IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025

DOI: 10.17148/IJARCCE.2025.14453

OWN JSON QUERY DATABASE: BRIDGING NoSQL and SQL

Dhayanithi S R[1], Diwakar P[2], M Maheswari[3]

Student, B.E. CSE, Anand Institute of Higher Technology, Chennai, India[1, 2]

Assistant Professor, CSE, Anand Institute of Higher Technology, Chennai, India [3]

Abstract: A traditional SQL plus NoSQL database has its major point of expansion as well as limitation in correlation to local storage situations for fast scalability. The paper presents the JSON Query Database (JQDB) as a lightweight local database application written in Python, which promises ease of such NoSQL-like flexibility but structured querying akin to SQL. This combination allows users to operate JQDB under SQL or NoSQL mode, specifying the mode at the database initialization, while CRUD is driven through a custom query language. JQDB operates entirely off local storage, hence not requiring any network database setup. JQDB is built with Python, employs JSON for data storage, has a basic command-line interface, and offers front-end integration for easy interactions. This paper discusses the architecture behind JQDB, command execution, and future work, thus establishing the capability of lightweight yet powerful local database solutions

Keywords: NoSQL Database, SQL-like Commands, Local Storage, Python Database, CRUD Operations.

I. INTRODUCTION

JSON Query Database (JQDB) comes as a solution to the rising complexity and heterogeneity in modern data management systems. With both NoSQL and SQL databases gaining momentum rapidly, applications no longer enjoy the luxury of handling structured, semi-structured, and unstructured data together. Traditional SQL databases hold fast to relational integrity and consistency and therefore are appropriate for well-structured data but not so accommodating to schema alterations. Conversely, NoSQL systems are optimized for scalability and adaptability but compromise the relationships among data and the querying facilities afforded by SQL.

To meet this challenge, the project gives way to the JSON Query Database, which is a lightweight and local storage database developed in Python. Unlike conventional databases that rely on external servers or cloud-based infrastructure, JQDB works entirely on the local machine to render it offline accessible and its deployment easy to implement. In selecting its initialization option, users would have the freedom of choosing the SQL-like structured querying or the NoSQL document-based storage, thus moving seamlessly between both models of data depending on user preference. This approach makes it highly flexible and suitable for standalone applications, local data analysis, and embedded systems where external database dependencies may be unwarranted.

JQDB, too, concentrates on workflows and front-end integration, making it suitable for applications depending on real-time data access and manipulation. It offers an easy-to-use API for interfacing with front-end interfaces, facilitating interaction between web and desktop applications. With role-based access control, data security can also be set in place, making specific data available only to a particular group of users. The objective is to provide a versatile, yet intuitive, database that may at some time function as either structured or semi-structured storage.

To meet the increasing demands of data flexibility and simplicity in local contexts, the JSON Query Database (JQDB) presents itself as a revolutionary hybrid solution that combines the structured reliability of SQL with the flexible nature of NoSQL. Written in Python, JQDB provides users with the option to work in SQL or NoSQL mode, within a light, portable system that runs completely offline—no internet connection or external database servers required. Its JSON storage model makes it especially ideal for students, teachers, data analysts, hobbyists, and developers in resource-constrained environments. The heart of JQDB is its command-based query interface that is designed specifically for it, and it allows basic CRUD operations alongside sophisticated filters and condition searches. It balances the nature of SQL-like syntax with the schema-less adaptability of NoSQL to enable users to set up database mode during initialization to best meet changing project demands. JQDB not only makes local data management easier but also promotes a more flexible, effective, and natural methodology in handling structured and semi-structured data.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025

DOI: 10.17148/IJARCCE.2025.14453

JQDB is a tool providing a new approach to local database management with an emphasis on performance, flexibility, and ease of use. Users can combine SQL-like querying with NoSQL storage so that they can effectively store, retrieve, and manage data without having to go through the complications of a typical DBMS. The architecture, command execution, and future outlook of JQDB are examined in this paper, revealing the potential of JQDB as a lightweight, powerful local database solution for desktop applications

