Paweł DRZYMAŁA, Henryk WELFLE

Politechnika Łódzka, Instytut Mechatroniki i Systemów Informatycznych

doi:10.15199/48.2023.02.33

Mechanizmy wsparcia standardu JSON w celu wydajnego przechowywania i przetwarzania danych w środowisku IBM DB2

Streszczenie. *W pracy zaprezentowano mechanizmy wsparcia standardu JSON (JavaScript Object Notation) dla przechowywania i przetwarzania danych w środowisku IBM DB2 LUV od wersji v11.1, które wykorzystywane są w szeroko rozumianych aplikacjach biznesowych. Coraz więcej danych JSON przechowywanych jest i przetwarzanych w środowiskach bazodanowych, zarówno bazach relacyjnych i nierelacyjnych (NoSQL). W środowisku bazodanowym IBM DB2 nie przewidziano specyficznego dla formatu JSON typu danych, tak jak jest to dla typu danych XML. Zastosowanie indeksów pozwala na wydajne sortowanie i filtrowanie danych, które przechowywane są we właściwościach dokumentów w formacie JSON.*

Abstract*. The paper presents the possibilities of storing, processing and exchanging JSON data (JavaScript Object Notation) in RDBMS IBM DB2 LUV from version v11.1, which are used in widely understood business applications. Increasingly JSON data is stored and processed in database environments, both relational and non-relational databases (NoSQL). The IBM DB2 database environment does not provide a JSON-specific data type as it is for the XML data type. The use of indexes allows efficient sorting and filtering of data, which are stored in the properties of documents in JSON format. (**JSON standard support mechanisms for efficient data storage and processing in IBM DB2 environment**).*

Słowa kluczowe: *IBM DB2, JSON (JavaScript Object Notation), indeksowanie danych JSON.*
Keywords: *IBM DB2, JSON (JavaScript Object Notation), indexing JSON data.*

Wprowadzenie

W pracy zaprezentowano mechanizmy wsparcia standardu JSON (JavaScript Object Notation) dla przechowywania i przetwarzania danych w środowisku IBM DB2 LUV od wersji v11.1. Składowane i przetwarzanie danych w formacie JSON w środowisku bazodanowym pozwala na bardziej naturalny dostęp do danych oraz wymianę informacji w aplikacjach wykorzystujących usługi REST, które wykorzystują ten format. W środowisku bazodanowym IBM DB2 nie przewidziano specyficznego dla formatu JSON typu danych, tak jak jest to dla typu danych XML. W przechowywaniu danych po stronie systemu DB2, w zależności od rozmiaru dokumentu JSON, wykorzystywane są typy: CHAR, VARCHAR, CLOB (Character Large OBject) lub BINARY, VARBINARY, BLOB (Binary Large OBject). Istnieje jednak możliwość sprawdzenia czy dane JSON są poprawnie sformułowane. W celu zwiększenia wydajności zapytań realizowanych na danych JSON należy odpowiednio indeksować tego typu pola.

Ponieważ dane JSON są przechowywane w bazie danych przy użyciu standardowych typów danych to zapytania SQL operują na danych JSON tak samo, jak na danych innych typów.

Aby wyszukać konkretne pola i atrybuty JSON lub odwzorować poszczególne pola formatu JSON na kolumny SQL używa się języka SQL/JSON i notacji kropki dla danych ścieżek.

W bazie danych IBM DB2 istnieją funkcje i rozszerzenia języka SQL/JSON, które zapewniają zwiększoną wydajność i elastyczność obsługi danych JSON:

- warunek JSON_EXISTS sprawdza obecność określonej wartości w danych JSON,
- funkcje BSON_TO_JSON i JSON_TO_BSON wykorzystywane w przypadku danych binarnych,
- funkcja JSON_VALUE wybiera wartość skalarną z danych JSON jako wartość SQL,
- funkcja JSON_TABLE wyświetla dane JSON jako wirtualną tabelę,
- funkcja JSON_QUERY wybiera jedną lub więcej wartości z danych JSON.

