Deerwalk Journal of Computer Science and Information Technology

SOCKETDB: DBMS WITH DATA STREAMING VIA WEBSOCKETS

Abhinav Gyawali*

abhizer@pm.me

ABSTRACT

This project introduces SocketDB, a novel real-time SQL database system specifically engineered to meet the evolving demands of contemporary applications requiring instantaneous data updates and robust query processing capabilities. By integrating a WebSocket-based notification system, SocketDB offers a unique approach, allowing clients to subscribe and instantly receive updates, thus facilitating real-time interactions and analytics. Built on a columnar data layout, it ensures efficient query performance and storage optimization, particularly beneficial for read-intensive scenarios. The core of SocketDB is enhanced by a recursive descent SQL parser, enabling comprehensive support for a wide array of SQL operations, making it highly adaptable to complex data manipulation needs. Additionally, SocketDB incorporates advanced features such as data compression with the Zstandard algorithm, ensuring reduced storage footprint without compromising access speed. Embedded fault tolerance mechanisms and scalable infrastructure further reinforce its reliability and applicability in diverse environments. Aimed at developers and businesses alike, SocketDB is poised to transform real-time data handling, offering a scalable, efficient, and user-friendly platform for dynamic data management and analysis.

Keywords: *Database; DBMS; WebSockets; SQL; storage; relational algebra*

1. INTRODUCTION

1.1. Overview

SocketDB presents a contemporary solution to the demand for real-time data management within Database Management Systems (DBMS). Utilizing WebSockets, it enables uninterrupted communication between the server and clients, ensuring immediate data retrieval and transmission. This innovative approach combines SQL-based robustness with WebSocket efficiency, making SocketDB a dynamic platform applicable in domains like financial markets and real-time analytics. This overview provides a concise glimpse into SocketDB's architecture, features, and potential impact on real-time data management across diverse fields.

1.2. Background and Motivation

The project stems from a genuine interest in unraveling the intricacies of database systems. The motivation behind it lies in a desire to comprehend how databases work. Seeking a hands-on approach to learning, the focus is on demystifying the inner workings of these systems and gaining a deeper understanding of data management. The project provides an opportunity to explore the complexities of database architecture, contributing to a broader perspective on the foundational elements of information technology.

1.3. Problem Statement

In the contemporary landscape of software applications, a pervasive challenge persists—the need for constant client-side polling to obtain timely updates from servers, and subsequently, from databases. This ubiquitous practice, exemplified by scenarios like fetching new social media posts, imposes an unwarranted burden on multiple facets of the application ecosystem. Specifically, it places added strain on client devices, web application servers, and underlying databases. This multifaceted strain manifests as increased resource consumption, elevated latencies, and suboptimal user experiences. The essential problem at hand is twofold: firstly, the prevalent reliance on traditional Database Management Systems (DBMS) and server-client communication paradigms, which necessitate client-side polling for data updates, and secondly, the subsequent consequences of this practice, encompassing heightened computational costs and diminished system efficiency. SocketDB, as an innovative DBMS solution, emerges as a prospective remedy

to alleviate these challenges by revolutionizing real-time data access and propagation, thereby redefining the landscape of data-driven application development.

1.4. Objectives

The objectives of this project are:

- to develop a compact SQL database capable of providing authenticated users with timely notifications regarding updates to the database
- to learn about the inner workings of modern databases

1.5. Scope and Limitation

SocketDB is purpose-built for applications with a pronounced focus on data retrieval, making it an ideal choice for read-heavy workloads. Its core objective is to enhance the efficiency of data access by harnessing the power of WebSocket technology. By enabling real-time data updates through WebSockets, SocketDB aims to significantly diminish the computational load on both application servers and databases, thereby optimizing overall system performance.

SocketDB's scope is primarily centered around academic institutions and small personal projects, where it can serve as a valuable and accessible resource. The system is designed to be user-friendly and lightweight, making it well-suited for educational purposes, research endeavors, and modest-sized applications that benefit from real-time data updates. In essence, SocketDB endeavors to streamline data retrieval processes for its target audience, contributing to improved application responsiveness and resource efficiency.