Beyond the command-line tool, JQDB is designed with extensibility as an underlying philosophy that includes APIs used to seamlessly interface with both web-based and desktop front-end apps. Developers use this to develop interactive admin panels, data visualizers, and dashboards which can read, write, and manipulate local data in real-time. To make it more useful in multi-user contexts, JQDB also supports role-based access control (RBAC), so permissions and data visibility can be controlled based on user roles—thus improving both security and convenience. In the future, JQDB has great potential to become an even more powerful local-first database system. Future releases will add functionality including data indexing, query optimization, import/export features, encryption for secure data, and adding a graphical user interface for those who wish to interact via graphics rather than command line. There is also potential for incorporating an optional hybrid sync capability that might allow backups to the cloud or integration of the cloud as a local storage area with local storage integrity and autonomy preserved.

Finally, the key findings highlight that JQDB functions as a secure, adaptable, and lightweight local database solution that empowers users to manage structured and semi-structured data effectively without relying on external servers. Its dual-mode capability supports both SQL-like and NoSQL-like operations, making it ideal for developers who prefer command-line control and local storage independence. JQDB ensures data privacy by keeping all operations confined to the local system, reducing risks associated with remote access and cloud vulnerabilities. As the demand for hybrid data systems grows, JQDB is positioned to evolve further—integrating enhanced features such as encryption, indexing, and GUI support. Future versions of JQDB aim to strengthen its role as a dependable local-first database system that bridges performance, simplicity, and security for modern application development.

II. RELATED WORK

We studied existing tools, visualization platforms, and hybrid database systems to design JSON Query Database (JQDB) as a flexible local storage solution. This includes analysing semi-structured data challenges, quality management gaps, and performance limitations in traditional SQL and NoSQL systems.

Background information was compiled and collected to better understand the aims and objectives of the study. The current literature review is covered in sections: historical evolution of database management systems-the demands of hybrid SQL and NoSQL solutions-the importance of query languages that are custom designed in achieving specialization for databases-the advantages of local storage databases for stand-alone applications-the perils of the traditional database model in dealing with dynamic data requirements.

Currently, some software applications and research papers have been devoted to assisting users in assessing and improving the quality of tabular datasets, leaving semi-structured data under-exploited. Semi-structured data, e.g., JavaScript Object Notation (JSON), XML, and NoSQL database, is ubiquitous and plays a progressively more important role with the rapid development of the Internet [1]. For example, in exploratory data analysis, one has to first solve quality issues to obtain valuable and reliable results. Besides, in NoSQL database quality assurance, data stewards must assess and ensure that the quality of data complies with predefined needs or standards. Chu et al acknowledged that quality issues for semi-structured data remain a challenge in data cleansing research [2]. There are two salient features that distinguish semi-structured data from structured data. First, semi-structured data is schema less as it is not restricted to a fixed or rigid schema defined in advance, allowing keys to be added, changed, or deleted at any time. Though flexible, this inconsistency impedes data comprehension and transformation. Second, unlike structured data that typically represents data in a table format, semi-structured data supports a hierarchical structure. Identifying quality issues that reside deep within the hierarchy requires traversal along the hierarchy [3].

It appears that email summarization has been addressed in numerous ways using natural language processing and also using machine learning algorithms. Some traditional methods as extractive summarization focusses on extracting key emails that may contain the salient information of the messages but most of the time, they don't preserve the cohesiveness and the context. The abstractive methods based on deep learning try to come up with brief summaries while relying heavily on the massive training data availability and facing definite cohesion issues. There are other techniques that have been used for keyword extraction such as TF-IDF and LSA but are not semantically aware.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025
DOI: 10.17148/IJARCCE.2025.14453

Bert and T5 are the latest transformer models that inspired superior performance in the stream of email summarization while they need to be adapted for the specific usage. This is can be attributed to the fact that whereas rule-based techniques provide the much-needed control over the resultant structures, they do not have flexibility of applying on different structural configurations of emails.

Some existing tools are able to extract and visualize the schema of semi-structured data. JSONDISCOVERER aims to facilitate the integration and composition of JSON-based Web APIs by discovering and visualizing the implicit schema of JSON documents. STEED is an analytical database system that learns and extracts a schema tree for tree-structured data (such as JSON data) and provides visualizations of the schema tree and its summary statistics.

