Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

JSON-based control model for SQL and NoSQL data conversion in hybrid cloud database

Lei Zhang[1,2], Ke Pang[3*], Jiangtao Xu[4] and Bingxin Niu[5]

Abstract

A data interaction transformation model, XYJSON, that is suitable for all data using current standard SQL syntax and JSON document data is proposed to solve the problem of increasing development workload and difficulty caused by using different control methods for corresponding types of databases under the cloud hybrid storage. A control program was studied to control relational and NoSQL data at the same time, by establishing a general conversion model between relational and NoSQL data and converting standard SQL statements into JSON. The performance of XYJSON was compared with that of the traditional mode. The results show that the performance difference between XYJSON and the traditional mode is small. In addition, a developer survey was conducted on XYJSON for user friendliness and compatibility. All developers rated XYJSON as excellent. The current cloud hybrid storage cannot use a unified control model to realize data control. XYJSON breaks through this bottleneck, making it easier and more efficient to control different types of databases under cloud hybrid storage.

Keywords: Simultaneous control, Data conversion, Hybrid cloud database, Native JSON, NoSQL, RDBMS

Introduction

With the rapid development of cloud computing and big data technology, the limitations of data processing brought by traditional single type database are more prominent. Using JSON text storage in a database requires expensive text processing each time a document is read by a query or updated by a DML statement [1], and NoSQL databases do not provide transaction mechanism functions [2]. Owing to the storing of complex logical relations, a relational database cannot meet the requirements of big data in application scenarios requiring high-performance data throughput. Therefore, hybrid cloud storage is being extensively used as a cutting-edge data exchange and storage method applicable to both relational and non-relational databases. Hybrid cloud storage integrates NoSQL and relational data formats. The NoSQL document storage system in such hybrid cloud storage mechanisms manages the substantial data exchanged and stored in JSON format [2], which aims to achieve efficient and simple access such that they are suitable for high-frequency access application scenarios with a single data table store structure. By contrast, relational databases are suitable for data with highly complex relations and compound queries based on these relations. Thus, they are utilized in scenarios that require data model statistics and predictions. Hybrid cloud storage integrates the advantages of these two types of databases, overcomes the limitations of single type database storage, and realizes efficient access to various data types.

At the same time, in hybrid cloud storage, developers need to use different types of control methods to control their corresponding types of databases, as shown in Fig. 1a. Relational data needs to control data storage through SQL control, while NoSQL data controls data access through data control. They lack a unified control

*Correspondence: xiaolei-zhl@163.com

[3] School of Software and Communication, Tianjin Sino-German University of Applied Sciences, Tianjin 300350, China
Full list of author information is available at the end of the article

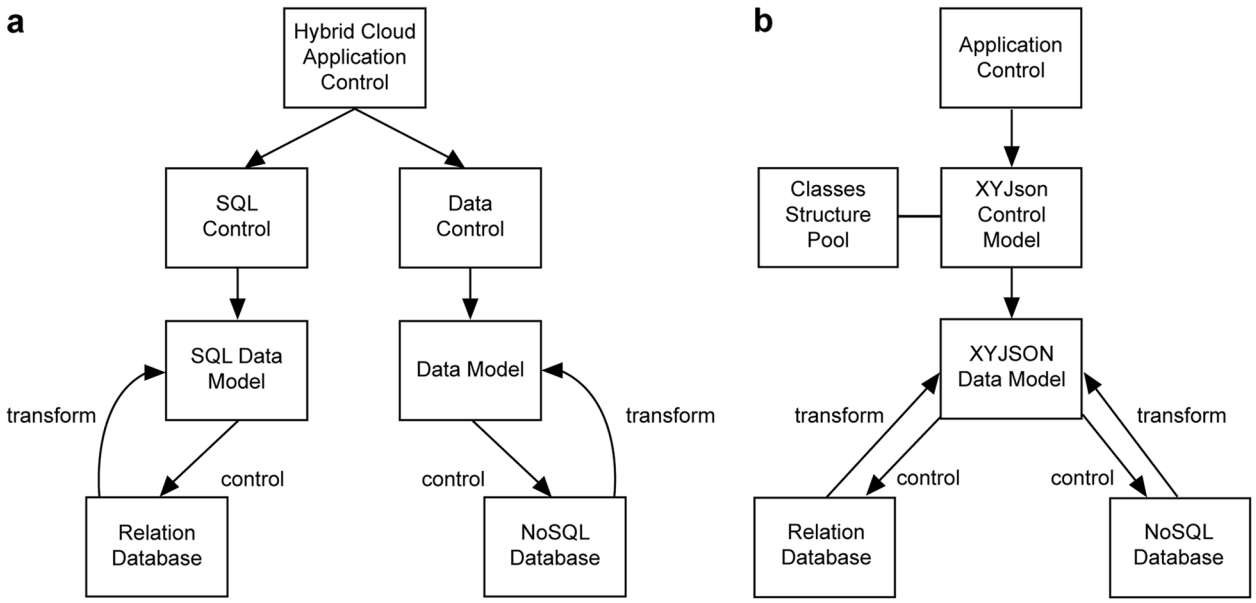

Fig. 1 Comparison of XYJSON and traditional hybrid database control structures (**a**) Traditional hybrid database control structure (**b**) XYJSON hybrid database control structure

method. This control method not only increases the development workload and difficulty of developers, but also becomes a bottleneck hindering the further development of hybrid cloud storage mechanism.

Facing this challenge, a data interactive conversion model suitable for all data using current standard SQL syntax and JSON document data is proposed in this study. By establishing a general conversion model between relational and NoSQL data and converting standard SQL statements into JSON, this study investigated the use of a program to control relational and NoSQL data simultaneously. This new JSON model is named XYJSON, as shown in Fig. 1b. The current cloud hybrid storage cannot use a unified control model to realize data control. XYJSON breaks through this bottleneck, making it easier and more efficient to control different types of databases under cloud hybrid storage.

Related work

Currently, many mainstream relational databases, such as Oracle [3], Microsoft SQL Server [4], MySQL [5], PostgreSQL [6], and TeraData [7], are being actively explored to identify ways of optimizing database performance to adapt to the big data era. Thus, attempts have been made to integrate JSON text storage into relational databases for compatibility with NoSQL databases, thereby achieving efficient hybrid cloud storage. Nevertheless, the characteristics of relational databases themselves have led to their inherent inability to perform JSON processing [8]. This deficiency has also made developers reluctant to use a single relational database to simultaneously process high-throughput data and high-complexity logical relational data in a modern hybrid cloud storage system. Moreover, researchers have proposed several methods of storing JSON text in relational databases. Storing native JSON data in commercial databases and using SQL for extended queries were discussed [2, 9]. A JSON hybrid query language based on SQL was proposed [10]. Two different mapping techniques, which were used to store JSON data in relational databases, were proposed and compared [11]. The entity–attribute–value data model was used to discuss the support of two open-source relational databases and two commercial relational databases for JSON documents [12]. The experimental results showed that JSON data can be used to simplify queries and reduce their execution time.

