An Efficiency of Native JSON+ for Data Integration

Nurul Anis Alia, Mohd Kamir Yusof, Suhailan Safei

Abstract— **To solve the problem of universal data schema in data integration, a native JSON+ method based on JSON is proposed to provide a universal data schema to improve the performance of data insertion response time and query processing response time. A native JSON+ provides a universal schema to allow data exchange or data sharing among applications. First, data is converted into JSON format. Second, elements and attributes are stored into relational database management systems (RDBMS). Third, each element and attribute are mapping to the data in JSON format. Based on the experiments have been done by using three different datasets (SigmodRecord, NASA and DBLP), the results indicate a native JSON+ performance in term of data insertion response time and query processing response time are better compared to NXD.**

Index Terms— **Data Integration, Data schema, Native XML, Native JSON+**

I. Introduction

SINCE decades ago, data integration has been learned widely especially, regarding its diverse approaches. Data integration is an important process in many different contexts, including the commercial and scientific spheres [1]. Data integration is the process of combining information from various sources into a single unified view [1]. A unified view is needed to ensure different applications can extract the data and dump it into their applications. Two processes are involved in data integration are data extraction and data conversion. Recent researchers have proved that data extraction and conversion are the essential role for data integration. Data extraction is the process of taking data out of data sources for later processing or data storage. It combines many data sources and extract them into various data formats [2]. Meanwhile, data conversion is the process of converting data in universal format [3]. Both processes are allowed to extract the data from different data sources and provided a single unified view in universal format. A few approaches have been used in data integration such as XML, and JSON. Extensible Markup Language (XML) is one of the data formats that is supported by data reading.

Manuscript received May 26, 2023; revised January 11, 2024.

Nurul Anis Alia is a postgraduate student of Universiti Sultan Zainal Abidin, Faculty of Informatics and Computing, 22200 Besut, Terengganu, Malaysia. (e-mail: nurulanis@gmail.com).

Mohd Kamir Yusof is a lecturer of Universiti Sultan Zainal Abidin, Faculty of Informatics and Computing, 22200 Besut, Terengganu, Malaysia (corresponding author to provide phone: +6013-944-6091; fax: +609-699-3333; e-mail: kamir2020@gmail.com).

Suhailan Safei is a lecturer of Universiti Sultan Zainal Abidin, Faculty of Informatics and Computing, 22200 Besut, Terengganu, Malaysia (e-mail: suhailan@unisza.edu.my).

It loads the collection of data from the disk into memory to be processed. A few approaches have been used in data integration such as XML, and JSON. Extensible Markup Language (XML) is one of the data formats that is supported by data reading. It loads the collection of data from the disk into memory to be processed. All of them are accommodated through a simple attribute-value pairs model [4]. A few XML approach has been implemented in data integration. First, XML document data is stored and searched by Hybrid XML to overcome the limitation of earlier research on the data model for XML document data [8]. Second, XML format data is supported by XML Enabled by allowing entries for a wide range of XML scheme languages as attribute values for specifying the attribute syntax [9]. Third, XML document data logical model is defined by Native XML by storing and retrieving a large amount of data efficiently [10]. However, performance of this approaches in term of data insertion response time and query processing response time are still considerably inefficient. JSON data model is proven efficient especially for data exchange via internet. In this research, a new approach a Native JSON+ data model will be proposed in data integration.

The first section of this paper discusses the introduction about data integration. This is followed by a review of the data integration models in second section. Third section will be explained about proposed JSON+ method in data integration. The last section of the paper discusses the performance of native JSON in data integration.

II. Data Integration Model

In this section, three data integration model has been reviewed. There are Hybrid-XML, XML-Enables, and Native XML.

A. Hybrid-XML

Hybrid XML stores and searches XML document data to overcome the limitation of earlier research on the data model for XML document data. The data and structure views of XML document is also represented for current database systems which are relational and object-oriented data model. As an instance, XML data sources and user requirements build XML data warehouses [8]. A path-based and node-based mapping approach where each inner and leaf node's path is tracked and recorded as a path expression in the path table. The findings of text nodes stored in the value table will

be simple and at once, the maintenance of the hierarchical structure of XML document [14].

