UT.7.8 BASES DE DATOS. ROOM

Programación Multimedia y Dispositivos Móviles

OBJETIVO

Almacenar información relevante en el dispositivo de forma que, cuando no haya conexión de red, el usuario pueda seguir viéndola fuera de línea

¿CÓMO?

Utilizaremos **Room**, una biblioteca desarrollada por Google, que proporciona una capa de abstracción por encima de la base de datos **SQLite**

DEPENDENCIAS

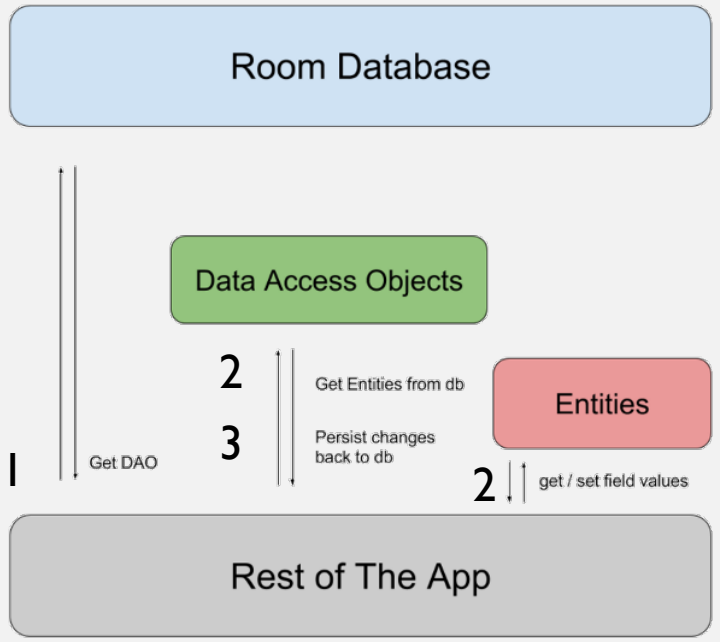
build.gradle del módulo

```
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"
(2.5.0)
```

ELEMENTOS

- **Base de datos**, punto de acceso a los datos
- **Entidades**, representan tablas
- **Data Access Objects** (DAOs), proporcionan métodos para realizar las operaciones *Query*, *Insert*, *Update* y *Delete* en la base de datos

1. La app pide DAOs a la base de datos
2. A través de los DAOs obtiene datos en forma de entidades
3. La app puede además modificar o añadir datos en entidades a través de los DAOs



ENTIDADES

- Anotación *@Entity*
  - Opcional: *tableName* especifica el nombre de la tabla
- *@PrimaryKey* para la clave primaria
- *@ColumnInfo* para columnas de datos
  - Opcional: *name* especifica el nombre que tendrá la columna

```
@Entity(tableName="user")
public class User {
   @PrimaryKey(autoGenerate = true)
   public int uid;

   @ColumnInfo(name = "first_name")
   public String firstName;

   @ColumnInfo(name = "last_name")
   public String lastName;
}
```

ENTITIES

- Claves compuestas, con *primaryKeys* en la @Entity
- Ignorar campos con *@Ignore*

```
@Entity(primaryKeys = {"firstName","lastName"})
public class User {
   public String firstName;
   public String lastName;

   @Ignore
   Bitmap picture;
}
```

DAO

- Definidas como interfaces
  - Room genera implementaciones en tiempo de ejecución
  - Se pueden generar implementaciones *mock* para los tests
- *@Query* utiliza ':' para sustituir valores pasados como parámetros

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    User findByName(String first, String last);

    @Insert
    void insertAll(User... users);

   @Update
   void update(User user);

    @Delete
    void delete(User user);
}
```

BASE DE DATOS

```
@Database(entities = {User.class}, version =
1)
public abstract class AppDatabase extends
RoomDatabase {
    public abstract UserDao userDao();
}
```

- Anotado con *@Database* a la que se pasa la lista de entidades que gestiona
- La clase debe ser abstracta y heredar de *RoomDatabase*
- Para cada DAO, definir exactamente un método sin parámetros que devuelve dicho DAO

USO

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "database-name").build();
UserDao userDao = db.userDao();
List<User> users = userDao.getAll();
User first = users.get(0);
first.firstName = "Jacinta";
userDao.update(first);
```