JSON Crack facilitates comprehension, discovery, and analysis of JSON structures by rendering JSON data as graphs. SCHEMADRILL enables the display of an aggregate schema representation that reduces the complexity of JSON schemas without losing significant information. These visualization tools are not intended for quality management of JSON. Such choices made of putting an information technology and the communication technology in storage and the control of inventory can be great opportunities to encourage the relational commitment behaviour. Newer or older version might exhibit significant differences in performance and reliability as restoration from failure. File system storage is also software version dependent but large variations in various versions of database software can be compensated by file system performance improvement between various file systems. If the speed difference cannot be improved considerably reliability features becomes important.

Performance comparisons between MySQL and MongoDB show that MongoDB outperforms MySQL by 55.77%, completing tasks in 42 minutes and 5 seconds compared to MySQL's 95 minutes and 8 seconds. This demonstrates MongoDB's superior efficiency in handling large datasets, particularly in scenarios where performance and speed are critical. Similarly, studies comparing HiveQL, Spark SQL, and Impala in cluster environments reveal that Impala delivers the fastest query speed, making it more suitable for quick queries and real-time data analysis in big data applications. Further evaluations of Hive and Spark SQL focused on analysing large web archive collections, specifically examining how the type of Hive table schemas impacts query performance. The study proposed a performance analysis framework designed to better understand and optimize query processing across these technologies. In terms of computational efficiency, Spark has been identified as more efficient than Hadoop, although it requires significantly higher memory allocation. Ahmed et al investigated the key parameters affecting Hadoop and Spark performance, including resource utilization, input splits, and shuffle operations, which have a direct impact on overall system performance and scalability. Additionally, two major concerns in the use of large-scale database systems are the protection of data in the event of failure and the presence of I/O bottlenecks, which can severely degrade performance. Solutions to these issues include implementing regular backups to ensure data integrity, utilizing RAID mirror systems for redundancy, and deploying load balancing mechanisms to distribute traffic evenly across multiple database servers. These strategies enhance system reliability and provide safeguards against unexpected failures or data loss.

Choosing different file system for database storage may decrease I/O bottleneck, resolve problem without investments and even introduce new features which can results in increased reliability of the database server [12].

III. PROPOSED METHODOLOGY

A. System Architecture

The system architecture of JQDB is built to support both SQL and NoSQL queries through a command-line interface. User queries are processed and optimized by the Query Processing module, which separates SQL and NoSQL commands. The Execution module handles CRUD operations and allows data export. Data is stored as either table-based (SQL) or document-based (NoSQL), then merged into a unified JSON format. This JSON output enables backend integration for frontend apps. Finally, results are displayed via the CMD interface, making JQDB a lightweight, flexible local database solution.

B. Database Work Flow

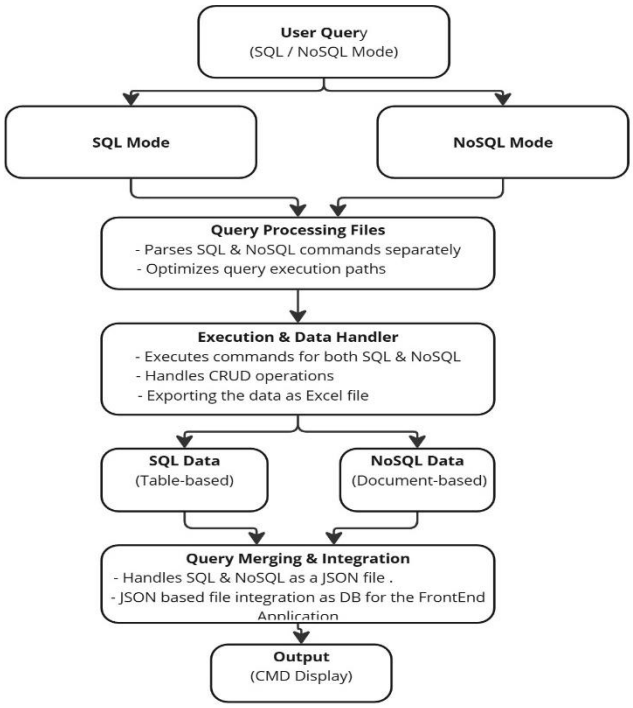
The database workflow of JQDB starts with the user initiating a query via the command-line interface, where they specify whether they want to operate in SQL or NoSQL mode. The system then parses the command and routes it through the appropriate execution engine. In SQL mode, the engine interprets the query using a structured schema-like format, performing traditional table-based CRUD operations.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025

