Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

Выполнил студент гр.3530904/90102 **Мэн Цзянин**

Руководитель **Прокофьев О.В.**

Санкт-Петербург

2022

1. Лабораторная работа No.3

1.1 Проектирование аналитической схемы базы данных

> http://www.postgres.cn/docs/14/plpython-funcs.html
>
> https://www.cnblogs.com/whitebai/p/12924270.html
>
> http://www.postgres.cn/docs/14/performance-tips.html
>
> https://juejin.cn/post/6960674004969455623

1.1.1 Постановка задачи

Седьмое практическое задание связано с проектированием схемы базы данных для аналитики. Будем исходить из того, что приложение, для которого была сделана база данных в задании стала очень популярной и по ней каждый день можно собирать большой объем статистической информации. Результатом данного практического задания являются: **скрипты создания базы данных, хранимая процедура (генератор) для ее заполнения**, **анализ плана выполнения запроса**.

Требования к БД

- Как минимум одна таблица должна содержать не меньше 10 млн. записей, которые со временем теряют актуальность.
- Другая таблица, связанная с первой, должна содержать не меньше 1 млн. записей.
- В одной из таблиц с количество записей больше 1 млн. должна быть колонка с текстом, по которой будет необходимо настроить полнотекстовый поиск.

Практическая часть включает:

1. наполнение таблицы, для этого нужно написать хранимую процедуру - генератор на языке plpython3u, которая использует словари (для строковых типов), случайные значения (для строковых, числовых типов).
2. оценку скорости выполнения запросов.
   Для этого могут быть использованы механизмы секционирования, наследования и индексов. Необходимо подготовить два запроса:

- Запрос к одной таблице, содержащий фильтрацию по нескольким полям.
- Запрос к нескольким связанным таблицам, содержащий фильтрацию по нескольким полям.

Для каждого из этих запросов необходимо провести следующие шаги:

- Получить план выполнения запроса без использования индексов (удаление индекса или отключение его использования в плане запроса).
- Получить статистику (IO и Time) выполнения запроса без использования индексов.
- Создать нужные индексы, позволяющие ускорить запрос.
- Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

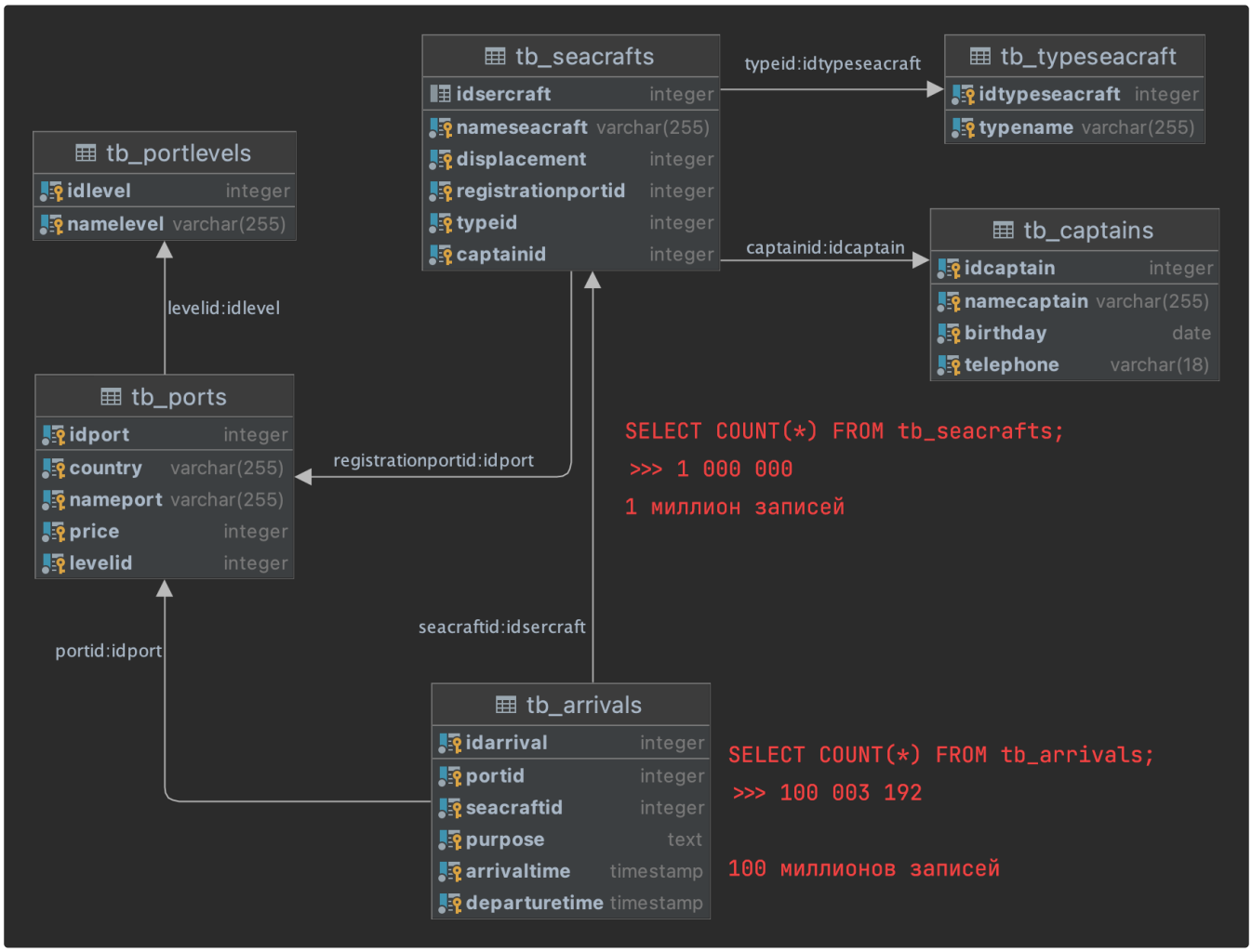
- Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.
- Оценить эффективность выполнения оптимизированного запроса.

