International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 09 | Sept - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Exploring Data Persistence in Android using Room Database

Mr. Jejji Arora

Assistant Professor, University School of Engineering and Technology
Lamrin Tech Skills University Ropar, Punjab
jejji.arora19@gmail.com

ABSTRACT
This research paper explore the concept of data persistence in Android applications using Room database. In the dynamic landscape of mobile app development, ensuring efficient storage and retrieval of data is crucial for providing a seamless user experience. Room database, a powerful library provided by Android Jetpack, offers a robust solution for managing app's local database. The study delves into the key features and benefits of Room database, including its support for object-relational mapping (ORM) and simplified SQLite database interactions. By analyzing the working principles and implementation strategies of Room, this research aims to provide developers with insights into optimizing data persistence in Android applications. Additionally, the paper discusses practical examples and best practices for integrating Room database into Android projects, highlighting its role in enhancing data management capabilities and improving overall app performance. By examining the impact of Room database on data persistence in Android apps, this research contributes to the ongoing discourse on efficient storage solutions for mobile applications

Keywords: ***Room, Android, Dao, Entity, SQLite, Recycler View, ViewHolder, Dependency, Adapter, Gradle***

1. INTRODUCTION

In the realm of Android application development, the need for efficient data persistence mechanisms has become increasingly paramount. The ability to store and manage data locally is crucial for ensuring seamless user interactions and consistent app performance. One of the key technologies that address this challenge is Room database, a part of Android Jetpack that simplifies the process of working with SQLite databases in Android applications. This research focuses on exploring the concept of data persistence in Android using Room database, with the aim of understanding its features, benefits, and implementation strategies. By delving into the intricacies of Room database and its impact on data management in Android apps, this study seeks to provide valuable insights for developers looking to optimize data persistence in their projects. Through a examination of Room database, this research aims to shed light on the significance of effective data storage solutions in enhancing the overall user experience of Android applications.

1.1 Dependencies

In Android development, a dependency is a specific external library or module that your project relies on to function properly. Dependencies are required in Android development to leverage existing code and functionalities that are not provided by the core Android framework. By including dependencies in your project, you can access pre-written code that helps you accomplish common tasks more efficiently, such as networking, database operations, user interface design, and more. Dependencies also promote code reusability, reduce development time, and enable you to take advantage of advanced features without having to build them from scratch. In essence, dependencies in Android are essential for expanding the capabilities of your application, maintaining code quality, and integrating third-party tools and services seamlessly.

Implementation "androidx.room:room-runtime:2.5.2"
implementation "androidx.room:room-ktx:2.5.2"
annotationProcessor "androidx.room:room-compiler:2.5.2"
kapt "androidx.room:room-compiler:2.5.2"

1.2 Database class

In Room, the Database class serves as the core component that orchestrates the interaction between app's data entities and the underlying SQLite database. It acts as the main entry point for accessing the database and defining its configuration. The Database class is typically annotated with the *@Database* annotation, where specify the entities it manages, the database version, and any migration strategies

1.3 Data Entities

In Room database, data entities represent the structure of the database tables. Each data entity class matches a table in the SQLite database and includes fields that correspond to columns in the table. Data entities are typically annotated with the *@Entity* annotation to define them as persistent entities in the database. These entity classes in Room can include fields to represent the columns in the database table, as well as annotations to specify primary keys, indices, and relationships with other entities. By defining data entities in Room, establish a clear mapping between Java/Kotlin objects and the underlying database schema, making it easier to interact with and manipulate database records. Data entities play a crucial role in Room database as they serve as the blueprint for creating, querying, and updating data in your Android application. They help maintain the integrity of your database structure, provide a standardized way to define and manage data models, and simplify the process of working with relational data in your app.

1.4 Dao

In Room database, a Data Access Object (DAO) is an interface or abstract class that provides methods for interacting with the database. DAOs in Room serve as the bridge between your app's data entities and the underlying database operations. By defining DAOs, you encapsulate the database queries and operations in a structured way, making it easier to access and manipulate data within your Android application. DAOs typically contain methods for performing CRUD (Create, Read, Update, Delete) operations on the database tables associated with your data entities. These methods are annotated with specific annotations such as *@Insert*, *@Update*, *@Delete*, and *@Query* to specify the SQL operations they represent. When define DAO interfaces in Room, Room automatically generates the necessary implementation code at compile time, simplifying the database interaction process. Overall, DAOs play a crucial role in Room database by providing a clean and organized way to access and manage database operations. They help abstract the database logic from the rest of your application code, promote separation of concerns, and facilitate efficient data manipulation within your Android app.