1.6. Development Methodology

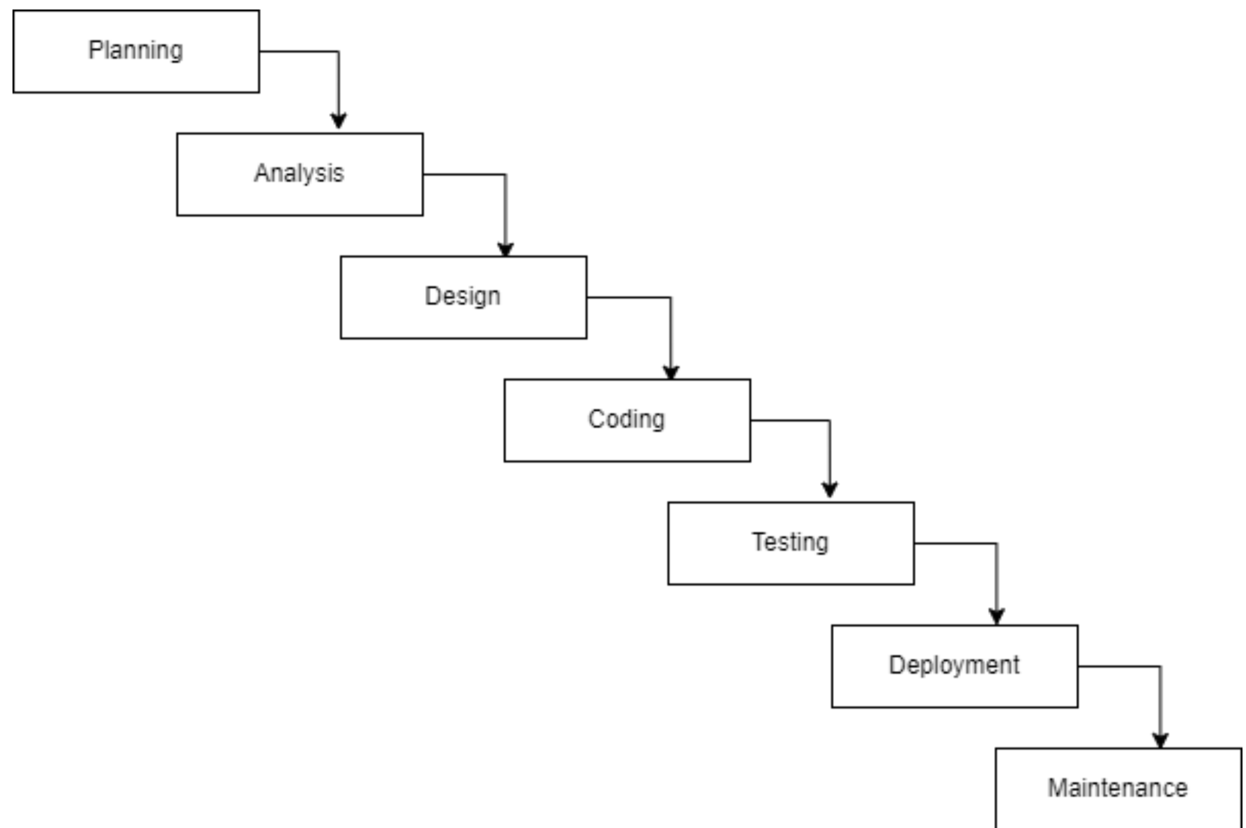

Figure 1: Waterfall Model

2. BACKGROUND STUDY AND LITERATURE REVIEW

2.1. Background Study

The background study for this project meticulously explores fundamental theories, general concepts, and integral terminologies crucial to its development. It begins by scrutinizing the WebSocket protocol, recognizing its pivotal role in enabling seamless bi-directional communication with SurrealDB. This feature consolidates queries through a single connection, thereby amplifying the efficiency of data retrieval and transmission. Delving deeper, the study elucidates SQLite's cornerstone status in database systems, dissecting its primary modes – rollback mode and write-ahead log mode – and elucidating their roles in ensuring data consistency and durability.

Moreover, prevalent practices in web applications, particularly their reliance on relational databases for storing and querying data, are meticulously examined. Notably, these applications often grapple with read-heavy query loads, accentuating the significance of the project's selected approach. This comprehensive background examination lays a robust foundation for a nuanced understanding of the project's key elements.

In parallel, the contemporary database research's focus on incremental view maintenance emerges as a critical aspect. This approach, which updates materialized views incrementally as underlying data changes occur, promises heightened query performance and responsiveness. While SocketDB does not directly implement incremental view maintenance, it pioneers a novel approach. By automatically transmitting updated records to clients via WebSockets, SocketDB enables real-time data dissemination without manual querying or recomputation. This proactive data dissemination underscores SocketDB's role in enhancing the responsiveness and utility of real-time data management systems.

2.2. Literature Review

The WebSocket protocol allows for easy bi-directional communication with SurrealDB. This allows you to maintain a single connection to run all your queries [1].

SQLite has two primary modes by which these guarantees are achieved: rollback mode and write-ahead log mode [2].

Many web applications use a relational database to store and query data. Page views generate database queries that frequently require complex computation, and the query load tends to be read-heavy [3].

2.3. Current System

In the sphere of real-time data management, currently, SurrealDB [1] emerges as a noteworthy attempt to address the demand for instant data retrieval and transmission. SurrealDB offers the WebSocket protocol to establish uninterrupted, bidirectional communication between the database server and connected clients. This unique approach allows for the swift dispatch of updates to authorized users, ensuring the perpetual currency of information. For instance, when a relevant change occurs in the database, SurrealDB promptly notifies connected users, presenting a departure from conventional databases that rely on periodic querying.

While SurrealDB represents a new entrant in the database landscape and is not yet widely adopted, it serves as a current system example in our exploration. This project draws inspiration from SurrealDB's endeavors, aiming to contribute to the evolution of real-time data management systems by exploring similar principles and enhancing predictive capabilities through innovative elements.