To a certain extent, the integration of JSON text stored in a relational database has addressed the interaction of different types of data in hybrid cloud storage. However, a relational database is not suitable for a JSON text storage because it is primarily designed to store relational data structures. Moreover, the compatibility of the JSON text stored in relational databases relies on its SQL statements, owing to the limitation of its structure.

Therefore, many researchers have attempted to address the interaction between different types of data in hybrid cloud storage from another perspective, that is, to achieve interaction through the mutual mapping between JSON and a relational database to realize the unified control of a JSON text storage and a relational database.

In terms of research on mapping JSON data to relational data, a mapping algorithm from JSON to a relational database was proposed [2], and JSON data were stored in the relational database [13–15]. JSON was defined in a web data request and a theoretical analysis to study the constraint method of JSON integrity was conducted [16]. A formal JSON data model was proposed [17], and a lightweight query language was defined. Interestingly, JSON conversion in MongoDB was specified as future work in that paper [18]. A data exchange format among RESTFul services was proposed, which is more inclined to store network attribute data [19].

Based on the existing mapping models between JSON and relational data and non-relational data, the paper combined these two mapping models and proposed a novel JSON model named XYJSON model. Using XYJSON's control model matching, this data model achieved unified control of different types of databases, helping fill the gap in the application control model for hybrid cloud databases and promoting research on unified control for hybrid cloud databases.

XYJSON model

XYJSON is divided into two parts: XYJSON data model and XYJson control model. As shown in Fig. 1b, the XYJSON data model is compatible with relational as well as NoSQL data and can realize data conversion according to different database types; that is, the XYJSON data model can be converted using relational data or NoSQL data. By transforming the XYJSON data model into an XYJSON control model driver, it can control the XYJSON data model, and consequently control relational and NoSQL databases.

XYJSON data model

In this study, a new native JSON-based data model is defined, which can interconvert standard SQL statements and JSON document structures. This JSON document data is referred to as the XYJSON model. Because the XYJSON data model was designed based on the native JSON syntax, it can be stored in a NoSQL database that adapts to native JSON and can be verified and parsed by program components that parse native JSON. This design method enhances the versatility of the XYJSON data model.

The syntax structure of the XYJSON data model is shown in Fig. 2. The XYJSON data model is divided into three first-level nodes, namely commandType, commandOp, and data nodes. "commandType" represents the corresponding SQL command, including four SQL command formats: INSERT, UPDATE, DELETE, and QUERY. "commandOp" represents the operation object, and the "tableName" contained within is the table name

```
{
    "commandType" : "INSERT | UPDATE | DELETE | QUERY",
    "commandOp" : {
        "tableName" : value,
        "colList" : [value,……],
        "queryList" : [@Fun(value
    },
    "data" : [{key : value,……},{key : value,……},……]
}
```

Fig. 2 XYJSON data model structure

corresponding to SQL; "colList" is the database field name, presented in the form of a string array; "queryList" is the query condition, presented as an array, and the corresponding query conditions are composed of query functions (see 2.3); and "data" is relational data, presented in the form of an array of Key–Value pairs.

XYJson control model

The XYJSON control model is the model driver of the XYJSON data model. The control model persistently drives the data model into XYJSON Bean. XYJSON Bean conforms to the JavaBean software component model design specifications, and because the XYJSON data model is based on native JSON syntax rules, any mapping component between objects and JSON data can map the XYJSON data model to XYJson Bean, that is, the XYJSON control model.

The XYJson control model is shown in Fig. 3a. The XYJson < T > class contains attributes corresponding to the first-level nodes of the data model, namely commandType, commandOp, and data. The commandType property indicates the command type. The values are INSERT, UPDATE, DELETE, and QUERY. The commandOp is an operation type attribute with tableName, colList, and queryList attributes under it, which is consistent with the meaning of the data model. The data type of the data is the Java class corresponding to the relational data, which exists as a T-generic class in the XYJson class. The T-generic class can vary according to the structure of the actual relational data. There are mainly two annotation classes in the T-generic class, including *Table* and *Column* annotations. The Table annotation class is applied to the class name of the T-generic class, implying that users can provide the table name value for the value method of this class. Conversely, the Column annotation class is

