Filipe Lutz Mariano 25956

Mobile Apps 2

Assignment Five

Submission Date 17/11/2024

Percent 10%

Include all workings and references

Section A.

1. Compare and contrast StateFlow and LiveData (*3 marks*)

StateFlow and **LiveData** are used for managing and observing state in Android applications, but they are designed for different purposes and have distinct characteristics.

Aspect	StateFlow	LiveData
Type of Data	State holder that always holds the latest value and is immutable. Part of Kotlin's Flow API.	A lifecycle-aware data holder that is specifically designed to be observed within UI components, holding and emitting the latest value.
Thread Safety	Thread-safe: Can be collected from any thread but typically used within a coroutine scope.	Thread-safe: Automatically handles thread switching when used with lifecycle-aware components.
Lifecycle Awareness	Not lifecycle-aware: StateFlow will continue emitting values even when the UI component is not active. Developer must manage lifecycle.	Lifecycle-aware: Notifies observers only when the lifecycle owner is in an active state.
Use Case	Ideal for state management and non-UI data, managing state across multiple parts of the app.	Primarily for UI-related data where updates need to be automatically delivered to the UI when the lifecycle is valid.
API Support	Supports Flow operators like map, combine, collect, etc. for data transformations.	Supports basic observe() and postValue() for background updates.
Mutability	MutableStateFlow for updateable data; StateFlow is read-only.	MutableLiveData can be updated; LiveData is read-only when exposed.

Summary:

- **StateFlow** is a more flexible and powerful tool for managing and observing data, especially when you need to work with **streams of data** in a **coroutine-based** environment.
- **LiveData** is simpler and integrates deeply with the **Android lifecycle** but is less flexible compared to StateFlow for complex use cases.

2. Referential Integrity is important to a relational database. Explain, giving an example and how it is enforced. (*2 marks*)

Referential Integrity ensures that relationships between tables in a relational database remain consistent. Specifically, it means that foreign keys in one table must match primary keys in another table or be null, ensuring that there are no orphan records (records that refer to non-existent entities).

Example: Consider a database with two tables: Orders and Customers.

- **Customers Table**:
  - customer_id (Primary Key)
  - name
- **Orders Table**:
  - order_id (Primary Key)
  - customer_id (Foreign Key)

In this case, the Orders table contains a customer_id, which refers to the customer_id in the Customers table. Referential integrity ensures that every customer_id in the Orders table corresponds to a valid customer_id in the Customers table.

Enforcing Referential Integrity:

1. **Foreign Key Constraints**: The database can enforce referential integrity by declaring customer_id in the Orders table as a **foreign key** that references the customer_id in the Customers table. The database will ensure:
   - **Insertions**: A record in the Orders table cannot reference a non-existing customer_id.
   - **Deletions**: If a customer_id is deleted from the Customers table, the database can either delete the corresponding rows in the Orders table (cascading delete), set them to null, or prevent the deletion (restrict).
   - **Updates**: If a customer_id in the Customers table is updated, the database can propagate the change to the Orders table or prevent the update if it would violate integrity.

Example SQL to create foreign key constraints:

```
CREATE TABLE Customers (
   customer_id INT PRIMARY KEY,
   name VARCHAR(100)
);

CREATE TABLE Orders (
   order_id INT PRIMARY KEY,
   customer_id INT,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

Section B.

1. Using Room database, set up the following table from the Query tab:

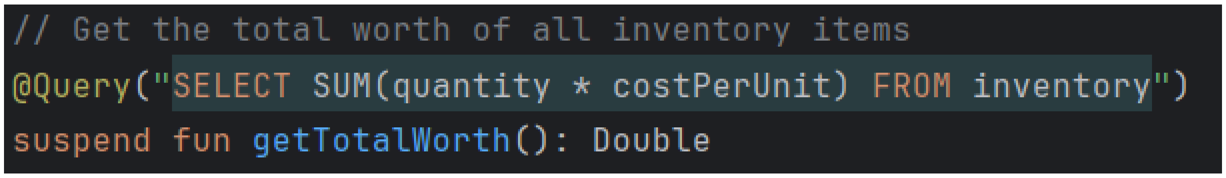
Inventory		
ID	Primary Key	Autogenerated
Name	Required	
Quantity	Numeric	
Supplier		Default “Co-op Store”
Cost Per Unit		

(3.5 *marks*)

2. Write a script that would get the total monetary worth of inventory

(*1.5 marks*)

Total Worth = SUM(quantity * costPerUnit)



```
// Get the total worth of all inventory items
@Query("SELECT SUM(quantity * costPerUnit) FROM inventory")
suspend fun getTotalWorth(): Double
```

References:

1. StateFlow and SharedFlow
2. Kotlin flows on Android
3. LiveData
4. Save data in a local database using Room
5. Guide to app architecture
6. Android Room with a View
7. LiveData vs StateFlow
8. Differences between StateFlow and LiveData?