Также необходимо продемонстрировать полезность индексов для организации полнотекстового поиска.

Для таблицы объемом 10 млн. записей произвести оптимизацию, позволяющую быстро удалять старые данные, ускорить вставку и чтение данных.

1.1.2 Реализация

1.1.2.1 ER диаграммы



1.1.2.2 Хранимая процедура (генератор)

Скрипты для генерации и вставки случайных записей:

Вставьте 100 000 000 случайных записей данных в tb_arrivals

```
CREATE EXTENSION plpython3u;

-------------------------------------------------------------------------------------------
-- Вставьте 100 000 000 случайных записей данных в tb_arrivals
CREATE OR REPLACE FUNCTION fc_InsertArrivals()
    RETURNS TEXT
    AS $$
import random
import datetime


COL_NUM = 100000000


def get_random_text():
    """
    Сгенерируйте случайный `текст`.

    :return: text -> string
    """
    text = ''
    num_word = random.randint(1, 20)
    for i in range(num_word):
        len_word = random.randint(1, 10)
        word = ''.join(random.sample('zyxwvutsrqponmlkjihgfedcba', len_word))
        text += (word + ' ')

    return """ + text + """


def get_random_start_end_time():
    """
    Получить строку со случайным начальным временем `start_time_str` и случайным конечным временем `end_time_str`,
    объединенными вместе

    :return: string -> start time and end time
    """
    start = '1900-01-01 00:00:00'
    end = '2022-04-15 00:00:00'
    frmt = '%Y-%m-%d %H:%M:%S'
    num_hour = random.randint(12, 7 * 24)
    stime = datetime.datetime.strptime(start, frmt)
    etime = datetime.datetime.strptime(end, frmt)
```

```
    time_datetime = random.random() * (etime - stime) + stime
    start_time_str = time_datetime.strftime(frmt)
    end_time_str = (time_datetime + datetime.timedelta(hours=num_hour)).strftime(frmt)

    return """ + start_time_str + "', '" + end_time_str + """


if __name__ == '__main__':
    try:
        num_port = plpy.execute("SELECT COUNT(*) FROM tb_ports")[0]['count']
        num_seacraft = plpy.execute("SELECT COUNT(*) FROM tb_seacrafts")[0]['count']
        # return str(num_port)

        for i in range(COL_NUM):
            portid = random.randint(1, num_port)
            seacraftid = random.randint(1, num_seacraft)
            purpose = get_random_text()
            arrival_leave_time = get_random_start_end_time()
            command = 'INSERT INTO tb_Arrivals (PortID, SeacraftID, Purpose, ArrivalTime, LeaveTime)
VALUES ('
            command = command + str(portid) + ', ' + str(seacraftid) + ', ' + purpose + ', ' + arrival_leave_time +
');'
            # return command
            plpy.execute(command)

    except plpy.SPIError as e:
        return "[ERROR] %s" % e.sqlstate

    else:
        return "[Info] Successful insert"

$$ LANGUAGE plpython3u;

-----------------------------------------------------------

TRUNCATE TABLE tb_arrivals CASCADE;
SELECT fc_InsertArrivals();  -- 运行时间 13040 s


SELECT COUNT(*) FROM tb_arrivals; -- 运行时间 26.195 s

