Room

Cómo guardar contenido en una base de datos local con Room

Room proporciona una capa de abstracción sobre SQLite que permite acceder a la base de datos sin problemas y, al mismo tiempo, aprovechar toda la potencia de SQLite.

Las apps que controlan grandes cantidades de datos estructurados pueden beneficiarse con la posibilidad de conservar esos datos localmente. El caso práctico más común es almacenar en caché datos relevantes. De esa manera, cuando el dispositivo no puede acceder a la red, el usuario de todos modos puede explorar ese contenido mientras está desconectado. Cualquier cambio de contenido iniciado por el usuario se sincroniza con el servidor una vez que el dispositivo vuelve a estar en línea.

Como Room se ocupa de estas inquietudes por ti, te **recomendamos** utilizar Room en lugar de SQLite. Sin embargo, si prefieres usar las API de SQLite directamente, lee [Cómo guardar datos mediante SQLite](https://developer.android.com/training/data-storage/sqlite.html).

**Nota**: Para usar Room en tu app, [declara las dependencias de Room](https://developer.android.com/jetpack/androidx/releases/room#declaring_dependencies) en el archivo **build.gradle**.

Estos son los 3 componentes principales de Room:

* [**Base de datos**](https://developer.android.com/reference/androidx/room/Database.html): Contiene el titular de la base de datos y sirve como punto de acceso principal para la conexión subyacente a los datos persistentes y relacionales de tu app.

La clase anotada con [@Database](https://developer.android.com/reference/androidx/room/Database.html) debería cumplir las siguientes condiciones:

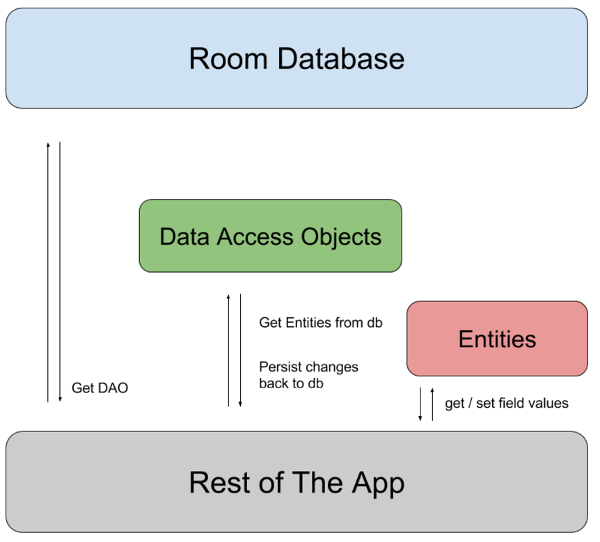
* + Ser una clase abstracta que extienda [RoomDatabase](https://developer.android.com/reference/androidx/room/RoomDatabase.html)
  + Incluir la lista de entidades asociadas con la base de datos dentro de la anotación
  + Contener un método abstracto que tenga 0 argumentos y muestre la clase anotada con [@Dao](https://developer.android.com/reference/androidx/room/Dao.html)

Durante el tiempo de ejecución, puedes adquirir una instancia de [Database](https://developer.android.com/reference/androidx/room/Database.html) llamando a [Room.databaseBuilder()](https://developer.android.com/reference/androidx/room/Room.html" \l "databaseBuilder(android.content.Context,%20java.lang.Class%3CT%3E,%20java.lang.String)) o [Room.inMemoryDatabaseBuilder()](https://developer.android.com/reference/androidx/room/Room.html" \l "inMemoryDatabaseBuilder(android.content.Context,%20java.lang.Class%3CT%3E)).

* [**Entidad**](https://developer.android.com/training/data-storage/room/defining-data.html): Representa una tabla dentro de la base de datos.
* [**DAO**](https://developer.android.com/training/data-storage/room/accessing-data.html): Contiene los métodos utilizados para acceder a la base de datos.

La app usa la base de datos de Room para obtener los objetos de acceso a los datos (DAO) asociados con esa base de datos. Luego, la app usa cada DAO para obtener entidades de la base de datos y guardar los cambios realizados en esas entidades en la base de datos. Por último, la app usa una entidad para obtener y configurar valores que corresponden a columnas de tabla dentro de la base de datos.

Esta relación entre los diferentes componentes de Room aparece en la figura 1:

**Figura 1:** Diagrama de arquitectura de Room

En el siguiente fragmento de código, se muestra una configuración de base de datos de ejemplo con una entidad y un DAO:

User

[KOTLIN](https://developer.android.com/training/data-storage/room#kotlin)[JAVA](https://developer.android.com/training/data-storage/room#java)

    @Entity  
    public class User {  
        @PrimaryKey  
        public int uid;  
  
        @ColumnInfo(name = "first\_name")  
        public String firstName;  
  
        @ColumnInfo(name = "last\_name")  
        public String lastName;  
    }

UserDao

[KOTLIN](https://developer.android.com/training/data-storage/room#kotlin)[JAVA](https://developer.android.com/training/data-storage/room#java)

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

AppDatabase

[KOTLIN](https://developer.android.com/training/data-storage/room#kotlin)[JAVA](https://developer.android.com/training/data-storage/room#java)

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

Después de crear los archivos anteriores, obtendrás una instancia de la base de datos creada con el siguiente código:

[KOTLIN](https://developer.android.com/training/data-storage/room#kotlin)[JAVA](https://developer.android.com/training/data-storage/room#java)

    AppDatabase db = Room.databaseBuilder(getApplicationContext(),  
            AppDatabase.class, "database-name").build();

**Nota**: Si tu app se ejecuta en un solo proceso, debes seguir el patrón de diseño único cuando creas instancias de un objeto **AppDatabase**. Cada instancia de **[RoomDatabase](https://developer.android.com/reference/androidx/room/RoomDatabase.html)** es bastante costosa, pero rara vez necesitas acceso a varias instancias dentro de un solo proceso.

Si tu app se ejecuta en múltiples procesos, incluye **enableMultiInstanceInvalidation()** en tu invocación del generador de bases de datos. De esta manera, cuando tienes una instancia de **AppDatabase** en cada proceso, puedes invalidar el archivo de base de datos compartida en un proceso y esta invalidación se propaga automáticamente a las instancias de **AppDatabase** dentro de los otros procesos.