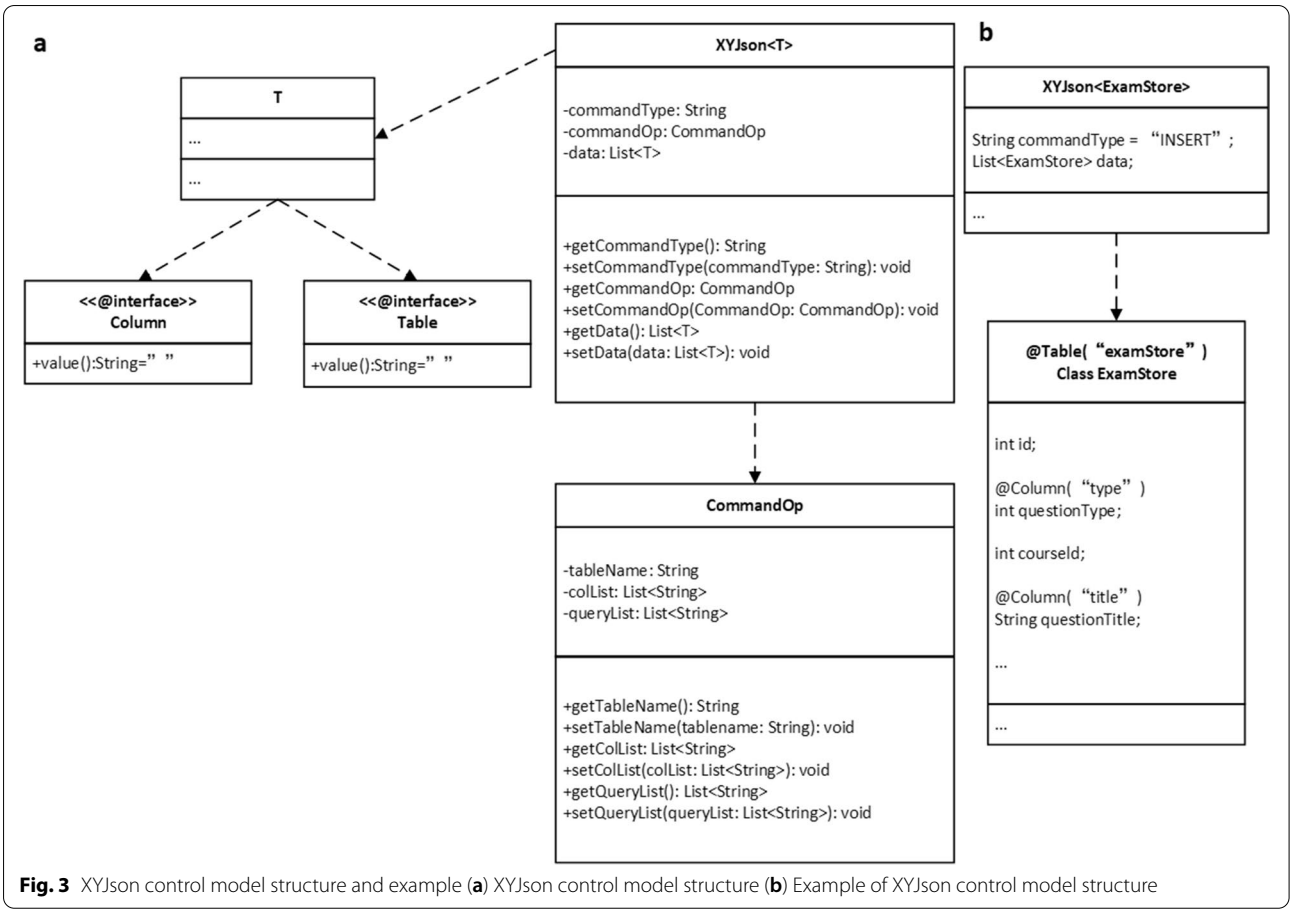

Fig. 3 XYJson control model structure and example (**a**) XYJson control model structure (**b**) Example of XYJson control model structure

applied to the attributes of the T-generic class, indicating that users can provide table field values for the value method of this class. Figure 3b is an example of the XYJson control model transform of IX1 in Fig. 4. The data model inserted in IX1 is transformed into the ExamStore Bean model. In the ExamStore Bean model, developers can use @table and @column annotations to map table names and column names in relational databases. The ExamStore Bean is used as the data attribute value of XYJson < ExamStore > in the form of an array. As the data model of IX1 only has a commandType node, only the commandType attribute in the XYJson control model corresponds to it, and its value is INSERT.

Using the model in Fig. 3b as an example, when developers need to insert the examStore table in the relational database, they can convert it into the SQL statement inserted into the examStore through XYJson < ExamStore > Bean and then operate the relational database, or they can directly operate the relational database through Bean. When they wish to operate a NoSQL database, it may be operated directly through XYJson < ExamStore > Bean. Developers can control different types of databases using XYJson < ExamStore > Bean alone and need not focus on the characteristics or programs of each database.

Query function node

Owing to the different characteristics and functions of various databases and the enhancement of the user friendliness, universality, and functionality of XYJSON, a variety of query functions are designed in XYJSON to realize different functions. The queryList node under the commandOp node represents the string array of the query function type in the XYJSON data model structure. Each query function starts with @, followed by the function name and the required parameters of the function (Fig. 5). The parameters in the query function can be nested into other functions, which also start with @. The query function represents an SQL query statement after WHERE, and it is presented by the query function in the XYJSON model document structure.

Take QX3 in Fig. 4 as an example. QX3 uses @NOTNULL and @EQ functions for business control and the @AND function for connecting query conditions. The query function @NOTNULL indicates that when the ID is not empty, the query condition with id = 1 will be used.

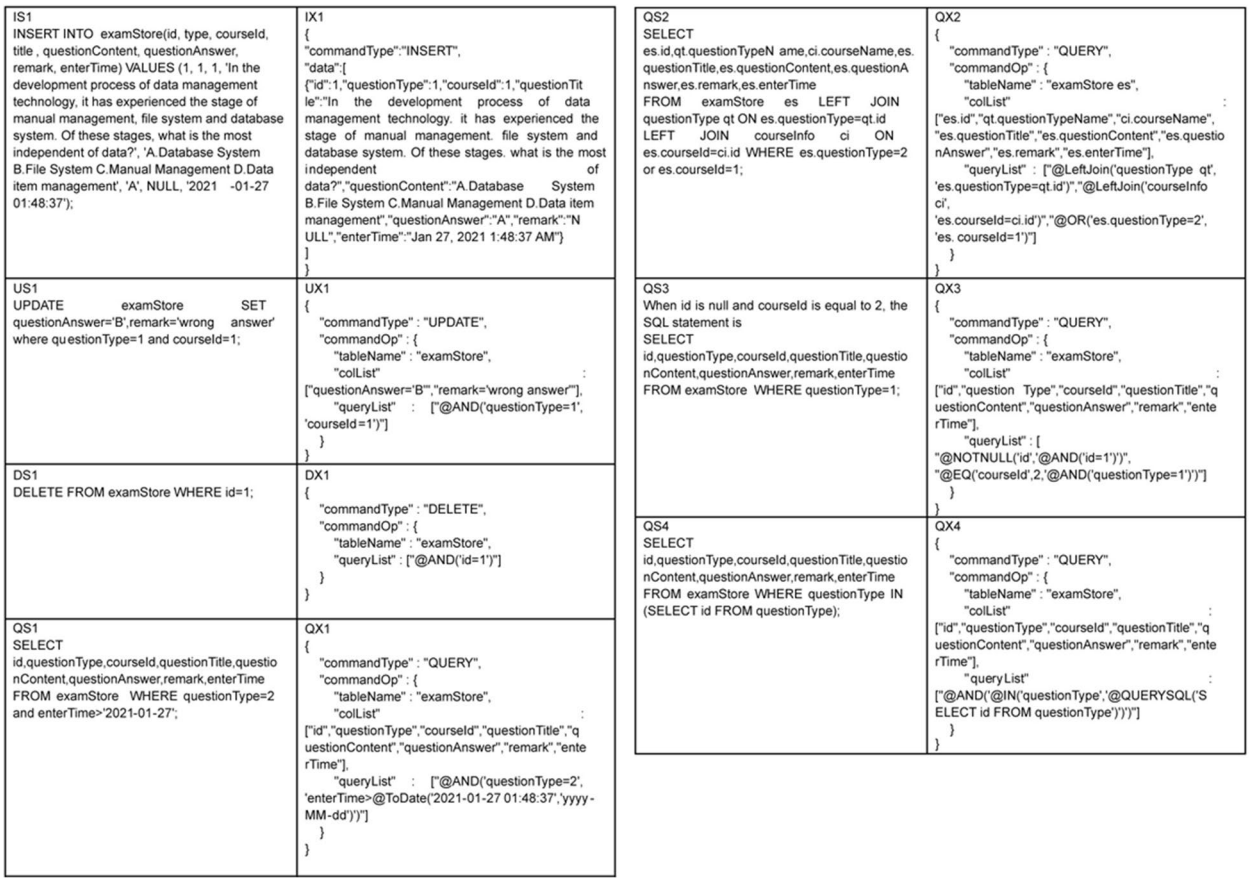

Fig. 4 XYJSON model with real examples of SQL statements

@EQ implies that when the course Id is equal to 2, the query condition with question Type = 1 will be used. @AND indicates that multiple query criteria are connected by "AND."