```

Вставьте 1 000 000 случайных записей данных в tb_arrivals

```
1
2  ----------------------------------------------------------------------------------------------
3  -- 向 tb_seacraft 中插入 1000000 条随机数据记录
4  -- Вставьте 1 000 000 случайных записей данных в tb_arrivals
5  CREATE OR REPLACE FUNCTION fc_InsertSeacrafts()
6     RETURNS TEXT
7     AS $$
8  import random
9  import datetime
10
11
12 COL_NUM = 1000000
13
14
15 def get_random_name():
16    """
17    生成一段随机 `name`
18    Сгенерируйте случайный `name`.
19
20    :return: text -> string
21    """
22    len_word = random.randint(1, 10)
23    word = ''.join(random.sample('zyxwvutsrqponmlkjihgfedcba', len_word))
24
25    return word
26
27
28 if __name__ == '__main__':
29    try:
30       num_captain = plpy.execute("SELECT COUNT(*) FROM tb_captains")[0]['count']
31       num_port = plpy.execute("SELECT COUNT(*) FROM tb_ports")[0]['count']
32       num_type = plpy.execute("SELECT COUNT(*) FROM tb_portlevels")[0]['count']
33
34       for i in range(COL_NUM):
35          regportid = random.randint(1, num_port)
36          captainid = random.randint(1, num_captain)
37          typeid = random.randint(1, num_type)
38          displacement = random.randint(10000, 1000000)
39          nameseacraft = get_random_word()
40
41          command = 'INSERT INTO tb_seacraft(NameSeacraft, Displacement, RegPortID, TypeID,
   CaptainID) VALUES ('
42          command = command + str(nameseacraft) + ', ' + str(displacement) + ', ' + str(regportid) + ', ' +
   str(typeid) + str(captainid) + ');'
43          # return command
44          plpy.execute(command)
45
46    except plpy.SPIError as e:
47       return "[ERROR] %s" % e.sqlstate
48
```

```
49      else:
50          return "[Info] Successful insert"
51
52  $$ LANGUAGE plpython3u;
53
54  ------------------------------------------------------------
```

1.1.2.3 Анализ плана выполнения запроса

> http://www.postgres.cn/docs/14/performance-tips.html
>
> http://www.postgres.cn/docs/14/using-explain.html#USING-EXPLAIN-BASICS
>
> http://www.postgres.cn/docs/14/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS
>
> https://blog.csdn.net/kmblack1/article/details/80761647

Константы стоимости для планировщика

Константы	Описание	Значение по умолчанию (стоимость)
cpu_tuple_cost(floating point)	Задаёт приблизительную стоимость обработки каждой строки при выполнении запроса	0.01
cpu_index_tuple_cost(floating point)	Задаёт приблизительную стоимость обработки каждой записи индекса при сканировании индекса	0.005
cpu_operator_cost(floating point)	Задаёт приблизительную стоимость обработки оператора или функции при выполнении запроса	0.0025

[1] **Запрос к одной таблице, содержащий фильтрацию по нескольким полям.**

1. Получить план выполнения запроса **без использования индексов** (удаление индекса или отключение его использования в плане запроса).

```
1  SELECT * FROM tb_arrivals WHERE ArrivalTime BETWEEN '2000-01-01 00:00:00'::TIMESTAMP
   AND '2001-01-01 00:00:00' AND PortID > 1000;
2
3  DROP INDEX IX_ArriverTime;
4  DROP INDEX IX_PortID;
```