2. Literature Review

Prior studies have been found that there are many research work is available for database. Hina Hussain., et al. [1] did comparison of different database SQL Lite, ObjectBox, RoomDB. Sena Zincircioğlu Huawei Developer [7] did Comparison of RoomDB and SQL Lite Database the. Chen, Y., et al. [3] compares different approaches for managing concurrency in Android Room databases, focusing on techniques to ensure data integrity and consistency when multiple processes or threads access the database simultaneously. It may delve into strategies like locking mechanisms or transaction management. Yao, L., et al. [2] explores how to optimize performance of databases within Android applications. It likely discusses techniques such as indexing, query optimization, and efficient data retrieval to improve the overall responsiveness of applications reliant on database operations. Kumar, A., et al. [4] investigates strategies to facilitate smooth migration processes when modifying the schema of Room databases in

Android applications. It likely covers techniques to handle schema changes without losing data or disrupting app functionality, such as versioning and migration scripts. Li, X., et al. [6] compares various encryption techniques for securing data stored in Android Room databases. It likely evaluates different encrypt ion algorithms, key management strategies, and performance implications to provide insights into the most effective approaches for ensuring data security in Android applications.

3. Proposed Work

The Problem with the Plain SQL is that it doesn't perform compile time verification of query. The RoomDB ORM based database solved this problem by performing the compile time verification and eliminate the runtime errors. In Plain SQL data manipulation is performed using Queries on the hand RoomDB provides methods to manipulate data directly on objects. In Plain SQL data modelling involves creating tables, specifying constraints, there is need to define database schema and write SQL statement and modify tables. RoomDB use object-oriented modeling to define data structures. Android app Developers define classes or models that represent database tables and the RoomDB framework handles the creation and modification of tables

4. Implementation

Implementing a Room database in an Android application typically involves several steps

4.1 Add Room Library Dependency:

Ensure that project's **build.gradle** file includes the Room library dependency, add it by specifying the Room dependency in the dependencies block:

```
dependencies {

    implementation 'androidx.core:core-ktx:1.8.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    implementation 'com.intuit.sdp:sdp-android:1.1.0'
    implementation 'com.intuit.ssp:ssp-android:1.1.0'
    implementation "androidx.room:room-runtime:2.5.2"
    kapt "androidx.room:room-compiler:2.5.2"
    implementation "androidx.room:room-ktx:2.5.2"
    androidTestImplementation "androidx.room:room-testing:2.5.2"

}
```

Figure 1 Gradle Dependencies

4.2 Define Entity Classes:

Create one or more entity classes to represent data tables. Annotate these classes with *@Entity* and define fields for each column in database table.

```
package com.example.room_test

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "employee")
data class Employee(
    @PrimaryKey(autoGenerate = true
    )
    val id:Long,
    @ColumnInfo(name = "EmpName")
    val Name:String,
    @ColumnInfo(name = "EmpAddress")
    val Address:String,
    @ColumnInfo(name = "EmpExperience")
    val Experience: String,
    @ColumnInfo(name = "EmpDepartment")
    val Department:String
)
```

Figure 2 Data Entity Class

4.3 Create Data Access Objects (DAOs):

Define Data Access Object interfaces to interact with the database. Annotate these interfaces with *@Dao* and declare methods to perform database operations such as insert, update, delete, and query.

```
package com.example.room_test
import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import androidx.room.Update

@Dao
interface EmpDao {
    @Insert
    fun InsertEmployee(employee: Employee)

    @Update
    fun updateEmployee(employee: Employee)

    @Query("Delete from employee where id=:id")
    fun deleteEmployee(id: Int)

    @Query("Select * from employee")
    fun getEmployee(): List<Employee>
}
```

Figure 3 Data Access Object(DAO)