Application experiment

The data conversion performances of different database types were compared with the traditional and XYJSON methods in a hybrid cloud database control experiment, including relational database MySQL, relational database standard SQL text, and NoSQL databases MongoDB and Redis. The experiment was divided into four operations: data INSERT, UPDATE, DELETE, and QUERY. Each conversion operation was repeated 10 times cyclically. The duration of each experiment was calculated from the beginning of the second cycle to avoid the long running time caused by loading various component packages required by the program during the first cycle. The conversion times of the remaining nine iterations were calculated, and the average value was taken as the time reference index. The experimental results showed that the proposed XYJSON model can achieve unified control operation for different types of databases, and the performance difference ranged from − 14.28% to 9.31% compared to that of the traditional method. Research conducted among software developers also showed that XYJSON has a high user friendliness and compatibility. All the developers who participated in the research rated XYJSON as "excellent."

The experimental setup comprised a workstation with an Intel Core i7 3.1 GHz/4 core CPU, 16 GB 2133 MHz LPDDR3 memory, 2 TB SSD, and MacOS Big Sur Operating System. The software used were MySQL Ver 5.7.21,

```
"queryList" : [@Fun(value,……),……]
Fun := ('QUERYSQL'| 'AND' | 'OR' | 'IN' | 'ORDER' | 'LIKE' | 'BETWEEN' | 'TOP' | 'NOTNULL' | 'NULL' | 'EQ' | 'NEQ' | 'TODATE' | 'JOIN' | 'INNERJOIN' | 'LEFTJOIN' | 'RIGHTJOIN' | 'FULLJOIN')
```

Fig. 5 Query function grammar

MongoDB Ver 4.4.4, Redis Ver 6.2.0, and JDK1.8. The cloud service hybrid storage environment was simulated in the workstation to evaluate the application of the proposed model in cloud hybrid storage. Four dockers were adopted to start four cloud servers loaded with MySQL database, MongoDB database, Redis database and XYJSON application cloud services, respectively. Here, the first three databases were controlled through the XYJSON application server.

Insert conversion experiment

In the INSERT experiment, the sample data in the test reached 1 million rows, with a volume of approximately 300 MB. Data conversion experiments were conducted on a relational database SQL text and NoSQL databases MongoDB and Redis, and compared the data conversion time between the traditional data conversion method and XYJSON in the simulated hybrid cloud database. The objective of the experiment was to use SQL text in a relational database. First, relational databases have been developed over a long period of time, and they include several types. The characteristics of each relational database are quite different, but they all follow the SQL standard; therefore, SQL standard text was used to replace the experiment for a single relational database. Secondly, the insertion conversion experiment can realize the data migration of different types of databases in the hybrid cloud database. In the process of data migration, developers often need to use SQL text as the basis for data migration; thus, the experiments were based on SQL text. During the experiment, the relational data were stored in a.SQL file in the form of SQL standard text lines.

The experimental results are shown in Fig. 6. The data format inserted into the data table is IX1. In the figure, red indicates the time performance gap between the traditional and XYJSON models, and the digital percentage is the excess percentage of time lost. It may be observed that different database types perform data conversion and insertion operations. In the mutual conversions between SQL and MongoDB or between SQL and Redis, the conversion performances of XYJSON are lower than the traditional methods. Its performance loss was within 7.97% because, in the conversion process, XYJSON data model conversion should be performed on SQL first, and then insert operations should be performed on other types of databases with XYJSON control models, resulting in performance degradation. In the mutual conversion between MongoDB and Redis, XYJSON achieved a slightly better performance than traditional methods, that is, 0.28% and 0.45%, because in non-relational databases, document data and key values have good compatibility with the XYJSON data model, enabling them to interact directly with native JSON. Therefore, the XYJSON data model can be efficiently transformed with these two database models. Meanwhile, XYJSON loads the required entity beans into the cache pool of the class structure by the control model when the project is started, thus reducing the performance loss. In addition, XYJSON realized

