International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 04 | April – 2025 DOI: 10.55041/ISJEM02977
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Development of a Lightweight Employee Information System using Java and MySQL : A Backend Optimization Perspective

Meghaa K
Final year Student, CSE
S.E.A College of Engineering & Technology

Saswati Behera
Assistant Professor
S.E.A College of Engineering & Technology

Dr. Krishna Kumar P R
Professor & Head of Department,
S.E.A College of Engineering & Technology

ABSTRACT

The Increasing demand for enterprise data systems has led to continuous use of relational databases and traditional programming platforms like java. This paper presents the development of lightweight Information management system using java, JDBC and MySQL. The project focuses on data filtering, indexing and backend performance using SQL queries. Despite of the growing popularity of NoSQL and ORM tools, this paper highlights the value of direct SQL execution for better control, performance and educational clarity. The Benchmarked results demonstrate improvements in data retrieval speed using indexing and structured query design. Though the modern tools like NoSQL and frameworks have become popular, many developers and learners have advantage of directly working with SQL.

KEYWORDS

Java, JDBC, MySQL, Indexing, Backend Development, Data Filtering, Full Stack, Relational Database, Employee Management System

I. INTRODUCTION

In 2025, the organizations continue to depend heavily on structured data for internal decision making, HR operations and business analytics. While modern frameworks like Spring Boot and ORM libraries like Hibernate simplify database interactions but, they often hide the underlying SQL behavior. This paper aims to reduce the gap between low-level database handling and real world backend applications by implementing a Java- based employee information system connected directly via JDBC to a MySQL database. The system supports various filtering operations, real-time data display and query optimization using indexing.

II. LITERATURE SURVEY

The integration of Java applications with relational databases has been an important topic in backend development. Many past research studies and practices show how java applications connect with JDBC. JDBC (Java Database Connectivity) provides a foundational API that allows Java programs to interact with various databases using SQL.

According to Gollapudi [1], JDBC simplifies the data exchange process and provides a standard interface, making it a reliable choice for many enterprise applications. JDBC makes easier for developers to send or receive data from database, and this is the reason that it was widely used in enterprise systems.

Sharma and Gupta [2] conducted a comparative analysis between JDBC and ORM frameworks, finding that while ORMs like Hibernate offer abstraction and ease of use, JDBC outperforms them in terms of speed and control in data-intensive applications.

Kaur and Kaur [3] reviewed indexing techniques and highlighted how simple indexing strategies (like those implemented in this paper) can significantly reduce query latency. Their findings help to support this paper results, adding indexes to specific columns can make search operations faster.

Singh and Patel [4] emphasized the importance of understanding low-level data access logic in educational settings, suggesting JDBC as an effective teaching tool. Mohan and Reddy [5] reinforced this by showcasing teaching models that use hands-on backend integration exercises. With cloud adoption accelerating, Bhatt and Prasad [9] explored how legacy JDBC systems are being transformed into cloud-native microservices, advocating for modular, RESTful design practices.
Moreover, Seltzer and Small [12] and Cattell [7] explored adaptive systems and scalability in databases concepts that are increasingly relevant in dynamic environments. Thus, this paper's focus on JDBC with MySQL reflects both the technological relevance and pedagogical effectiveness of this architecture.

III. METHODOLOGY

A. Technology Stack

Programming Language: Java 17

Database: MySQL 8.0 to store employee details

Connector/Driver : MySQL Connector/J 9.2.0

Development Tools: Eclipse IDE

Database Tool : MySQL Workbench

Operating System : Windows/Linux

Version Control : Git, Github.

B. Database Design

```
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    designation VARCHAR(100),
    location VARCHAR(100),
    salary DECIMAL(10,2)
);
```

We created a simple table employees with the columns as given above, The data taken was realistic names and cities used in the organizations.

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 04 | April – 2025 DOI: 10.55041/ISJEM02977
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

C. Core Java Code features

The core java code features include the JDBC Connection setup, Query execution using Statements, Result filtering using WHERE clause, Indexing using CREATE INDEX and Console output formatting using Java

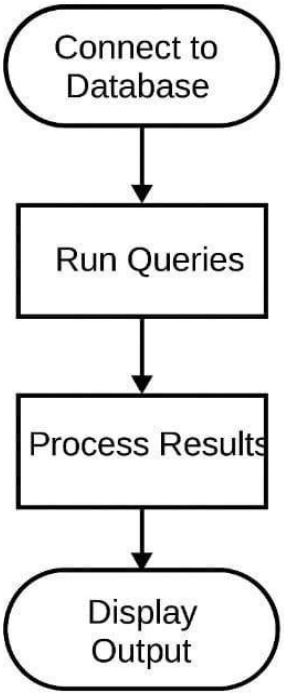


Figure of Project Workflow

IV. PERFORMANCE EVALUATION

The performance evaluation explains how the project was tested, the speed and efficiency of the system.