In addition to its innovative use of the WebSocket protocol, SurrealDB's approach aligns with the growing focus in contemporary database research on incremental view maintenance. This technique involves updating materialized views incrementally as underlying data changes occur, rather than recomputing the entire view from scratch. While SurrealDB does not directly implement incremental view maintenance, its real-time data transmission mechanism shares similarities with this approach. By automatically notifying connected users of relevant database changes, SurrealDB proactively disseminates updated information without the need for manual querying or recomputation. This alignment with principles of incremental view maintenance underscores SurrealDB's role in enhancing the responsiveness and utility of real-time data management systems. As a notable example in the evolving landscape of database technologies, SurrealDB inspires the exploration of similar principles and innovative approaches, contributing to the ongoing evolution of real-time data management systems.

2.4. The problem with Current System

Current mainstream database management systems, including MySQL and SQLite, lack a crucial feature for real-time data management—they do not inherently offer the option of receiving notifications via WebSocket. These traditional systems rely on periodic querying mechanisms, making it challenging for users to receive timely updates. The absence of WebSocket integration in widely used databases limits their responsiveness, hindering the seamless bidirectional communication necessary for immediate data retrieval and transmission. This inherent limitation in the current systems underscores the need for a more contemporary approach to address the exigent demand for real-time data management.

3. SYSTEM ANALYSIS

3.1. Requirement Analysis

3.1.1. Functional Requirement

- The DBA shall be able to create databases and tables to store data.
- The DBA shall be able to store, query, update and remove the existing data.
- The DBA shall have a query prompt to run SQL commands.
- The end user shall be able to receive updates to a table after authenticated via web sockets.

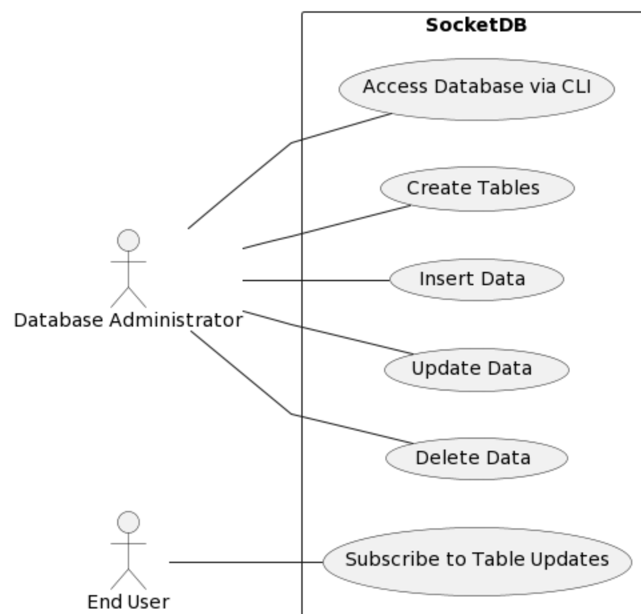

Figure 2: Use Case Diagram of SocketDB

3.1.2. Non-Functional Requirement

- The DBMS must be fast and responsive while querying.
- The DBMS must compress the data to store it for persistence.
- The DBMS must use minimal network traffic while providing updates to clients.