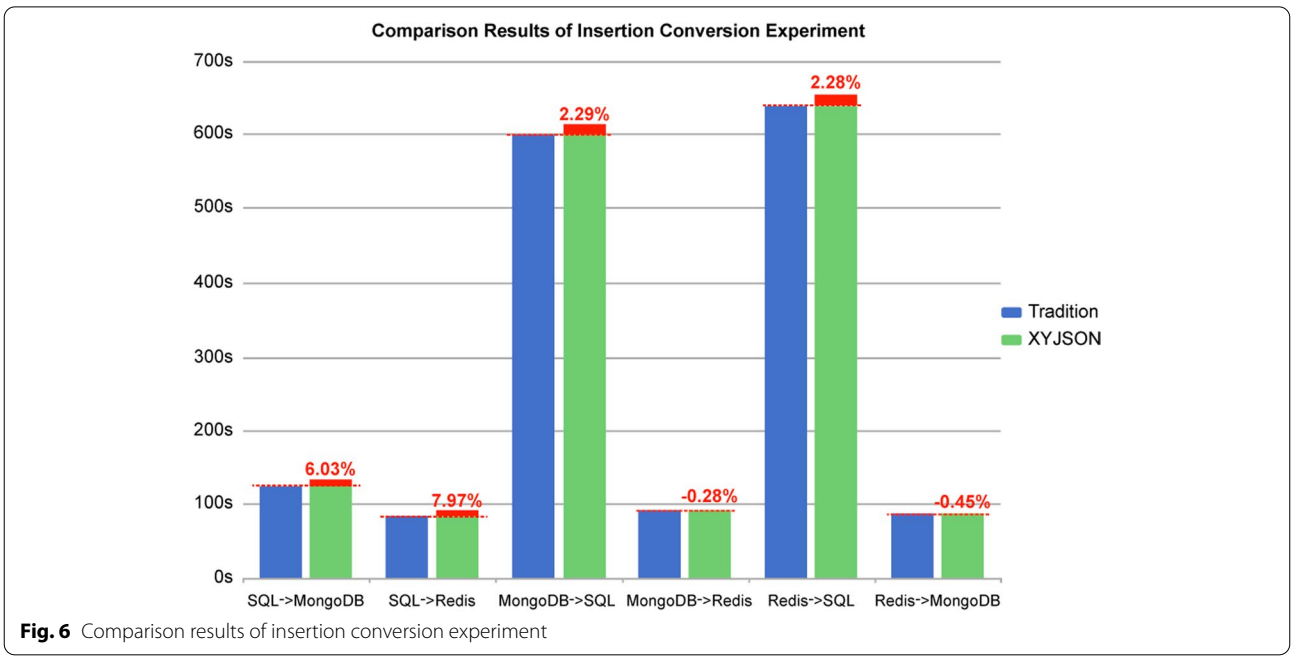

Fig. 6 Comparison results of insertion conversion experiment

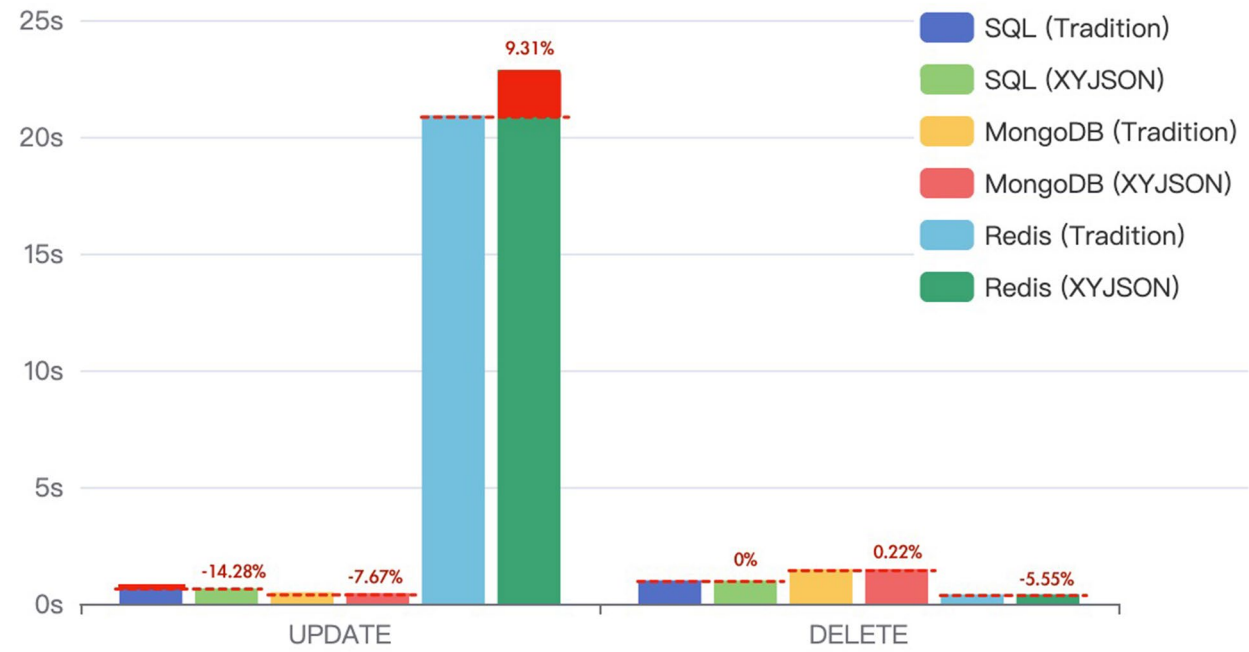

Fig. 7 Comparison results of update and delete conversion experiment

the unified data conversion operation between different types of databases, which reflects the compatibility of the XYJSON model.

Experiment of updating and deleting conversions

In this group of experiments, the UPDATE and DELETE operations of different types of databases were completed using traditional and XYJSON methods, considering UX1 and DX1 in Fig. 4 as examples. As shown in Fig. 7, the experimental results were a comparison between the time taken for the UPDATE and DELETE operations by different database types using the traditional and XYJSON methods. The experimental results showed that in the modification operation, the neutral performance of the XYJSON method and MongoDB in the relational database was slightly better than that of the traditional method by 14.28% and 7.67%, respectively. XYJSON performs conversion operations according to different modified fields and loads the data model into the class structure cache pool after parsing. In the next operation, when the model performs the same modification operation again, it does not need to be parsed again. Therefore, the performance of XYJSON is slightly better than the traditional methods of SQL and MongoDB. Owing to the characteristics of the key–value database, Redis is considered unfriendly to meta-child modifications. Therefore, the traditional and XYJSON models consumed a relatively large amount of time, and the time loss of the XYJSON model was 9.31% greater than that of traditional methods. Traditional Redis implements the modification operation by overwriting the old data with the new data after querying the data to be modified. Based on this process, XYJSON also needs to perform the data model transformation of query data and new data coverage operations; hence, its performance is lower than that of the traditional methods. The difference in time consumption between the two models in the deletion experiment was not significant, i.e., − 5.55%–0.22%. In the experiment of SQL deleting operation, the performance of the XYJSON method is similar to that of the traditional SQL method because the model transformation of the SQL delete statement is relatively simple; thus, the XYJSON model consumes minimal time during data model parsing. In the experiment of MongoDB deleting operation, XYJSON converts the data model to document data, and the performance is almost similar to that of the traditional method.In the experiment of Redis deleting operation, considering Redis is a key-value database, XYJSON can easily implement key-value pair control with its control model; hence, its performance is slightly higher compared to that of the traditional method. The experimental results showed that the XYJSON model