DOI: 10.17148/IJARCCE.2025.14453



In NoSQL mode, the system handles data stored in a schema-less JSON format, enabling flexibility for semi-structured or unstructured data. After processing, the query results are stored or updated within the local JSON files that act as the core database storage. Users can choose to export the data into Excel format, supporting quick sharing and analysis. The results are also merged and converted into a unified JSON structure to ensure compatibility with frontend applications for visualization or interaction. JQDB logs each operation to maintain traceability and provides real-time output back to the command line for user feedback. This seamless local-first workflow enables users to manage data efficiently without the need for an internet connection or external server dependencies.

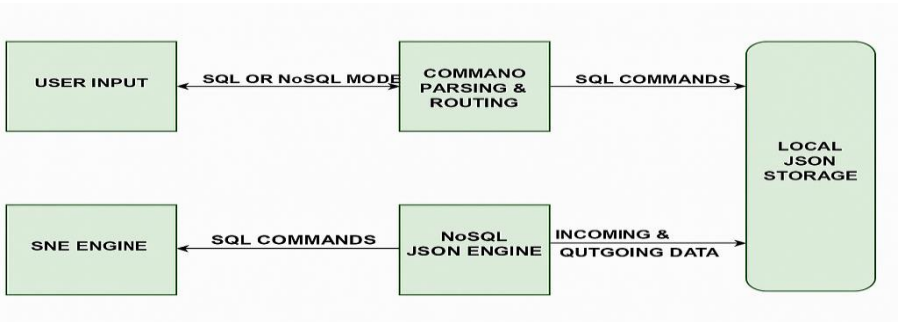


Fig (b) Work Flow estimation

C. SERVER

To enhance accessibility, scalability, and remote data management, JQDB introduces a **server feature** that allows users to interact with the database over a network rather than being restricted to local storage. This feature is implemented using **Fast API**, a high-performance Python framework, enabling seamless execution of SQL and NoSQL queries through RESTful APIs. With **multi-user support**, multiple clients can access, modify, and retrieve data simultaneously, making JQDB suitable for team-based applications. Security is a key aspect, with **JWT (JSON Web Token) authentication** ensuring that only authorized users can perform database operations. Additionally, the server mode enables **real-time data access**, allowing instant updates across connected applications. It also facilitates **web and mobile integration**, enabling front-end applications built with React, Angular, or mobile frameworks to fetch and modify data remotely. By supporting **lightweight cloud-based deployment**, this feature extends JQDB's usability to **collaborative applications, real-time analytics, and IoT-based systems**, making it a versatile solution for both offline and online environments.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025

DOI: 10.17148/IJARCCE.2025.14453

D. IMPLEMENTATION

The JSON Query Database implementation, nicknamed JQDB, is built around the integration of SQL and NoSQL capabilities into a single system for managing structured and unstructured data. Users may utilize SQL-like operations in structured (table-based) data and document-based NoSQL data in JSON format. The system would thus allow queries processing through custom-built parsers, run an add, update, retrieve, and delete (CRUD) operations, and export data if needed. Query Parsing identifies whether the input is an SQL or NoSQL query and processes it accordingly. Execution Engine executes the parsed query, interacting with the appropriate data storage model. Storage Management maintains table-based storage for SQL mode and document-based storage for NoSQL mode. Data Exporting converts and exports query results into an Excel file for further analysis. Integration Layer merges SQL and NoSQL data into a unified JSON format, making it accessible for front-end applications.

