Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

Отчет по лабораторной работе

3.2 «Использование документно-ориентированных

объектов типа Json»

по дисциплине

«Системы управления базами данных»

Выполнила

Студентка гр. 5130904/10101 Никифорова Е. А.

Руководитель Прокофьев О. В.

Санкт-Петербург

2024

Постановка задачи

PostgreSQL стала первой реляционной базой данных, поддерживающей слабоструктурированные данные. В PostgreSQL для этого используется JSON (JavaScript Object Notation, Запись объекта JavaScript RFC 7159), который имеет два представления: json и jsonb. Для реализации эффективного механизма запросов к этим типам данных в Postgres также имеется тип jsonpath. Официально JSON появился в PostgreSQL в 2014 году. PostgreSQL с JSONB совмещает гибкость NoSQL , а также надёжность и богатство функциональности реляционных СУБД.

В практической части необходимо:

* Cоздать БД IMDB test, использующую стандартные атрибуты и атрибут jsonb. Ссылка на интефейсы. Используется один файл **actors.list.txt** с я.диска из папки DataSet.
* Составить 3-4 запроса с использованием api postgresql (3 варианта) для jsonb.
* Измерить время доступа к полю jsonb для каждой строчки (в виде таблицы или графика). Оценить влияние **длины строки** на **скорость доступа** (ожидается ступенька до 2kB, после линейная зависимость). Для этого следует измерить время **чтения year первой роли** актера (explain analyze). Для точности оценки важно учитывать хранимую длину (jsonb хранится в сжатой форме). Как можно это влияние уменьшить?
* Составить запрос на **изменение year у первой роли** актера. Сравнить изменение объема БД для актера с малым кол-вом ролей и актера с большим количеством ролей (toasted roles).

Решение:

Преобразование actors.list.txt в .csv

import regex  
import json  
  
regx = r'(?<title>("?\w\*\w.+?["\}\!]?)|(\w\s\*)|(\.\.\.\s\*))\s+\((?<year>(\d+|\?+)(\/[IVX]\*)?)\)(\s+\((?<type1>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\((?<type2>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\((?<type3>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\((?<type4>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\((?<type5>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\((?<type6>(also )?(V|TV|VG|archive footage|uncredited|voice))\))?(\s+\{+(?<seriesname>.+?)\}+)?(\s+\((?<ascharacter>as.+)\))?(\s+\[(?<charactername>.+)\])?(\s+\<(?<credit>.+)\>)?'  
  
count = 0  
output = open("output.csv", "w")  
output.write("ActorName,RolesName\n")  
  
def add(match, role\_data: dict, group\_name: str, data\_name: str = None):  
 value = match.group(group\_name)  
 if value:  
 role\_data[data\_name if data\_name else group\_name] = value  
  
def parse\_role(role\_text: str) -> dict:  
 role\_data = {}  
 match = regex.search(regx, role\_text)  
 if not match:  
 global count  
 print(role\_text)  
 count += 1  
 return  
  
 headers = ["title", "year", "type1", "type2", "type3", "type4", "type5", "type6",  
 ("seriesname", "series name"), ("ascharacter", "as character"),  
 ("charactername", "character name"), "credit"]  
  
 for i in headers:  
 if type(i) is tuple:  
 add(match, role\_data, \*i)  
 else:  
 add(match, role\_data, i)  
 return role\_data  
  
def add\_row(ActorName: str, RolesName: dict):  
 global output  
 if len(ActorName) == 0:  
 return  
 ActorName = ActorName.replace("'", "''")  
 roles\_json = json.dumps(RolesName).replace("'", "''")  
 output.write(f"\'{ActorName}\',\'{roles\_json}\'\n")  
  
file\_path = 'actors.list.txt'  
actor\_name = ""  
roles = []  
i = 0  
with open(file\_path, 'r', encoding='cp1251') as file:  
 for line in file:  
 line = line.replace('\t\t', '\t').replace('\t\t', '\t').replace('\n', '')  
 if len(line) < 3:  
 continue  
 if line[0] == '\t':  
 roles.append(parse\_role(line[1:]))  
 else:  
 roles\_dict = {}  
 roles\_dict['roles'] = roles  
 add\_row(actor\_name, roles\_dict)  
  
 line = line.split('\t')  
 actor\_name = line[0]  
 roles = []  
 roles.append(parse\_role(line[1]))