B. XML-Enabled

XML-enabled are extended by the key relational DBMSs typically through the SQL/XML standard or a variant thereof. As a directory, XML-enabled allows entries to have XML formatted data as attribute values since a wide range of XML schema languages for specifying the attribute syntax is supported by the directory. The JSON format was often added to XML-enabled databases. Besides, other types of DBMSs that combined their raw data format with other data formats also been discovered [9].

C. Native-XML (NXD)

Native XML is specialized for XML data and defines a logical model for XML document. Furthermore, Native XML stores and retrieves a large amount of data in efficient way. As mentioned before, Native XML originally created to process XML data using standardized querying languages. Nowadays, JSON have mostly taken the place of previous technology [10]. There is also a small number of works on data manipulation in Native XML which are multi-temporal and multi-version but the three fundamental operations-insertion, modification, and deletion are used to manage multi-temporal XML data. Therefore, Native XML still act as a database for a wide variety of contemporary applications across many different fields for XML data storage until today [13].

D. JSON

JSON is a universal standard format for the representation and exchanging the information [15]. Nowadays, JSON is widely by major Web APIs such as Twitter, Facebook, and many Google services for data exchange format [16]. In term of storage and query retrieval, JSON is more powerful compared to XML [17][18]. When XML has been said to contribute with a lot of unnecessary baggage, JSON document can contain the same information with much lightweight and easier to read. JSON is commonly used when exchanging or storing structured. data. On the other hand, XML and JavaScript Object Notation (JSON) are two different data serialization formats used in web applications [19]. Two approaches which typically has been used: Asynchronous JavaScript and XML (AJAX). XML is a platform independent language for representing data and has been used in the development of web service application. However, the performance of web services has shown a significant decrease when using XML data because of the low efficiency of reading and parsing XML data during execution of services. Based on the measurement of metrics such as the number of objects sent, total time to send the number of objects, average time per object transmission, use CPU utilization, system CPU utilization, and memory utilization, it has been proved that it is significantly faster and has higher parsing efficient than XML. Based on these approaches, JSON model has a potential for improvement by integrate with NXD model. In this paper, Native JSON (N-JSON) are proposed to get better performance in term of data insertion response time and query processing response time compared to NXD.

III. METHODOLOGY

This section explains the development and formulation model for data extraction in data integration. Four (4) processes are involved in this formulation model which are map to data sources, extract the data and dump the data. Fig.1 show the process of data extraction from selected data sources. Then, a specified data model will be generated for data retrieval purposes.

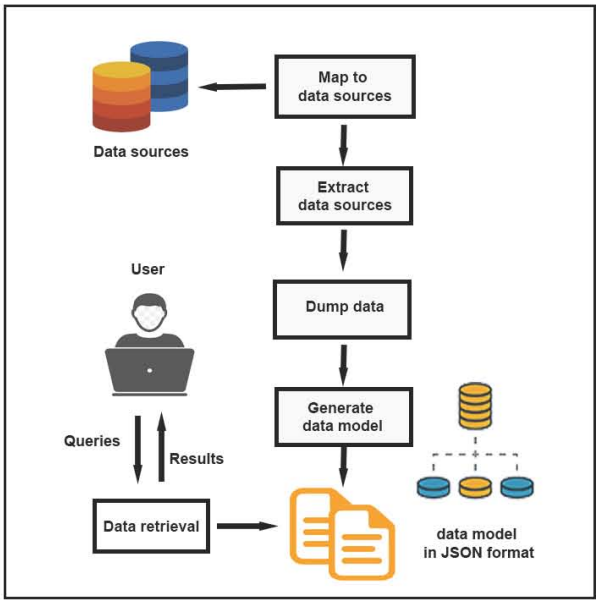

Fig.1. Data extraction in data integration

Map to data sources

In this process, data sources are selected and mapped for data extraction process.

Definition 1:

Assume S represents a dataset of data sources. S = {ds_1, ds_2, ds_3, ds_n}, where ds_1 until ds_n is an element of S.

