Experiment 5

Experiment 5: Implement code to pre-populate the local database with sample restaurant data.

Lab Objective:
To implement a code solution that pre-populates a local SQLite database with sample restaurant data.

Prerequisites:

1. Basic understanding of databases (tables, records).
2. Familiarity with SQL and SQLite.
3. Basic Kotlin programming knowledge.
4. Familiarity with Android's SQLiteOpenHelper and DB connection.
5. Understanding of schema design and data types.

Outcome:
• Write code that creates the required table(s).
• Insert multiple sample restaurant records during database creation.
• Verify inserted data by querying the database.
• Learn basic error handling for insert operations.

Step 1: Database schema (SQL)

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

Step 2: DatabaseHelper.kt (pre-populate in onCreate)

```
import android.content.ContentValues
```

```
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.util.Log

class DatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DB_NAME, null, DB_VERSION) {

    companion object {
        private const val DB_NAME = "restaurant.db"
        private const val DB_VERSION = 1

        private const val TABLE_RESTAURANTS = "restaurants"
        private const val COL_ID = "id"
        private const val COL_NAME = "name"
        private const val COL_CATEGORY = "category"
        private const val COL_ADDRESS = "address"
        private const val COL_RATING = "rating"
    }

    override fun onCreate(db: SQLiteDatabase) {
        val createTable = """
            CREATE TABLE $TABLE_RESTAURANTS (
                $COL_ID INTEGER PRIMARY KEY AUTOINCREMENT,
                $COL_NAME TEXT NOT NULL,
                $COL_CATEGORY TEXT NOT NULL,
                $COL_ADDRESS TEXT,
                $COL_RATING REAL
            );
        """.trimIndent()
        db.execSQL(createTable)

        // Pre-populate with sample data
        val sampleData = listOf(
            RestaurantInsert("Spice Garden", "Indian", "12 MG Road", 4.2),
            RestaurantInsert("The Burger Joint", "Fast Food", "5 Park Avenue", 4.0),
            RestaurantInsert("Sushi House", "Japanese", "18 Lake Street", 4.6),
            RestaurantInsert("Green Bowl", "Vegan", "33 Market Lane", 4.4),
            RestaurantInsert("Pasta Palace", "Italian", "7 River Road", 4.1),
            RestaurantInsert("Taco Town", "Mexican", "99 Sunset Blvd", 4.0)
        )

        db.beginTransaction()
        try {
            for (r in sampleData) {
                val cv = ContentValues().apply {
                    put(COL_NAME, r.name)
```

```
                put(COL_CATEGORY, r.category)
                put(COL_ADDRESS, r.address)
                put(COL_RATING, r.rating)
            }
            val id = db.insert(TABLE_RESTAURANTS, null, cv)
            if (id == -1L) {
                Log.w("DatabaseHelper", "Failed to insert row for ${r.name}")
            }
        }
        db.setTransactionSuccessful()
    } catch (e: Exception) {
        Log.e("DatabaseHelper", "Error pre-populating DB", e)
    } finally {
        db.endTransaction()
    }
}

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

// Utility: fetch all restaurants (for verification)
fun getAllRestaurants(): List<RestaurantRecord> {
    val list = mutableListOf<RestaurantRecord>()
    val db = readableDatabase
    val cursor: Cursor = db.query(
        TABLE_RESTAURANTS,
        arrayOf(COL_ID, COL_NAME, COL_CATEGORY, COL_ADDRESS, COL_RATING),
        null, null, null, null, "$COL_NAME ASC"
    )
    cursor.use {
        while (it.moveToNext()) {
            val id = it.getInt(it.getColumnIndexOrThrow(COL_ID))
            val name = it.getString(it.getColumnIndexOrThrow(COL_NAME))
            val category = it.getString(it.getColumnIndexOrThrow(COL_CATEGORY))
            val address = it.getString(it.getColumnIndexOrThrow(COL_ADDRESS))
            val rating = if (!it.isNull(it.getColumnIndexOrThrow(COL_RATING)))
                it.getDouble(it.getColumnIndexOrThrow(COL_RATING)) else null
            list.add(RestaurantRecord(id, name, category, address, rating))
        }
    }
    return list
}

// Simple data holders
data class RestaurantInsert(val name: String, val category: String, val address: String?,
val rating: Double?)
```

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

Step 3: Verification snippet (MainActivity.kt)

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

class MainActivity : AppCompatActivity() {

    private lateinit var dbHelper: DatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // setContentView(...) -- not required for pre-population verification

        dbHelper = DatabaseHelper(this)

        // Trigger DB creation and pre-population by getting writable DB
        val db = dbHelper.writableDatabase

        // Verify by reading all records and logging them
        val restaurants = dbHelper.getAllRestaurants()
        for (r in restaurants) {
            Log.i("MainActivity", "Restaurant: ${r.id} | ${r.name} | ${r.category} | ${r.address} |
${r.rating}")
        }
    }
}
```

Explanation:

1. `DatabaseHelper.onCreate()` creates the `restaurants` table and inserts multiple sample rows inside a DB transaction for atomicity and performance.

2. Inserts are done with `ContentValues` and `db.insert()`; failures are logged.

3. `getAllRestaurants()` queries the table and returns a list of records for verification.

4. `MainActivity` opens the writable database to trigger `onCreate()` and logs inserted rows to Logcat for verification.

Test Cases:

1. Fresh install → `onCreate()` runs → table created → sample rows inserted → `getAllRestaurants()` returns inserted rows.

2. Corrupted insert (simulated by forcing an exception) → transaction rolls back → no partial inserts.

3. Re-run app without uninstall → `onCreate()` not called (DB exists) → data persists across launches.

4. `onUpgrade()` with version change → table dropped and recreated with sample data.

5. Validate special characters (e.g., "Café Plaza") are inserted and retrieved correctly.