3.2. Analysis

3.2.1 Sequence Diagram

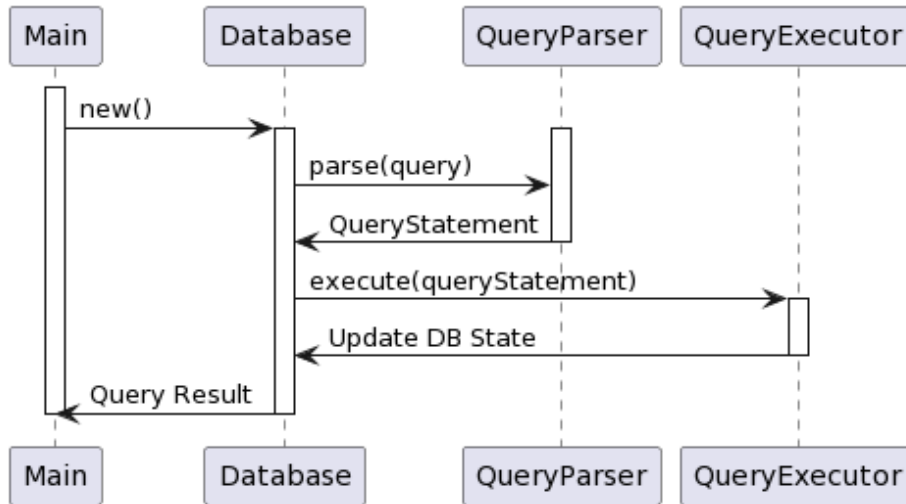

Figure 3: Sequence Diagram of SocketDB

3.2.2 Activity Diagram

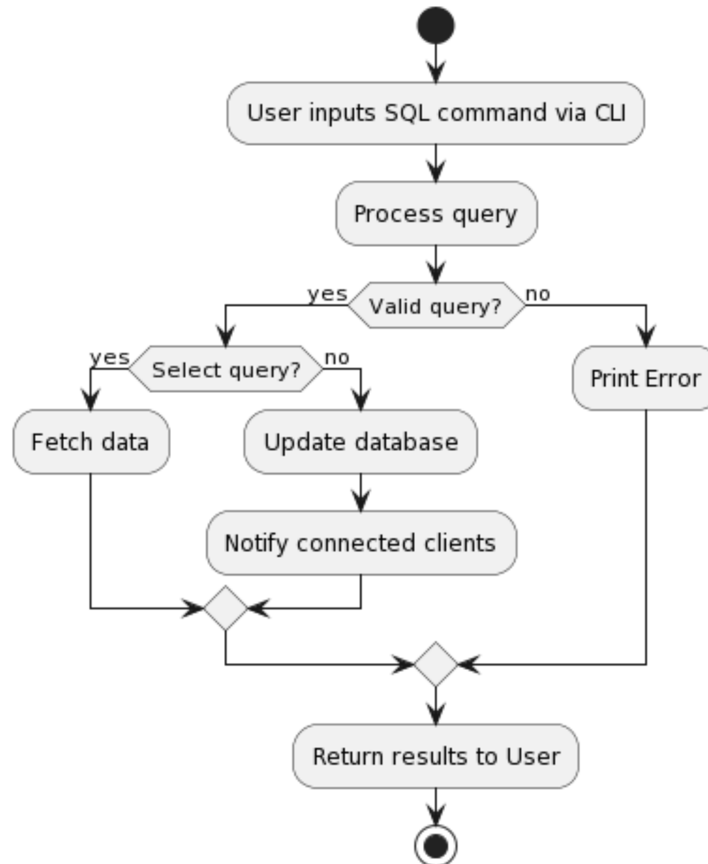

Figure 4: Activity Diagram of SocketDB

4. SYSTEM DESIGN

4.1. Design

4.1.1. Block Diagram

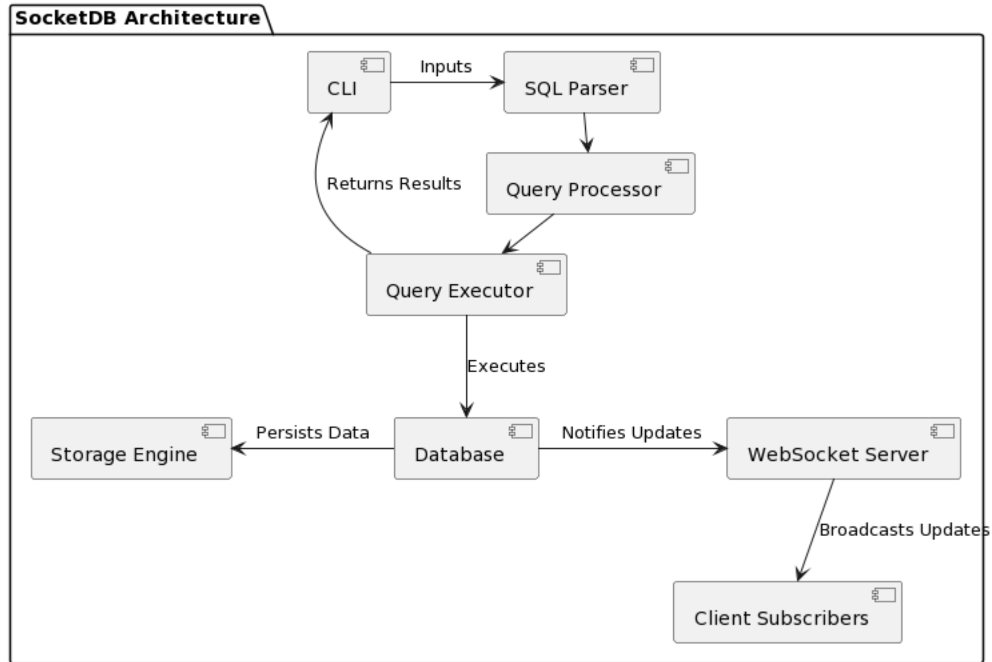

Figure 5: System Block Diagram of SocketDB

4.1.2. Sequence Diagram

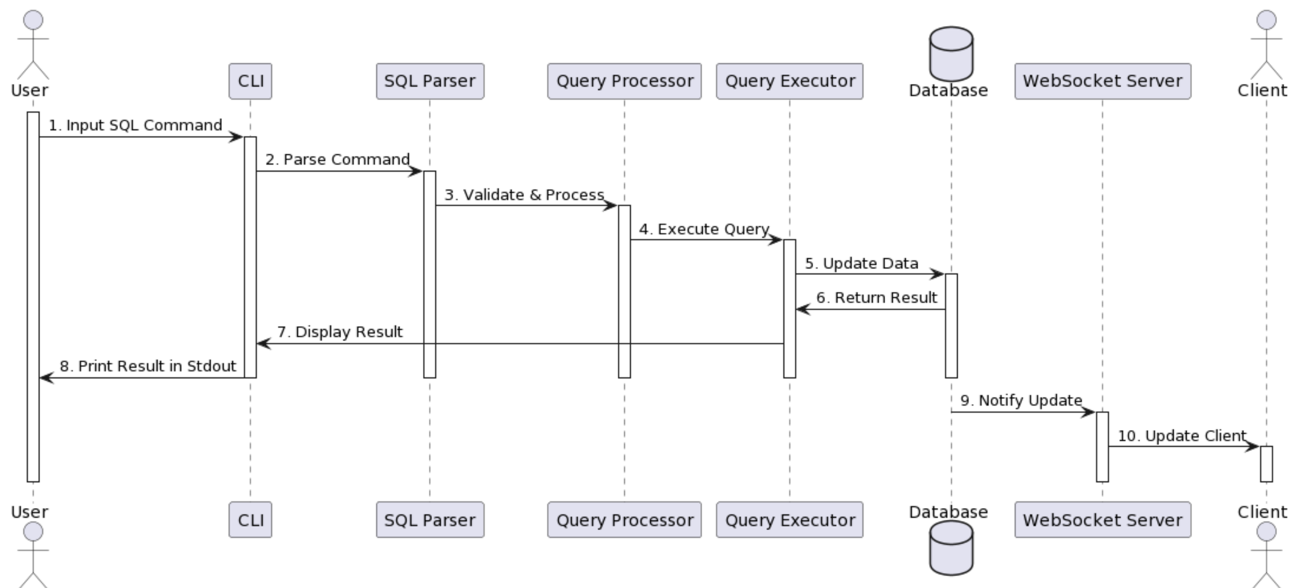

Figure 6: Refinement of Sequence Diagram of SocketDB

4.1.3. Activity Diagram

