



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Ingeniería en Sistemas Computacionales.

Tarea 6_ Unidad 6 Persistencia de Datos

Asignatura: Programación Nativa para Móviles

Docente: Jorge Peralta Escobar

Hora: 14:00 – 15:00 pm

Integrantes:

Del Angel Del Angel Erika Yaneth #20071926

Villaseñor Grimaldo Alejandro #19071548

Semestre: Enero – Junio 2025.



GITHUB - [Workspace/Programacion_movil/Tarea6U6 at master · Alejandro19071548/Workspace](#)

RUTA DE APRENDIZAJE 1 – INTRODUCCIONA SQL

VIDEO 1 – RUTA DE APRENDIZAJE 1

Esta unidad introduce el concepto de persistencia de datos en aplicaciones móviles, específicamente mediante el uso de bases de datos locales SQL Lite en Android. Se explica que almacenar datos en variables no es suficiente porque se pierden al cerrar la aplicación, y que una base de datos permite guardar datos estructurados que persisten. Se describen las características de las bases de datos SQL Lite: organizadas en tablas con filas y columnas, cada tabla con una clave principal única. Se enseña el uso del lenguaje SQL para interactuar con estas bases de datos, destacando las sentencias básicas para seleccionar, insertar, actualizar y eliminar datos. Además, se presentan cláusulas y operadores para filtrar y ordenar datos con SELECT, como WHERE, AND, OR, NOT, ORDER BY y LIMIT. Finalmente, se anima a los estudiantes a practicar este conocimiento para desarrollar aplicaciones que manejen datos persistentes.

CUENTIONARIO RUTA 1

1. ¿Cuáles de las siguientes afirmaciones sobre las bases de datos relacionales y SQLite son verdaderas?
Selecciona todas las respuestas que consideres correctas.

☒ Hacer referencia a la clave primaria de una tabla en otra tabla te permite modelar relaciones entre tablas. ✓ ¡Correcto!

☐ Una base de datos SQLite consiste en columnas, que constan de tablas y filas.

☐ Cada tabla de datos debe tener al menos una clave externa.

☒ Las filas contienen los elementos individuales de la base de datos. ✓ ¡Correcto!

2. Finalizar una instrucción de SQL con un punto y coma es opcional.

☐ Verdadero

☒ Falso ✓ ¡Correcto!

3. ¿Qué usas para calcular la suma de todos los valores de una columna de base de datos?

☒ Función agregada ✓ ¡Correcto!

4. ¿Qué sentencia `SELECT` muestra la cantidad de direcciones de correos electrónicos únicas para los mensajes en la carpeta de spam?

- ☐ `SELECT COUNT(DISTINCT folder) FROM email WHERE spam != sender;`
- ☐ `SELECT DISTINCT COUNT(sender) FROM email WHERE folder = 'spam';`
- ☒ `SELECT COUNT(DISTINCT sender) FROM email WHERE folder = 'spam';`
- ☐ `SELECT DISTINCT COUNT('spam') FROM email WHERE sender = folder;`

✓ ¡Correcto!

5. La instrucción de SQL `SELECT * FROM contacts WHERE name LIKE '%Milton'` muestra todas las filas en las que el valor de la columna de nombre comienza con `Milton`.

☐ Verdadero

☒ Falso

✓ ¡Correcto!

6. ¿Cuáles de las siguientes afirmaciones sobre `GROUP BY` y `ORDER BY` son verdaderas?

Selecciona todas las respuestas que consideres correctas.

⊖ Parcialmente correcto.

- ☐ La cláusula `ORDER BY` va antes de la cláusula `GROUP BY`.
- ☐ En una cláusula `ORDER BY`, el orden descendente es el predeterminado.
- ☐ Si una consulta contiene una cláusula `GROUP BY`, anula la cláusula `ORDER BY`.
- ☒ Las cláusulas `ORDER BY` y `GROUP BY` pueden aceptar varias columnas.

✓ ¡Correcto!

7. La condición `WHERE NOT read = false` y la condición `WHERE read != true` son equivalentes.

☐ Verdadero

☒ Falso

✓ ¡Correcto!