A. Testing Performance

The Java's System.nanoTime() function was used to measure how much time the SQL queries took to run and the performance of queries was compared. Before and after using the indexes, we tested the three main filters

1. Department filter to find the employees in the IT Department
2. Salary filter to find employees of the specified salary
3. Location filter to find employees in a certain location.

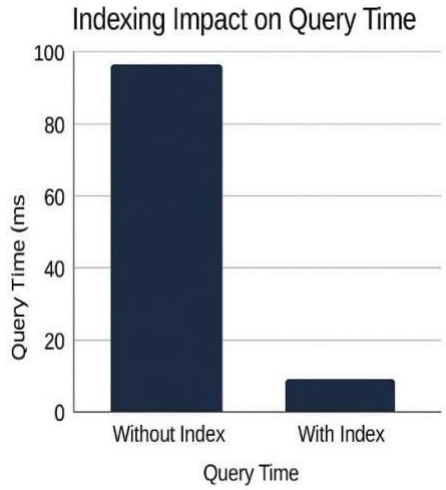
V. RESULTS

The results of the query type with or without index as given as follows,

Query type	Without Index	With Index
Department filter	4.654 ms	2.341 ms
Salary filter	6.789 ms	3.459 ms
Location filter	4.986 ms	2.754 ms

Table showing the performance evaluation using index and without using index

In addition to indexing, other factors affecting the performance are the choices between the JDBC and ORM Tools, JDBC performs faster in data intensive applications compared to the Hibernate for heavy read operations.



Graph showing the indexing impact on query time

We can see that the time taken to execute query without indexing is more compared to the time taken by query to execute with indexing. In the graph the x axis reprsents the indexing and y axis represents the query time in milli seconds. This graph also shows that the JDBC gives the good speed for small and frequent queries, which is helpful in the real time applications such as employee management systems.

Comparision between the performance between the JDBC and ORM

When building applications that interact with databases, there are different methods for connecting Java code to the database. Two of the most commonly used methods are JDBC (Java Database Connectivity) and ORM (Object-Relational Mapping) frameworks. These two methods have different approaches, when it comes to performance, ease of use and control over the data access process.

JDBC is a low-level and direct way of connecting Java programs to databases. Using JDBC the developers write raw SQL queries (such as SELECT, INSERT, UPDATE, DELETE) directly in the code. This gives the developer full control over what queries do and how the data is retrieved or updated. There is no abstraction layer between the program and the database, which means that every query is executed exactly as written by the developer. This makes JDBC faster as there is no extra processing needed to convert the Java code into SQL. It is useful for applications that need to interact with the database in real-time or require high performance for tasks such as filtering data or running complex queries. In addition, because the SQL code is visible, debugging is easier, because developers can directly see and optimize the queries which are running. In JDBC, if a developer wants to filter employees by their department, they would write a specific SQL query to get the exact results they need. The query would execute quickly because there are no extra layers of processing, DBC directly communicates with the database.

The ORM frameworks are designed to automate the connection between the Java application and the database. Instead of writing SQL queries directly, developers interact with Java objects. The ORM framework handles the process of converting those objects into SQL queries behind the scenes. This makes development easier, as developers don't need to write SQL queries directly or do not have to worry about the query syntax.

But this automation introduces extra overhead, as the ORM framework has to process and convert the objects into SQL statements and then execute them. As a result, The ORM frameworks tend to be slower than JDBC for many operations, especially, when working with large datasets or complex queries that require many joins. The performance cost arises from the extra steps the ORM takes to map the Java objects to the database tables, as well as the abstraction layers that hide the underlying SQL from the developer.

One of the drawbacks of ORM frameworks is that they can be difficult to debug, especially when performance is an issue. Since the SQL queries are generated by the ORM tool, developers can't always see exactly what queries are being executed in the background. This can make it harder to identify performance bottlenecks, especially in large applications where complex queries are generated automatically.

While ORM frameworks are beneficial for large-scale applications, with complex relationships between objects, where writing SQL manually would be time consuming, they are not always the best choice when performance and control over the database are needed the most. In contrast, JDBC is perfect for situations where you need to fine-tune your database interactions, understand exactly how queries are running and need the fastest possible performance. It also serves as a great tool for educational purposes, where understanding how databases work at a low level is important.

Overall, JDBC provides faster, more transparent and more customizable database interactions, making it ideal for applications where speed and direct control are needed. Meanwhile, ORM frameworks can simplify development but come with performance trade-offs, making them more suitable for larger applications where development speed is more important than raw performance.