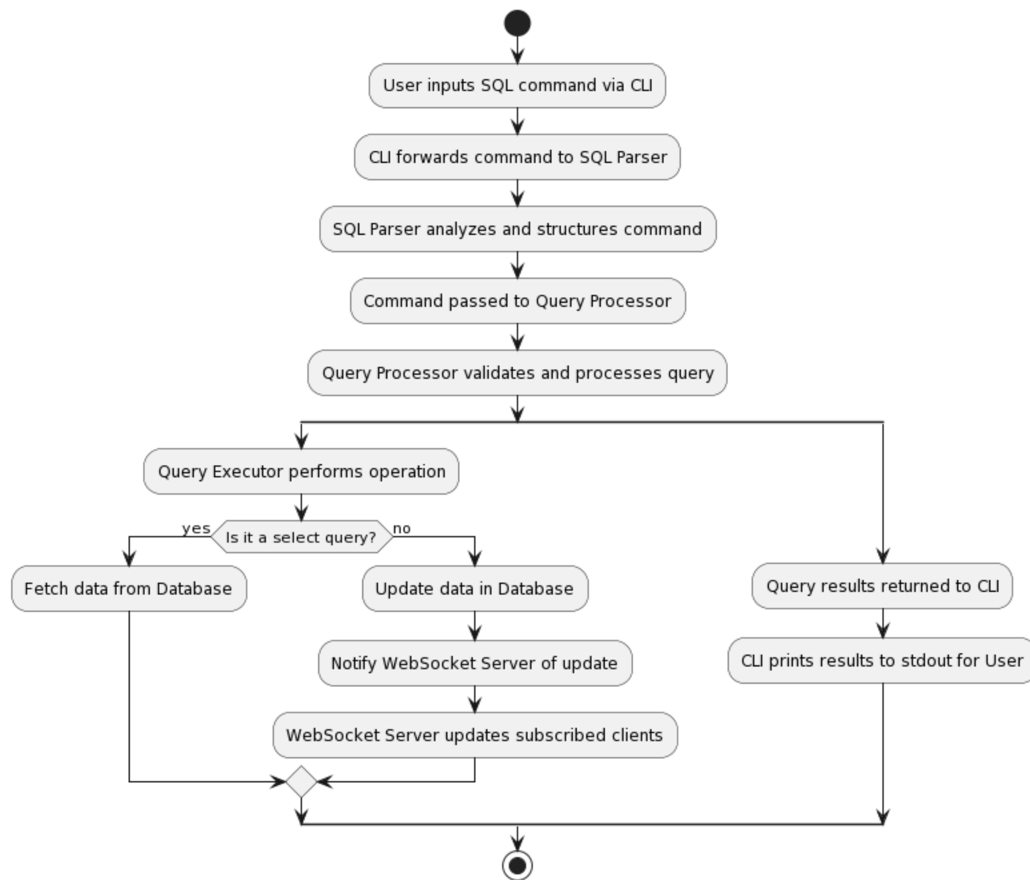

Figure 7: Refinement of Activity Diagram of SocketDB

4.2. Algorithm Details

4.2.1. SQL Parser

The SQL Parser in SocketDB follows a Recursive Descent Algorithm. This algorithm involves breaking down the parsing process into multiple smaller functions, each responsible for parsing a segment of the SQL query syntax. Recursive descent parsing is a top-down approach that starts from the highest level of the grammar and works its way down, making it well-suited for SQL due to its hierarchical structure.

