Module 5

[Assignment]

Que-1) Discuss the differences between Shared Preferences, SQLite, and Room in Android. When would you choose each for data storage?

Ans:

Features	Shared Preference	SQLite	Room
Purpose	Stores simple key-value pairs	Stores structured relational data	Modern database layer on top of SQLite
Use case	Small settings, user preferences	Complex relational data storage	Simplifies SQLite with ORM features
Data Structure	Key-Value pairs (primitive types)	Tables with rows & columns	Uses entity classes (objects) mapped to tables
Query Language	No queries (only get/set values)	SQL queries required	Uses DAO (Data Access Object) with annotations
Performance	Fast for small data	Can be slow with complex queries	Optimized with LiveData, Flow, caching
Thread Safety	Not designed for multi-threading	Requires manual handling (e.g., using AsyncTask or Coroutines)	Built-in support for multi-threading via LiveData and Flow

Scalability	Not scalable, only good for simple data	Scalable but requires more management	More scalable and maintainable

1. SharedPreferences

- **Use when you need to store small amounts of simple data.**
- **Ideal for:**
  - **User settings (e.g., dark mode preference, language selection)**
  - **Session tokens (temporary, non-secure)**
  - Last opened page or app state

Avoid when:

- You need to store complex data or large amounts of structured data.

---

2. SQLite

- **Use when you need to store structured relational data.**
- **Ideal for:**
  - Storing large, complex data (e.g., user profiles, transactions)
  - Performing advanced queries, joins, indexing
  - Local database storage for offline use

Challenges:

- Requires manual SQL queries and boilerplate code.
- Managing migrations and data consistency can be complex.

---

3. Room (Recommended over SQLite)

- **Use when you need a structured database with easier management.**
- **Ideal for:**
  - Storing relational data like SQLite, but with less boilerplate code

    - Using Kotlin Coroutines, LiveData, or Flow for reactive data updates
    - Handling complex database migrations smoothly

Why Choose Room over SQLite?

- Less boilerplate: Eliminates writing raw SQL for common operations.
- Compile-time validation: Ensures queries and schemas are correct.
- Better integration: Works well with Jetpack components (ViewModel, LiveData).

Conclusion:

- Use SharedPreferences for small, simple key-value data.
- Use SQLite for raw database control (but avoid it if possible).
- Use Room for a modern, maintainable database experience.

Since you're working with Room and Kotlin, you'd likely use Room in most cases for database needs, while SharedPreferences would be helpful for user settings.