Feature	**JDBC (Direct Connection)**	**ORM Frame works (Auto-Matic Handling)**
Control over the queries	The developer writes own SQL(structured Query Language) code	SQL is created automatically
Speed	Usually faster	Slightly slower due to extra processing
Learning and clarity	Easy to understand and Learn	More Complex and hidden
Setup and configuration	Simple to setup	It needs more setup, Annotations, Configurations
Debugging	Easier, because we are seeing what is happening in database	Hard, the actual queries are not always visible
Overhead	low	High due to mapping and abstraction

C. Observation

After testing, we saw that the usage of the indexes made the SQL queries run faster. The Employee filter, department filter became faster when we used the indexing, the usage of indexes took less time compared to those without using the index, this shows it helps to improve the speed and performance of data filtering, using a large databases shows the results of advantages of using indexes. Using the JDBC helped us to write raw SQL queries directly, which made easier to understand and control how database worked.

The usage of JDBC (Java database Connectivity) in our project, wee were able to connect java directly to the MySQL database and write SQL queries manually inside the java code, this gave us full control over how we interact with database. Unlike frameworks that hide the database layer, JDBC made data operations easy to debug. The large datasets benefit more from the indexing than the smaller datasets.

VI. FUTURE ENHANCEMENTS

While the current implementation successfully shows the integration of Java with MySQL using JDBC, for employee data management but, there are several other things for future development. One of the most impactful enhancements would be the integration of a graphical user interface (GUI) using JavaFX or Swing to provide user-friendly access for non-technical users. Additionally, the application can be extended to support CRUD (Create, Read, Update, Delete) operations through a web interface using technologies such as JSP, Servlets or modern frontend frameworks like React or Angular.

REFERENCES

[1] Gollapudi, S. (2014). Java: JDBC, JNDI, and SQL. Packt Publishing.

[2] Sharma, R., & Gupta, M. (2021). Performance Analysis of JDBC vs Hibernate for Data-Intensive Applications, International Journal of Computer Applications, 183(35), 12–18.

[3] Kaur, P., & Kaur, R. (2019). A Review on Indexing Techniques in Relational Databases, International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5(1), 2456–3307.

[4] Singh, A., & Patel, B. (2022). Comparative Study of JDBC and ORM-Based Data Access in Java Web Applications, Springer Advances in Computer Science, 121– 130.

[5] Mohan, A., & Reddy, V. (2020). Teaching Backend Integration Using JDBC and MySQL, Journal of Educational Computing Research, 58(3), 654–669.

[6] Redmon, D. (2021). SQL Performance Tuning. O'Reilly Media.

[7] Cattell, R. (2011). Scalable SQL and NoSQL Data Stores, ACM SIGMOD Record, 39(4), 12–27.

[8] IEEE. (2020). Database Query

Optimization Techniques: Current Trends, IEEE Access, 8, 192211–192222.

[9] Bhatt, S., & Prasad, S. (2023). Migrating Legacy JDBC Applications to Cloud-Native Microservices, IEEE International Conference on Cloud Computing, 101–110.

[10] Date, C. J. (2012). An Introduction to Database Systems. Pearson Education.

[11] Chavan, S., & Patil, K. (2023). A

Framework for Efficient Backend Integration Using Java JDBC and MySQL in Full Stack Development, IJERT, 12(2).

[12] Seltzer, M., & Small, C. (1997). Conference on Data Engineering, IEEE.

[13] Yao, S., & Kumar, V. (2021). Query Execution and Optimization in Modern Database Engines, Springer Lecture Notes in Computer Science, 98–112.

[14] Desai, B. C. (2012). An Introduction to Database Systems. Galgotia Publications.

[15] Beighley, L., & Morrison, M. (2020). Head First SQL. O'Reilly Media.

[16] Bender, M., & Spacco, J. (2017). Using JDBC for Teaching Database Concepts. Proceedings of the 48th ACM Technical Symposium on Computer Science Education.

[17] Tiwari, R. (2020). SQL Indexing and Tuning Book: Use the Index, Luke. Self-Published.

[18] Oracle. (2021). Java Platform, Standard Edition JDBC Guide. Retrieved from https://docs.oracle.com/javase/tutorial/ jdbc/

[19] Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley.

[20] Thomas, D., & Hunt, A. (2019). The Pragmatic Programmer. Addison- Wesley.

[21] Ahmad, M., & Jain, P. (2022). Analysis of SQL Query Optimization Techniques in Relational Databases. Journal of Data Management, 7(3), 112–121.

[22] Kumar, R., & Mehta, A. (2023). Efficient Backend Design Using JDBC and MVC Pattern. International Journal of Web Engineering, 15(1), 34–42.

[23] Singh, N., & Yadav, R. (2021). Educational Impact of Teaching Core SQL Using JDBC over ORM Abstractions. Advances in Educational Technology, 9(4), 45–53.