IV. RESULTS AND DISCUSSION

A. Overcomes

The server component in JQDB greatly improves its functionality by overcoming the shortcomings of conventional local storage databases, especially in accessibility, security, and scalability. JQDB was previously limited to a single-machine setup, limiting collaborative access and necessitating manual data sharing. With the addition of a server-based architecture, users are now able to access the database remotely via RESTful API endpoints, supporting multi-user access and real-time data modifications without any hassles. Security has been a major issue in local databases since they do not have proper authentication mechanisms, thus exposing them to unauthorized access and data breaches. To address this, JQDB now also supports JWT (JSON Web Token) authentication such that only specific users are capable of performing database operations. The inclusion fortifies user authentication, avoids data modification, and advances data security since it allows the implementation of role-based access control.

B. Discussion

The integration of a server-based architecture in JQDB marks a significant advancement, transforming it from a local database system into a scalable, network-accessible solution that supports multi-user access, real-time data updates, and secure remote operations, making it ideal for modern applications. Leveraging RESTful APIs, JQDB can seamlessly connect with web and mobile applications, enabling dynamic data retrieval and updates. The introduction of JWT authentication strengthens security by ensuring that only authorized users can perform database operations. However, this server-based architecture also introduces challenges such as increased server load, network dependency, and potential latency issues. To maintain high performance and responsiveness in large-scale deployments, solutions like query optimization, caching mechanisms, and load balancing strategies will be essential. Moreover, JQDB's ability to handle both SQL and NoSQL queries enhances its versatility, making it suitable for applications requiring structured and unstructured data processing. Real-time data synchronization across multiple users can present concurrency control challenges, which require advanced transaction management techniques such as ACID compliance, optimistic locking, and data versioning to prevent conflicts. Additionally, scalability improvements, including database sharding and horizontal scaling, can further optimize JQDB for cloud-based environments with high data loads.

V. CONCLUSION

In summary, JQDB's server feature provides a highly accessible, scalable, and secure approach to managing both structured and unstructured data over a network, making it an ideal solution for modern applications. This enhancement transforms JQDB into a robust tool that enables remote access, multi-user collaboration, and secure data operations, benefiting developers, system administrators, and data analysts who need a lightweight yet powerful database solution. By integrating RESTful APIs and JWT authentication, JQDB ensures efficient data management and protects sensitive data, allowing seamless interaction with web and mobile applications.

Despite being an evolving solution, JQDB continues to focus on enhancing its security, performance, and multi-user collaboration features, positioning itself as a promising, essential tool for modern database management. With continuous development, it offers users greater control, flexibility, and efficiency in handling data, ensuring it will be a must-have solution for managing data in today's increasingly complex digital landscape.

Presently, JQDB is compatible with hybrid SQL and NoSQL queries, so it can support a variety of application needs, whether structured or unstructured. The ability to support both kinds of queries gives JQDB great versatility to support varied usage scenarios ranging from the usual relational database activities to newer, NoSQL-driven applications.

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2025

DOI: 10.17148/IJARCCE.2025.14453

Functions like real-time data consistency, API-driven communications, and deployment through the cloud only enhance its attraction, in that it becomes not only readily accessible but efficient and forward-compatible. Such functionalities help JQDB fit quite comfortably within distributed systems and clouds and offer ease of scalability as well as high availability with massive deployment.

REFERENCES

[1]. Convert JSON to other formats and vice-versa, 2023.
[2]. Json Curer: Data Quality Management for JSON Based on an Aggregated Schema, June 2024.
[3]. A Comparative Study of HiveQL and SparkSQL Query Performance in a Cluster Environment, ICRAIE 2023.
[4]. Database System Performance in Correlation with Different Storage File Systems, Polytechnic of Zagreb, 2021. An Overview of Hybrid Databases, 2024.
[5]. Database System Performance in Correlation with Different Storage File Systems, Polytechnic of Zagreb, 2021.
[6]. Extracting Information from JSON Database as Simple as Extracting in SQL Using JSONiq, 2022.
[7]. QWRL: A TSQL2-Like Query Language for Temporal Ontologies Generated from JSON Big Data.