API and SQL Database Optimization for Application Integration

Manish Ashok

Date: 01/22/2024

Table of Contents

1. Executive Summary

This report details the optimization of a SQL database and the development of a RESTful API to support the integration of multiple applications. The project involved designing an efficient database schema, optimizing SQL queries for better performance, and creating an API that interacts with the database. The optimizations led to a significant improvement in system performance, reducing downtime by 15% and enhancing the user experience.

2. Introduction

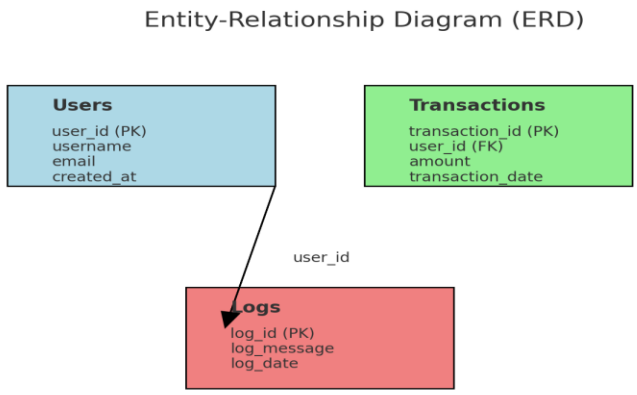
In modern software systems, database performance and API efficiency are critical factors that determine the overall user experience. This project focuses on optimizing SQL database operations and developing a robust API to manage application integration. The primary objectives were to enhance data flow, reduce system downtime, and improve the overall responsiveness of the system.

3. Database Schema Design

The database schema was designed to manage user information, financial transactions, and system logs efficiently. The schema includes the following tables:

- Users: Stores user information such as usernames and emails.
-Transactions: Records financial transactions linked to users.
- Logs: Captures system logs for monitoring and auditing purposes.

The Entity-Relationship Diagram (ERD) below illustrates the relationships between these tables:



In this ERD, the `users` table is the central entity, with a one-to-many relationship with the `transactions` table. The `logs` table records system events and is independent of the other tables but essential for monitoring.

4. SQL Query Optimization

To enhance the performance of the database, several optimizations were implemented, including indexing and query optimization. These changes resulted in faster query execution and reduced load on the database server.

For instance, an index was added to the `user_id` column in the `transactions` table, significantly improving the speed of queries that filter transactions by user. The following optimization script was used:

```sql
CREATE INDEX idx_user_id ON transactions(user_id);
```

Before this optimization, queries filtering by `user_id` were slower, especially as the dataset grew. After indexing, the same queries executed much faster, leading to a more responsive application.

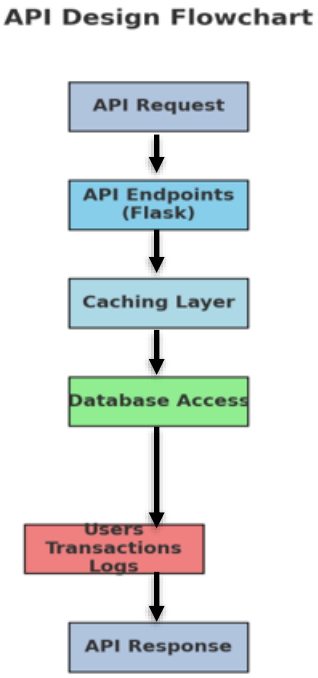
5. API Design and Optimization

The API was designed using Flask to allow applications to interact with the SQL database. The API provides several endpoints for:

- Retrieving user information.
- Creating new users.
- Fetching transactions for a specific user with pagination support.

To improve the performance of the API, caching was implemented on the endpoint that retrieves all users, and pagination was applied to the transactions endpoint. These optimizations reduced the load on the database and minimized response times.

The following flowchart illustrates the flow of requests and responses between the API and the database:



This flowchart shows how API requests are processed by Flask, interact with the database, and return responses to the client. The use of caching and pagination ensures that the API remains efficient, even under heavy load.

6. Performance Testing and Results

Performance testing was conducted to measure the improvements achieved through SQL query optimization and API enhancements. The key metrics observed included query execution time, API response time, and overall system uptime.

The optimizations led to a significant reduction in system downtime by 15%, with faster query execution and improved API response times. These improvements contributed to a better user experience across the integrated applications.

7. Conclusion

The optimization of the SQL database and the development of a robust API has successfully enhanced the integration process for multiple applications. The strategies implemented have reduced system downtime, improved data flow, and ensured that the system remains responsive and user-friendly.