8. La cláusula `LIMIT` `LIMIT 30 SKIP 60` muestra lo siguiente:

☐ La fila 60

☐ Las filas 31 a 60

☒ Las filas 61 a 90

✓ ¡Correcto!

☐ La fila 90

9. Una declaración `UPDATE` usa una cláusula ____ para asignar valores a las columnas.

☐ `WHERE`

☒ `SET`

✓ ¡Correcto!

☐ `ASSIGN`

☐ `LIKE`

10. Las declaraciones `UPDATE` y `DELETE` pueden incluir una cláusula `WHERE` y pueden afectar a varias filas.

☒ Verdadero

✓ ¡Correcto!

Results

Tu puntuación es 9 de 10. ¡Felicitaciones! Aprobaste el cuestionario.

**RUTA DE APRENDIZAJE 2 – COMO USAR ROOM PARA LOGRAR LA
PERSISTENCIA DE DATOS**

VIDEO 1 – RUTA DE APRENDIZAJE 2

La presentación aborda conceptos básicos y avanzados sobre Kotlin Flow, una API para manejar flujos de datos asíncronos en Android. A diferencia de las funciones suspendidas, los Flows pueden emitir múltiples valores con el tiempo, lo que los hace ideales para eventos como cambios en la base de datos, acciones del usuario o datos de sensores.

Se destaca la importancia de integrar Flow con el ciclo de vida de los componentes de Android usando herramientas como `repeatOnLifecycle`, lo que permite recolectar datos de forma segura, evitando fugas de memoria y errores comunes.

También se muestran ejemplos prácticos de su uso, como observar datos de Room, responder a eventos de UI y procesar entradas en tiempo real. Finalmente, se recomiendan buenas prácticas como estructurar el código para facilitar el mantenimiento y recolectar los flujos considerando el ciclo de vida, promoviendo un código más limpio y robusto.

VIDEO 2 – RUTA DE APRENDIZAJE 2

En esta entrega de MAD Skills se presenta la biblioteca Room de Jetpack, destacando su integración con Kotlin para facilitar el manejo de bases de datos locales en Android. Room actúa como una capa de abstracción sobre SQLite, permitiendo definir entidades, relaciones y consultas de forma declarativa con anotaciones, lo que mejora la legibilidad y reduce errores.

Se muestra cómo Room se integra con las corrutinas de Kotlin para realizar operaciones asíncronas (insertar, actualizar, eliminar) sin bloquear el hilo principal. También se explica el uso de Kotlin Flow para observar en tiempo real los cambios en la base de datos y actualizar dinámicamente la interfaz de usuario.

Además, se recomiendan buenas prácticas como el uso de DAOs para organizar el acceso a los datos y el uso de extensiones KTX para escribir código más limpio y conciso en Kotlin.

CUESTIONARIO RUTA 2

1. ¿Cuál de las siguientes afirmaciones sobre la anotación `@Query` no es verdadera?

- ☐ En el DAO, la anotación `@Query` se usa con un método.
- ☐ La anotación `@Query` corresponde a una búsqueda `SELECT`.
- ☐ Cuando hay dos puntos antes de su nombre, la anotación `@Query` puede pasar argumentos a una instrucción de SQL.
- ☒ La anotación `@Query` solo se puede usar con una función de suspensión. ¡Correcto!

2. ¿Cuáles de las siguientes afirmaciones sobre el DAO son verdaderas?

- ☐ Las funciones DAO usan anotaciones como `@Insert` y `@Update` que corresponden a una operación en la base de datos.
- ☐ Las funciones DAO pueden mostrar un flujo.
- ☐ En la clase `AppDatabase`, se hace referencia a las instancias de clases DAO.
- ☒ Todas las opciones anteriores ¡Correcto!

3. La clase `Database`, que se hereda de la clase `RoomDatabase`, es responsable de ____.

- ☒ Crear una instancia de la base de datos y proporcionar acceso al DAO ¡Correcto!

4. ¿Cuál es el propósito del DAO?

- ☐ Conserva la referencia a los modelos de vista y la base de datos.
- ☒ Define funciones que se asignen a instrucciones de SQL, como las consultas `SELECT` y `INSERT`. ¡Correcto!
- ☐ Proporciona un método de fábrica para crear una instancia de base de datos.
- ☐ Crea una nueva instancia de base de datos.

