ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

UNRAVELING DATABASE CHOICES IN ANDROID STUDIO: A COMPARATIVE ANALYSIS OF SQLITE AND ROOM PERSISTENCE LIBRARY

Meer Usman Amjad
su92-mspmw-f24-002@superior.edu.pk

Mudassir Abbas
su92-mspmw-s24-007@superior.edu.pk

Muhammad Ramzan Yasin
ramzan4167768@gmail.com

Yusra Shafiq
yusrashafiq99@gmail.com

Abstract
A mobile app heavily relies on user data. If an app doesn't address user inquiries effectively it may struggle to stay competitive in the market. The blame, for this setback lies with the developer. The demand for Android apps is on the rise. This study delves into. Contrasts two database systems. Room Persistence Library. To complement existing research in this area we carried out experiments by setting up both databases and executing queries. We aimed to determine which database is more user-friendly and efficient. This study proves beneficial for developers seeking to make decisions regarding the database for their projects. It offers insights, into how each database operates and responds aiding developers in grasping their functionality and performance levels.

Keywords: SQLite, Room Persistence Library, MVVM, Android Studio, Database, Java, CURD Operation,

1. Introduction
The (Holla and Katti, 2012) world has seen a significant shift from websites to mobile applications. Android Studio plays a vital role in developing mobile applications. It gets updated frequently, introducing new versions regularly. Android Studio offers a variety of databases for applications. For developers it can be tricky to determine the database, in terms of efficiency, reliability, scalability, backup, recovery and security (Pujari et al., 2020). When it comes to Android Studio and local databases choosing the " practices" for database selection can be subjective and influenced by factors like the applications needs, complexity and performance requirements. While developers may have preferences based on their experiences a research study that compares the behavior and impact of databases such as SQLite and the Room persistence library can offer objective insights and empirical evidence. Through an assessment of performance, efficiency, scalability, data integrity, security aspects among others for both databases. This study aims to provide results that can assist developers in making decisions when selecting the most suitable database for their Android applications based on data-driven evidence rather than personal opinions. This research could greatly benefit the Android development community by offering a guide backed by evidence to help optimize application performance and enhance user experience through selecting a local database solution, for their projects. To tackle this issue we evaluated the performance of two databases SQLite and the Room persistence library. This study aims to provide reassurance to readers regarding the efficiency of these databases enabling them to make informed choices for their apps.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

2. Literature Review

(Dobslaw and Dobslaw n.d) conducted a study comparing Room and Green DAO databases in Android applications. They focused on assessing the performance of CRUD operations in terms of time efficiency, resource consumption (RAM and CPU usage), and the size of the apps. The results indicated that Green DAO outperformed Room and SQLite, for CRUD operations. Nonetheless, Room demonstrated performance for databases containing up to 5000 entries. Is suggested for managing relational data with cascading requirements. On the other hand, Green DAO is recommended for handling non-relational data. The measurements were carried out using an Android Virtual Device (AVD) to ensure outcomes. The primary objective of the research was to investigate performance variances between Room and Green DAO with SQLite serving as a benchmark, for app size comparisons. In their paper, (Obradovic et al., 2019) analyze the performance of the SQLite database concerning fundamental data operations, namely CRUD. The study explores whether SQLite can meet the requirements of modern applications and primarily concentrates on the following three aspects: database operations on unencrypted data, operations on encrypted data, and concurrent access to the database. The paper's findings show how well the SQLite database performs when dealing with big sets of data. (Lyu et al., 2017) this paper presents a comprehensive study of mobile app database behaviors using dynamic and static analysis techniques. It delves into query types, performance implications, and problematic construction patterns. In summary, the research offers practical guidance for app developers and outlines potential areas for future exploration in the field of software engineering.

(Gyorodi et al., 2015) conducted quantitative research on the comparative analysis of two databases MongoDB and MySQL. In his research paper, he studied load Balancer. He had two datasets (SQL Dataset and Mongo Dataset) for web applications, one for login and the other for registration. (Wałachowski and Kozieł, 2020) This article conducted a comparative analysis of various Android databases, highlighting that there's no one-size-fits-all solution. The choice of a database depends on your specific app requirements. For basic operations with a small number of records, most databases performed similarly. However, Realm was notably slower for extensive record operations, especially for saving, editing, and deleting. SQLite excelled in editing records, while Realm outperformed in string-related operations. Complex queries had the fastest response times in SQLite and Object Box, with Object Box being slightly slower.

