🎯 1. SharedPreferences vs DataStore

☑ Câu 1 – SharedPreferences hoạt động như thế nào? Có những nhược điểm gì?

🎯 Trả lời mẫu

SharedPreferences lưu dữ liệu theo dạng key–value trong file XML nằm trong Internal Storage. Khi gọi `edit().commit()` hoặc `apply()`, dữ liệu sẽ được ghi xuống file XML.

Nhược điểm:

1. **Synchronous (đồng bộ)** → Gây **ANR** nếu ghi file lớn hoặc ghi nhiều lần liên tục.
2. **Không an toàn trong đa luồng** → dễ lỗi ghi đè hoặc mất dữ liệu.
3. **Không đảm bảo atomically** (tính nguyên tử) khi thao tác nhanh.
4. Giao diện API cũ, Google **khuyên dùng DataStore thay thế**.

☑ Câu 2 – Điểm khác nhau giữa SharedPreferences và DataStore?

SharedPreferences	DataStore
Đồng bộ (blocking)	Bất đồng bộ (coroutines + Flow)
Dễ gây ANR	Không gây ANR
Không an toàn đa luồng	An toàn, atomic
Dễ mất dữ liệu	Được thiết kế an toàn
Lưu XML	Preferences / Proto (Protobuf)
Code cũ	API mới của Jetpack

Tóm lại: 👉 Với dữ liệu key–value, nên dùng **DataStore** thay vì SharedPreferences.

☑ Câu 3 – Khi nào dùng Preferences DataStore và khi nào dùng Proto DataStore?

🎯 Trả lời mẫu

- **Preferences DataStore**: dữ liệu key-value đơn giản (giống SharedPreferences).
- **Proto DataStore**: lưu dữ liệu có **schema rõ ràng**, dùng Protobuf → mạnh mẽ, type-safe. Ví dụ dùng Proto DataStore: UserSettings(name, age, themeMode, notificationEnabled)

☑ Câu 4 – Vì sao DataStore không gây ANR như SharedPreferences?

🎯 Trả lời mẫu

- DataStore chạy hoàn toàn trên **coroutines (Dispatchers.IO)** → tất cả thao tác I/O đều bất đồng bộ.
- Mọi đọc/ghi đều qua Flow + suspend function → không block main thread.
- Có transaction bên trong → đảm bảo không ghi đè.

🎯 2. Room Database (ORM)

☑ Câu 5 – Room gồm các thành phần nào? Nhiệm vụ từng phần?

🎯 Trả lời mẫu

Room có 3 thành phần chính:

1. **Entity** → đại diện 1 bảng trong SQLite (dùng annotation @Entity).
2. **DAO (Data Access Object)** → chứa hàm thao tác với DB (@Insert, @Query, @Update, @Delete).
3. **RoomDatabase** → nơi khởi tạo Database, kết nối Entity và DAO.

☑ Câu 6 – Khi dùng Room, bạn có cần viết code quản lý Cursor không?

🎯 Trả lời mẫu

Không. Room tự động thực hiện:

- quản lý Cursor, close Cursor
- ánh xạ Cursor → object Kotlin
- kiểm tra SQL query lúc compile

→ Giảm lỗi và code sạch hơn rất nhiều.

☑ Câu 7 – Room hỗ trợ Data Observability như thế nào?

🎯 Trả lời mẫu

Room hỗ trợ trả dữ liệu dạng:

- **LiveData**
- **Flow**

Khi dữ liệu thay đổi, Flow hoặc LiveData tự động emit lại → UI update realtime.

Ví dụ:

```
@Query("SELECT * FROM user_table")
fun getAllUsers(): Flow<List<User>>
```

☑ Câu 8 – Sự khác nhau giữa @Insert, @Update, @Delete và @Query?

🎯 Trả lời mẫu

- **@Insert** → thêm record
- **@Update** → cập nhật đối tượng, dựa vào primary key
- **@Delete** → xoá theo object
- **@Query** → thao tác SQL tùy ý (select, delete by id, update value…)

☑ Câu 9 – Tại sao Room yêu cầu hàm insert/update/delete phải là suspend?

🎯 Trả lời mẫu

Vì thao tác database là I/O → mất thời gian. Room bắt buộc chạy các tác vụ này trong coroutine để tránh **bị block UI → gây ANR**.

🎯 3. Internal Storage vs External Storage

☑ Câu 10 – Sự khác biệt của Internal và External Storage?

🎯 Trả lời mẫu

Internal Storage	External Storage
Private cho app	Public cho user
Không cần permission	Cần permission
An toàn hơn	Dễ bị xoá, bị truy cập
File bị xoá khi uninstall app	File có thể còn lại
Dùng cho: dữ liệu nhỏ, private	Dùng cho file lớn, media, export

---

☑ Câu 11 – Khi nào cần xin quyền READ/WRITE_EXTERNAL_STORAGE?

🎯 Trả lời mẫu

Chỉ khi ứng dụng cần:

- ghi file vào **bộ nhớ ngoài** không thuộc thư mục app
- truy cập ảnh/video trong thư viện người dùng (Android < 10)

Lưu ý:

- Trên Android 10+ sử dụng **Scoped Storage** → không cần quyền cho thư mục riêng của app.
- Trên Android 13+, quyền đọc ảnh/video được tách thành:
  - READ_MEDIA_IMAGES
  - READ_MEDIA_VIDEO
  - READ_MEDIA_AUDIO

☑ Câu 12 – Internal Storage có an toàn tuyệt đối không?

🎯 Trả lời mẫu

Không tuyệt đối. Mặc dù file private, nhưng:

- máy root vẫn đọc được
- malware có thể truy cập nếu có quyền root → Nếu dữ liệu nhạy cảm (password) → nên **mã hóa**.

🎯 4. Câu hỏi nâng cao – để gây ấn tượng

⭐ Câu 13 – DataStore xử lý multi-process thế nào? Có an toàn hơn SharedPreferences không?

🎯 Trả lời mẫu

SharedPreferences **không an toàn** khi nhiều process cùng read/write. DataStore sử dụng:

- Transaction
- Single writer
- Coroutine + Mutex

→ đảm bảo an toàn trong multi-thread + multi-process.

⭐ Câu 14 – Room dùng SQLite bên trong, vậy Room nhanh hơn SQLite không?

🎯 Trả lời mẫu

Room KHÔNG nhanh hơn SQLite vì nó chính là SQLite.

Room chỉ:

- tránh code thừa
- tránh lỗi SQL
- hỗ trợ coroutine + Flow

- mapping object nhanh hơn vì dùng annotation processing

→ Nhưng tốc độ chạy SQL thì không thay đổi.

⭐ Câu 15 – Làm thế nào để encrypt database Room?

🎯 Trả lời mẫu

Dùng thư viện SQLCipher:

```
implementation "net.zetetic:android-database-sqlcipher:4.5.0"
```

Rồi cấu hình trong Room:

```
val factory = SupportFactory(SQLiteDatabase.getBytes("password123".toCharArray()))

val db = Room.databaseBuilder(context, AppDatabase::class.java, "secure.db")
    .openHelperFactory(factory)
    .build()
```