5. ¿Por qué necesitas usar la función `synchronized()` cuando creas la base de datos?

Selecciona todas las respuestas que consideres correctas.

- ☐ Te permite crear varias copias de la base de datos.
- ☒ Te permite acceder de forma segura al código desde varios subprocesos a la vez. ¡Correcto!
- ☒ Se usa para evitar condiciones de carrera. ¡Correcto!
- ☒ Garantiza que solo un subproceso pueda ingresar al bloque de código a la vez. ¡Correcto!

8. Selecciona todas las afirmaciones que sean verdaderas sobre el Inspector de bases de datos:

Selecciona todas las respuestas que consideres correctas.

- ☒ Te permite inspeccionar, consultar y modificar las bases de datos de la app mientras se está ejecutando. ✓ ¡Correcto!
- ☐ Funciona con otras bibliotecas de SQLite que empaquetas con tu app.
- ☒ Es particularmente útil para depurar bases de datos. ✓ ¡Correcto!
- ☒ Funciona con SQLite simple y bibliotecas compiladas sobre SQLite, como Room. ✓ ¡Correcto!

9. Las entidades representan tablas de datos individuales en la base de datos de Room.

- ☒ Verdadero ✓ ¡Correcto!
- ☐ Falso

6. Puedes usar las anotaciones `@Insert` y `@Delete` sin proporcionar una instrucción de SQL.

- ☒ Verdadero ✓ ¡Correcto!
- ☐ Falso

7. Completa los espacios en blanco

Ingresa una o más palabras para completar la oración.

Para manejar conflictos cuando haces una inserción en una base de datos, puedes pasar un parámetro

`onConflict` (como IGNORE) a la anotación `@Insert`.

✓ ¡Correcto!


8. Selecciona todas las afirmaciones que sean verdaderas sobre el Inspector de bases de datos:

Selecciona todas las respuestas que consideres correctas.

- ☒ Te permite inspeccionar, consultar y modificar las bases de datos de la app mientras se está ejecutando. ✓ ¡Correcto!
- ☐ Funciona con otras bibliotecas de SQLite que empaquetas con tu app.

10. ¿Cuál de las siguientes afirmaciones sobre la clave primaria no es verdadera?

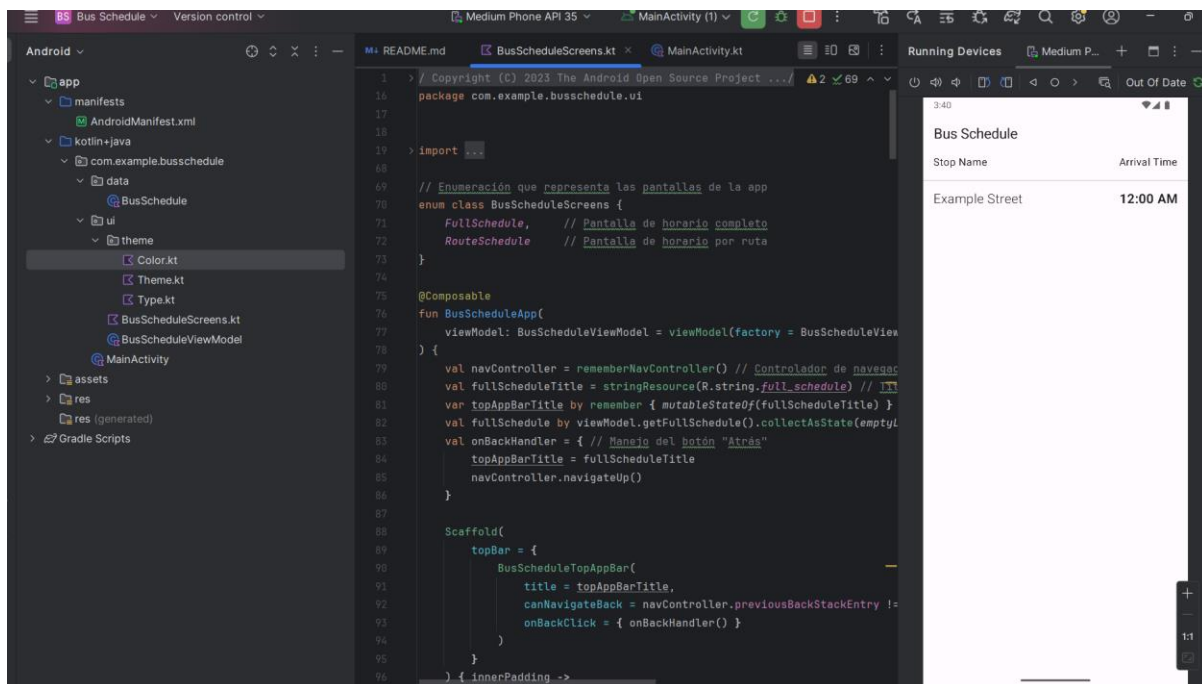
- ☐ Puedes usar la clave primaria para identificar de manera única cada registro o entrada en las tablas de tu base de datos.
- ☐ Después de asignar la clave primaria, no podrás modificarla.
- ☐ Room genera un valor de clave primaria incremental para cada entidad de forma predeterminada.
- ☒ La clave primaria representa el objeto de la entidad, siempre que exista en la base de datos.