(Tesone et al., 2022) This paper guides the selection of a database management system (DBMS) for mobile apps based on specific requirements. With the growth of data types and improved mobile hardware, NoSQL databases have emerged. SQLite, Firebase Realtime Database, and Realm are recommended for their simplicity and integration with object-oriented programming. Firebase Realtime Database may not suit complex search needs. For high schema flexibility, Couchbase Lite is ideal. In experiments involving various needs, SQLite is favored due to its alignment with the relational model. (Phiri and Kunda, 2017) conducted research comparing NoSQL and Relational Databases, focusing on performance, weaknesses, models, and features. Relational Databases prioritize consistency and security with the ACID model but face scalability issues, weak performance, and high costs.

3. Databases in Android

According to the Android Studio Documentation, saving data to a database is ideal for repeating or structuring data. The APIs you'll need to use a database on Android are available in the Android database sqlite package. SQ Lite Database contains functions for generating,

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

erasing, executing SQL commands, and accomplishing other standard database administration duties.

One of the fundamental principles within SQL databases pertains to the scheme an official declaration outlining the database's organization. The schema's manifestation occurs through the SQL statements employed in database creation. You might find it beneficial to establish a companion class, commonly referred to as a contract class. This contract class serves as a repository for constants that delineate URIs, tables, and columns. By utilizing the contract class, you can consistently apply these constants across various classes within the same package. This approach facilitates alterations to a column name in a single location, with the change reflected throughout your code. An effective method of structuring a contract class is to position overarching definitions for the entire database at the root level of the class. Subsequently, you can create inner classes for individual tables, each inner class itemizing the corresponding table's columns.

3.1 Architecture

SQLITE has seven layers and we'll go through every layer one by one.

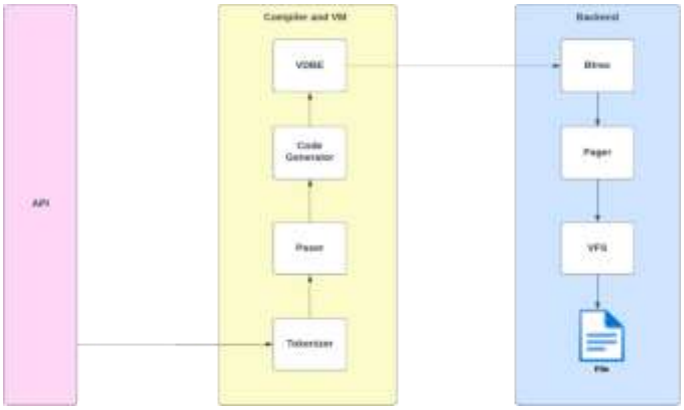


Figure 1 SQ lite architecture

The initial tier is referred to as the Tokenizer. The tokenizer's role involves generating tokens for a provided SQL query. This component scans through the SQL query in a left-to-right manner, resulting in the creation of a collection of tokens. A Parser is employed to create a parse tree by processing a sequence of tokens. In SQLite, the Lemon parser is utilized for parsing SQL queries. This Lemon parser enables the SQLite C compiler to produce appropriate C code based on predefined language rules. These language rules are established using Context-Free Grammar (CFG). During SQL query parsing, SQLite generates a set of instructions or codes for result generation.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

4. ROOM PERSISTENCY LIBRARY

According to the Android Studio Documentation, Room is a Database Object Mapping tool designed to simplify database access in Android apps. Instead of concealing SQLite intricacies, Room aims to incorporate them, offering user-friendly APIs for database querying and compile-time query validation. This grants you the ability to harness SQLite's capabilities while enjoying Java SQL query builders' type safety. Room functions as an Abstraction Layer that sits atop an SQLite database. While SQLite employs a specialized language (SQL) for database tasks, Room streamlines the process of database setup, configuration, and interaction. Moreover, Room offers compile-time assessments of SQLite statements, enhancing the integrity of your code. This annotation database designates a class as a database. This class should be abstract and extend the Room Database. During runtime, you can obtain an instance of it using Room Database Builder. Within this class, you outline the entities and data access objects present in the database. Additionally, it serves as the primary entry point for accessing the underlying connection. This annotation entity designates a class as a database row. A corresponding database table is generated to house the items within each Entity. The Entity class needs to be specified in the Database entities array. Every field within the Entity (including those inherited from its superclass) is stored in the database unless stated otherwise (refer to Entity documentation for specifics). This annotation Dao designates a class or interface as a Data Access Object (DAO). Data access objects serve as a pivotal part of Room, responsible for outlining methods that interact with the database. The class carrying the Database annotation must contain an abstract method devoid of arguments, returning the class marked with Dao.