Wbudowane funkcje do przetwarzania danych w formacie JSON

W środowisku IBM DB2 do przechowywania danych w formacie JSON wykorzystuje się dostępne standardowe typy CHAR, VARCHAR, CLOB, BINARY, VARBINARY lub BLOB, stosowane np. przy określaniu typu kolumn w definicjach tabel,

```
CREATE TABLE emp_json
(  empno      INTEGER NOT NULL PRIMARY KEY,
   flex_json  VARCHAR(4000));
```

Aby być pewnym, że struktura przechowywanego dokumentu jest zgodna ze standardem JSON, można zdefiniować własną funkcję użytkownika np. o nazwie CHECK_JSON następującej postaci:

```
CREATE OR REPLACE FUNCTION CHECK_JSON(JSON
VARCHAR(4000))
RETURNS INTEGER
CONTAINS SQL LANGUAGE SQL DETERMINISTIC NO
EXTERNAL ACTION
BEGIN
   DECLARE RC BOOLEAN;
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
RETURN(FALSE);
   SET RC = JSON_EXISTS(JSON,'$' ERROR ON
ERROR);
   RETURN(TRUE);
END@
```

Następnie wykorzystać własną funkcję CHECK_JSON do zdefiniowania ograniczenia CHECK na konkretnej kolumnie z danymi JSON:

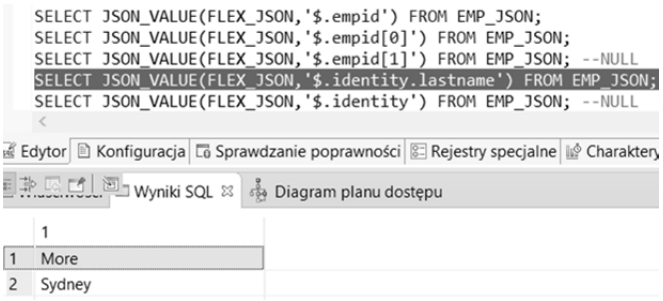
```
ALTER TABLE emp_json
    ADD CONSTRAINT JSON_OK
    CHECK (CHECK_JSON(FLEX_JSON);
```

Struktura przykładowej tabeli z polem typu JSON jest odpowiednikiem zbiorów, które można znaleźć w klasycznych bazach typu klucz-dokument. Taka struktura jest dobrym wyborem dla klasycznych scenariuszy NoSQL, w których użytkownik zamierza przetwarzać dane JSON.

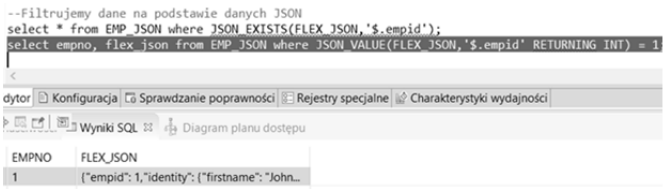
```
INSERT INTO DB2ADMIN.EMP_JSON(empno, flex_json) VALUES
(
1,
'{
    "empid": 1,
    "identity": {
        "firstname": "John",
        "lastname": "More",
        "birthdate": "1998-04-15"
    }
}'
);
```

Rys.1. Przykład wstawienia danych JSON do tabeli z danymi JSON.

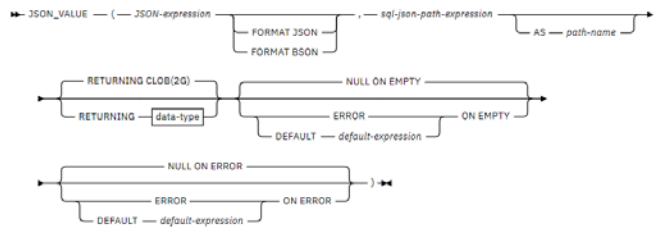
Wyszukiwanie i filtrowanie danych przechowywanych w bazie danych w formacie JSON jest dostępne z wykorzystaniem funkcji JSON_VALUE.