Let S consists of data sources ds_1, ds_2, and ds_3. Each data source consists of components which are attributes and elements. These components will be used in extraction process. Figure

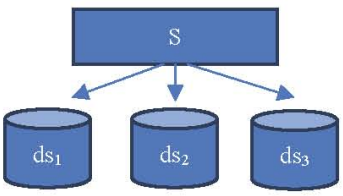

Fig.2. Mapping of data sources

Data extraction

In this process, data from different sources will be extracted which are consists of attributes and data for each object.

Definition 2:
Assume M represents a dataset of object. M = {e_1, e_2, e_3, e_n}, where e_1 until e_n is an element of M. Then, each element of e consists of attributes and data. Let e_n = {a_n, d_n} where a_n is attribute and d_n is data of attribute a_n

Dump data
In this process, each object consist of attributes and tuples will be dump into structured database.

Definition 3:
Assume S represent a dataset of object consists of attributes and data. S = {(e1 → a_n, d_n), (e2 → a_n, d_n), (e_n → a_n, d_n), … }, where e_1 until e_n is object which are consists of attributes (a_n) and data (d_n).

Generate data model
In this process, a specified data model will be generated and match with data in structured database.

Definition 4:
Assume DM represent a dataset of data model. DM = {m_1, m_2, m_3, m_n} where m_1 until m_n is data model.

Data retrieval
In this process, a few queries statement will be executed to evaluate the performance in term of data insertion response time and query processing response time.

Definition 5:
Assume Q represent dataset of query statement. Q = {q_1, q_2, q_3, q_n} where q_1 until q_n is list of query statement.

IV. IMPLEMENTATION

This section presented experimental results for data integration model. This section shows the experimentation for data integration using Native JSON, Native XML, Hybrid XML and XML-Enabled. The experimental results are analysed and comparisons are done with the baseline benchmarked algorithms. The benchmark datasets are employed together in this experiment.

A. Experiment set-up: Database platform and programming language

This study performed all its experiments on a cloud environment, 1 vCPU with 2GB RAM using a Ubuntu 20.04 (LTS) x64. The software specification for algorithm development is deployed using open-source software, including MySQL version 8.0.31, MySQL community server (GPL) for our database server, Apache/2.4.41 for our web server, PHP as a programming language and phpMyAdmin with administration of MySQL over the Web. The phpMyAdmin (phpMyAdmin: Bringing MySQL to the web) is a free software tool, written in PHP, that supports a wide range of operations on MySQL, MariaDB and Drizzle. The user interface helps one to perform frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc.), with the ability to directly execute any SQL statement. Table I exhibits the software configuration.

TABLE I
DATABASE CONFIGURATION

Software	Configuration
Database	MySQL
Web Server	Apache/2.4.41 OpenSSL/1.0.1i PHP/7.4.3 Database client version PHP extension
MySQL Administration	phpMyAdmin Version information: 5.2.0, latest stable version: 5.2.0
Programming Language	PHP

B. Database Source and Characteristics

SigmodRecord, NASA and DBLP datasets will be used in experiments. These benchmark datasets have been used for NXD approach in experiments purposes in term of data insertion time and query processing response time (Mohsin Marjani et al., 2018). These datasets are download and saved in a *.xml file format. Size of these data are 467KB, 23.9MB and 127.7MB. These datasets provide bibliographical information about computer sciences journals, books, thesis, URL, and proceedings. The overall characteristics of benchmark datasets is tabulated in Table II. The size in MB represents physical file size, the length represents attributes or labelled as attributes name and the records defines the total number or records.

TABLE II
DATABASE CHARACTERISTICS

File name	Size (MB)	Length (Attributes)	Record
SigmodRecord	0.479	4-7	3280
NASA	23.9	4-8	10000
DBLP	127.7	6-10	50000

C. Data Pre-processing

Data pre-processing in this context refers to the conversion of data format. The downloaded data benchmark dataset in .xml file format is unzipped and opened in Notepad. Three process involves in pre-processing: map to data source, and data extraction.