CONSULTAS ASÍNCRONAS

ROOM

CONSULTAS ASÍNCRONAS

- No se puede llamar a Room en el hilo principal ☹
- Podemos utilizar consultas asíncronas de un solo uso
- Dependencia con room-rxjava3, rxjava y rxandroid
  - `implementation "androidx.room:room-rxjava3:$room_version“`
  - `implementation 'io.reactivex.rxjava3:rxjava:3.1.6'`
  - `implementation "io.reactivex.rxjava3:rxandroid:3.0.2"`
- Modificar
  - Métodos DAO que devuelven `void` a devolver `Completable`
  - Métodos DAO que devuelven un resultado de un `tipo` a devolver `Single<tipo>`

CONSULTAS ASÍNCRONAS DE UN SOLO USO

- *Completable*, *Single* son *observables*
- Se ejecutan en un hilo determinado por *subscribeOn()*
  - Usamos uno que no corresponda con el hilo principal
- El resultado se devuelve en un hilo determinado por *observeOn()*
  - Elegimos el hilo principal que es el único que puede modificar componentes de UI
- Cuando se completa la operación se llama al *Consumer* (para *Single*) o *Action* (para *Completable*) pasado al método *subscribe()*

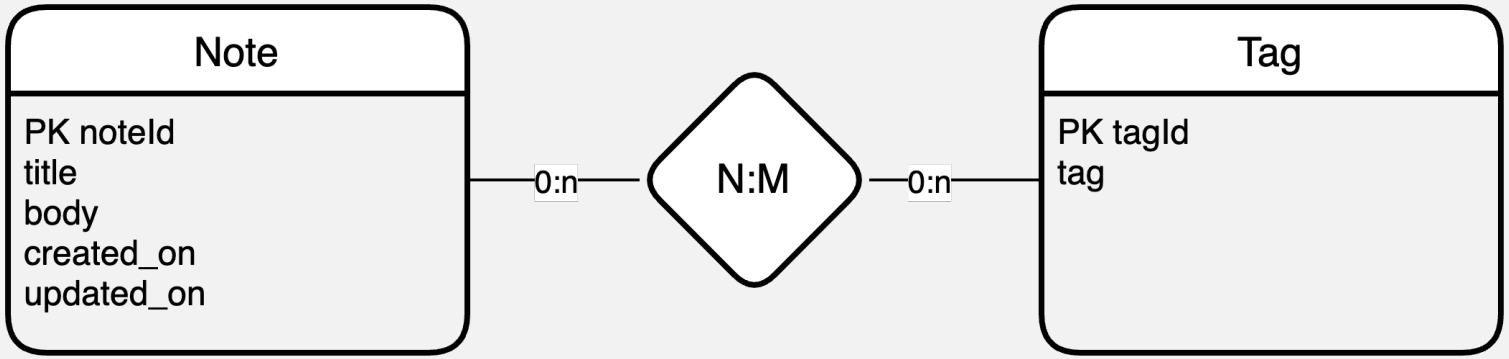
```
userDao()
.getAll()
.subscribeOn(Schedulers.io())
.observeOn(AndroidSchedulers.mainThread())
.subscribe(
  new Consumer<List<Student>>() {
    @Override
    public void accept(List<Student> students)
      throws Throwable {
    }
});
```