2. Получить статистику (IO и Time) выполнения запроса **без использования индексов**.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE ArrivalTime BETWEEN '2000-01-01 00:00:00'::TIMESTAMP AND '2001-01-01 00:00:00' AND PortID > 1000;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..2348941.17 rows=326215 width=97) (actual time=1.252..27359.813 rows=332165 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on tb_arrivals  (cost=0.00..2315319.67 rows=135923 width=97) (actual time=0.427..27321.096 rows=110722 loops=3)
         Filter: ((arrivaltime >= '2000-01-01 00:00:00'::timestamp without time zone) AND (arrivaltime <= '2001-01-01 00:00:00'::timestamp without time zone) AND (portid > 1000))
         Rows Removed by Filter: 33222612
 Planning Time: 3.485 ms
 Execution Time: 27368.246 ms
(8 rows)
```

3. Создать нужные индексы, позволяющие ускорить запрос.

```
CREATE INDEX IX_ArriverTime ON tb_arrivals(ArrivalTime);
CREATE INDEX IX_PortID ON tb_arrivals(PortID);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE ArrivalTime BETWEEN '2000-01-01 00:00:00'::TIMESTAMP AND '2001-01-01 00:00:00' AND PortID > 1000;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=459271.16..1940524.67 rows=326215 width=97) (actual time=557.093..23659.834 rows=332165 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Bitmap Heap Scan on tb_arrivals  (cost=458271.16..1906903.17 rows=135923 width=97) (actual time=537.187..23605.583 rows=110722 loops=3)
         Recheck Cond: ((arrivaltime >= '2000-01-01 00:00:00'::timestamp without time zone) AND (arrivaltime <= '2001-01-01 00:00:00'::timestamp without time zone) AND (portid > 1000))
         Rows Removed by Index Recheck: 12492294
         Heap Blocks: exact=14079 lossy=199261
         ->  BitmapAnd  (cost=458271.16..458271.16 rows=326215 width=0) (actual time=551.432..551.432 rows=0 loops=1)
               ->  Bitmap Index Scan on ix_arrivertime  (cost=0.00..16861.76 rows=804919 width=0) (actual time=96.028..96.028 rows=818208 loops=1)
                     Index Cond: ((arrivaltime >= '2000-01-01 00:00:00'::timestamp without time zone) AND (arrivaltime <= '2001-01-01 00:00:00'::timestamp without time zone))
               ->  Bitmap Index Scan on ix_portid  (cost=0.00..441246.04 rows=40527663 width=0) (actual time=448.839..448.839 rows=40516196 loops=1)
                     Index Cond: (portid > 1000)
```

```
 Planning Time: 0.142 ms
 Execution Time: 23671.716 ms
(14 rows)
```

4. Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.

	cost	actual time	Planning Time	Execution Time
без использования индексов	1000.00..2348941.17	1.252..27359.813	3.485 ms	27368.246 ms
с использованием индексов	459271.16..1940524.67	557.093..23659.834	0.142 ms	23671.716 ms
Разница	-866687.6599	-4255.82	-3.343 ms	-3696.5299 ms

5. Оценить эффективность выполнения оптимизированного запроса.

   После использования индекса приблизительная стоимость запуска(cost) значительно увеличилась, однако приблизительная общая стоимость (cost) и фактическое время (Execution Time) значительно уменьшились. Фактическое время (Execution Time) сократилось на 13,506%

[2] Запрос к нескольким связанным таблицам, содержащий фильтрацию по нескольким полям

1. Получить план выполнения запроса **без использования индексов** (удаление индекса или отключение его использования в плане запроса).

