

JSON w bazie danych PostgreSQL

Maciej Kubicki



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Fizyki i Informatyki Stosowanej

24 stycznia 2017

- 1 Cel pracy
- 2 Dane testowe
 - Pierwszy zestaw danych
 - Drugi zestaw danych
- 3 JSON w PostgreSQL
 - Operatory
 - Funkcje przetwarzające
 - Funkcje tworzące
 - Indeksy
- 4 Porównanie obsługi JSON w PostgreSQL z rozwiązaniami NoSQL
 - Wprowadzanie danych
 - Przeszukiwanie danych
 - Modyfikacja danych
 - Rozmiar danych
 - Wnioski

Cel pracy oraz użyte narzędzia

Celem pracy jest przedstawienie możliwości formatu JSON zaimplementowanego w relacyjnej bazie danych jaką jest PostgreSQL oraz porównanie tychże narzędzi z innymi rozwiązaniami NoSQL-owymi.

W pracy inżynierskiej używałem PostgreSQL 9.6.1, MongoDB 3.2.1, Couchbase Server Enterprise 4.5.1 oraz Python 2.7 wraz z odpowiednimi bibliotekami.

Pierwszy zestaw testowy

Pierwszy zestaw został wygenerowany na podstawie AdventureWorks2008 z Microsoft SQL Server. Oto kod, który wygenerował zbiór 18798 dokumentów JSON.

```
1 from DatabasePreparation import *
2 import pyodbc
3 cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER=localhost\SQLEXPRESS;
    DATABASE=AdventureWorks2008;')
4 cursor = cnxn.cursor()
5 cursor.execute("SELECT e.EmailAddress, pp.FirstName, ISNULL(pp.
    MiddleName, '') AS MiddleName, pp.LastName, p.BusinessEntityID, [
    AddressLine1], ISNULL([AddressLine2], '') AS AddressLine2, [City], [
    PostalCode] FROM [AdventureWorks2008].[Person].[Address] a, [
    AdventureWorks2008].[Person].[EmailAddress] e, [AdventureWorks2008
    ].[Person].[BusinessEntityAddress] p, [AdventureWorks2008].[Person
    ].[Person] pp WHERE p.AddressID = a.AddressID and pp.
    BusinessEntityID=p.BusinessEntityID and e.BusinessEntityID=pp.
    BusinessEntityID;")
6 rows = cursor.fetchall()
7 with open('person.json', 'w') as pl:
8     for row in rows:
9         if is_json(to_json(row)):
10             pl.write(to_json(row, False) + '\n')
```

Pierwszy zestaw testowy

Funkcja `to_json` na podstawie wyniku wiersza generuje dokument JSON. Struktura wygenerowanych dokumentów nie jest jednakowa. Mianowicie nie każdy człowiek (odpowiadający mu dokument) posiada klucz `MiddleName` oraz `address_line2`. Dodatkowo liczba adresów e-mail przypisanych do osoby może wynosić 0,1,2.

```
1 {  
2     "person": {  
3         "personID": 12,  
4         "FirstName": "Thierry",  
5         "MiddleName": "B",  
6         "LastName": "DHers"  
7     },  
8     "address": {  
9         "city": "Bothell",  
10        "address_line1": "1970 Napa Ct.",  
11        "postal_code": "98011"  
12    },  
13    "email": ["thierry0@adventure-works.com", "DHers12@mail.com"]  
14 }
```

Listing 1: Wygenerowany dokument JSON

Pierwszy zestaw testowy - PostgreSQL

```
1 CREATE TABLE PersonJSONB (  
2     "ObjectID" SERIAL NOT NULL ,  
3     data JSONB NOT NULL,  
4     PRIMARY KEY("ObjectID"));  
5 CREATE TABLE PersonJSON(  
6     "ObjectID" serial NOT NULL,  
7     data JSON NOT NULL,  
8     PRIMARY KEY ("ObjectID"));
```

Listing 2: Tworzenie tabeli w PostgreSQL z typami jsonb i json

Drugi zestaw testowy - PostgreSQL

```
1 CREATE TYPE Gender AS ENUM( 'M' , 'F' );
2 CREATE TABLE Users (
3     idUsers SERIAL NOT NULL ,
4     username VARCHAR(20) NOT NULL ,
5     password VARCHAR(255) NOT NULL ,
6     mail VARCHAR(255) NOT NULL ,
7     Name VARCHAR NOT NULL ,
8     Surname VARCHAR NOT NULL ,
9     town VARCHAR(20) NOT NULL ,
10    gender Gender NOT NULL ,
11    country VARCHAR(255) NOT NULL,
12    PRIMARY KEY(idUsers));
```

Listing 3: Tworzenie tabeli z drugim zestawem danych

Funkcjonalność wspierająca JSON w PostgreSQL-u:

- Typy danych obsługujące format JSON: json i jsonb,
- Operatory działające na powyższych typach,
- Funkcje przetwarzające dokumenty JSON,
- Funkcje tworzące dokumenty JSON,
- Indeksy na kluczach dokumentów JSON.