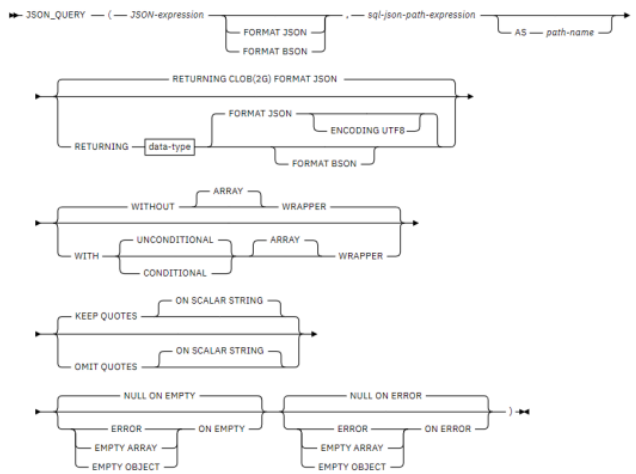
Rys.2. Wykorzystanie funkcji JSON_VALUE operującej na danych JSON



Rys.3. Wykorzystanie funkcji JSON_VALUE do filtrowania danych JSON



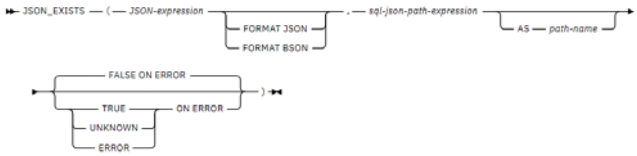
Rys.4. Składnia funkcji JSON_VALUE [1]



Rys.5. Składnia funkcji JSON_QUERY [1]

Funkcja JSON_QUERY zwraca wartość SQL/JSON z określonego tekstu JSON przy użyciu danej ścieżki co pokazano na przykładzie:

```
select JSON_QUERY(CUSTOMER_INFO,
'$.identity') from CUSTOMERS_JSON
```



Rys.6. Predykat JSON_EXISTS [1]

Predykat JSON_EXISTS określa, czy dane typu JSON zawierają wartość JSON, którą można zlokalizować przy użyciu określonej ścieżki w strukturze JSON. Jeżeli dane JSON istnieją zwracana jest wartość True. Wykorzystanie predykatu JSON_EXISTS pokazano na przykładzie:

```
select * from CUSTOMERS_JSON
where JSON_EXISTS(CUSTOMER_INFO,
'$.customerid');
```

Predykat JSON_EXISTS można użyć do określenia, czy w dokumencie występują pola kluczowe formatu JSON. Możemy użyć wyniku tej operacji, aby określić, czy zawartość dokumentu JSON jest zgodna z oczekiwaniami i zdecydować, czy podjąć dalsze działania lub pobrać wartość.

Funkcja JSON_TO_BSON sprawdzi, czy dokument jest poprawnie sformatowany w formacie JSON. Jeśli nie przechowujemy dokumentów w formacie BSON, możesz użyć funkcji JSON_EXISTS w funkcji zdefiniowanej przez użytkownika, aby sprawdzić dokument przed uruchomieniem jakichkolwiek innych funkcji względem niego.

Różne sposoby przechowywania danych JSON

Standard ISO JSON nie określa, jakiego formatu danych należy używać do przechowywania rekordów JSON. Ponieważ JSON może być przechowywany w jego natywnym formacie znakowym lub w formacie binarnym (BSON), decyzja, którego użyć, należy do użytkownika. Funkcje Db2 JSON akceptują jako dane wejściowe zarówno wartości w formacie BSON, jak i JSON, co oznacza, że z perspektywy programistycznej nie ma potrzeby konwertowania z formatu JSON na BSON (lub odwrotnie), aby używać funkcji Db2 JSON.

Przykładowe możliwości reprezentowania danych:

- JSON; dane przechowywane w formacie JSON w kolumnie VARCHAR,
- BLOB; dane przechowywane w formacie BSON w kolumnie BLOB,
- BSON; dane przechowywane w formacie BSON w kolumnie VARBINARY.

W zależności od ilości dokumentów i zapytania realizowanego na danych JSON najwolniej wykonywane są operacje na danych JSON przechowywanych w kolumnie typu VARCHAR, następnie BLOB a na końcu VARBINARY. VARBINARY jest ograniczone do wartości maksymalnego rozmiaru strony DB2 (32K), więc jeśli dokumenty będą większe to musimy użyć BLOB.