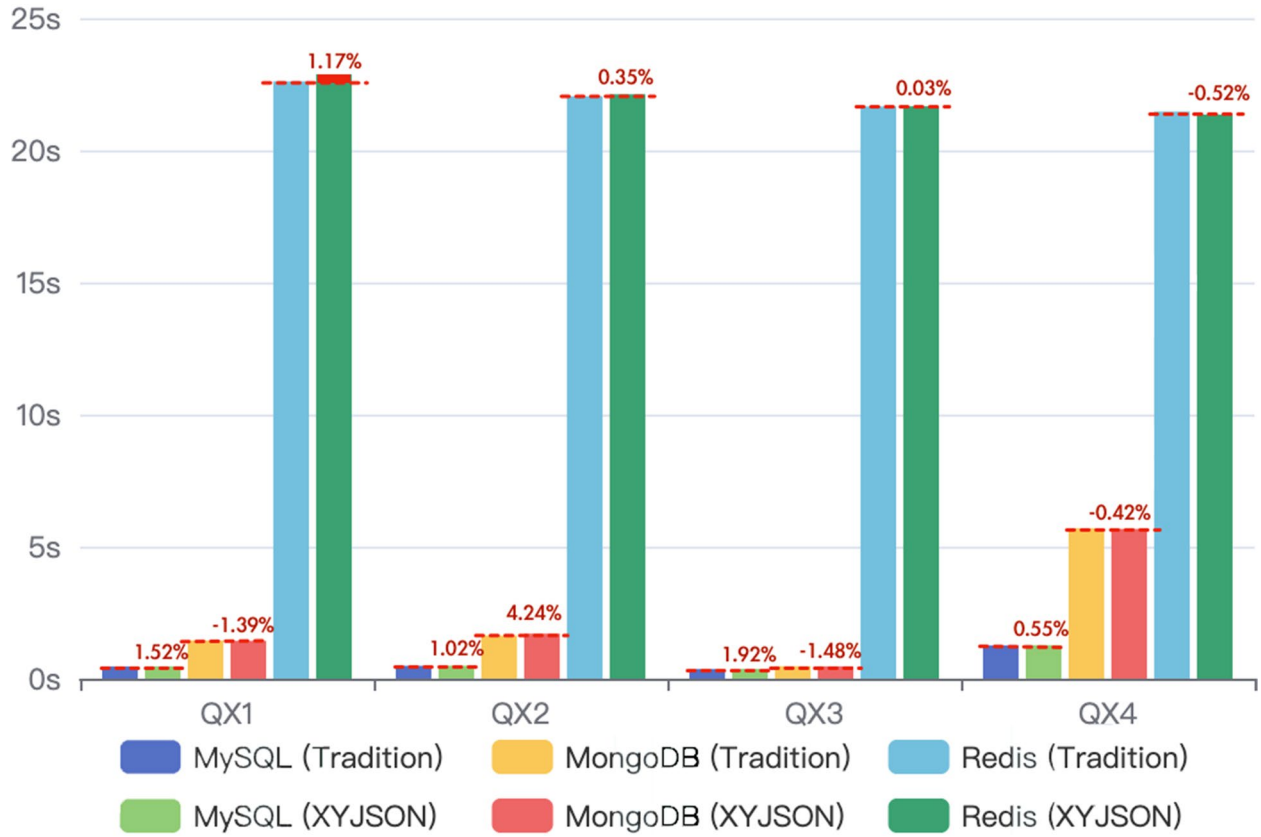

Fig. 8 Comparison results of query conversion experiment

realized the modification and deletion of different types of databases using one command mode, and in the relational data and MongoDB database, the modification performance was slightly better than that of the traditional model, and the gap between other operation performances was small.

Query conversion experiment

In the query operation conversion experiment, the query time comparison of MySQL, MongoDB, and Redis databases was realized using the traditional and XYJSON methods. Consider QX1-4 in Fig. 4 as an example. The experimental results are shown in Fig. 8. It can be observed that there was a small difference in time consumption between the two during query operations, ranging from − 1.48% to 4.24%. In the query operation of MySQL, whether a single table query or a query associated with multiple tables, XYJSON adds the data model transformation based on traditional methods. The performance degradation caused by the transformation is negligible, ranging from 0.55% to 1.92%. In the query operation of MongoDB, excluding the QX2 example, XYJSON has a slightly higher performance than the traditional method. Due to the document data type of MongoDB, the data structure of MongoDB and XYJSON can interact directly with the native JSON; hence, the XYJSON data model can efficiently implement the conversion with MongoDB data. Coupled with the class structure cache pool method, XYJSON has a slightly higher performance than the traditional method. The QX2 example involves the associated query of three tables, requiring XYJSON to perform the model transformation on three tables during parsing; therefore, its performance is lower than that of the traditional method. In the query operation of Redis, because Redis itself has poor support for the conditional query operation, both the traditional Redis method and XYJSON method consume more query time. XYJSON needs to parse and convert date query conditions in the QX1 example; therefore, its performance is slightly lower than that of traditional methods. In the QX2 example, aimilar to MongoDB, XYJSON needs to parse the query conditions of three

Table 1 Questionnaire Details

Id	Question	Answer
Q1	How many years have you worked in the company?	/
Q2	What type of program do you usually use?	Cloud Development; Microservice Development; Single Application Service; Android Development; IOS Development; Web program development
Q3	What type of database do you usually use?(Multiple choice)	MySQL; Oracle; Redis; MongoDB; SQL Server; SQLite; PostgreSQL; Hbase; Others
Q4	Please fill in other types of databases	/
Q5	What are the difficulties in hybrid database development?(Multiple choice)	Complex Development; High learning cost; Various database types; Various data types; Code Redundancy; High maintenance cost; Others
Q6	Please fill in other difficulties	/
Q7	Can XYJSON help you achieve unified control over different types of databases?	YES; NO
Q8	Can't help, please fill in the reason	/
Q9	What databases do you use when using XYJSON?	MySQL; Oracle; Redis; MongoDB; SQL Server; SQLite; PostgreSQL; Hbase; Others
Q10	Please fill in other types of databases	/
Q11	What are the advantages of XYJSON?(Single choice)	Low learning cost; High stability; High performance; High compatibility; High security; High maintainability; User friendliness; Others
Q12	Other advantages please fill in	/
Q13	What are the disadvantages of XYJSON?	/
Q14	Can query function of XYJSON help you realize data processing?	YES; NO
Q15	Can't help, please fill in the reason	/
Q16	In the XYJSON query function, which function do you use most?	QUERYSQL; AND; OR; IN; ORDER; LIKE; BETWEEN; TOP; NOTNULL; NULL; EQ; NEQ; TODATE; JOIN; INNERJOIN; LEFTJOIN; RIGHTJOIN; FULLJOIN
Q17	Do you use XYJSON to migrate data from different types of databases?	YES; NO
Q18	Which databases are used for database migration (please fill in for those who have performed data migration)?	MySQL; Oracle; Redis; MongoDB; SQL Server; SQLite; PostgreSQL; Hbase; Others
Q19	Please fill in other databases	/
Q20	What is the approximate amount of database migration data?	[100 M-500G]
Q21	What indicators do you care most about for cloud hybrid database control (single choice)?	High performance; User friendliness; Low learning cost; High security; High maintainability; High stability; High compatibility
Q22	What do you think of XYJSON?	Excellent; Good; Average; Fair; Poor
Q23	What is your suggestion for XYJSON?	/

tables and then perform data model transformation; therefore, its performance is slightly lower than that of the traditional method. In the QX3 example, XYJSON parses non-null functions and equal functions to control Redis queries smartly; therefore, its performance is slightly lower than that of traditional methods. In the QX4 example, there are subquery statements in the query statements. Redis adopts the traditional method of controlling program condition filtering, while XYJSON implements the nested operation of the query by parsing the query function directly; therefore, the performance is slightly higher than that of the traditional method.