4.1 Architecture

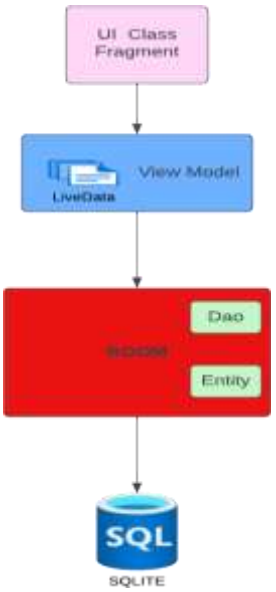


Figure 2 Room DB Architecture

Within Figure 2, the initial components are the UI and View Model, fundamental to Android Studio. These components manage user interaction and maintain the application's state. Moving forward, we encounter the Entity class—a fundamental construct that defines a table in the database. Each instance of this class signifies a row in the table. The Entity class is equipped with mappings that instruct Room on how to present and engage with data in the

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

database. In your app, this entity will hold details, about the items in stock such as their names, prices, and availability. The Data Access Object (DAO) follows a design that separates the storage layer from the rest of the application. By providing an interface this design principle aligns with the idea of having responsibilities. The app can utilize these data entities to make updates or insert records, into the database tables. To achieve this you'll need to create a Room Database class and annotate it with @Database.

5. Development Architecture

When I incorporate the MVVM (Model-View-View-Model) architecture, into my app it can impact performance in ways. Using MVVM brings advantages like dividing responsibilities and improving UI responsiveness with operations. This division simplifies app maintenance and testing making it easier for me to detect and fix any performance issues efficiently. However, mishandling MVVM can add complexity and extra work potentially impacting performance. Neglecting data binding optimization could result in updates and calculations.

5.1 Device profiling

Property	Value
Android Version	13
RAM	6.00 GB
Internal Storage	128.00 GB
CPU	Octa Core
Size, Pixel Density	6.3 inches, ~537 ppi density

5.2 Development Environment

Architecture	Architecture
Processor	Intel(R), Xeon(R), CPU E5-1650 v3 @ 3.50GHz 3.50 GHz
RAM	16.00 GB
System Type	64-bit Operating System
Window	10
Language	Java

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

6. CRUD Operations Performance Test

Create

SQLite

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	1	6	3.5
10	7	13	13.5
100	63	76	69.5

ROOM Library Persistency

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	3	6	4.5
10	45	61	53
100	303	338	320.5

Read

SQLite

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	0	1	0.5
10	0	1	0.5
100	0	1	0.5

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

Room Library persistency

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	0	1	0.5
10	0	3	1.5
100	0	6	3

Update

SQLite

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	4	14	9

ROOM Library Persistency

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	4	14	9

Delete

SQLite

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	0	0	0
100	0	1	0.5

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

ROOM Library Persistency

No. of Execution	Min. Time ms	Max. Time ms	Mean Time ms
1	0	0	0
100	10	12	11

7. Results & Discussion

The software assesses the utilization of the Room database monitoring the apps efficiency by tracking its memory and CPU usage on the device. When numerous data entries are added to the Room database and multiple queries are executed simultaneously it demonstrates the databases performance. Likewise, when running queries, in SQLite and observing how the device responds you can see the effects, on RAM and CPU utilization.



Figure 3 Execution time graph

SQLite usually responds, to queries because it interacts directly with the file system in a manner. On the other hand Room despite having abstraction enhances query execution efficiency resulting in response times comparable to SQLite. The abstraction in Room offers advantages such as compile time safety without impact on performance. In some cases, custom SQL queries in Room can. Even outperform raw SQLite queries but for most applications the difference in query response time is minimal. Developers can make their choice based on their requirements and optimization strategies. SQLite is recognized for its transaction processing due to its design and direct interaction with the file system. It manages transactions effectively. With its abstraction layer Room maintains a level of transaction processing speed as SQLite while optimizing SQL queries and providing user friendliness without compromising performance. In real-world scenarios, the disparity in transaction processing speed between SQLite and Room is generally insignificant. Developers often consider factors beyond speed

