insert record → Record count increases → record retrievable.
3. Update record → Changes are persisted and returned on read.

4. Delete record → Record no longer returned.
5. App restart → Data persists across launches.
6. Special characters (e.g., "Café Plaza") stored and retrieved correctly.

Notes

• Use Room for safer, modern implementation with compile-time checks.
• Run DB operations off the main thread (coroutines, Executors).
• For pre-population with Room, consider `Room.databaseBuilder(...).addCallback(...)` to insert initial data.