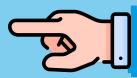
Acceso a datos en Android

La librería ROOM (Parte II)

{desafío}
latam_

Implementar capa de acceso a datos en un aplicativo móvil utilizando la librería ROOM para otorgar persistencia de estados resolviendo el problema planteado

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android



{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Describe el rol de la capa de abstracción Room y sus principales elementos tales como Entidad y DAO*

{desafío}
latam_

¿Cómo te imaginas que se guardan las tablas creadas en Room? ¡Escribe tu respuesta en el chat!

{desafío} latam_

/* Definiendo relaciones entre objetos */

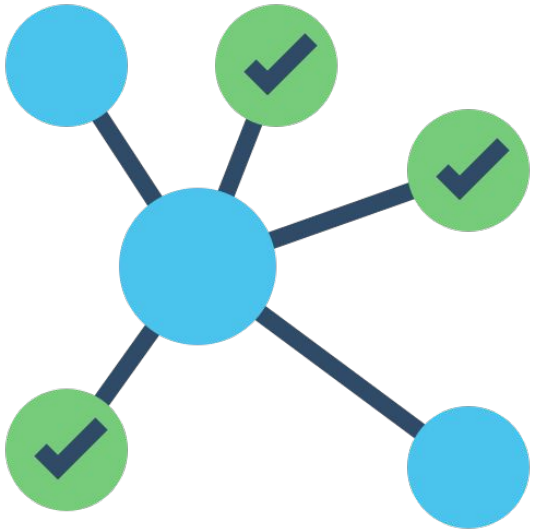
Definiendo relaciones entre objetos

En Room, puede definir relaciones entre tablas utilizando las anotaciones `@Entity` y `@Embedded`.

Hay dos tipos de relaciones que se pueden definir en Room:

- Relación **uno a uno**
- Relación de **uno a varios**

La relación uno a uno se define mediante la anotación `@Embedded`. Te permite incluir una entidad dentro de otra como un campo simple.



Room - Relación uno a uno - ejemplo

```
@Entity
public class User {
    @PrimaryKey
    public int id;

    @Embedded
    public Address address;
}

@Embeddable
public class Address {
    public String street;
    public String city;
    public String state;
    public String zip;
}
```

Room - Relación uno a varios

La relación de uno a muchos se define mediante la anotación `@Relation`. Le permite incluir una lista de una entidad dentro de otra.

```
@Entity
public class User {
    @PrimaryKey
    public int id;

    public String name;
}

@Entity
public class Order {
    @PrimaryKey
    public int id;

    public int userId;

    @Relation(parentColumn = "userId", entityColumn = "id")
    public User user;
}
```

Room - Relación uno a varios - Ignore

Además, en el caso de una relación de uno a muchos, puede usar `@Ignore` en la clase de entidad secundaria para evitar que la clase secundaria cree su propia tabla y solo sea parte de la tabla principal.

```
@Entity
public class User {
    @PrimaryKey
    public int id;

    public String name;

    @Ignore
    @Relation(parentColumn = "id", entityColumn = "userId")
    public List<Order> orders;
}
```

También debes asegurarte de que la clave externa esté definida en la tabla secundaria y esté configurada como no nula. Es importante tener en cuenta que no puedes usar `@Relation` para clases incrustadas.

/* Guardar datos en Room */

Guardar datos en Room

En Android Room se puede guardar datos en la base de datos utilizando la interfaz de objeto de acceso a datos (DAO) para definir métodos que correspondan a las operaciones que desea realizar.

Aquí hay un ejemplo de cómo guardar datos usando Room:

```
@Entity
data class User(
    @PrimaryKey val id: Int,
    val name: String,
    val email: String
)

@Dao
interface UserDao {
    @Insert
    fun insert(user: User)

    @Update
    fun update(user: User)

    @Delete
    fun delete(user: User)
}
```

Guardar datos en Room

```
val userDao =
AppDatabase.getDatabase(this).userDao()
val user = User(1, "John Doe",
"johndoe@example.com")

userDao.insert(user)
userDao.update(user)
userDao.delete(user)
```

En este ejemplo, tenemos una entidad `User` y una interfaz `UserDao` que contiene tres métodos: **insertar, actualizar y eliminar**. Las anotaciones `@Insert`, `@Update` y `@Delete` indican que estos métodos corresponden a las operaciones SQL correspondientes.

Los métodos de inserción, actualización y eliminación toman un objeto `User` como parámetro y realizan la operación correspondiente en la base de datos.

Guardar datos en Room - TypeConverter

En Room, se usa un **`TypeConverter`** para convertir un objeto personalizado en un tipo conocido que se puede almacenar en una base de datos. Para usar un TypeConverter, primero debe definirlo creando una clase que implemente la interfaz TypeConverter. Esta clase debe tener dos métodos: toRoomType y fromRoomType.

- **`toRoomType`** debe convertir el objeto personalizado en un tipo que se pueda almacenar en la base de datos (por ejemplo, cadena, entero, etc.).
- **`fromRoomType`** debería volver a convertir el tipo de base de datos en el objeto personalizado.