 Incorrecto.

Results

Tu puntuación es **9 de 10**. ¡Felicitaciones! Aprobaste el cuestionario.

BUS SCHEDULE



Esta app de Android desarrollada con Jetpack Compose permite visualizar los horarios de autobuses en dos vistas principales:

1. FullScheduleScreen: Muestra todos los horarios disponibles.
2. RouteScheduleScreen: Muestra los horarios filtrados por una parada específica.

El usuario puede navegar entre estas vistas tocando un elemento de la lista. El encabezado se actualiza dinámicamente con el nombre de la parada seleccionada. Usa un ViewModel para obtener los datos en tiempo real y Jetpack Navigation para manejar las pantallas.

También incluye un TopAppBar con botón de retroceso cuando se navega a la pantalla de ruta. Todo el contenido se adapta a diferentes tamaños y direcciones de layout, utilizando Scaffold, NavHost, y LazyColumn.

RUTA DE APRENDIZAJE 3 – COMO ALMACENAR DATOS Y ACCEDER A ELLOS MEDIANTE CLAVES CON DATASTORE

VIDEO 1 – RUTA DE APRENDIZAJE 3

La presentación compara SharedPreferences con DataStore, señalando que SharedPreferences tiene limitaciones importantes, como una API sincrónica que puede bloquear el hilo principal y falta de seguridad en la escritura de datos. En cambio, DataStore ofrece una API completamente asincrónica basada en Kotlin Coroutines y Flow, lo que permite realizar lecturas y escrituras sin bloquear el hilo principal, mejorando el rendimiento y la respuesta de la app. Se describen dos tipos principales de DataStore: **Preferences DataStore**, que usa pares clave-valor como SharedPreferences pero con ventajas de asincronía y seguridad; y **Proto DataStore**, que permite almacenar objetos tipados mediante Protocol Buffers, ofreciendo una estructura de datos más sólida y segura.

VIDEO 2 – RUTA DE APRENDIZAJE 3

Preferences DataStore es una alternativa moderna a SharedPreferences que utiliza una API completamente asincrónica basada en Kotlin Coroutines y Flow. A diferencia de SharedPreferences, que es sincrónica y vulnerable a problemas de concurrencia, Preferences DataStore permite realizar lecturas y escrituras sin bloquear el hilo principal, lo que mejora notablemente el rendimiento y la capacidad de respuesta de la aplicación.

Para implementarlo, primero se debe agregar la dependencia en el archivo build.gradle del módulo. Luego, se crea una instancia de DataStore mediante el delegado preferencesDataStore, y se definen las claves de preferencias utilizando funciones como booleanPreferencesKey o stringPreferencesKey. La lectura de datos se realiza a través del flujo datastore.data, mientras que la escritura se hace con el método edit, que garantiza operaciones transaccionales y seguras.

CUESTIONARIO RUTA 3

1. Estas son las implementaciones de `DataStore`:

Selecciona todas las respuestas que consideres correctas.