Poniższy kod SQL demonstruje różne sposoby definiowania kolumny JSON w tabeli.

Tworzenie tabeli z tekstowymi kolumnami JSON:

```
CREATE TABLE JSON_DATA
(
FIELD1 CHAR(255),
FIELD2 VARCHAR(300),
FIELD3 CLOB(1000)
--INLINE LENGTH 1000
--wydajniejszy sposób przechowywania danych
);
```

Tworzenie tabeli z binarnymi kolumnami JSON (BSON):

```
CREATE TABLE BSON_DATA
(
FIELD1 BINARY(255),
FIELD2 VARBINARY(300),
FIELD3 BLOB(1000)
);
```

Decyzja o użyciu BSON lub JSON jako formatu przechowywania sprowadza się do tego, czy aplikacja musi regularnie wyszukiwać pola w dokumencie JSON. Jeśli większość dostępu do JSON polega na przechowywaniu i pobieraniu całych dokumentów, wówczas obciążenie związane z konwersją BSON jest niepotrzebne. Jeśli jednak wzorzec dostępu do dokumentu JSON jest nieznany, warto przekonwertować dokumenty na BSON w celu szybszego wyszukiwania. Inną opcją jest użycie indeksów omówionych w następnej sekcji.

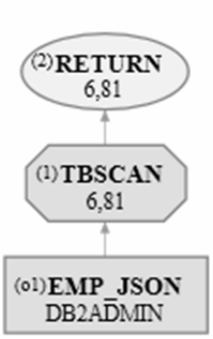
Oznacza to, że istnieją dwa obszary, w których ten niejawny narzut z JSON na BSON może wpłynąć na wydajność zapytań podczas uzyskiwania dostępu do dokumentu JSON:
- Ile wartości musisz zmaterializować jako część listy kolumn SELECT,
- Do ilu wartości trzeba się odwoływać w predykatach SQL.

Indeksowanie danych JSON

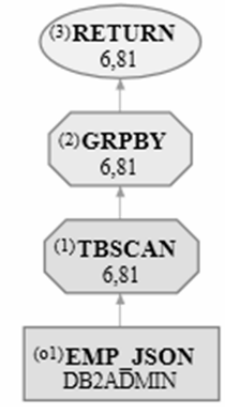
Aby dostroić wydajność zapytań, można indeksować pola JSON. Zastosowanie indeksów pozwala na wydajne sortowanie i filtrowanie danych, które przechowywane są we właściwościach dokumentów w formacie JSON. Bez implementacji indeksów założonych na danych w formacie JSON, środowisko IBM DB2 wykonuje pełne skanowanie tabeli za każdym razem, gdy dane są pobierane. Nie należy bezpośrednio indeksować kolumn z danymi formacie JSON, gdyż taki indeks nie jest użyteczny. Natomiast często interesują nas elementy występujące w formacie JSON. W takim przypadku system DB2 obsługuje tzw. indeksy obliczane (computed indexes), które wykorzystują funkcję JSON_VALUE.

```
-- Wykonujemy zapytanie bez indeksu
-- (VISUAL EXPLAIN wykorzystujemy do obserwacji
--  planu wykonania zapytania)
select * from emp_json
where JSON_VALUE(FLEX_JSON, '$.empid' RETURNING INT) = 1;
-- Wykonujemy zapytanie bez indeksu z opcją grupowania
-- (VISUAL EXPLAIN wykorzystujemy do obserwacji
--  planu wykonania zapytania)
select count(*) from emp_json
where JSON_VALUE(FLEX_JSON, '$.empid' RETURNING INT) = 1;
```

Rys.7. Wykorzystanie funkcji JSON_VALUE operującej na danych JSON z operacją wybierania wszystkich kolumn i grupowania.



