**Building database with Room Persistence Library**

Hello everyone! Last week at Google IO 2017, Google introduced Room persistence library which allows fluent database access while harnessing the full power of SQLite. Basically, it is abstraction layer over SQLite.

**Room is a new way to create a database in your android apps, it is much similar [OrmLite](http://ormlite.com/" \t "_blank).**

*The core framework provides built-in support for working with raw SQL content. Although these APIs are powerful, they are fairly low-level and require a great deal of time and effort to use:*

*There is no compile-time verification of raw SQL queries.*

*As your schema changes, you need to update the affected SQL queries manually. This process can be time consuming and error prone.*

*You need to write lots of boilerplate code to convert between SQL queries and Java data objects.*

*Room takes care of these concerns for you while providing an abstraction layer over SQLite.*

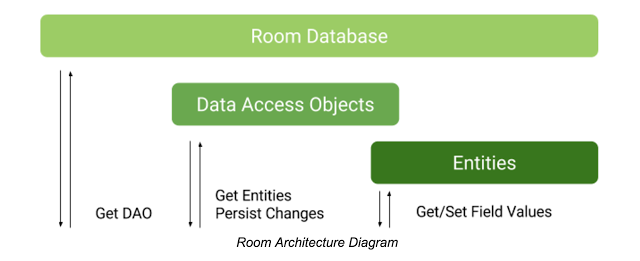
***Database, Entity, DAO***

*There are three major components in Room:*

***Entity****represents data for a single table row, constructed using an annotated java data object. Each entity is persisted into its own table.*

***DAO****(Data Access Object) defines the method that access the database, using annotation to bind SQL to each method.*

***Database****is a holder class that uses annotation to define the list of entities and database version. This class content defines the list of DAOs.*



Let’s implement an example to have more understanding how it works. In this example, we will create an app that allows to save user details ex first name, last name, age. You can see full source code [here](https://github.com/ajaysaini-sgvu/room-persistence-sample).

Alright! So far so good!!

1. Create a new project in Android Studio with empty activity.
2. Add this dependencies in module *build.gradle*

compile 'android.arch.persistence.room:runtime:' + rootProject.archRoomVersion;  
annotationProcessor 'android.arch.persistence.room:compiler:' + rootProject.archRoomVersion;

Define project dependencies and add Google Maven repository in root *build.gradle*

ext {  
 buildToolsVersion = "25.0.2"  
 supportLibVersion = "25.3.1"  
 archRoomVersion = "1.0.0-alpha1"  
}

allprojects {  
 repositories {  
 jcenter()  
 maven { url 'https://maven.google.com' }  
 }  
}

3. Now, lets create an entity called User. It defines a attributes of your table, it is must to declare one field as primary key. It has property to auto generate values. Class is annotated with **@Entity** and name of the table. To make field primary key, you need to annotate a field with **@PrimaryKey** and property **autoGenerate**which assign automatic IDs. Room will create a user table with defined attributes.

@Entity(tableName = "user")  
public class User {  
  
 @PrimaryKey(autoGenerate = true)  
 private int uid;  
  
 @ColumnInfo(name = "first\_name")  
 private String firstName;  
  
 @ColumnInfo(name = "last\_name")  
 private String lastName;  
  
 @ColumnInfo(name = "age")  
 private int age;  
  
 public int getUid() {  
 return uid;  
 }  
  
 public void setUid(int uid) {  
 this.uid = uid;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
}

4. Create a data access object using an interface. This class is annotated with **@Dao** annotation. Room will generate an implementation of defined methods. There are four annotations **@Query**, **@Insert**, **@Update**, **@Delete**to perform CRUD operations. @Query annotation is used to perform read operation on database.

@Dao  
public interface UserDao {  
  
 @Query("SELECT \* FROM user")  
 List<User> getAll();  
  
 @Query("SELECT \* FROM user where first\_name LIKE :firstName AND last\_name LIKE :lastName")  
 User findByName(String firstName, String lastName);  
  
 @Query("SELECT COUNT(\*) from user")  
 int countUsers();  
  
 @Insert  
 void insertAll(User... users);  
  
 @Delete  
 void delete(User user);  
}

5. Create a database holder called AppDatabase extends RoomDatabase, we will define list of entities and database version. Class is annotated with **@Database** annotation. It is good practice to use singleton approach for the database, so you need to create an static method which will return instance of AppDatabase.

@Database(entities = {User.class}, version = 1)  
public abstract class AppDatabase extends RoomDatabase {  
  
 private static AppDatabase *INSTANCE*;  
  
 public abstract UserDao userDao();  
  
 public static AppDatabase getAppDatabase(Context context) {  
 if (*INSTANCE* == null) {  
 *INSTANCE* =  
 Room.*databaseBuilder*(context.getApplicationContext(), AppDatabase.class, "user-database")  
 // allow queries on the main thread.  
 // Don't do this on a real app! See PersistenceBasicSample for an example.  
 .allowMainThreadQueries()  
 .build();  
 }  
 return *INSTANCE*;  
 }  
  
 public static void destroyInstance() {  
 *INSTANCE* = null;  
 }  
}

Perfect!! You are done! Now you can add users in the user table of the database. You must perform queries on worker thread, otherwise your application will crash.

private static User addUser(final AppDatabase db, User user) {  
 db.userDao().insertAll(user);  
 return user;  
}  
  
private static void populateWithTestData(AppDatabase db) {  
 User user = new User();  
 user.setFirstName("Ajay");  
 user.setLastName("*Saini*");  
 user.setAge(25);  
 *addUser*(db, user);  
}

Nevertheless, it gives you an option to execute queries on main thread by calling `allowMainThreadQueries` method on RoomDatabase Builder. Don’t do this on production!

You can find complete source code [here](https://github.com/ajaysaini-sgvu/room-persistence-sample)

Please, if you find something to improve or any suggestion, don’t hesitate to contact me, I’ll try to do my best to answer any question or improve this tutorial.