INTERRELACIONES

ROOM

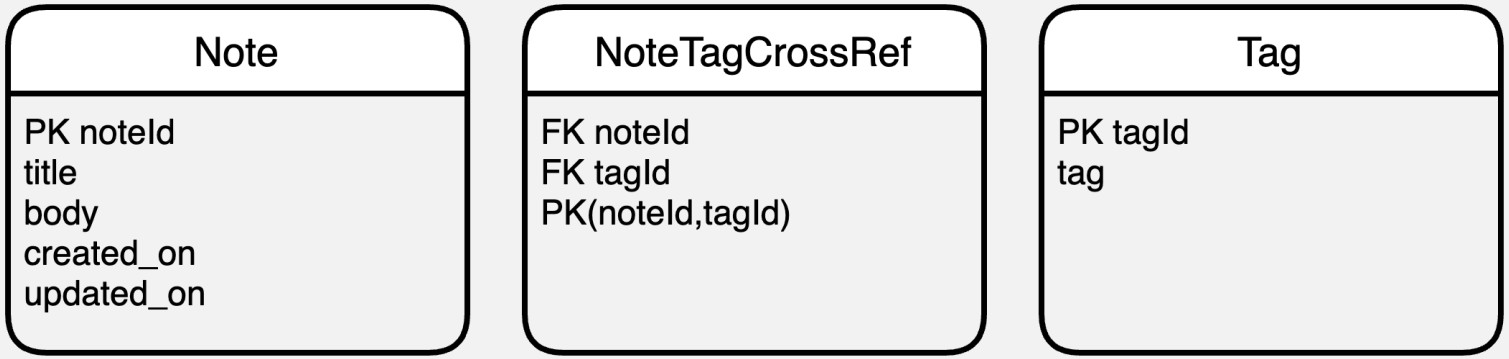
INTERRELACIONES

- Con Room se pueden modelar todos los tipos de interrelaciones en bases de datos
  - One-to-one
  - One-to-many
  - Many-to-many
- Nos centramos en estas últimas por su mayor complejidad



RELACIÓN MANY-TO-MANY

- Una nota puede tener varias etiquetas asociadas
- Una etiqueta puede estar asociada a varias notas



RELACIÓN MANY-TO-MANY

- Se implementa con una table intermedia de referencias cruzadas entre ambas tablas
- La clave primaria está compuesta de las claves foráneas que referencias las claves primarias de las tablas referenciadas 🤓

ENTIDADES NOTA Y ETIQUETA

```
@Entity(tableName = "notes")
public class Note {
    @PrimaryKey(autoGenerate = true)
    public int noteId;
    public String title;
    public String body;
}
```

```
@Entity(tableName = "tags")
public class Tag {
    @PrimaryKey(autoGenerate = true)
    public int tagId;
    public String tag;
}
```

TABLA INTERMEDIA

```
@Entity(primaryKeys = {"noteId", "tagId"})
public class NoteTagCrossRef {
    public int noteId;
    @ColumnInfo(index = true)
    public int tagId;
}
```

- La clave primaria está compuesta por noteId y tagId
- @ColumnInfo(index = true) hace que se cree un índice para esa columna (Room no lo hace por defecto)

OJO, AÑADIR TODAS LAS ENTIDADES

```
@Database(entities = {Note.class, Tag.class, NoteTagCrossRef.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {
    public abstract NotesDao notesDao();
    public abstract TagsDao tagsDao();
}
```

NOTA CON ETIQUETAS

```
public class NoteWithTags {
    @Embedded
    public Note note;
    @Relation(
            parentColumn = "noteId",
            entityColumn = "tagId",
            associateBy = @Junction(NoteTagCrossRef.class)
    )
    public List<Tag> tags;
}
```

ETIQUETA CON NOTAS

```
public class TagWithNotes {
    @Embedded
    public Tag tag;
    @Relation(
            parentColumn = "tagId",
            entityColumn = "noteId",
            associateBy = @Junction(NoteTagCrossRef.class)
    )
    public List<Note> notes;
}
```

DAOS

```
@Dao
public interface NotesDao {

    @Query("SELECT * FROM notes")
    Single<List<Note>> getAll();

    @Query("SELECT * FROM notes WHERE noteId = :noteId")
    Single<Note> find(int noteId);

    @Transaction
    @Query("SELECT * FROM notes")
    Single<List<NoteWithTags>> getNotesWithTags();

    @Transaction
    @Query("SELECT * FROM notes WHERE noteId = :noteId")
    Single<NoteWithTags> findWithTags(int noteId);

    @Insert
    Completable insertNote(Note note);

    @Update
    Completable updateNote(Note note);

    @Delete
    Completable deleteNote(Note note);
}
```

```
@Dao
public interface TagsDao {

    @Query("SELECT * FROM tags")
    Single<List<Tag>> getAll();

    @Query("SELECT * FROM Tags WHERE TagId = :tagId")
    Single<Tag> find(int tagId);

    @Transaction
    @Query("SELECT * FROM tags")
    Single<List<TagWithNotes>> getTagsWithNotes();

    @Transaction
    @Query("SELECT * FROM tags WHERE TagId = :tagId")
    Single<TagWithNotes> findWithTags(int tagId);

    @Insert
    Completable insertTag(Tag Tag);

    @Update
    Completable updateTag(Tag Tag);

    @Delete
    Completable deleteTag(Tag Tag);
}
```