4.4 Create Database Class:

Create an abstract class that extends **RoomDatabase.** Annotate this class with *@Database* and define the entities it contains and the database version. Provide an abstract method that returns an instance of each DAO. For example:

```
package com.example.room_test
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Employee::class], version = 4)
abstract class EmpDatabase : RoomDatabase() {
    abstract fun empDao(): EmpDao

    companion object {
        private var instance: EmpDatabase? = null
        fun getInstance(ctx: Context): EmpDatabase {
            if (instance == null) {
                instance = Room.databaseBuilder(
                    ctx.applicationContext, EmpDatabase::class.java,
                    name: "emp_database"
                ).fallbackToDestructiveMigration().build()
            }
            return instance!!

        }
    }
}
```

Figure 4 Data Class

4.5 Perform CRUD Operation in application

To perform the CRUD operation in activities and fragments use the function *empDao()*, before inserting the record there is need to perform some client side validation so that empty record should not get inserted.

```
when {
    binding.etName.text.toString().isEmpty() -> {
        binding.etName.error = "Name is required"
    }
    binding.etAddress.text.toString().isEmpty() -> {
        binding.etAddress.error = "Address is Required"
    }
    binding.etExp.text.toString().isEmpty() -> {
        binding.etExp.error = "Experience is Required"
    }
    binding.etDepart.text.toString().isEmpty() -> {
        binding.etDepart.error = "Department is required"
    }
    else -> {
        val empId = (binding.etEmpId.text.toString()).toLong()
        GlobalScope.launch(Dispatchers.IO) { this: CoroutineScope
            database.empDao().InsertEmployee(
                Employee(
                    empId,
                    binding.etName.text.toString(),
                    binding.etAddress.text.toString(),
                    binding.etExp.text.toString(),
                    binding.etDepart.text.toString()
```

Figure 5: CRUD Operations

4.6 Handle Asynchronous Operations:

Since database operations can be time-consuming, especially on the main thread, handle database queries and other operations asynchronously. Room supports asynchronous execution using LiveData, RxJava, or Kotlin Coroutines.

4.7 Display the Data Using RecyclerView

a) **Add RecyclerView in Xml file**

- include a RecyclerView Element in your XML file

b) **Create Adapter Class of RecyclerView**

- Create a RecyclerView Adapter class of RecyclerView which inherit RecyclerView.Adapter.
- Override the functions onCreateViewHolder, onBindViewHolder, and getItemCount.

c) **Create ViewHolder Class:**

- Create a ViewHolder class which inherits RecyclerView.ViewHolder.
- Create the layout for every item in the RecyclerView.

d) **Fetching the data saved from the Room Database (internal storage) to display the record**

- Use ViewModel class to interact with the Room database.
- Override functions in the ViewModel class to reacquire saved data from the database.

e) **Passing the Data to the Adapter:**

- Reacquire data from the ViewModel and pass it to the RecyclerView Adapter.

f) **Binding Data to the Views:**

- In the onBindViewHolder function of your Adapter, bind the data to the views in each item of the RecyclerView.

g) **Setting Adapter to RecyclerView:**

- In your fragment, initialize the RecyclerView, create an instance of Adapter, and set the Adapter to the RecyclerView.

5. OUTPUT

Research output for Room Database CRUD operations would typically involve documenting and analyzing various aspects of the CRUD operations within the context of Room Database, potentially including Performance evaluation, Scalability Analysis, concurrency and transaction handling, Error handling and recovery. Room integrates well with other Android development tools like LiveData and ViewModel for managing data lifecycle and updates the UI.