```
SELECT * FROM tb_arrivals INNER JOIN tb_seacrafts ON
tb_arrivals.seacraftID=tb_seacrafts.IDSeacraft INNER JOIN tb_ports ON
tb_arrivals.portID=tb_ports.IDPort WHERE NameSeacraft='cmari' AND NamePort='St Petersburg';

DROP INDEX IX_NameSeacraft;
DROP INDEX IX_NamePort;
```

2. Получить статистику (IO и Time) выполнения запроса **без использования индексов**.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals INNER JOIN tb_seacrafts ON
tb_arrivals.seacraftID=tb_seacrafts.IDSeacraft INNER JOIN tb_ports ON
tb_arrivals.portID=tb_ports.IDPort WHERE NameSeacraft='cmari' AND NamePort='St Petersburg';
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=1008.31..2140204.13 rows=1 width=161) (actual time=698.274..29483.075 rows=6 loops=1)
   Join Filter: (tb_arrivals.seacraftid = tb_seacrafts.idseacraft)
   Rows Removed by Join Filter: 59539
   ->  Seq Scan on tb_seacrafts  (cost=0.00..20033.00 rows=1 width=29) (actual time=0.177..55.336 rows=1 loops=1)
         Filter: ((nameseacraft)::text = 'cmari'::text)
         Rows Removed by Filter: 999999
   ->  Gather  (cost=1008.31..2119427.53 rows=59488 width=132) (actual time=4.875..29424.577 rows=59545 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Hash Join  (cost=8.31..2112478.73 rows=24787 width=132) (actual time=6.244..29400.751 rows=19848 loops=3)
               Hash Cond: (tb_arrivals.portid = tb_ports.idport)
               ->  Parallel Seq Scan on tb_arrivals  (cost=0.00..2002819.67 rows=41666667 width=97) (actual time=0.345..27659.885 rows=33333333 loops=3)
               ->  Hash  (cost=8.29..8.29 rows=1 width=35) (actual time=0.165..0.165 rows=1 loops=3)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Index Scan using uq_ports_nameport on tb_ports  (cost=0.28..8.29 rows=1 width=35) (actual time=0.118..0.119 rows=1 loops=3)
                           Index Cond: ((nameport)::text = 'St Petersburg'::text)
 Planning Time: 7.004 ms
 Execution Time: 29483.181 ms
(18 rows)
```

3. Создать нужные индексы, позволяющие ускорить запрос.

```
CREATE INDEX IX_NameSeacraft ON tb_seacrafts(NameSeacraft);
CREATE INDEX IX_NamePort ON tb_ports(NamePort);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals INNER JOIN tb_seacrafts ON
tb_arrivals.seacraftID=tb_seacrafts.IDSeacraft INNER JOIN tb_ports ON
tb_arrivals.portID=tb_ports.IDPort WHERE NameSeacraft='cmari' AND NamePort='St Petersburg';
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1016.76..2113212.11 rows=1 width=161) (actual time=9437.606..29046.762 rows=6
loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Hash Join  (cost=16.76..2112212.01 rows=1 width=161) (actual time=6601.278..29032.137 rows=2
loops=3)
         Hash Cond: (tb_arrivals.portid = tb_ports.idport)
         ->  Hash Join  (cost=8.46..2112203.59 rows=42 width=126) (actual time=6.039..29030.866
rows=3333 loops=3)
               Hash Cond: (tb_arrivals.seacraftid = tb_seacrafts.idseacraft)
               ->  Parallel Seq Scan on tb_arrivals  (cost=0.00..2002819.67 rows=41666667 width=97) (actual
time=0.461..27245.982 rows=33333333 loops=3)
               ->  Hash  (cost=8.44..8.44 rows=1 width=29) (actual time=0.353..0.356 rows=1 loops=3)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Index Scan using ix_nameseacraft on tb_seacrafts  (cost=0.42..8.44 rows=1 width=29)
(actual time=0.349..0.350 rows=1 loops=3)
                           Index Cond: ((nameseacraft)::text = 'cmari'::text)
         ->  Hash  (cost=8.29..8.29 rows=1 width=35) (actual time=0.222..0.223 rows=1 loops=3)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Index Scan using ix_nameport on tb_ports  (cost=0.28..8.29 rows=1 width=35) (actual
time=0.215..0.215 rows=1 loops=3)
                     Index Cond: ((nameport)::text = 'St Petersburg'::text)
 Planning Time: 3.746 ms
 Execution Time: 29047.011 ms
(18 rows)
```