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

when choosing between them such as preferences and additional features offered by Room. Both SQLite and Room demonstrate CPU utilization for standard database operations. SQLite performs well in scenarios where resources are limited and tasks are straightforward while Room maintains efficiency while also providing benefits, like abstraction, compile-time safety, and ease of use. The variances, in CPU usage are usually small. Rely on the intricacy of tasks and workloads. SQLite and Room are adept at utilizing storage space retaining their efficiency as data volume grows. Though storage efficiency may slightly decrease with datasets the impact is typically minimal. The room, in Room, ensures that SQL queries are checked for validity during compilation than runtime helping developers catch errors early. By enforcing typing and query verification during compilation Room assists developers in avoiding compile time failures and runtime errors associated with raw SQL queries. The structured approach promoted by Room in managing database-related code within the app makes it simpler to maintain and comprehend. Through high-level abstractions, Room simplifies interactions with databases reducing the complexity of database code. Its user-friendly API design enhances developers' ability to understand and work with it for those managing databases. Additionally, Room provides features like encryption to bolster data security, an aspect of applications. Being part of Androids architecture components Room benefits from documentation and community support. These advantages collectively establish Room as a choice, among Android developers for managing databases in Android applications.

8. Conclusion

In our study, we tested operations, on both databases in the context of our MVVM application structure. We found that when running the queries on both databases SQLite performed better because developers can interact directly with it. On the other hand, Room uses a layer that acts as a middleman between the database and the developer resulting in slightly lower performance. This is similar to the comparison between SQL and MySQL where MySQL, like SQLite's faster but more susceptible to vulnerabilities. As a result, Room is considered the option as it focuses on user-friendliness and integration with Android components. It prioritizes safety during compilation. Strikes a balance between convenience and performance while following best practices. However, for developers looking for control over queries and schemas SQLite remains a viable choice. Room aligns closely with Androids principles by adding layers of abstraction over SQLite and integrating View Model and Live Data for data management. Particularly noteworthy is Room's ability to handle volumes of data without interruptions. Although Room may have higher time complexity and query compilation compared to SQLite due to its abstraction layer it offers the benefit of reducing errors and creating connections, between the database and programming language.

Reference

Dobslaw, D.F., Dobslaw, F., n.d. Examiner Supervisor Author Program Course Field of study Semester, year.

Gyorodi, C., Gyorodi, R., Pecherle, G., Olah, A., 2015. A comparative study: MongoDB vs. MySQL, in 2015 13th International Conference on Engineering of Modern Electric Systems (EMES). Presented at the 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), IEEE, Oradea, Romania, pp. 1–6. https://doi.org/10.1109/EMES.2015.7158433

Holla, S., Katti, M.M., 2012. ANDROID-BASED MOBILE APPLICATION. Int. J. Comput. Trends Technol.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.01 (2025)

Liu, Y., Gui, J., Wan, M., Halfond, W.G.J., 2017. An Empirical Study of Local Database Usage in Android Applications, in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). Presented at the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, Shanghai, pp. 444–455. https://doi.org/10.1109/ICSME.2017.75

Obradovic, N., Kelec, A., Dujlovic, I., 2019. Performance analysis on Android SQLite database, in 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH). Presented at the 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH), IEEE, East Sarajevo, Bosnia and Herzegovina, pp. 1–4. https://doi.org/10.1109/INFOTEH.2019.8717652

Phiri, M.H., Kunda, D.D., 2017. A Comparative Study of NoSQL and Relational Database 1.

Pujari, Mr.V., Patil2, Dr.R., Sutar, Mr.S., 2020. A Review on Best Practices in Mobile Application Development. Natl. Semin. “Trends Geogr. Commer. IT Sustain. Dev. 77, 4.

Tesone, F., Thomas, P., Marrero, L., Olsowy, V., Pesado, P., 2022. A Comparison of DBMSs for Mobile Devices, in Pesado, P., Gil, G. (Eds.), Computer Science – CACIC 2021, Communications in Computer and Information Science. Springer International Publishing, Cham, pp. 201–215. https://doi.org/10.1007/978-3-031-05903-2_14

Wałachowski, K., Kozieł, G., 2020. Comparative analysis of database systems dedicated to Android. J. Comput. Sci. Inst. 15, 126–132. https://doi.org/10.35784/jcsi.2043