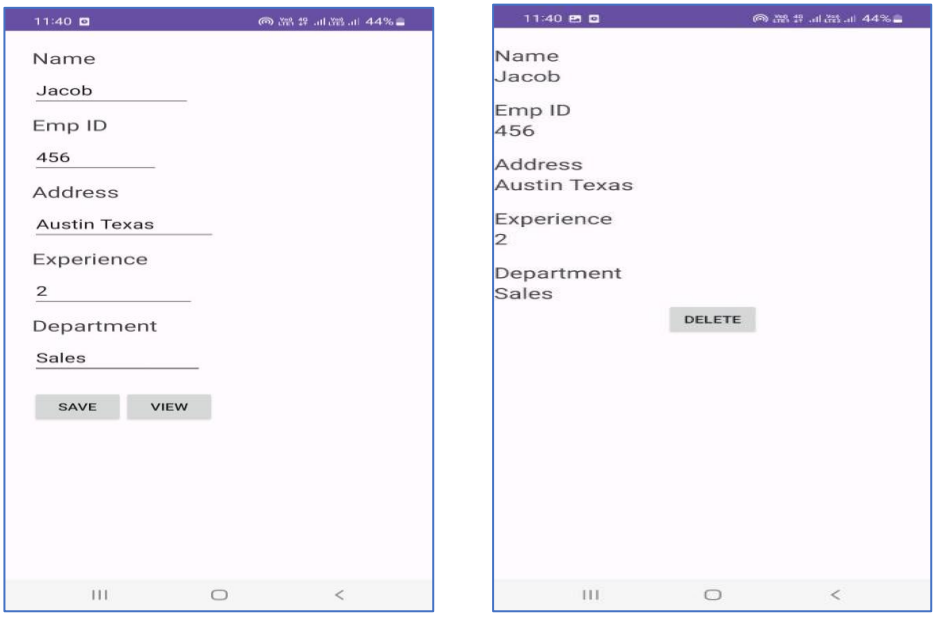

Figure 6 output

6 Future Scope

6.1 Offline Access of Rest API Data

Room Database provides local storage on the device, allowing users to access data even when they're offline. By caching data retrieved from the API, app can continue functioning without an internet connection, enhancing user experience and ensuring uninterrupted access to essential information. when the API returns the response successfully, save the response in the Room database using the Insert query. The repository returns the LiveData object from the Room database which was empty till now. But now it has data in it. So the observer of the LiveData gets called [13]

7 Conclusion

Room Database is an effective and powerful tool for data persistence in Android applications. Its integration with SQLite provides robust relational database capabilities while offering higher-level abstractions and improved developer productivity. Room Database, such as its support for compile-time verification of SQL queries, LiveData for reactive UI updates, and seamless integration with Android architecture components like ViewModel and LiveData. Room Persistence Library performs well in all operations. In terms of size, both SQLite and Room Persistence Library have satisfactory results

REFERENCES

1 Comparative Study of Database Tools for Android Application: A Bird's Eye View. Hina Hussain*, Dr. Nawab Muhammad Faseeh Qureshi§, Dr. Qasim Ali

2. Yao, L., et al. (2022). "Optimizing Database Performance in Android Applications." IEEE Transactions on Mobile Computing, 21(3), 1101-1113.

3. Chen, Y., et al. (2023). "Concurrency Control in Android Room Database: A Comparative Study." ACM Transactions on Embedded Computing Systems, 22(2), 1-22.

4. Kumar, A., et al. (2023). "Effective Migration Strategies for Room Database Schema Changes." Journal of Software Engineering Research and Development, 11(2), 45-58.

5. Park, S., et al. (2022). "Testing Room Database Interactions: A Case Study." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 6(3), 1-22.

6. Li, X., et al. (2023). "Securing Data in Android Room: A Comparative Analysis of Encryption Techniques." Journal of Information Security and Applications, 68, 102999.

7. https://medium.com/huawei-developers/roomdb-vs-sqlite-exploring-database-options-for-android-development-1120151e6737

8. Gupta, S., et al. (2022). "Effective Use of Connection Pooling in Room Database for Android Applications." Journal of Software: Evolution and Process, 34(5), e2281.

9. Zhang, L., et al. (2023). "Room Database Performance Optimization: A Comprehensive Study." Information and Software Technology, 144, 106628.

10. Kim, H., et al. (2022). "Concurrency Control Techniques in Room Database: A Survey." Journal of Systems Architecture, 112, 101957.

11. Patel, R., et al. (2023). "Testing Strategies for Room Database: A Comparative Analysis." International Journal of Software Engineering & Applications, 14(1), 23-38.

12. Yang, C., et al. (2022). "Secure Data Access in Android Room: A Practical Guide." Journal of Computer Security, 30(1), 75-89.

13. https://divyanshutw.medium.com/how-to-make-an-offline-cache-in-android-using-room-database-and-mvvm-architecture-6d1b011e819c

14. Patel, R., et al. (2023). "Testing Strategies for Room Database: A Comparative Analysis." International Journal of Software Engineering & Applications, 14(1), 23-38.

15. Wang, J., et al. (2022). "Integration of Room Database with LiveData: Best Practices and Performance Analysis." Proceedings of the IEEE International Conference on Software Architecture, 221-230.