Developer research experiment

For XYJSON, a developer survey questionnaire was launched. The online questionnaires were provided to 246 front-line developers from software companies anonymously, as well as the XYJSON development kit in the form of compiled components. After being explained how to use XYJSON, the developers were asked to use or test XYJSON model in real projects and evaluate the model in the form of questionnaires. In the process of the project development, developers can use XYJSON components to control different types of databases uniformly. Finally, 223 valid questionnaires were received.

There were 23 questions in the questionnaire, including the developer's work experience, the indicators concerned with the development of hybrid cloud databases, and the advantages of XYJSON, as shown in Table 1.

Q1 and Q2 are designed to investigate the length of service of developers and their technical fields to ensure the breadth of survey, so as to achieve the statistics of the use experiences of XYJSON model from developers with different length of service and different technical

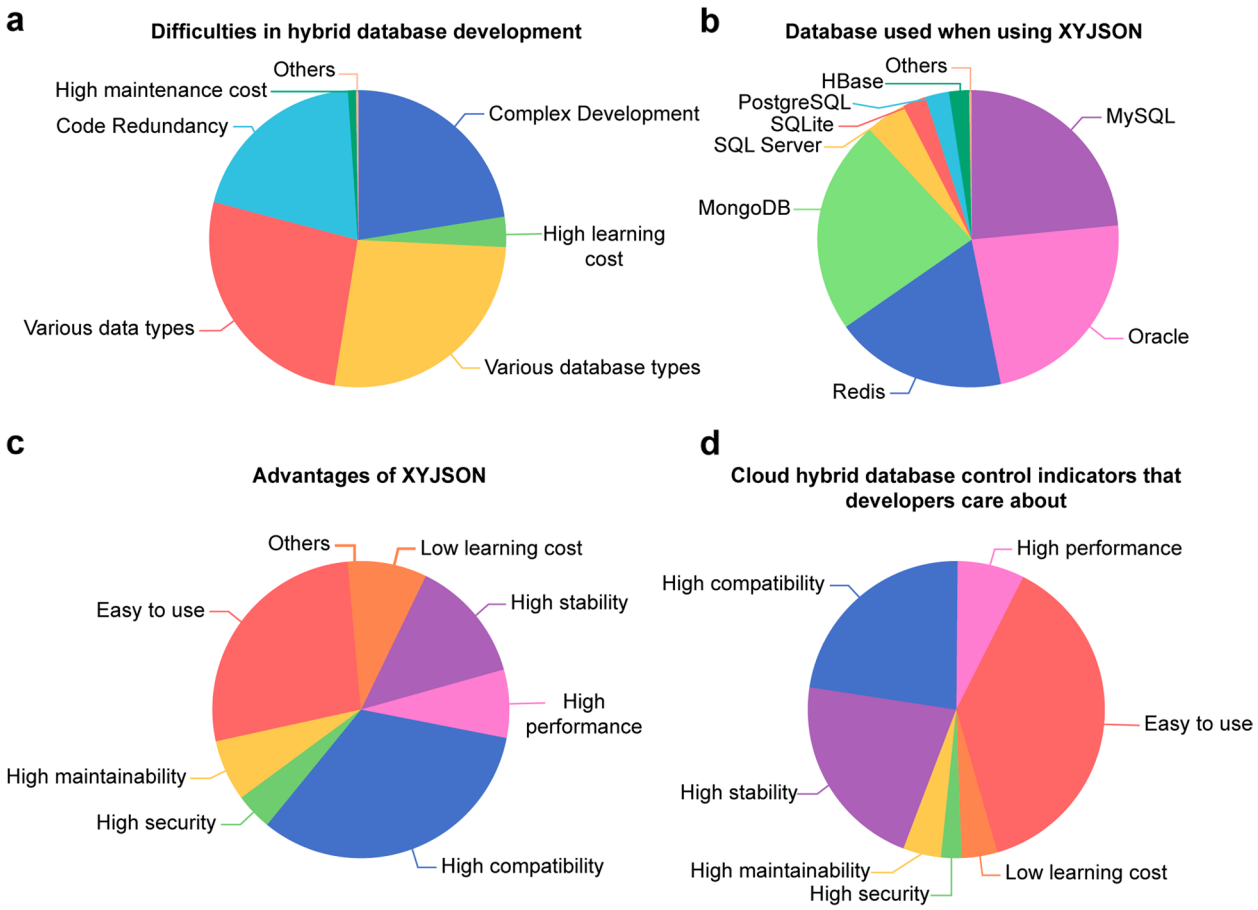

Fig. 9 Developer survey statistics (**a**) Difficulties in hybrid database development (**b**) Database used with XYJSON (**c**) Advantages of XYJSON (**d**) Cloud hybrid database control indicators considered by developers

fields in the largest range. The results show that most of the developers participating in this survey have worked for 4 years and most worked in the field of cloud development and micro service development. Q3-Q6 investigate what types of databases developers often use during project development and what difficulties they encounter in using hybrid databases. The results show that MySQL and Oracle are mostly used in relational databases, while MongoDB and Redis are mostly used in non-relational databases. From Q7 to Q17, the specific user experiences of XYJSON model as it is used are mainly investigated, including whether XYJSON can help developers achieve unified database operations, which databases are used when using XYJSON model, what are the advantages and disadvantages of xyjson, the user experiences of XYJSON query functions, and so on. Q18-Q20 specifically investigate the user experiences of XYJSON model by developers in the process of big data migration. Q21-Q23 mainly aims at the subjective feelings and suggestions of XYJSON model. For the usability and compatibility of developers discussed in this survey, the statistical data of four questions are listed in detail, as shown in Fig. 9.

According to the developer survey statistics in Fig. 9a, developers consider four aspects to be the most difficult in the development of a hybrid cloud database: varying database types, varying data types, complex development, and code redundancy. The authors attribute the four difficulties put forward by the developers to one issue. During the development of a hybrid database, different types of databases should be considered. The characteristics of different databases and the inconsistency between data field types in each database should be distinguished separately in the program.

Developers often have to write more code to control different databases, resulting in more complex and redundant code.

As observed in Fig. 9b, MySQL, Oracle, MongoDB, and Redis are the most commonly used databases among developers when using XYJSON. The survey highlighted that developers typically used more than one database,

which is consistent with the results. It shows that developers use mostly MySQL and Oracle for relational databases and MongoDB and Redis for NoSQL databases. Therefore, the databases used in the experiments are in line with the current usage habits of developers.