4.2.2. Query Processing

For SQL SELECT queries, the processing is done in two main stages: Selection and Projection.

Selection: This step involves filtering data based on the criteria specified in the WHERE clause of the SQL query. Since data is stored in a columnar layout, this step can efficiently scan and retrieve relevant rows without needing to process entire records.

Projection: After the relevant rows are identified, the projection step extracts only the specified columns, as indicated in the SELECT part of the query. This allows for a more efficient data retrieval, tailored to the user's request.

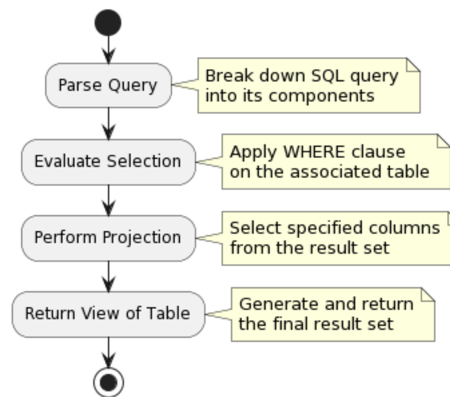

Figure 8: Activity Diagram of Select Query

4.2.3. Data Storage

Data in SocketDB is stored in a Columnar Layout. Each column is stored independently in a BTreeMap, which facilitates efficient access and manipulation. The columns are linked by the Row ID, enabling the reconstruction of records when needed. This layout is particularly efficient for read-heavy operations and analytical queries where operations are often performed on specific columns.

4.2.4. Indexing

Indexing in SocketDB is primarily done via the Primary Key. Each table's primary key is indexed to allow for fast lookup, retrieval, and update operations. This indexing strategy significantly speeds up query processing for operations that involve searching by the primary key.

4.2.5. Encoding and Storage

For data encoding and compression, SocketDB utilizes the Zstandard (Zstd) algorithm. Zstd provides a balance between speed and compression ratio, making it an excellent choice for database storage. It ensures that the stored data consumes less disk space while remaining quick to access and decompress when needed.

4.2.6. Websocket Notifier

The connection between the database and the WebSocket notifier follows the Actor Model. In this model, the database and the notifier operate as independent actors that communicate through message passing. This decouples the database operations from the real-time notification system, allowing for scalable and maintainable real-time data delivery. Clients connected via WebSocket can subscribe to table updates and receive notifications asynchronously, enabling efficient real-time data streaming.

Overall, the combination of these algorithmic and architectural choices makes SocketDB a powerful tool for managing and querying data efficiently while providing real-time updates to clients.

5. IMPLEMENTATION AND TESTING

5.1. Implementation

The SQL parser in SocketDB is implemented following the recursive descent algorithm. This method allows for flexible adaptation to complex SQL syntax and extensibility for future SQL features. The parser breaks down the SQL statements into identifiable components, such as SELECT, FROM, WHERE, and other clauses.

Key Steps:

- **Lexical Analysis:** Convert the input SQL text into a stream of tokens.
- **Syntax Analysis:** Apply recursive descent to parse tokens into a parse tree based on SQL grammar.
- **Syntax Tree Construction:** Build an abstract syntax tree representing the hierarchical structure of the SQL statement.

The execution of SELECT queries in SocketDB involves several stages: parsing, selection, projection, and result generation.

- **Selection:** After parsing, the query executor evaluates the selection criteria defined in the WHERE clause. This evaluation is performed directly on the data stored in a columnar layout to enhance the efficiency of the search and filtering processes.
- **Projection:** Once the relevant records are identified, SocketDB performs projection, extracting only the required columns specified in the SELECT clause. This step reduces the volume of data to be processed and sent to the client.
- **View Generation:** Finally, the system combines the results into a coherent view, effectively constructing the result set to be returned to the user.

SocketDB employs a columnar storage format, where each column is stored separately, facilitating efficient query processing, especially for column-specific operations. Data encoding and compression are handled using the Zstandard (Zstd) algorithm, balancing compression ratio and speed to optimize storage and query performance.

The real-time notification system in SocketDB leverages the actor model to decouple database changes from client notifications. This system is integral for applications requiring real-time data updates.

Implementation Strategy:

- **Actor Model Setup:** Establish actors for the database and WebSocket server to handle messages asynchronously.
- **Subscription Management:** Clients can subscribe to changes in specific tables or datasets.
- **Update Broadcasting:** Upon data changes, the affected table's actor sends messages to the WebSocket server actor, which then broadcasts these updates to all subscribed clients.

The implementation of SocketDB encompasses careful design choices and algorithm implementations tailored to provide efficient data storage, fast query processing, and real-time updates. By adhering to these implementation details, SocketDB ensures high performance and scalability, meeting the needs of modern applications requiring real-time data interaction.