V. FINDINGS AND DISCUSSION

This study compared the performance in term of data insertion response time and query processing response time between JSON+ and NXD. Three different datasets will be used are SigmodRecord, NASA, and DBLP. Data insertion response time will be measured in milliseconds (ms) based on data extraction process from XML file and dump it into DBMS and generate a new XML (NXD) or JSON (JSON+). Meanwhile, a query processing response time is also measured in milliseconds (ms) and will be evaluated based on queries with different complexity in Table III, Table IV, and Table V.

TABLE III
QUERIES WITH DIFFERENT COMPLEXITY (SIGMODRECORD)

Query	Query description
I	Retrieve and list all the information where the tag is "number" which is a child node of tag "issue"
II	Retrieve and list all the information where tag is "article" on condition that the value of one its child node – tag "author" is "Amihai Motro"
III	Retrieve and list all the information for all tags with name "title" where the attribute article Code is greater than "152010" and less than or equal to "152010"

TABLE IV
QUERIES WITH DIFFERENT COMPLEXITY (NASA)

Query	Query description
I	Retrieve and list all the information where the tag is "title" which is a child node of tag "dataset"
II	Retrieve and list all the information where the tag is "other" on condition that the value of one of its child nodes - tag "lastName" is 'Jackson'
III	Retrieve and list all the information for all tags with name "other" where the value of its child node-tag "year" is greater than '1970' and less than or equal to '1990'

TABLE V
QUERIES WITH DIFFERENT COMPLEXITY (DBLP)

Query	Query description
I	Retrieve and list all the information where the tag is "title" which is a child node of tag "www"
II	Retrieve and list all the information where the tag is "masterthesis" on condition that the value of one of its child nodes – tag "year" is "2006"
III	Retrieve and list all the information for all tags with name "www" where the attribute key is "www/org/tpc" or the attribute mdate is "2004-12-02"

1) Comparison of NXD and JSON+ on data insertion response time

In this section, data are extracted from different data sources and converted it into three data models: NXD and JSON+. In this experiment, response time for data insertion have been calculated 4 times. The last column shows the average response time for each data model. Based on Table VI, the result of data insertion response time JSON+ is reduce to 9.11% compared to NXD using SigmodRecord dataset. Meanwhile, based Table VII, the result of data insertion *response* time JSON+ is reduce to 9.11% compared to NXD using NASA dataset. Then, based Table VIII, the result of data insertion response time JSON+ is reduce to 17% compared to NXD using DBLP dataset. The number of percentages for each data model can be calculated based on the following formula:

$$\frac{Avg.of\ insertion\ time\ NXD - Avg.of\ insertion\ time\ NJSON}{Avg.of\ NXD} \times 100$$

Meanwhile, Fig. 10, Fig. 11 and Fig. 12 represent response time for data insertion in line graph based on result in Table 6, Table 7 and 8. JSON+ was produced better result compared to NXD because of data schema and raw data has been split and mapping according to definition 3 and definition 4. Through this technique, process of data insertion will be divided into two ways compared to NXD.

TABLE VI
RESPONSE TIME FOR DATA INSERTION OF SIGMODRECORD

Data model	1th	2th	3th	4th	Avg.
NXD	652	635	643	661	648
JSON+	595	587	583	590	589

TABLE VII
RESPONSE TIME FOR DATA INSERTION OF NASA

Data model	1th	2th	3th	4th	Avg.
NXD	2608	2540	2572	2644	2591
JSON+	2380	2348	2332	2360	2355

TABLE VIII
RESPONSE TIME FOR DATA INSERTION OF DBLP

Data model	1th	2th	3th	4th	Avg.
NXD	25789	25750	25712	25770	25755
JSON+	21256	21278	21285	21219	21260