Скрипт создания таблицы

CREATE TABLE actors (

id SERIAL PRIMARY KEY,

actorname VARCHAR(255) NOT NULL,

rolesname JSONB

);

Загрузка данных в таблицу через psql

Psql –U postgres –d lab4

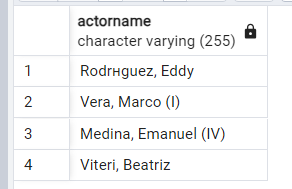
\copy actors(actorname, rolesname) FROM 'D:\DBM\3\_2\output.csv' CSV HEADER QUOTE E'\''

Запросы для jsonb

SELECT actorname

FROM actors

WHERE rolesname @> '{"roles": [{"title": "The gift"}]}';



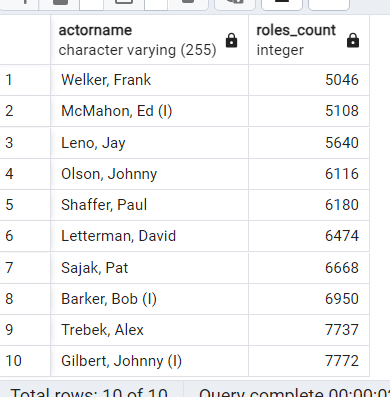
Найти актеров, у которых количество ролей превышает 5000, и вывести их имена и количество ролей в порядке возрастания количества ролей

SELECT actorname, jsonb\_array\_length(rolesname->'roles') as roles\_count

FROM actors

WHERE jsonb\_array\_length(rolesname->'roles') >5000

ORDER BY roles\_count ASC;

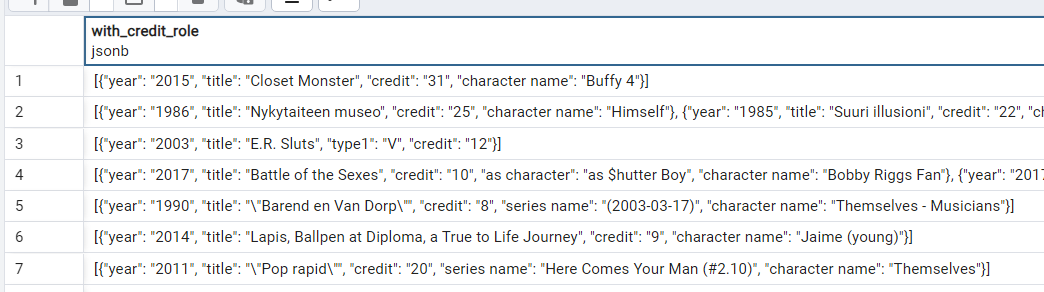


Получить все элементы JSON из массива ‘roles’, где у первого элемента имеется ключ ‘credit’

Select jsonb\_path\_query(rolesname, '$.roles') AS with\_credit\_role

FROM actors

WHERE jsonb\_path\_exists(rolesname->'roles'->0, '$.credit') = TRUE;



Измерение времени доступа к полю jsonb

Создание дополнительной таблицы

create table access\_time(time INTERVAL, row\_size INT);

Заполнение таблицы времени чтения year первой роли актера

DO $$

DECLARE

row RECORD;

start\_time TIMESTAMP;

end\_time TIMESTAMP;

access\_time INTERVAL;

size\_bytes INT;

foo TEXT;

BEGIN

FOR row IN SELECT rolesname, pg\_column\_size(rolesname) as size\_column FROM actors ORDER BY size\_column LOOP

start\_time := clock\_timestamp();

foo := row.rolesname->'roles'->0->>'year';

end\_time := clock\_timestamp();

access\_time := end\_time - start\_time;

size\_bytes := row.size\_column;

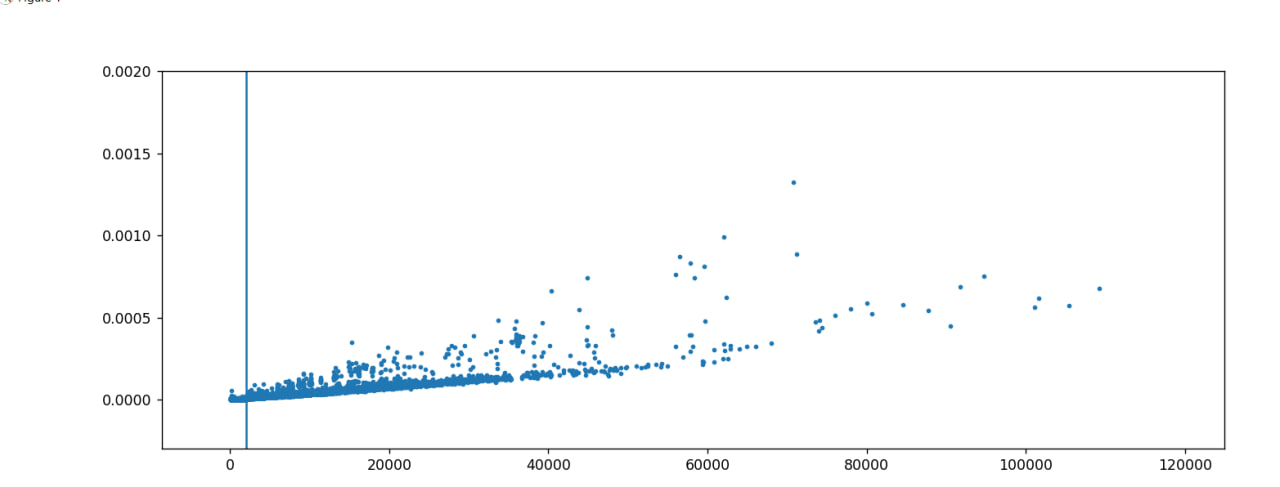
insert into access\_time (time, row\_size) values (access\_time, size\_bytes);

END LOOP;

END $$;

Построение графика зависимости времени доступа от размера jsonb

from psycopg2 import connect  
import matplotlib.pyplot as plt  
  
addr = "dbname=lab4 user=postgres password=root"  
conn = connect(addr)  
cur = conn.cursor()  
  
times = []  
row\_sizes = []  
cur.execute("SELECT time, row\_size FROM access\_time")  
i = 0  
prev\_row\_size = 0  
for time, row\_size in cur:  
 if row\_size > prev\_row\_size:  
 times.append(time.total\_seconds())  
 row\_sizes.append(row\_size)  
 else:  
 if i > 100:  
 times.append(time.total\_seconds())  
 row\_sizes.append(row\_size)  
 i = 0  
 i += 1  
 prev\_row\_size = row\_size  
  
plt.scatter(row\_sizes, times, s=5)  
plt.axvline(x=2048)  
plt.ylim(top=0.002)  
plt.xlim(right=125000)  
  
plt.show()



По графику можно заметить, что зависимость времени от размера линейная.

Сравнение изменение объема БД

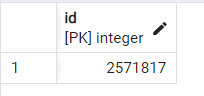
Получение id актера с минимальным количеством ролей

SELECT id

FROM actors

ORDER BY jsonb\_array\_length(rolesname->'roles') ASC

LIMIT 1;



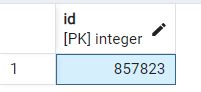
Получение id актера с максимальным количеством ролей

SELECT id

FROM actors

ORDER BY jsonb\_array\_length(rolesname->'roles') DESC

LIMIT 1;



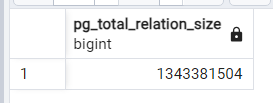
SELECT pg\_total\_relation\_size('actors');



UPDATE actors

SET rolesname = jsonb\_set(rolesname, '{roles,0,year}', '"2022"')

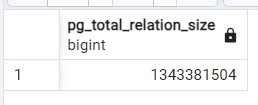
WHERE id = 857823;



UPDATE actors

SET rolesname = jsonb\_set(rolesname, '{roles,0,year}', '"2022"')

WHERE id = 2571817;



Таким образом, при изменении year у актера с маленьким количеством ролей размер не увеличивается, а при изменении у актера с большим количеством – увеличивается.

Это происходит, так как при обновлении какого-то ключа переписывается все: вся строка продублируется и появится новая версия.

Вывод:

В ходе лабораторной работы был изучен тип объекта jsonb, исследована зависимость времени доступа от размера jsonb