5.1.1. Tools Used

SocketDB is developed using Rust and heavy utilizes the availability of algebraic datatypes in Rust.

5.1.2. Implementation Details of Modules

5.1.2.1. Parser Module

Objective:

- To interpret and validate SQL queries submitted by users, converting them into an internal representation that can be processed by the database.

Implementation:

- Utilizes a recursive descent algorithm, breaking down SQL text into tokens and constructing a parse tree.
- Handles syntax errors by providing meaningful error messages to the user.
- Supports parsing of key SQL components: SELECT, INSERT, UPDATE, DELETE, WHERE clauses.

- Outputs an abstract syntax tree (AST) that represents the structured interpretation of the SQL command.

5.1.2.2. Evaluator Module

Objective:

- To apply query logic on the data stored within the tables, primarily handling the WHERE clause and condition evaluation.

Implementation:

- Processes the AST generated by the Parser module to identify and evaluate conditions specified in the WHERE clause.
- Employs efficient search algorithms to filter data based on the selection criteria.
- Supports various operators and expressions, including logical, arithmetic, and comparison operators.
- Integrates with the Table module to fetch and manipulate the actual data.

5.1.2.3. Table Module

Objective:

- To define the structure and handle operations for individual tables within the database, including data storage, retrieval, and manipulation.

Implementation:

- Manages data in a columnar format, where each column is stored as an independent entity, facilitating efficient query execution.
- Leverages BTreeMaps to organize data within columns, enabling fast lookups and updates.
- Supports definition and manipulation of columns, including data types, constraints, and default values.
- Handles indexing, particularly through the primary key, to speed up data retrieval processes.

5.1.2.4. Database Module

Objective:

- To manage the collection of tables and provide an interface for executing queries, managing transactions, and ensuring data integrity.

Implementation:

- Stores metadata about tables, columns, and relationships between tables.
- Implements transaction management features, including commit, rollback, and concurrency control mechanisms.
- Ensures data integrity through constraints and validation checks.
- Facilitates connection management, allowing multiple clients to interact with the database simultaneously.
- Incorporates the Storage Engine, which utilizes Zstandard (Zstd) compression for efficient data storage and retrieval.

5.1.2.5. Notifier Module

Objective:

To manage real-time notifications and updates, allowing clients to subscribe to changes in the database and receive updates instantly.

Implementation:

- Built on the actor model to handle asynchronous messaging and updates between the database and clients.
- Allows clients to subscribe to specific tables, receiving updates via WebSockets when data changes.
- Manages a list of active subscriptions and associated clients, ensuring timely and relevant updates.
- Integrates with the Database module to capture and broadcast changes, including inserts, updates, and deletes.

Each of these modules plays a critical role in the overall functionality and performance of SocketDB, ensuring that the system can efficiently parse queries, evaluate conditions, manage data, and notify clients about relevant updates in real-time.

5.2. Result Analysis

The SocketDB system integrates various components and algorithms to enhance performance, reliability, and real-time data interaction capabilities. The system employs a recursive descent algorithm for SQL parsing, ensuring comprehensive and accurate parsing of user queries. This approach allows SocketDB to handle complex SQL statements effectively, maintaining high adaptability to varied query structures.

The data evaluation mechanism, particularly for SELECT queries, demonstrates the system's efficiency in data retrieval. By employing a two-step process—first executing selection criteria and then performing projection—SocketDB ensures that data processing is both precise and resource-efficient. This methodology aligns well with the columnar data layout, optimizing the retrieval process and enhancing overall system performance.

In terms of real-time data handling, the WebSocket notifier module stands out by providing instant data updates to clients. This feature, powered by the actor model, facilitates asynchronous communication and update dissemination, proving essential for applications requiring real-time data synchronization.

The storage engine, utilizing Zstandard compression, contributes significantly to reducing storage space and improving I/O operations. This choice reflects a well-considered balance between compression efficiency and performance, underpinning the system's scalability and speed.

System testing reveals that SocketDB maintains robust performance and accuracy across various scenarios, from basic data operations to complex transaction handling and real-time notifications. The tests confirm the system's resilience and capability to recover from failures, ensuring data integrity and continuous service availability.

However, increasing complexity or load beyond a certain point could potentially lead to diminishing returns, such as reduced performance or increased resource consumption. It's critical for SocketDB to maintain a balance between functionality and efficiency, ensuring that enhancements, such as increasing concurrency levels or adding more complex query capabilities, do not adversely impact the core system performance.

In conclusion, SocketDB's modular design and the integration of specialized algorithms for parsing, data evaluation, and real-time notifications form a solid foundation for a high-performance, reliable database system. The system testing results underscore its capability

to handle a variety of data workloads and operational demands, making it a viable solution for applications requiring fast, consistent, and real-time data management.

6. CONCLUSION AND FUTURE RECOMMENDATION

6.1. Conclusion

The development and testing of SocketDB have demonstrated its capability as a robust, efficient, and real-time SQL database system. By integrating advanced algorithms and modern data management techniques, SocketDB offers a comprehensive solution tailored for applications requiring instantaneous data updates and high throughput.