Rys.8. Plan wykonania zapytania dla tabeli bez wykorzystania indeksu (skanujemy całą zawartość tabeli)

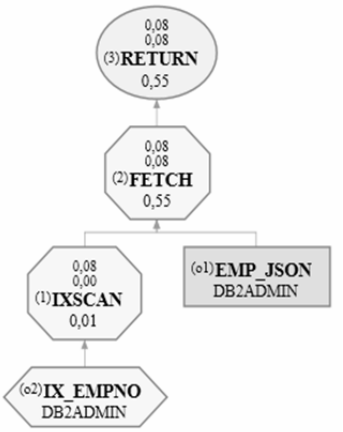


Rys.9. Plan wykonania zapytania dla tabeli bez wykorzystania indeksu z opcją grupowania danych (skanujemy całą zawartość tabeli)

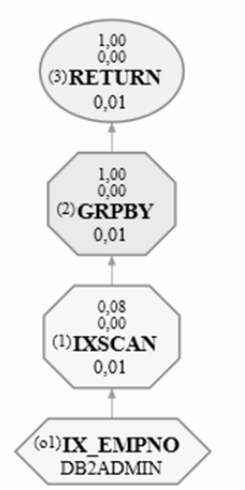
```
--Tworzymy indeks
CREATE INDEX IX_EMPNO
    ON EMP_JSON
    (JSON_VALUE(FLEX_JSON,'$.empid' RETURNING INT) ASC);

-- Wykonujemy zapytanie korzystające z indeksu
-- (VISUAL EXPLAIN wykorzystujemy do obserwacji
--  planu wykonania zapytania)
select * from emp_json
where JSON_VALUE(FLEX_JSON, '$.empid' RETURNING INT) = 1;
-- Wykonujemy zapytanie korzystające z indeksu z opcją grupowania
-- (VISUAL EXPLAIN wykorzystujemy do obserwacji
--  planu wykonania zapytania)
select count(*) from emp_json
where JSON_VALUE(FLEX_JSON, '$.empid' RETURNING INT) = 1;
```

Rys.10. Tworzenie indeksu przed wykonaniem zapytań.



Rys.11. Plan wykonania zapytania dla tabeli z wykorzystaniem indeksu dla przykładowego zapytania.



Rys.12. Plan wykonania zapytania dla tabeli z wykorzystaniem indeksu dla zapytania z grupowaniem danych

Jedną z kwestii związanych z tworzeniem indeksów w dokumentach JSON jest to, że funkcja JSON_VALUE musi zawierać klauzulę RETURNING. Instrukcja CREATE INDEX nie może określić typu danych z polecenia i zgłosi komunikat o błędzie podczas próby utworzenia indeksu.
Patrząc na wydajność zapytań dla różnych typów przechowywania danych JSON a korzystających tylko z indeksu to ich wydajność będzie porównywalna. Ale już jeśli wybieramy dane nie przechowywane w indeksie dostęp będzie różny co przedstawiono wcześniej.

Podsumowanie

W artykule zaprezentowany został proces składowania i przetwarzania danych typu JSON w strukturach baz relacyjnych. Przedstawiono również mechanizmy indeksowania takich danych oraz wpływ tego indeksowania na podniesienie wydajności przetwarzania informacji.

W porównaniu do formatu JSON obsługa danych formatu XML jest bardziej zaawansowana oraz zapewnia więcej mechanizmów wsparcia przez środowisko IBM DB2. Jednak docelowo z każdą nową wersją tego środowiska można spodziewać się implementacji nowych mechanizmów przetwarzania danych JSON. Zasadniczą różnicą jest sposób przechowywania danych XML wykorzystującą technologię PureXML, a danymi JSON w postaci normalnych danych tekstowych lub binarnych.

Możliwe jest również zdefiniowanie własnych funkcji, które zwiększą funkcjonalność przetwarzania danych typu JSON a w połączeniu z wykorzystaniem indeksów zapewnią wzrost wydajności przetwarzania.

Przedstawione funkcje są proste w użyciu, co powoduje łatwość wykorzystania ich przy obsłudze danych. Należy zwrócić uwagę iż nie ma osobnego typu danych do przechowywania formatu JSON tak jak istnieje to w przypadku formatu XML a do przechowywania tego typu danych wykorzystujemy typ VARCHAR a dodatkowo dla zapewnienia poprawności przechowywanych danych musimy zbudować i wykorzystać własną funkcję.