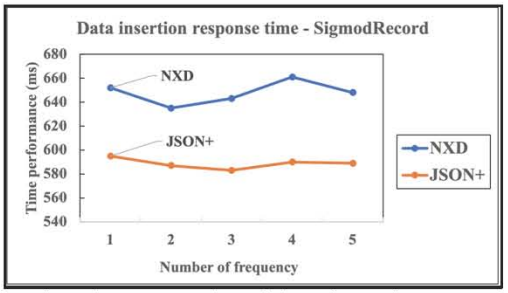

Fig. 10. Data insertion response time of SigmodRecord

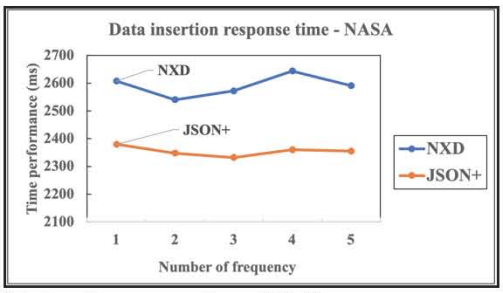

Fig. 11. Data insertion response time of NASA

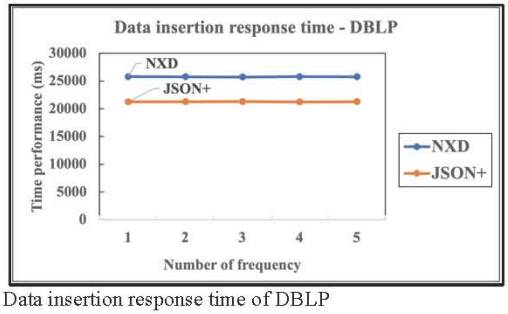

Fig. 12. Data insertion response time of DBLP

2) Comparison of NXD and JSON+ on query processing response time

In this section, we evaluated the performance of NXD and JSON+ based on query processing response time. Three (3) different queries were executed based on specified statement

in Table 3, Table 4 and Table 5, and query processing response time were executed 4 times.

Tables IX – XI show the query processing response time for SigmodRecord dataset. The result shows JSON+ in three different queries complexity are reduce by 8.87%, 9.36%, and 4.46% respectively compared to NXD. In this case, JSON+ can reduce query processing time during searching and retrieving process. The number of percentages for each data model can be calculated based on the following formula:

$$\frac{Avg.of\ response\ time\ (NXD) - Avg.response\ time\ (NJSON)}{Avg.response\ time\ (NXD)} \text{ x } 100$$

Meanwhile, Fig. 13 - 15 represent the line graph for query processing response time for SigmodRecord dataset in milliseconds (ms).

TABLE IX
QUERY PROCESSING RESPONSE TIME FOR SIGMODRECORD DATASET (I)

Data model	1th	2th	3th	4th	Avg.
NXD	375	380	368	365	372
JSON+	335	342	340	339	339

TABLE X
QUERY PROCESSING RESPONSE TIME FOR SIGMODRECORD DATASET (II)

Data model	1th	2th	3th	4th	Avg.
NXD	430	440	420	430	430
JSON+	390	398	383	388	390

TABLE XI
QUERY PROCESSING RESPONSE TIME FOR SIGMODRECORD DATASET (III)

Data model	1th	2th	3th	4th	Avg.
NXD	130	125	128	133	129
JSON+	121	120	125	119	123

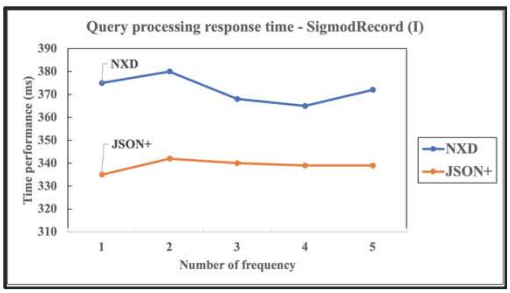

Fig. 13. Query processing response time of SigmodRecord – Query I

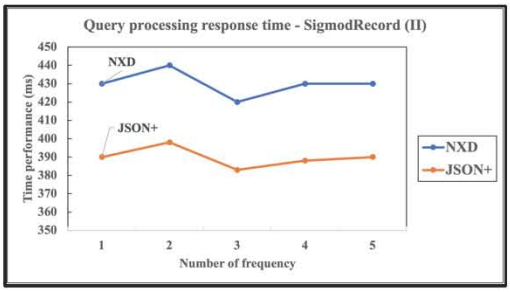

Fig. 14. Query processing response time of SigmodRecord – Query II

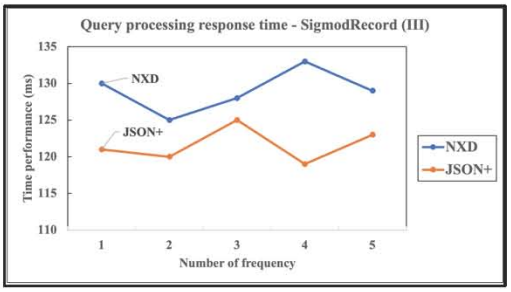

Fig. 15. Query processing response time of SigmodRecord – Query III

Tables XII –XIV show the query processing response time for NASA dataset. The result shows JSON+ in three different queries complexity is reduced by 9.07%, -24.26%, and 3.12% respectively compared to NXD. In this case, JSON+ can reduce query processing time during searching and retrieving process. Meanwhile, Fig 16 - 18 represent the line graph for query processing response time for NASA dataset in milliseconds (ms).

TABLE XII
QUERY PROCESSING RESPONSE TIME FOR NASA DATASET (I)

Data model	1th	2th	3th	4th	Avg.
NXD	1507	1524	1472	1462	1491
JSON+	1340	1368	1360	1356	1356

TABLE XIII
QUERY PROCESSING RESPONSE TIME FOR NASA DATASET (II)

Data model	1th	2th	3th	4th	Avg.
NXD	1722	1768	1676	1754	1730
JSON+	1614	1599	1528	1558	1575

TABLE XIV
QUERY PROCESSING RESPONSE TIME FOR NASA DATASET (III)

Data model	1th	2th	3th	4th	Avg.
NXD	515	511	521	538	521
JSON+	494	517	511	498	505

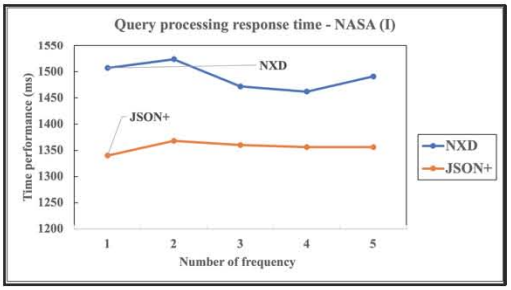

Fig. 16. Query processing response time of NASA Dataset I

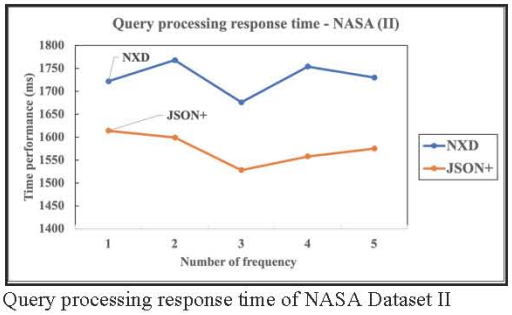

Fig. 17. Query processing response time of NASA Dataset II

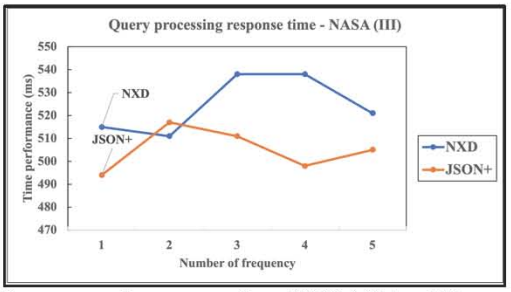

Fig. 18. Query processing response time of NASA Dataset III

Tables XV –XVII show the query processing response time for DBLP dataset. The result shows JSON+ in three different queries complexity are reduce by 14.67%, 7.81%, and 5.95% respectively compared to NXD. In this case, JSON+ can reduce query processing time during searching and retrieving process. Meanwhile, Fig. 19 - 21 represent the line graph for query processing response time for DBLP dataset in milliseconds (ms).