Operatory

Jakie? Do czego służą?

```
1 SELECT "ObjectID", data->'person'->>'LastName' AS Surname FROM  
   personjsonb WHERE data->'person'->>'LastName' LIKE 'T%'  
   LIMIT 5;
```

Listing 4: Przykład użycia operatorów

ObjectID	Surname
30	Tsoflias
37	Tiedt
41	Tibbott
77	Tejani
161	Turner
(5 wierszy)	

Rysunek: Wynik zapytania z przykładu

Funkcje przetwarzające

```
1 SELECT jsonb_pretty(data) FROM public.personjsonb WHERE "ObjectID" = 1922;  
2 UPDATE public.personjsonb SET data = (SELECT jsonb_set(data, '{email,1}', 'newmail@hotmail.com' ,  
   , false) FROM public.personjsonb WHERE "ObjectID" = 1922) WHERE "ObjectID" = 1922;  
3 SELECT jsonb_pretty(data) FROM public.personjsonb WHERE "ObjectID" = 1922;
```

Listing 5: Przykład przetwarzania

```
      jsonb_pretty  
-----  
{  
  "email": [  
    "jeremiah2@adventure-works.com",  
    "Taylor3991@email.com"  
  ],  
  "person": {  
    "LastName": "Taylor",  
    "personID": 3991,  
    "FirstName": "Jeremiah",  
    "MiddleName": "M"  
  },  
  "address": {  
    "city": "Royal Oak",  
    "postal_code": "V8X",  
    "address_line1": "5906 Walnut Place"  
  }  
}  
(1 wiersz)
```

Rysunek: Przed modyfikacją

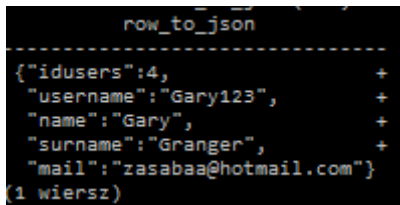
```
      jsonb_pretty  
-----  
{  
  "email": [  
    "jeremiah2@adventure-works.com",  
    "newmail@hotmail.com"  
  ],  
  "person": {  
    "LastName": "Taylor",  
    "personID": 3991,  
    "FirstName": "Jeremiah",  
    "MiddleName": "M"  
  },  
  "address": {  
    "city": "Royal Oak",  
    "postal_code": "V8X",  
    "address_line1": "5906 Walnut Place"  
  }  
}  
(1 wiersz)
```

Rysunek: Po modyfikacji

Jakie? Po co?

```
1 SELECT row_to_json(row, true) FROM (SELECT idUsers, username,  
    Name, Surname, mail FROM Users WHERE idUsers=4) row;
```

Listing 6: Przykład tworzenie dokumentu JSON



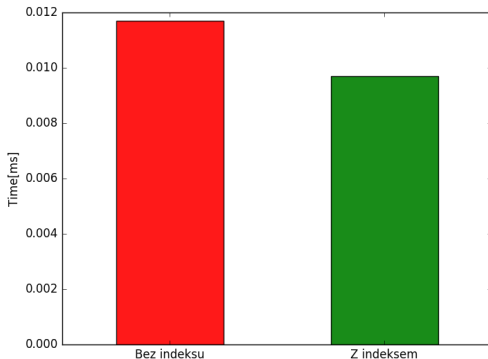
```
row_to_json  
-----  
{\"idusers\":4, +  
  \"username\":\"Gary123\", +  
  \"name\":\"Gary\", +  
  \"surname\":\"Granger\", +  
  \"mail\":\"zasabaa@hotmail.com\"}  
(1 wiersz)
```

Rysunek: Wynik z przykładu

Używanie indeksów w dokumentach JSON

```
1 SELECT "ObjectID", data->'person'->>'LastName'::text AS Surname FROM personjsonb WHERE data->'
   person'->>'LastName'::text LIKE 'T%';
2 CREATE INDEX surname ON personjsonb (((data->'person'->>'LastName')::text));
3 SELECT "ObjectID", data->'person'->>'LastName'::text AS Surname FROM personjsonb WHERE data->'
   person'->>'LastName'::text LIKE 'T%';
```

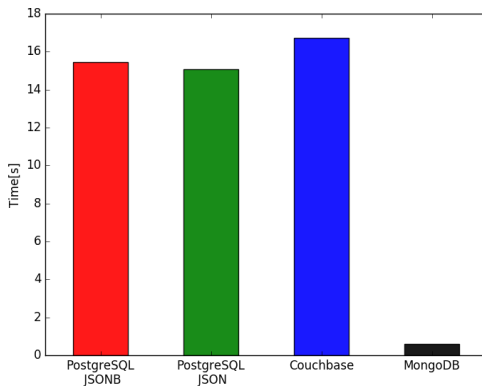
Listing 7: Przykład użycia indeksów



Rysunek: Czas wykonania przykładowego zapytania z użyciem i bez użycia indeksu

Porównanie - wprowadzanie danych

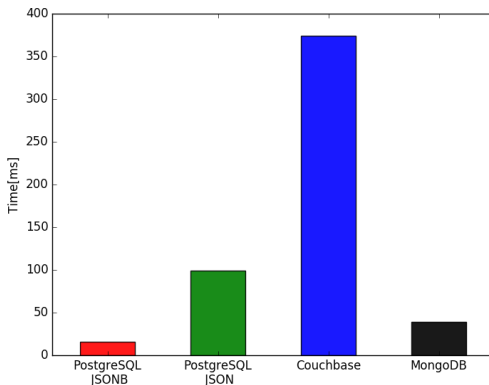
```
1 \i 'person_insert_jsonb.sql'
2 \i 'person_insert_json.sql'
3 mongoimport --db test --collection person_col --drop < person.json
4 cbdocloader -n localhost:8091 -u Administrator -p password -b Person_Bucket Person_Bucket.zip
```



Rysunek: Czas wprowadzania danych do poszczególnych baz

Porównanie - przeszukiwanie danych

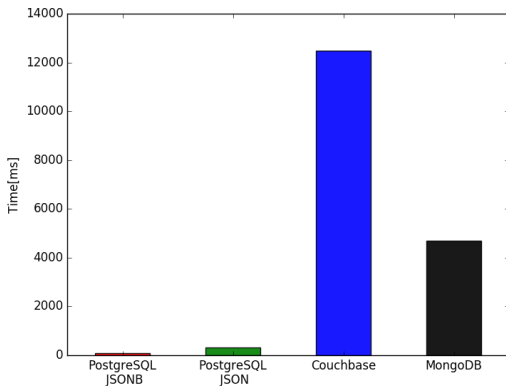
```
1 SELECT max(cast(data->'person'->>'personID' AS int)) FROM personjsonb;  
2 SELECT max(cast(data->'person'->>'personID' AS int)) FROM personjson;  
3 SELECT max(person.personID) FROM 'Person_Bucket';  
4 db.person_col.find({}, {"person.personID":1}).sort({"person.personID": -1}).limit(1)
```



Rysunek: Czas wyszukiwania w poszczególnych bazach

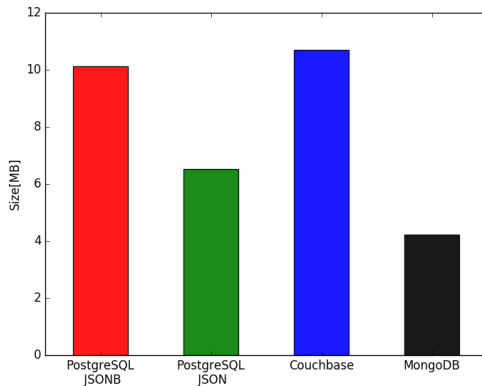
Porównanie - modyfikowanie danych

W tym przykładzie modyfikacja polega na dodaniu tablicy "email", wraz z jedną wartością do dokumentów, w których nie ma takiej tablicy.



Rysunek: Czas modyfikacji w poszczególnych bazach

Porównanie - rozmiar danych



Rysunek: Rozmiar danych w poszczególnych bazach

W przypadku wprowadzania danych zdecydowanym liderem okazała się baza MongoDB, która wyprzedza pozostałe rozwiązania. Potwierdziła się teza zgodnie, z którą json (PostgreSQL) jest szybszy we wprowadzaniu danych od jsonb (minimalnie), a wolniejszy w ich przetwarzaniu. Prawie we wszystkich testach najwolniej działał Couchbase, jednak prawie wszystkie testy zakładały użycie N1QL, a więc zmuszony byłem do korzystania z Couchbase Bucket (alternatywą jest Memcached Bucket, który nie wspiera N1QL). Typ jsonb z PostgreSQL i baza MongoDB działają ze zbliżoną prędkością jeśli chodzi o przeszukiwanie danych i ich grupowanie, a json jest trochę wolniejszy. W aspekcie aktualizacji i kasowania danych najwydajniej pracuje Postgres. Ostatnim przypadkiem testowym był pomiar rozmiaru zajętego na dysku. Najkorzystniej wypadła baza MongoDB.