Una vez que haya definido el TypeConverter, debe registrarlo con Room mediante la anotación `@TypeConverter` en la clase.

Demostración "Guardar datos en Room usando TypeConverter"

{desafío} latam_

Guardar datos en Room - TypeConverter

Aquí hay un ejemplo de un TypeConverter que convierte un objeto Date a un valor Long que representa el número de milisegundos en Unix epoch:

```
public class DateConverter {
    @TypeConverter
    public static Long fromDate(Date date) {
        return date == null ? null : date.getTime();
    }
    @TypeConverter
    public static Date toDate(Long timestamp) {
        return timestamp == null ? null : new Date(timestamp);
    }
}
```

Guardar datos en Room - TypeConverter

Luego, debe agregar la clase TypeConverter a la clase de base de datos con:

```
@TypeConverters ({DateConverter.class})

@Database(entities = {User.class}, version = 1)
@TypeConverters({DateConverter.class})
public abstract class AppDatabase extends RoomDatabase {
    // ...
}
```

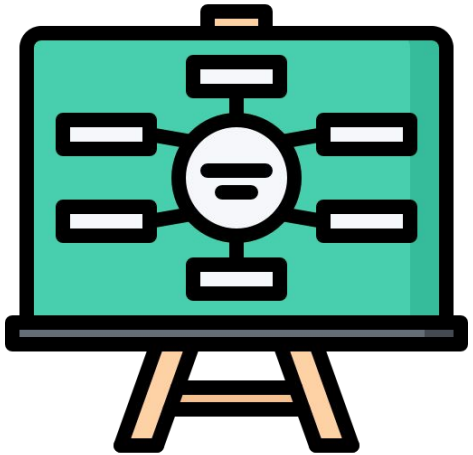
Luego puede usar el objeto personalizado en una clase de Entidad y Room usará automáticamente el TypeConverter para convertir el objeto hacia y desde un tipo compatible con la base de datos al insertar o consultar los datos.

/* El schema de datos en Room */

El schema de datos en Room

- En Android Room, el esquema se refiere a la estructura de la base de datos y sus tablas. El esquema se define utilizando una serie de clases llamadas Entidades, que se asignan a tablas en la base de datos. Cada clase de Entidad define las columnas de la tabla y los tipos de datos que se almacenarán en cada columna.

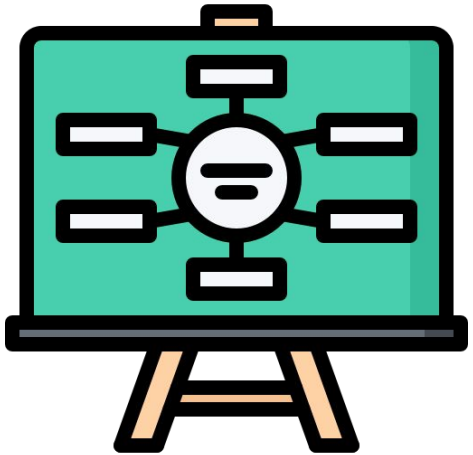
- La biblioteca Room usa anotaciones para definir el esquema, como la anotación @Entity para definir una clase de entidad y la anotación `@ColumnInfo` para definir las columnas de la tabla. La clave principal de la tabla se define mediante la anotación `@PrimaryKey`. Room también usa la anotación `@ForeignKey` para definir relaciones entre tablas.



El schema de datos en Room

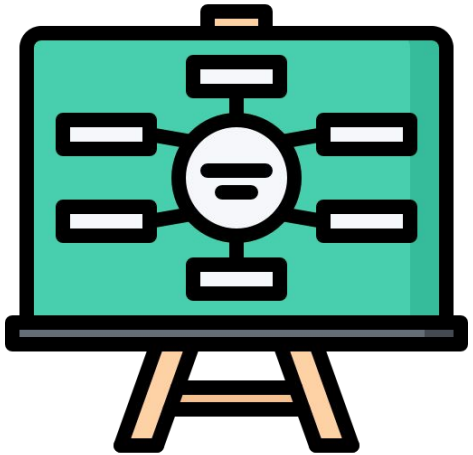
- Cuando se crea la aplicación, Room utiliza la información del esquema para generar el código necesario para crear la base de datos y sus tablas, así como para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los datos.

- Cuando realice algún cambio en el esquema, Room actualizará automáticamente el esquema de la base de datos en la próxima compilación. Room utiliza una técnica llamada "Migración" para manejar los cambios. Es un proceso que permite a Room conservar los datos cuando cambia el esquema. Room verificará la versión actual del esquema, la comparará con la nueva versión del esquema y aplicará los cambios necesarios a la base de datos.



El schema de datos en Room

Es importante tener en cuenta que Room no admite la evolución del esquema durante el tiempo de ejecución, lo que significa que no puede cambiar el esquema mientras se ejecuta la aplicación.



Antes de empezar a desarrollar una app, deberías tener claro el modelo de datos.
Esto simplificará enormemente la creación de tablas y la relación entre ellas.

Próxima sesión...

- *Describe el rol de la capa de abstracción Room y sus principales elementos tales como Entidad y DAO*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales