Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

NATIVE JSON MODEL FOR DATA INTEGRATION IN BUSINESS INTELLIGENT APPLICATIONS

MOHD KAMIR YUSOF[1],MUSTAFA MAN[2], WAN AMIR FAZAMIN WAN HAMZAH[1], SUHAILAN SAFEI[1], ISMAHAFEZI ISMAIL[1]

[1]Faculty of Informatics & Computing, Universiti Sultan Zainal Abidin, 22200 Besut, Terengganu, Malaysia
[2]Faculty of Ocean Engineerinf Technology & Informatics, 21300 K.Terengganu, Terengganu, Malaysia

E-mail: {mohdkamir,wamir,suhailan,isma}@unisza.edu.my, mustafaman@umt.edu.my

ABSTRACT

Collection of data is important component for most business intelligent (BI) application to make a best decision regarding to company. Successful of BI application is consist how much this application can provide useful data for decision maker to achieve the company critical goals such as achieving or exceeding revenue, opportunity to reduce cost, etc. However, 80 percent of BI application has a problem to produce useful data because of data integration issue. This issue is occurred because of increasing amount of data., integration difficulties, and high complexity. Universal data integration model is required to provide useful data for BI application. Native XML (NXD) and JSON are two approaches have been implemented in data integration. These approaches are proven efficient for data integration in term of data insertion response time and query processing response time. Based on NXD and JSON, this paper proposed Native JSON (N-JSON) as a new alternative data integration for BI application. Three experiments such as data insertion response time, query processing response time and CPU usage has been done by using two different datasets; SigmodRecord and DBLP. The results indicate N-JSON produces a better performance compared NXD. As a result, N-JSON can be used as an alternative data integration for BI application in order to provide useful data to decision maker.
Keywords: *Data Integration, Business Intelligent, Native XML, JSON, Native JSON*

1. INTRODUCTION

Business intelligent (BI) is the process of collecting operational data and use these data to make a best decisions regarding to company [1]. The success of a BI is consists how much it can helps the users, managers, technicians, and organizations to achieve the company critical goals such as achieving or exceeding revenue figures, seeking opportunities to reduce cost throughout the organization and maximize profitability by identifying the most profitable customers, most profitable products, services or programs. Another success key in BI is providing useful data coming from multiple heterogeneous sources of data [2]. However, almost 80 percent of any BI projects or applications face with data integration problem in order to provide useful data [3][5]. This problem occured 5384uet o the increasing amount of data, integration difficulties, high complexity and manual effort to manage the data. A suitable data integration model is looking to overcome this problem, thus to produce a successful BI projects or applications. Data integration is combining data residing at different data sources and providing the user with a unified view of these data [4]. Data integration is providing a uniform view of a set of heterogeneous data sources [6]. Current practice in most of organizations for data integration is relational database. In this paper, we find out two alternative approaches have been used in data integration: Native XML (NXD) and JSON. These approaches are proven efficient in term of data insertion response time and query processing response time for data integration especially involved with huge amount of data [7][8]. Based on these approaches, we proposed Native JSON (N-JSON) as another alternative approach for data integration.

2. RELATED WORK

This section discusses about two current data integration approaches: Native XML (NXD) and JSON.

A. Native XML (NXD)

Native XML (NXD) database is the database that stores XML documents directly [9]. NXD promise to play a key role in the near future as management system for XML data [9*]. NXD is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the best described as a database that has an XML document (or its rooted part) as its fundamental unit of (logical) storage and defines a (logical) model for an XML document, as opposed to the data in that document (its contents) [11]. It is represented logical XML document model, and stores and manipulates documents according to that model [12]. The basic characteristic of NXD as below: -

- A logical unit of an NXD is XML document or its footed part, and it corresponds to a row in a relational database
- Includes at least the following components: elements, attributes, textual data (PCDATA), and document order
- Physical model (and type or persistent NXD storage) is unspecified

In an NXD, XML is invisible inside the database. There is unique database for all XML schemas and documents. NXD are especially suitable for storing irregular, deeply hierarchical and recursive data. There are example of "menu" XML document in NXD stored in fixed relational database tables.

Documents	Doc_id	Doc_name
	1	simple.xml
	…	…

Elements	Doc_id	El_id	Parent_id	Name	OrdInParent
	1	1	NULL	menu	1
	1	2	1	food	1
	1	3	2	name	1
	1	4	2	price	2
	…	…	…	…	…

Attributes	Doc_id	Atr_id	Parent_id	Name	Value
	1	1	1	Date	5.10.2006
	…	…	…	…	…

Text	Doc_id	Text_id	Parent_id	Value
	1	1	3	Homemade Bean Soup
	1	2	4	100.00
	…	…	…	…

The example above shows how to build a NXD on top of a relational database, most the NXD are built from scratch, as stand-alone document in management systems. Those systems manage collections of documents, allowing users to query and manipulate those documents as a set, which is like the relational concept of a table.

B. JSON

JSON is lightweight data-interchange format which is easy for humans to read and write, and for machines to parse and generate [13]. JSON is becoming the universal standard format for the representation and exchanging the information [14]. JSON approach is widely adopted as a data exchange format, which is used by major Web APIs such as Twitter, Facebook, and many Google services [15]. JSON approach is more powerful compared XML in term of storage and query retrieval [16][17][19]. JSON is a friendly alternative to XML, and often used as a substitute to it [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 [20][21]. These two approaches which typically are the application in 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.

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. METHODOLOGY

This section described the model of N-JSON. In this model, three steps involved; fetch the data from data sources, integration from data sources and produce output in N-JSON model. Figure. 1 shows the workflow of N-JSON.

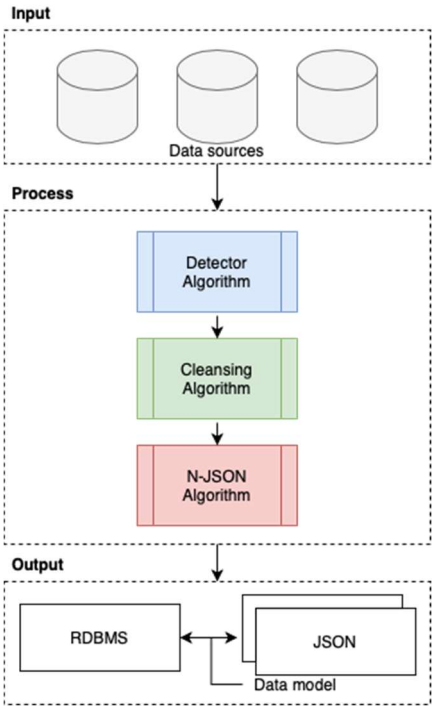

Figure. 1. N-JSON Model Workflow

C. Fetch data from data sources

Assume G represent a dataset of semi-structured data. A ={s1, s2, s3 ...sn} where s1 until sn is an element of A. Figure. 2 shows the datasets representation for semi-structured.

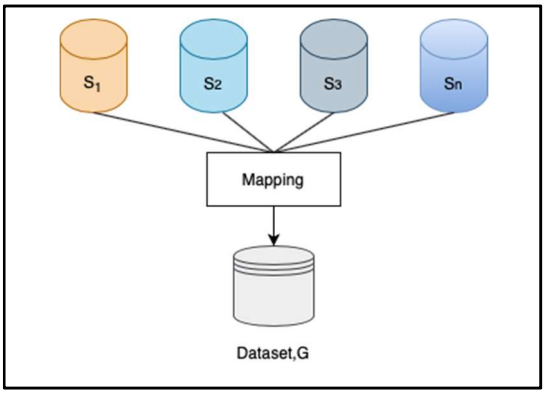

Figure. 2. Mapping To Data Sources

Input : Relevant data source s
Output : Selected data sources, Ds
Begin For each GS relation R in GS relevant to s.
Construct a view sV to be mapped to R.
Query the mapping helper table from the repository.
Insert a new record into mapping helper table.
Enrich mR with a new union element and add sV
Extract the detectors of R from s (Ds)
Unify the naming between the attributes common on the 2 sets of Ds and DR Apply difference operator between Ds and DR Ds_DR_Diff
If Ds_DR_DIFF is empty, then
Rebuild the mapping assertion of R_Detectors to include view over s
Continue
End If
Else then
Insert new record into the GSRelationDetector table with values (R,Ds)
Apply different operator between Ds_DR_DIFF and attributes of R to get R_Diff_Atts
Else then
Rebuild the mapping assertion of R_Detectors to include R_Diff_Atts
Enrich the R_Detectors with view for s
Modify the views over other sources include corresponding attributes for R_Diff_Atts
End if
End if
End loop

Figure. 3. Algorithm For Addition Of A New Data Source

D. Cleansing data

Assume C represent a dataset of clean semi-structured data. The special characters, as an example, are removed from the data. Figure. 4 shows the diagram for removing special characters from the dataset. B represents dataset with special characters.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

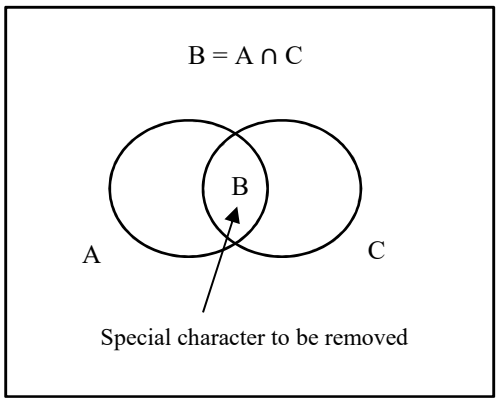

Figure. 4. Removing Special Characters

Figure. 5 shows the algorithm to remove any special characters from data sources. The purpose of this cleaning process is to improve the performance of data insertion response time and query processing response time.

Input : Data source, DS and data dictionary, dd
Output : Clean data source, C_{ds}
Begin :
Step 1: Read data source, DS
Step 2: Identify any special characters, w & refer dd
Step 3: If w is exist
Step 3.1: Remove the value
Step 3.2: Repeat step 2
Step 4: Write a new data source, C_{ds}
Step 5: Display clean data source, C_{ds}

Figure. 5: Algorithm For Removing Special Characters

E. Data Extraction

This section describes how data is extract from different data sources and convert into standard format using native JSON.

Assume B = {s_1, s_2, s_3… s_n}. Let B represent set of clean data sources, where s_1 until s_n is a clean data source that contains different elements, attributes, and text.

Figure. 6 represent algorithm for NXD. This algorithm has been designed to produce different types of documents or data models.

Input : Clean Data Source, C_{ds}
Output : Data schema (DS) in RDBMS, *M*, insertion time (*t*)
Begin :
1: Read element and attributes from C_{ds}
1.1: Identify elements of data source, *E*
1.2: Identify attributes of data source, *AT*
1.3: Write a value into E into table element, *tbl_element*
1.4: Write a value into E into table attributes, *tbl_attribute*
1.5: Repeat Step 1.1 to 1.4 until end of file
2: Map *tbl_element* and *tbl_attribute* to C_{ds}
3: Save *M* to extension of *.xml
4: Display data insertion response time, *t*

Figure. 6: Algorithm (Data Extraction – NXD)

In an NXD, XML is invisible inside the database. There is a unique database for all XML schemas and documents. NXD are especially suitable for storing irregular, deeply hierarchical and recursive data. There are example of "record" XML document in NXD stored in fixed relational database tables.

Documents	Doc_id	Doc_name
	1	dblp.xml
	2	sigmodrecord.xml

Elements	Doc_id	Elm_id	Name	OrdInParent
	1	1	article	1
	1	2	author	1
	1	3	title	1
	1	4	journal	1
	2	29	articlesTuple	29
	2	30	title	29
	…	…	…	…

Attributes	Doc_id	Atr_id	Elm_id	Name	Value
	1	1	1	Date	5.10.2006
	…	…	…	…	…

In XML approach, a basic unit file is an entity or chunk that contains content and markup. The markup database describes the content. Generally, a markup consists of tags, the name and any additional information surrounded by the "<" and ">" characters. Similarly, an end tag consists of the tag name surrounded by the "</" and ">". XML is case sensitive, so the start and end tag names must match exactly. Figure. 7

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and Figure. 8 shows how publication data is represented in XML format.

```
:
<article mdate="2003-03-31" key="tr/trier/MI97-12">
<author>Christoph W. Kebler</author>
<title>Practical PRAM Programming with Fork95 – A Tutorial</title>
<journal>Universitat Trier, Mathematik/Informatik, Forschungsbericht</journal>
<volume>97-12</volume>
<year>1997</year>
</article>
<article mdate="2003-02-11" key="tr/trier/MI99-24">
<author>Markus R. Schmidt</author>
<title>Dual Characterization of Super-Hedging Prices in a Currency Market with Proportional Transaction Costs</title>
<journal>Universitat Trier, Mathematik/Informatik, Forschungsbericht</journal>
<volume>99-24</volume>
<year>1999</year>
</article>:
:
```

Figure. 7: DBLP In XML (Dblp.Xml)

```
:
<article mdate="2003-03-31" key="tr/trier/MI97-12">
<author>Christoph W. Kebler</author>
<title>Practical PRAM Programming with Fork95 – A Tutorial</title>
<journal>Universitat Trier, Mathematik/Informatik, Forschungsbericht</journal>
<volume>97-12</volume>
<year>1997</year>
</article>
<article mdate="2003-02-11" key="tr/trier/MI99-24">
<author>Markus R. Schmidt</author>
<title>Dual Characterization of Super-Hedging Prices in a Currency Market with Proportional Transaction Costs</title>
<journal>Universitat Trier, Mathematik/Informatik, Forschungsbericht</journal>
<volume>99-24</volume>
<year>1999</year>
</article>:
:
```

Figure. 8: Sigmodrecord In XML (Sigmodrecord.Xml)

Figure. 9 represent algorithm for N-JSON. This algorithm has been designed to produce different types of documents or data models.

Input : Clean Data Source, C_{ds}
Output : Data schema (DS) in RDBMS, *M*, insertion time (*t*)
Begin :
1: Read element and attributes from C_{ds}
1.1: Identify elements of data source, *E*
1.2: Identify attributes of data source, *AT*
1.3: Write a value into E into table element, *tbl_element*
1.4: Write a value into E into table attributes, *tbl_attribute*
1.5: Repeat Step 1.1 to 1.4 until end of file
2: Map *tbl_element* and *tbl_attribute* to C_{ds}
3: Save *M* to extension of *.json
4: Display data insertion response time, *t*

Figure. 9: Algorithm (Data Extraction – N-Json)

In N-JSON, JSON is invisible inside the database. There is a unique database for all JSON schemas and documents. N-JSON are especially suitable for storing irregular, deeply hierarchical and recursive data. These are example of "record" JSON document in N-JSON stored in fixed relational database tables.

Documents	Doc_id	Doc_name
	3	dblp.json
	4	sigmodrecord.json

Elements	Doc_id	Elm_id	Name	OrdInParent
	3	34	"article"	34
	3	35	"id"	34
	3	36	"author"	34
	3	37	"title"	34
	3	38	"pages"	34
	4	49	"articlesTuple"	49
	4	49	"title"	49
	…	…	…	…

Attributes	Doc_id	Atr_id	Elm_id	Name	Value
	3	1	34	Date	5.10.2006
	…	…	…	…	…

The JSON approach represents data in an array format. JSON is built through two structures. The first is a collection of name/value pairs. In various languages, this is realized as an object, structure, dictionary, hash table, keyed list, or associate array. The second is an ordered list of values. In most languages, this is realized as an array or list of sequences. Each object begins

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with " [" and ends with "] ". Meanwhile, a value can be a string in double quotes, a number, true or false, an object, or an array. Figure. 10 and Figure. 11 shows how publication data is represented in JSON format. A JSON file is simple in that each data /record is separated by line.

{"article":[{"id":"274222","author":"N. Prati","title":"A Partial Model of NP with E.","pages":"1245-1253","year":"1994","volume":"59","journal":"J.Symb. Log.","url":"db\/journals\/jsyml\/jsyml59.html#Prati94"}],
:
"book":[{"id":"211","isbn":"3-540-60058-2","author":"Marco Cadoli","title":"Tractable Reasoning in Artificial Intelligence","series":"Lecture Notes in Computer Science","volume":"941","publisher":"Springer","year":"1995","url":"..."}],

"www":[{"id":"1","editor":"...","title":"Java Language Home Page","booktitle":"...","year":"...","url":"http:\/\/java.sun.com\/"}
:
:

Figure. 10: DBLP In JSON (Dblp.Json)

{"articlesTuple":[{"title":"Announcements.","initPage":"4","endPage":"6","author":"J. S. Knowles"}],

"articlesTuple":[{"title":"A Progress Report on the Activities of the CODASYL End User Facility Task Group.","initPage":"1","endPage":"19","author":"Henry C. Lefkovits"}],

"articlesTuple":[{"title":"SIGMOD Chairman's Message.","initPage":"20","endPage":"20","author":"James P. Fry"}],

"articlesTuple":[{"title":"Problems of Basic and Applied Research in Database Systems.","initPage":"16","endPage":"12","author":"Edgar H. Sibley,W. Terry Hardgrave"}],

:
:

Figure. 11: Sigmodrecord In JSON (Dblp.Json)

F. Data Retrieval

Assume M be a dataset including steps for query of data. M query of data are fetched from ∈ Z. M = {Q1, Q2, Q3, Q4}. Figure. 12 – 13 shows the algorithms to execute a list of queries start with Q1 until Q3.

Input : keyword, *Z* and query, *Q*, *tbl_element*, *tbl_attribute*
Output : Data retrieved, *r*, query processing response time, *c* and CPU usage, *cp*
Begin :
1. Read keyword, Z.
2. Read query, *Q*
3. Assign Z → Q
4. Assign variable *a,* for start time, *b* for end time and *cp* for CPU usage
5.Search elements and attributes from *tbl_element* & *tbl_attributes*
5.1 If data is found, map directly to XML document
5.2 Retrieve all data, *r*
5.3 *r*++
6. Assign variable *c* for different time
7. Calculate value of *c*, which *b* minus *a*
8. Display value of *r* and *c* and *cp*

Figure. 12: Algorithm (Data Retrieval – Nxd)

Input : keyword, *Z* and query, *Q*, *tbl_element*, *tbl_attribute*
Output : Data retrieved, *r*, query processing response time, *c* and CPU usage, *cp*
Begin :
1. Read keyword, Z.
2. Read query, *Q*
3. Assign Z → Q
4. Assign variable *a,* for start time, *b* for end time and *cp* for CPU usage
5.Search elements and attributes from *tbl_element* & *tbl_attributes*
5.1 If data is found, map directly to JSON document
5.2 Retrieve all data, *r*
5.3 *r*++
6. Assign variable *c* for different time
7. Calculate value of *c*, which *b* minus *a*
8. Display value of *r* and *c* and *cp*

Figure. 13: Algorithm (Data Retrieval – N-Json)

4. EXPERIMENTAL RESULTS

This study performed all its experiments on a Dell XPS13, IntelI Core I i7-5500U CPU @ 2.39 GHz with 8GB RAM using a Windows 10 64-bit platform. The software specification for algorithm development is deployed using open source software, including MySQL version 5.6.20, MySQL community server (GPL) for our database server, Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.5.15 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.

a. Data Source and Characteristics

In our experiments, SigmodRecord and DBLP will be used as a benchmark dataset. These benchmark datasets have been used for NXD approach in experiments purposes in term of data insertion time and query processing response time [7]. These datasets are download and saved in a *.xml file format. Size of these data are 704KB and 22.9MB. These datasets provide bibliographical information about computer sciences journals, books, thesis, URL, and proceedings. The overall characteristics of benchmark datasets is tabulated in Table 1. 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 1: Database Characteristics

File Name	URL	Size	Length	Records
SigmodRecord	http://www.dia.uniroma3.it/Araneus/Sigmod/	704 KB	4-7	3820
DBLP	https://hpi.de/naumann/projects/repeatability/datasets/dblp-dataset.html	22.9 MB	6-10	50000

b. Data Extraction and Data Retrieval

The main motivation for this research is to extract data from different data sources and convert into three different data models. Three experiments will be executed in this research: data insertion response time, query processing response time, and CPU usage. Query processing response time, and CPU usage are evaluated based on queries with different complexity in Table 2 and Table 3.

Table 2: Query Complexity (SIGMOD Record)

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 articleCode is greater than "152010" and less than or equal to "152010"

Table 3: Query complexity (DBLP)

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 node – 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"

- **Data Insertion Response Time**

In this section, data are extracted from different data sources and converted into two data models: NXD and N-JSON. Two types of datasets in Table 1 have been used in this experiment. In this experiment, response time for data insertion have been calculated 4 times.

Based on Figure. 14, the result of data insertion response time N-JSON is reduce to 7% compared to NXD using SigmodRecord dataset. Meanwhile, based Figure. 15, the result of data insertion response time N-JSON is reduce to 15% and 22% compared to NXD using DBLP dataset. The number of percentages for each data model can be calculated based on the following formula:

$$\frac{Avg.of\ data\ insertion\ time\ (NXD) - Avg.of\ data\ insertion\ time\ (N-JSON)}{Avg.of\ data\ insertion\ time\ (NXD)} \text{ x } 100$$

$$\text{N-JSON (SigmodRecord)} = \frac{629-586}{629} \text{ x } 100 = 6.83 \approx 7\%$$

$$\text{N-JSON (DBLP)} = \frac{25327-2152}{25327} \text{ x } 100 = 15.01 \approx 15\%$$

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

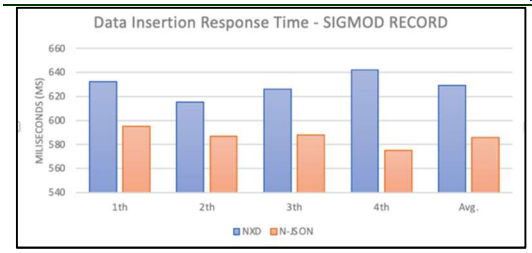

Figure 14. Data insertion response time – Query I

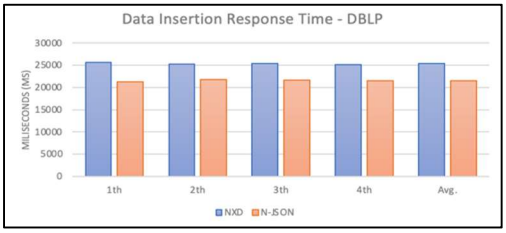

Figure. 15. Data insertion response time – Query II

- **Query Processing Response Time**

In this section, we evaluated the performance of NXD and N-JSON based on query processing response time. Three (3) different queries were executed based on specified statement in TABLE II and TABLE III, and query processing response time were executed 4 times.

The result shows N-JSON in three different queries complexity are reduce by 7%, 8%, and 5% respectively compared to NXD using SigmodRecord dataset. The number of percentages for each data model can be calculated based on the following formula:

$$\frac{Avg.of\ (NXD)\ -Avg.of\ (N-JSON)}{Avg\ of\ (NXD)}$$

$$= \frac{353-3}{353} \text{ x } 100 = 7.08 \approx 7\%$$

$$= \frac{426-393}{426} \text{ x } 100 = 7.74 \approx 8\%$$

$$= \frac{105-1}{105} \text{ x } 100 = 4.76 \approx 5\%$$

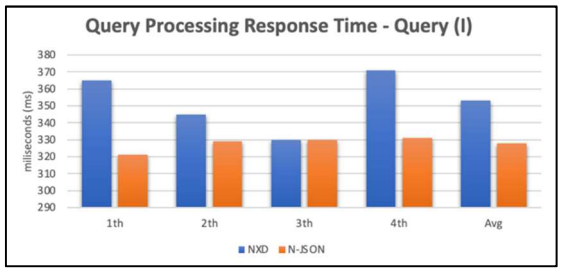

Figure. 16. Query processing response time – SIGMOD (Query I)

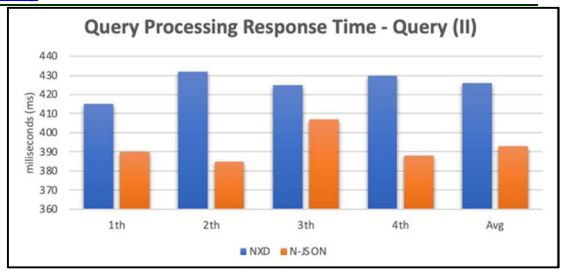

Figure. 17. Query processing response time – SIGMOD (Query II)

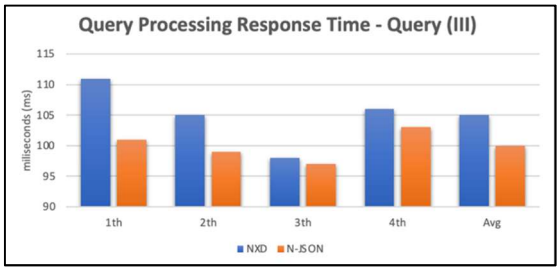

Figure. 18. Query processing response time – SIGMOD (Query III)

The result shows N-JSON in three different queries complexity are reduce by 7%, 8%, and 5% respectively compared to NXD using DBLP. The number of percentages for each data model can be calculated based on the following formula:

$$\frac{Avg.of\ (NXD)\ -Avg.of\ (N-JSON)}{Avg\ of\ (NXD)}$$

$$= \frac{353-314}{353} \text{ x } 100 = 11.08 \approx 11\%$$

$$= \frac{426-378}{426} \text{ x } 100 = 11.26 \approx 11\%$$

$$= \frac{105-95}{105} \text{ x } 100 = 9.52 \approx 9\%$$

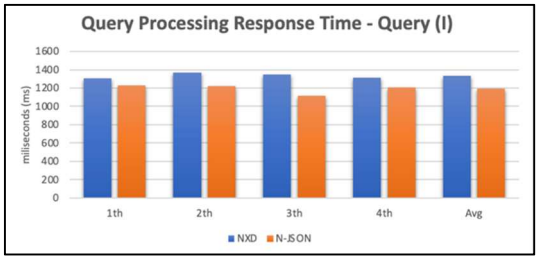

Figure. 19. Query processing response time – DBLP (Query I)

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

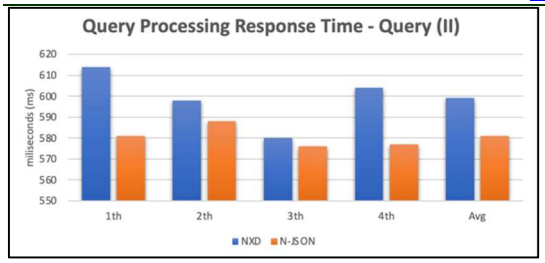

Figure. 20. Query Processing Response Time – DBLP (Query II)

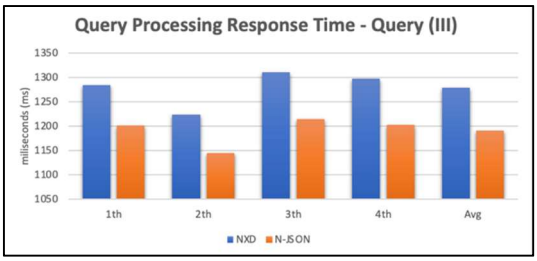

Figure. 21. Query Processing Response Time – DBLP (Query III)

- **CPU Usage**

This section evaluated the performance of CPU usage based on queries complexity in TABLE II and TABLE III by using two different datasets: SigmodRecord and DBLP.

Based on Fig. 22 – Fig. 24, the result shows N-JSON in three different queries complexity are reduce by 12%, 8%, and 6% respectively compared to NXD using SigmodRecord dataset.

$$\frac{Avg.of\ CPU\ usage\ (NXD) - Avg.of\ CPU\ usage\ (N-JS\quad)}{Avg.of\ CPU\ usage\ (NXD)} \text{ x } 100$$

$$= \frac{6.0-5.3}{6.0} \text{ x } 100 = 11.66 \approx 12\%$$

$$= \frac{10.1-9.3}{10.1} \text{ x } 100 = 7.92 \approx 8\%$$

$$= \frac{9.9-9.3}{9.9} \text{ x } 100 = 6.06 \approx 6\%$$

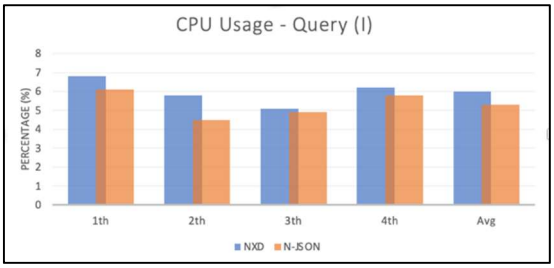

Figure. 22. CPU Usage – SIGMOD (Query I)

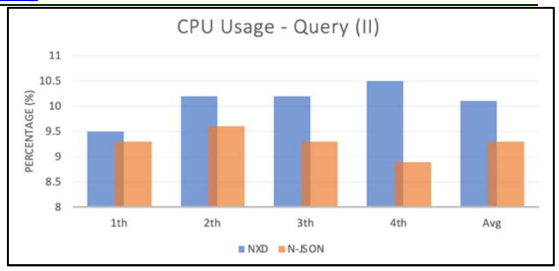

Figure. 23. CPU Usage – SIGMOD (Query II)

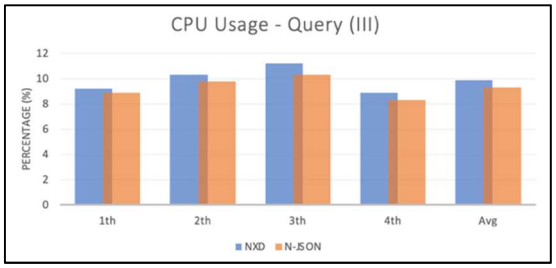

Figure. 24. CPU Usage – SIGMOD (Query III)

Based on Figure. 25 – Figure. 27, the result shows N-JSON in three different queries complexity are reduce by 11%, 7%, and 6% respectively compared to NXD using DBLP dataset.

$$\frac{Avg.of\ CPU\ usage\ (NXD) - Avg.of\ CPU\ usage\ (N-JSON)}{Avg.of\ CPU\ usage\ (NXD)} \text{ x } 100$$

$$= \frac{26.5-23.5}{26.5} \text{ x } 100 = 11.32 \approx 11\%$$

$$= \frac{27.2-25.3}{27.2} \text{ x } 100 = 6.98 \approx 7\%$$

$$= \frac{26.3-24.8}{26.3} \text{ x } 100 = 5.70 \approx 6\%$$

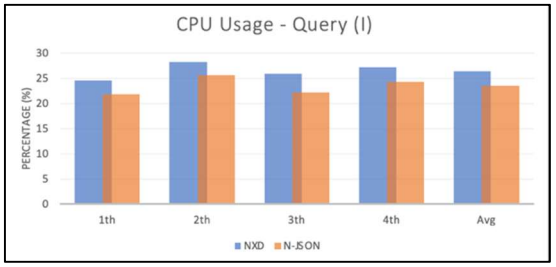

Figure. 25. CPU Usage – DBLP (Query I)

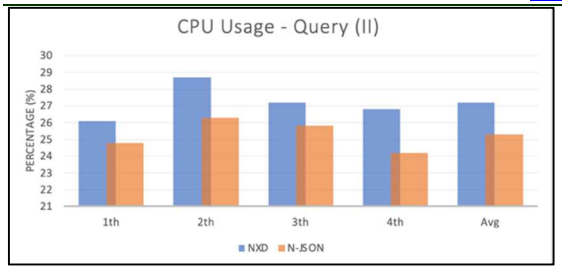

Figure. 26. CPU Usage – DBLP (Query II)

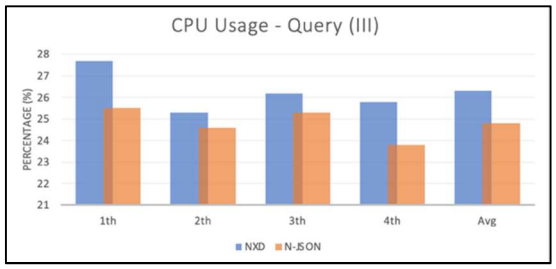

Figure. 27. CPU Usage – DBLP (Query III)

c. Performance Analysis

To evaluate the efficiency and accuracy for each database approaches, this section is focusing on the performance of extraction time for extracting information, we also want to extract the significant information of data by removing the noisy information.

$$Precision = \frac{Data\ Retrieved}{Data\ Retrieved + Data\ False}$$

$$Recall = \frac{Data\ Retrieved}{Total\ of\ Data}$$

$$FMeasure = 2.\frac{Precision.Recall}{Precision + Recall}$$

Table 4 – 6 shows the accuracy results based on three different complexity queries using SigmodRecord dataset.

Table 4: Result Of Experiments -SIGMOD RECORD (I)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	3820	34	1	0.97	0.0089	0.0176
N-JSON	3820	37	2	0.94	0.0097	0.0192

Table 5: Result Of Experiments – SIGMOD RECORD (II)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	3820	14	2	0.87	0.0036	0.0071
N-JSON	3820	18	2	0.90	0.0047	0.0094

Table 6: Result Of Experiments – SIGMOD RECORD (III)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	3820	8	4	0.66	0.0021	0.0042
N-JSON	3820	10	2	0.83	0.0026	0.0052

*Fi*gure. 28 represent the line graph based on the results TABLE 4 – 6 stated.

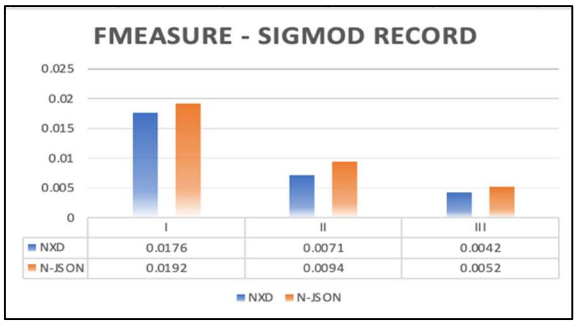

Figure. 28. Fmeasure – SIGMOD

TABLE 7 – 9 shows the accuracy results based on three different complexity queries using DBLP dataset.

Table 7: Result Of Experiments – DBLP (I)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	50000	34	1	0.97	0.0068	0.00144
N-JSON	50000	37	1	0.97	0.0074	0.00145

Table 8: Result Of Experiments – DBLP (II)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	50000	14	2	0.88	0.0038	0.0027
N-JSON	50000	18	1	0.95	0.0036	0.0035

Table 9: Result Of Experiments – DBLP (III)

Database Approach	Total Data	Data Retrieved	Data (false)	Precision	Recall	F1
NXD	50000	8	0	1.00	0.0016	0.0032
N-JSON	50000	10	1	0.91	0.0020	0.0040

Figure. 29 represent the line graph based on the results Table 7 – 9 stated.

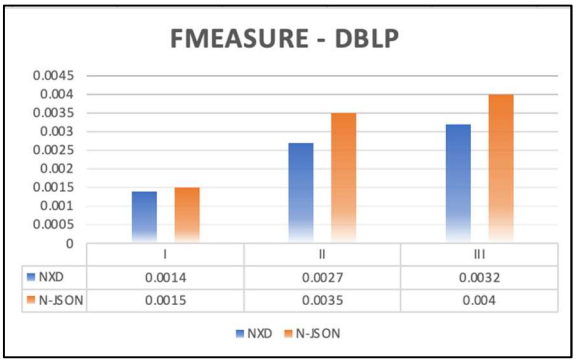

Fig. 29. Fmeasure – DBLP

5. CONCLUSION

NXD and JSON are two approaches have been reviewed in this research for data integration. N-JSON approach is proposed by combination of NXD and JSON elements to produce better performance in data integration. SigmodRecord and DBLP datasets have been used for experiment purposed. Three experiments have been done, N-JSON indicated a better performance in term of data insertion response time, query processing response time and CPU usage compared to NXD. In this case, N-JSON is proven efficient and reliable to become as an alternative data integration approach especially in business intelligent applications.