Jeśli dokumenty JSON są bardzo duże, narzut indeksowania może być jeszcze większy niż w przypadku niewielkich dokumentów. Wielu programistów korzysta z magazynów dokumentów zaprojektowanych z myślą o wyjątkowo dużej przepustowości. Większość indeksów JSON to indeksy oparte na funkcjach, co oznacza, że narzut związany z konserwacją jest większy niż zwykły indeks B-Tree. Te same zasady dotyczą konserwacji wszystkich indeksów (B-drzewa, bitmapowych i pełnotekstowych).

Przy rozważaniach dotyczących wydajności zawierających dane JSON należy uwzględniać sposób, w jaki dokumenty JSON będą dostępne dla aplikacji oraz wymagań dotyczących wydajności aplikacji. Działania takiego będą musiały zrównoważyć korzyści i koszty każdego możliwego rozwiązania.

Jeśli zamierzamy uzyskiwać dostęp do wielu indywidualnych kluczy w dokumentach JSON lub chcemy zmaksymalizować wydajność, konwersja wszelkich przychodzących danych JSON na format BSON podczas przechowywania danych poprawi wydajność w momencie dostępu. Jednocześnie dostrajanie wydajności zapytań z wykorzystaniem indeksów w kluczowych predykatach JSON znacznie poprawią ich wydajność.

Z perspektywy poleceń Insert, jeśli dane przychodzące nie są jeszcze w formacie BSON, przechowywanie danych w formacie BSON jest droższe ponieważ konieczne będzie wywołanie funkcji konwersji JSON_TO_BSON. Rozmiar tabel dla różnych typów będzie podobny z minimalnymi różnicami. Istnieją pewne oszczędności miejsca związane z używaniem BSON, ale z dodatkowym kosztem wzrostu czasu przetwarzania podczas wstawiania (zakładając, że musi być dokonana konwersja).

Chociaż BSON ma niewielką przewagę nad JSON, biorąc pod uwagę zaawansowane możliwości kompresji DB2, nie ma zbyt wielu korzyści z oszczędności miejsca między BSON i JSON.

W Db2 V11 wprowadzono osiem nowych funkcji JSON, które umożliwiają użytkownikom przechowywanie, pobieranie i publikowanie dokumentów JSON bez potrzeby korzystania z bibliotek zewnętrznych. Dokumenty JSON mogą być przechowywane w DB2 w formacie JSON (znakowym) lub w formacie BSON (binarny JSON). Standardowa składnia SQL może służyć do wstawiania i pobierania dokumentów JSON bez wymaganej konwersji.

Wybór używanego formatu przechowywania zależy od wymagań dotyczących wydajności, użycia i pochodzenia danych. Nowe funkcje DB2 JSON będą działać z dowolnym formatem bez konieczności określania przez użytkownika podstawowej struktury dokumentu JSON.

***Autorzy**: dr inż. Paweł Drzymała, Politechnika Łódzka, Instytut Mechatroniki i Systemów Informatycznych, Stefanowskiego 22, 90-537 Łódź, E-mail: pawel.drzymala@p.lodz.pl; dr inż. Henryk Welfle, Politechnika Łódzka, Instytut Mechatroniki i Systemów Informatycznych, Stefanowskiego 22, 90-537 Łódź, E-mail: henryk.welfle@p.lodz.pl.*

LITERATURA

[1] Dokumentacja IBM DB2 11.5: https://www.ibm.com/ docs/en/db2/ 11.5?topic=functions-json-value
[2] Specification Version 1.1: http://bsonspec.org/spec.html
[3] Introducing JSON - www.json.org/
[4] JSON Schema - json-schema.org
[5] G. Baklarz, P. Bird ; Db2 for Linux, Unix, and Windows: Version 11 JSON Highlights; 2019
[6] Drzymała P., Welfle H Drzymała A.J. „Efektywne przetwarzanie i integracja dużych zbiorów danych w środowisku Hadoop”, Przegląd Elektrotechniczny 2019, 1, 31–34, doi: 10.15199/48.2019.01.08.