The recursive descent algorithm used in the SQL parser ensures accurate and efficient parsing of complex queries, laying a strong foundation for reliable data operations. The evaluator and table modules, designed with performance in mind, facilitate effective data manipulation and retrieval, capitalizing on the strengths of a columnar data layout. This structure not only enhances query performance but also optimizes storage utilization, a critical aspect for large-scale applications.

The implementation of the WebSocket notifier, adhering to the actor model, stands out as a pivotal feature of SocketDB, enabling real-time data delivery and ensuring that client applications remain consistently synchronized with the database state. This real-time capability is particularly beneficial in today's fast-paced data environments, where timely information can provide significant competitive advantages.

Furthermore, the system's resilience, demonstrated through comprehensive testing, confirms its reliability and robustness under various operational conditions. The use of Zstandard for data compression strikes an effective balance between storage efficiency and performance, further enhancing the system's scalability.

However, as with any system, there is always room for improvement and adaptation, especially in the rapidly evolving landscape of database technology. Future enhancements could include expanding the query capabilities, improving the scalability to handle larger datasets, and refining the real-time notification system to cater to more diverse and complex application requirements.

In conclusion, SocketDB represents a novel setp forward in the realm of database systems, particularly for applications demanding real-time interactions and high data throughput. Its modular design, combined with efficient algorithms and a user-centric approach, positions it as a valuable tool for developers and businesses alike. As the system continues to evolve,

it is poised to meet and exceed the growing demands of modern data management and analysis.

6.2. Future Recommendation

For the future development of SocketDB, enhancing support for SQL joins and extending the feature set for more robust SQL capabilities stand out as crucial improvements. By integrating advanced join types and expanding the system's handling of complex queries, SocketDB can better meet the needs of users requiring detailed data analysis and manipulation. Additionally, broadening support for SQL standards, including sophisticated filtering, grouping, and aggregation functionalities, will significantly increase the system's utility and alignment with industry-standard practices.

Improving data ingress and egress mechanisms, alongside bolstering fault tolerance, will also be pivotal for SocketDB's evolution. Sophisticated data handling capabilities will facilitate efficient large-scale data operations, crucial for big data applications, while enhanced fault tolerance ensures database reliability and stability, particularly in distributed environments. Implementing these improvements will position SocketDB as a more competitive and versatile solution in the database technology landscape.

REFERENCES

[1] ‘SurrealDB | The ultimate multi-model database’, SurrealDB [Online]. Available: https://surrealdb.com/ [Accessed: 20-Sept-2023]

[2] Gaffney, Kevin & Prammer, Martin & Brasfield, Larry & Hipp, D. & Kennedy, Dan & Patel, Jignesh. (2022). SQLite: past, present, and future. Proceedings of the VLDB Endowment. 15. 3535-3547. 10.14778/3554821.3554842.

[3] Gjengset, Jon & Schwarzkopf, Malte & Behrens, Jonathan & Araújo, Lara & Ek, Martin & Kohler, Eddie & Kaashoek, M & Morris, Robert. (2018). Noria: dynamic, partially-stateful data-flow for high-performance web applications

[4] M. Budiu, F. McSherry, L. Ryzhyk, and V. Tannen, "DBSP: Automatic Incremental View Maintenance for Rich Query Languages," 2022. [Online]. Available: arXiv:2203.16684 [cs.DB].

APPENDIX I

```
socketdb on  main is 📦 v0.1.0 via 🦀 v1.76.0
❯ ./target/release/socketdb
>> select 1 + 1;
+----------+
| ?column? |
+----------+
| 2        |
+----------+

>> .tables
+------+---------+
| name | columns |
+------+---------+

>> create table mytbl (id int primary key, val int);
>> .tables
+-------+---------+
| name  | columns |
+-------+---------+
| MYTBL | id      |
|       | val     |
+-------+---------+
```

Figure A: Creating a Table

```
>> insert into mytbl values(1, 2), (3, 4), (5, 6);
>> select * from mytbl;
+----+-----+
| id | val |
+----+-----+
| 1  | 2   |
+----+-----+
| 3  | 4   |
+----+-----+
| 5  | 6   |
+----+-----+

>> select * from mytbl where id > 2;
+----+-----+
| id | val |
+----+-----+
| 3  | 4   |
+----+-----+
| 5  | 6   |
+----+-----+
```

Figure B: Inserting and Selecting from a Table

APPENDIX II

```
table: MYTBL updated +-------+-----+ | id | val |

Text

1  table: MYTBL updated
2   +-------+-----+
3  | id    | val |
4  +-------+-----+
5  | 1     | 2   |
6  +-------+-----+
7  | 3     | 4   |
8  +-------+-----+
9  | 5     | 6   |
10 +-------+-----+
11 | 23635 | 76  |
12 +-------+-----+

Connected to ws://localhost:8080/ws?table=mytbl
```

Figure C: Client receiving updates via WebSocket