6. ACKNOWLEDGMENT

We wish to thank FRGS of Kementerian Pengajian Tinggi (KPT) for providing financial support for this study with FRGS/1/2020/ICT06/UNISZA/03/2.

REFERENCES

[1] Marian, P., 2017. Business Intelligent Integrated Solutions. Land Forces Academy Review, Vol. XXII, No. 4(88), 2017.

[2] Ana, R., & Razwan, B. (2011). Integrating Data Sources From Different Development Environments: An E-LT Approach. Quality- Access To Success, Special Issue 123, Vol 11, 2011, Pp: 785 – 792.

[3] Kisker, H., Garbani, J.P., Evelson, B., Green, C., Lisserman, 2010. The State Of Business Intelligence Software And Emerging Trends, 2010. Forrester Research Report, May 10, 2010, Available Online At Http://Www.Forrester.Com/Rb/Research/State_Of_Business_Intelligence_Softwa E_And_Emerging/Q/Id/56749/T/2, Accessed On 31 Januari 2020.

[4] Lenzerini, M. (2002). Data Integration: A Theoretical Perspective. Proceedings Of The Twenty-First ACM SIGMOD-SIGART Symposium On Principles Of Database Systems, Madison, Wisconsin, USA, June 2002.

[5] Mustafa Man, Nurul Aqilah Ruslan, Julaily Aida Jusoh, Wan Aezwani Abu Bakar, (2020). Conceptual Model Of Incremental R-Eclat Algorithm For Infrequent Itemset Mining, International Journal Of Engineering Trends And Technology (IJETT), Vol. 10, Pp: 129 – 133.

[6] Bouzeghoub, M., Loscio, B. F., Kedad, Z., Soukane, A. (2002). Heterogeneous Data Source Integration And Evolution. DEXA 2002, LNCS 2453, Pp: 751 – 757.

[7] Marjani, M., Nasaruddin, F., Ghani, A., Shamshirband, S. (2018). Measuring Transaction Performance Based On Storage Approaches Of Native XML Database. Measurement 114, 91 -101.

[8] Eriksson, M., & Hallberg, V. (2011). Comparison Between JSON

And YAML For Data Serialization. KTH Computer Science And Communication.

[9] Saba, A. M., Shahab, E., Abdolrahimpour, H., Hakimi, M., Moazzam, A. (2017). A Comparative Analysis Of XML Documents, XML Enabled Databases And Native XML Databases. Computer Science: Databases.

[9*] Gonzalez, M., Prieto, M., Nieto, M. (2009). A Study Of Native XML Databases – Document Update, Querying, Access Control And Application Programming Interfaces In Native XML Databases. 5th International Conference On Web Information System And Technologies, Pp: 89 – 92.

[11] Lazetic, G. M. P. (2007). Native XML Databases Vs. Relational Databases In Dealing With XML Documents. Kragujevac J. Math. Vol 30, Pp: 181 – 199.

[12] Bourret, R. (2007). XML Database Products. Http://Www.Rpbourret.Com/Xml/Xmlanddatabase-Products.Htm. Last Updated March 2007. Visited The 2008/10/20.

[13] Haw, S. C., & Lee, C. S. (2014). Efficient Preprocesses For Fast Storage And Query Retrieval In Native XML Database. IETE Technical Review. Vol 26, Issue 1, 28 - 40.

[14] Nurseitov, N., Paulson, M., Reynolds, R., Izurieta, C. (2009). Comparison Of JSON And XML Data Interchange Formats: A Case Study.CAINE 2009, Computer Science.

[15] Wang, L., Hassanzadeh, O., Zhang, S., Shi, J., Jiao, L., Zou, J., Wang, C. (2015). Schema Management For Document Stores. Proceedings Of The VLDB Endowment, Vol. 8, No. 9. Pp: 922-933.

[16] Meneghello, M. 2001. XML (Extensible Markup Language) – The New Language Of Data Exchange. Cartography, Vol 30, No. 1, 51 – 57

[17] Yusof, M.K., & Man, M. (2016). Efficiency Of JSON Approach For Data Extraction And Query Retrieval. Indonesian Journal Of Electrical Engineering And Computer Science. Vol 4, 201 - 214.

[18] Eriksson, M., & Hallberg, V. (2011). Comparison Between JSON And YAML For Data Serialization. KTH Computer Science And Communication.

[19] Kamir, Y., Mustafa, Man. (2018). Efficiency Of Flat File Database Approach In Data Storage And Data Extraction For Big Data, Indonesian Journal Of Electrical Engineering And Computer Science, Vol. 9, No. 2, Pp: 460 - 473, Doi: 10.11591/Ijeecs.V9.I2.Pp460-473

[20] Afsari, K., Eastman, C. M., Lacouture, D. C. (2017). Javascript Object Notation (JSON) Data Serialization Of IFC Schema In Web-Based BIM Data Exchange. Automation In Construction. 24 - 51.

[21] Mustafa Man, Wan Aezwani Wan Abu Bakar, Zailani Abdullah, Masita Abdul Jalil, 2016. Mining Association Rules: A Case Study On Benchmark Dense Data, Indonesian Journal Of Electrical Engineering And Computer Science, Vol. 3, No. 3, Pp: 546 – 553.