After using XYJSON, developers answered the question regarding the advantages of XYJSON, as shown in Fig. 9c. It should be noted that out of 223 developer questionnaires, it was revealed that 98 developers used XYJSON to migrate database data, and the amount of migrated data were mostly between 100 MB and 500 GB. They believed that the advantages of XYJSON were evident in three aspects: high compatibility, user friendliness, and high stability. XYJSON can help developers reduce the time required for development, improve the user friendliness, and reduce the code redundancy and complexity caused by controlling different types of databases. The results shown in Fig. 9c are also consistent with the statistical results in Fig. 9d. Figure 9d presents the indicators that developers care about when controlling a hybrid cloud database, which mostly includes user friendliness, high compatibility, and high stability. Surprisingly, the performance indicators are not the most important indicators for developers in hybrid cloud database control. Based on communications with some developers, it was found that owing to the continuous improvement of cloud development, cloud distribution, cloud server hardware level, and the excellent performance of various databases in their fields of expertise, in terms of hybrid cloud database control, performance factors are not the key indicators that developers pay most attention to. On the premise of sacrificing a small amount of performance, XYJSON has improved its high user friendliness and compatibility, which developers think is worthwhile; therefore, all the developers finally rated XYJSON as "excellent."

Conclusion

A data conversion model, named XYJSON, is proposed to solve the problem of increasing development workload and the difficulty of different types of control methods for their corresponding types of databases under the cloud hybrid storage. The model can be adapted to relational and NoSQL data based on native JSON. It can support the conversion between different data types in different types of databases, and realize the persistence of application-level objects by controlling beans through the XYJson control model, to realize the unified control of relational database and NoSQL database.

Taking the hybrid database controlled by traditional and XYJSON methods as an example, it was experimentally demonstrated that XYJSON slightly sacrifices performance in exchange for improvements in compatibility in four different operations. Simultaneously, the results of the generated research report show that XYJSON has the advantages of high user friendliness and high compatibility. All the developers rated XYJSON as "excellent." The results showed that the establishment of a general conversion model between relational and NoSQL data can effectively help developers realize the data interaction between different types of databases.

In addition, in order to further optimize the XYJSON model, we will also increase the function compatibility and increase the survey data and scope according to the suggestions of the developers, and have obtained more objective evaluation results.

Abbreviations

SQL: Structured Query Language; JSON: JavaScript Object Notation; NoSQL: Non-relational; DML: Data manipulation language; RESTfu: Representational state transfer.

Acknowledgements

We thank Editage for their English language editing.

Authors' contributions

Lei ZHANG: Conceptualization, Methodology, Software, Writing—Original Draft; Ke PANG: Formal analysis, Investigation, Data Curation, Writing—Original Draft; Jiangtao XU: Conceptualization, Writing—Review & Editing, Supervision; Bingxin NIU: Methodology, Visualization, Data Curation, Validation. The author(s) read and approved the final manuscript.

Funding

This work was supported by the Tianjin Science and Technology Plan project [grant number 20YDTPJC00890]. The funding sources had no role in the study design; in the collection, analysis, and interpretation of data; in the writing of the report; or in the decision to submit the article for publication.

Availability of data and materials

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests

The authors have no competing interests to declare.

Author details

[1]School of Microelectronics, Tianjin University, Tianjin 300072, China. [2]School of Software and Communication, Tianjin Sino-German University of Applied Sciences, Tianjin 300350, China. [3]School of Software and Communication, Tianjin Sino-German University of Applied Sciences, Tianjin 300350, China. [4]School of Microelectronics, Tianjin University, Tianjin 300072, China. [5]School of Artificial Intelligence, Hebei University of Technology, Hebei 300401, China.

Received: 10 May 2022 Accepted: 26 July 2022
Published online: 03 August 2022

References

1. Liu ZH, Hammerschmidt B, McMahon D, Chang H, Lu Y, Spiegel J, Sosa AC, Suresh S, Arora G, Arora V (2020) Native JSON datatype support: maturing SQL and NoSQL convergence in Oracle database. In: Proceedings of the VLDB Endowment 13(12):3059–3071. https://doi.org/10.14778/3415478.3415534

2. Bahta R, Atay M (2019) Translating JSON data into relational data using schema-oblivious approaches. In: Proceedings of the 2019 ACM Southeast Conference, New York
3. Oracle JSON support. (2021). https://docs.oracle.com/en/database/other-databases/nosql-database/20.3/admin/index.html. Accessed on 15 Apr 2021
4. Microsoft SQL Server JSON support. (2021). https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver15. Accessed on 10 Jun 2021
5. MySQL JSON DataType. https://dev.mysql.com/doc/refman/8.0/en/json.html. Accessed on 10 Apr 2021
6. PostgreSQL JSON Types. (2021). https://www.postgresql.org/docs/13/datatype-json.html. Accessed on 25 May 2021
7. Teradata Database JSON Data Type. (2021). https://docs.teradata.com/r/C8cVEJ54PO4~YXWXeXGvsA/root. Accessed on 20 May 2021
8. Petković D (2017) JSON integration in relational database systems. Int J Comput Appl 168:14–19. https://doi.org/10.5120/ijca2017914389
9. Liu ZH, Hammerschmidt B, McMahon D (2014) JSON data management: Supporting schema-less development in RDBMS. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Utah. https://doi.org/10.1145/2588555.2595628
10. Chasseur C, Li Y, Patel JM (2013) Enabling JSON document stores in relational systems, WebDB, New York. pp 1–6
11. Petković D (2020) Non-native techniques for storing JSON documents into relational tables. In: Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services, New York. https://doi.org/10.1145/3428757.3429103
12. Piech M, Marcjan R (2018) A new approach to storing dynamic data in relational databases using JSON. Comput Sci. https://doi.org/10.7494/csci.2018.19.1.2505
13. json-schema.org: The home of json schema. (2021). http://json-schema.org/. Accessed on 20 May 2021
14. Bray T (2014) The JavaScript Object Notation (JSON), Data Interchange Format. p 1
15. ECMA. The JSON Data Interchange Format. (2021). http://www.ecma-international.org/publications/standards/Ecma-404.htm. Accessed on 10 Aug 2021
16. Pezoa F, Reutter JL, Suarez F, Ugarte M, Vrgoč D (2016) Foundations of JSON schema. In: Proceedings of the 25th International Conference on World Wide Web, Republic and Canton of Geneva, CHE. https://doi.org/10.1145/2872427.2883029
17. Bourhis P, Reutter JL, Vrgoč D, Json VD (2017) Data model and query languages. Inf Syst 89:101478. https://doi.org/10.1145/3034786.3056120
18. MongoDB Inc, The MongoDB4.4 manual. (2021). https://docs.mongodb.org/manual/. Accessed on 20 May 2021
19. Lanthaler M, Gütl C (2012) On using JSON-LD to create evolvable RESTful services. In: Proceedings of the Third International Workshop on RESTful Design, New York. https://doi.org/10.1145/2307819.2307827

Publisher's Note