☒ Proto

☒ ¡Correcto!

☒ Preferencias

☒ ¡Correcto!

☐ Room

☐ SQLite

2. Preferences `DataStore` usa un esquema predefinido.

☐ Verdadero

☒ Falso

☒ ¡Correcto!

3. ¿Qué función proporciona `DataStore` para su modificación?

☐ `preferencesDataStore()`

☐ `updatePreferences()`

☒ `edit()`

☒ ¡Correcto!

4. Preferences DataStore utiliza claves para acceder a los valores almacenados.

☒ Verdadero  ¡Correcto!

☐ Falso

5. ¿Qué excepción puede ocurrir cuando se intenta leer desde Preferences DataStore?

☐ IllegalArgumentException

☒ IOException  ¡Correcto!

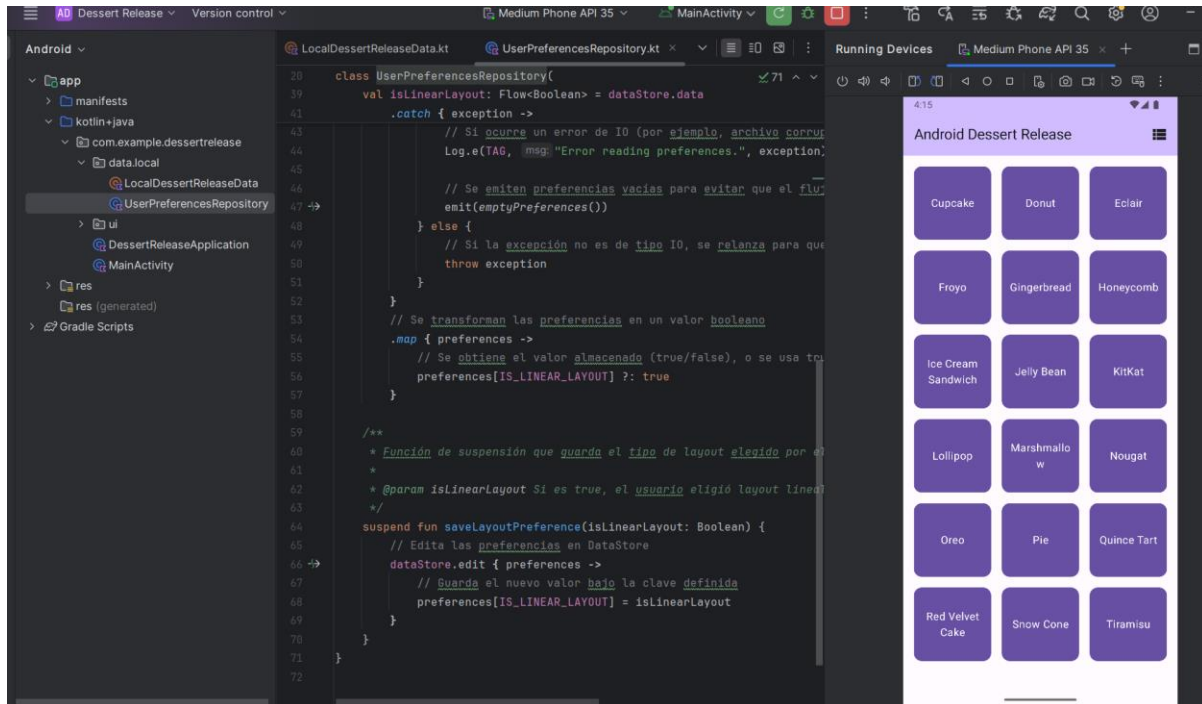
☐ IllegalStateException

☐ NumberFormatException

Results

Tu puntuación es 5 de 5. ¡Felicitaciones! Aprobaste el cuestionario.

ALMACENAR DATASTORE



Esta clase es un repositorio de preferencias de usuario que utiliza Jetpack DataStore para:

- Leer si el usuario prefiere una vista lineal o grid (flujo isLinearLayout).
- Guardar esa preferencia de forma persistente (saveLayoutPreference).

En una app donde puedes cambiar el diseño de lista (por ejemplo, productos o fotos), esta clase permite recordar la elección del usuario incluso después de cerrar la app.