TABLE XV
QUERY PROCESSING RESPONSE TIME FOR DBLP DATASET (I)

Data model	1th	2th	3th	4th	Avg.
NXD	1456	1420	1440	1450	1442
JSON+	1230	1229	1223	1238	1230

TABLE XVI
QUERY PROCESSING RESPONSE TIME FOR DBLP DATASET (II)

Data model	1th	2th	3th	4th	Avg.
NXD	654	620	633	640	637
JSON+	581	588	589	590	587

TABLE XVII
QUERY PROCESSING RESPONSE TIME FOR DBLP DATASET (III)

Data model	1th	2th	3th	4th	Avg.
NXD	1294	1284	1289	1289	1289
JSON+	1201	1224	1215	1209	1212

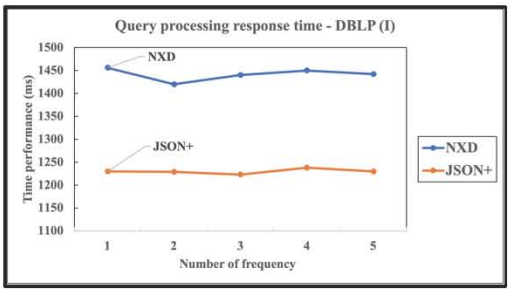

Fig. 19. Query processing response time of DBLP – Query I

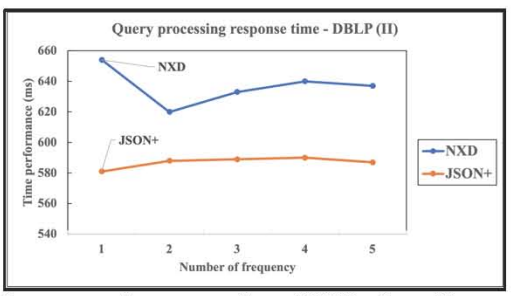

Fig. 20. Query processing response time of DBLP – Query II

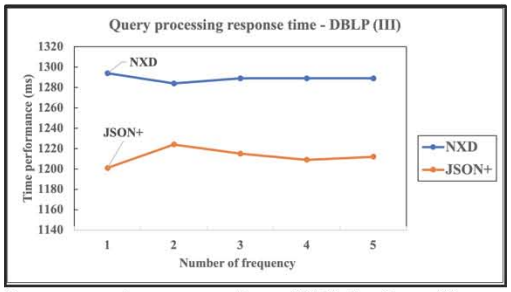

Fig. 21. Query processing response time of DBLP – Query III

Three different datasets have been used for query processing response time. Based on the result in Table IX – XVII, JSON+ is fast compared to NXD by execute different query complexity in Table III – V. JSON+ performance is better than NXD because of removing and eliminating unused characters during mapping data sources in definition 1. In this process, unused characters such as "&","@","~","#" have been remove before transferring it into MySQL database (data schema) and clean data into XML and JSON format. Meanwhile, process of dividing data schema and original sources in XML and JSON format also effect to performance of JSON+, because the query process has been executed in MySQL, then mapping data is executed after the query process. Response time for query processing is become fast because of limited data compared execution from original data which is involve huge amount of data.

VI. CONCLUSION

Native XML and Native JSON+ are two approaches have been reviewed in this research for data integration. Native JSON+ approach is proposed by combination of Native XML and JSON+ elements to produce better performance in data integration. SigmodRecord, NASA and DBLP datasets have been used for experiment purposed. Two experiments have been done, Native JSON+ indicated a better performance in term of data insertion response time and query processing response time compared to Native XML. In this case, Native JSON+ is proven efficient and reliable to become as an alternative for data integration.

REFERENCES

[1] E. L. Sibanda, K. Webb, C. A. Fahey, M. S. K. Dufour, S. I. McCoy, C. Watadzaushe, J. Dirawo, M. Deda, A. Chimwaza, I. Taramusi, A. Mushavi, S. Mukungunugwa, N. Padian, F. M. Cowan, "Use of data from various sources to evaluate and improve the prevention of mother-to-child transmission of HIV programme in Zimbabwe: a data integration exercise," *Journal of the International AIDS Society, 23, e25524.W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993*, pp. 123–135, 2020.

[2] Q. Zhou, J. Xing, L. Hou, R. Xu, K. Zheng, "A novel rate and channel control scheme based on data extraction rate for LoRa networks," *In 2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE.E. H. Miller, pp. 1-6, 2019, doi: 10.1109/WCNC.2019.8885860

[3] I. Mearaj, P. Maheshwari, M. J. Kaur, "Data conversion from traditional relational database to MongoDB using XAMPP and NoSQL," *In 2018 Fifth HCT Information Technology Trends (ITT)*, pp. 94-98, 2018.

[4] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, M. Koubarakis, "The return of jedai: End-to-end entity resolution for structured and semi-structured data," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1950-1953, 2018.

[5] Y. Lin, Z. Jun, M. Hongyan, Z. Zhongwei, F. Zhanfang, "A method of extracting the semi-structured data implication rules," *Procedia computer science*, 131, pp. 706-716, 2018.

[6] D. Durner, V. Leis, T. Neumann, "JSON Tiles: Fast Analytics on Semi-Structured Data," *In Proceedings of the 2021 International Conference on Management of Data*, pp. 445-458, 2021.
[7] A. O. Erkimbaev, V. Y. Zitserman, G. A. Kobzev, A. V. Kosinov, "Standardization of Storage and Retrieval of Semi-structured Thermophysical Data in JSON-documents Associated with the Ontology," *Proceedings of the XIX International Conference Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL'2017), Moscow, Russia*, pp. 287 – 292, October 10–13, 2017.
[8] Z. Ouaret, D. Boukraa, O. Boussaid, R. Chalal, "AuMixDw: Towards an automated hybrid approach for building XML data warehouses," *Data & Knowledge Engineering*, vol. 120, pp. 60-82, 2019
[9] J. Lu, I. Holubova, "Multi-model databases: a new journey to handle the variety of data," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1-38, 2019
[10] C. O. Truica, E. S. Apostol, J. Darmont, T. B. Pedersen, "The forgotten document-oriented database management systems: An overview and benchmark of native xml dodbmses in comparison with json dodbmses," *Big Data Research*, vol. 25, pp. 100-205, 2021.
[11] D. Song, X. Zhu, F. Ma, "High Camouflage Intrusion Detection Method for Structured Database Based on Multi Pattern Matching," *In Journal of Physics: Conference Series*, vol. 2066, no. 1, 2021.
[12] C. Lockard, X. L. Dong, A. Einolghozati, P. Shiralkar, "Ceres: Distantly supervised relation extraction from the semi-structured web,", 2018.
[13] Z. Brahmia, H. Hamrouni, R. Bouaziz, "TempoX: A disciplined approach for data management in multi-temporal and multi-schema-version XML databases," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 1, pp. 1472-1488, 2022.
[14] E. Song, S. C. Haw, "XML-REG: Transforming XML into relational using hybrid-based mapping approach," *IEEE Access*, 8, pp. 177623-177639, 2022.
[15] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study.CAINE 2009," *Computer Science*, 2009.
[16] L. Wang, O. Hassanzadeh, S. Zhang, J. Shi, L. Jiao, J. Zou, C. Wang, "Schema Management for Document Stores," *Proceedings of the VLDB Endowment*, vol. 8, no. 9, pp. 922-933, 2015.
[17] M. Meneghello, "XML (eXtensible markup language) – The new language of data exchange," *Cartography*, vol. 30, no. 1, pp. 51 – 57, 2001.
[18] M. K. Yusof, M. Man, "Efficiency of JSON approach for data extraction and query retrieval," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 4, pp. 201 – 214, 2021.
[19] K. Afsari, C. M. Eastman, D. C. Lacouture, "JavaScript Object Notation (JSON) data serialization of IFC schema in web-based BIM data exchange," *Automation in Construction*, pp. 24 – 51, 2017.

Nurul Anis Alia is a postgraduate student in the Universiti Sultan Zainal Abidin. His major is computer science. His research direction includes data integration and software evolution. Her email address is nurulanis@gmail.com

Mohd Kamir Yusof is a lecturer in the Universiti Sultan Zainal Abidin. Her major is computer science and technology engineering. Her research direction includes data integration, mobile and web application developments. His email address is kamir2020@gmail.com

Suhailan Safei is a lecturer in the Universiti Sultan Zainal Abidin. Her major is computer science and technology engineering. Her research direction includes web application development and socket programming. His email address is suhailan@unisza.edu.my