4. Создать нужные индексы, позволяющие ускорить запрос.

```
CREATE EXTENSION pg_trgm;
CREATE INDEX IX_NameSeacraft ON tb_seacrafts USING GIN(NameSeacraft gin_trgm_ops);
CREATE INDEX IX_NamePort ON tb_ports USING GIN(NamePort gin_trgm_ops);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals INNER JOIN tb_seacrafts ON
tb_arrivals.seacraftID=tb_seacrafts.IDSeacraft INNER JOIN tb_ports ON
tb_arrivals.portID=tb_ports.IDPort WHERE NameSeacraft='cmari' AND NamePort='St Petersburg';
```

```
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1088.34..2321635.00 rows=1 width=161) (actual time=11391.572..35022.070 rows=6 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Hash Join  (cost=88.34..2320634.90 rows=1 width=161) (actual time=12996.658..35002.537 rows=2 loops=3)
         Hash Cond: (tb_arrivals.portid = tb_ports.idport)
         ->  Hash Join  (cost=80.03..2320626.48 rows=42 width=126) (actual time=10.302..34999.074 rows=3333 loops=3)
               Hash Cond: (tb_arrivals.seacraftid = tb_seacrafts.idseacraft)
               ->  Parallel Append  (cost=0.00..2211170.01 rows=41667034 width=97) (actual time=1.530..32921.062 rows=33333333 loops=3)
                     ->  Parallel Seq Scan on tb_arrivals tb_arrivals_1  (cost=0.00..2002819.67 rows=41666667 width=97) (actual time=1.517..31469.568 rows=33333333 loops=3)
                     ->  Parallel Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..15.18 rows=518 width=60) (actual time=0.001..0.001 rows=0 loops=1)
               ->  Hash  (cost=80.02..80.02 rows=1 width=29) (actual time=1.348..1.349 rows=1 loops=3)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Bitmap Heap Scan on tb_seacrafts  (cost=76.01..80.02 rows=1 width=29) (actual time=1.346..1.347 rows=1 loops=3)
                           Recheck Cond: ((nameseacraft)::text = 'cmari'::text)
                           Heap Blocks: exact=1
                           ->  Bitmap Index Scan on ix_nameseacraft  (cost=0.00..76.01 rows=1 width=0) (actual time=1.339..1.339 rows=1 loops=3)
                                 Index Cond: ((nameseacraft)::text = 'cmari'::text)
         ->  Hash  (cost=8.29..8.29 rows=1 width=35) (actual time=0.151..0.152 rows=1 loops=3)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Index Scan using uq_ports_nameport on tb_ports  (cost=0.28..8.29 rows=1 width=35) (actual time=0.083..0.084 rows=1 loops=3)
                     Index Cond: ((nameport)::text = 'St Petersburg'::text)
 Planning Time: 1.025 ms
 Execution Time: 35022.279 ms
(23 rows)
```

5. Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.

	cost	actual time	Planning Time	Execution Time	Разница Execution Time с неиспользуемыми индексами
без использования индексов	1008.31..2140204.13	698.274..29483.075	7.004 ms	29483.181 ms	
с использованием индексов	1016.76..2113212.11	9437.606..29046.762	3.746 ms	29047.011 ms	-436.17 ms
с использованием индексов (GIN)	1088.34..2321635.00	11180.878..35380.249	1.025	35022.279 ms	+5539.09 ms

6. Оценить эффективность выполнения оптимизированного запроса.

При добавлении индексов к двум полям varchar подтаблицы значительного улучшения **в скорости запросов не наблюдается.** GIN-индексирование в этом случае вместо того, чтобы ускорить запрос, делает его медленнее.

[3] **Также необходимо продемонстрировать полезность индексов для организации полнотекстового поиска.**

1. Получить план выполнения запроса **без использования индексов** (удаление индекса или отключение его использования в плане запроса).

```
SELECT * FROM tb_arrivals WHERE Purpose LIKE '%at%';

DROP INDEX IX_Purpose;
```

2. Получить статистику (IO и Time) выполнения запроса **без использования индексов**.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE Purpose LIKE '% at %';
                                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..2108981.63 rows=9953 width=97) (actual time=2.708..35207.244 rows=145614 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on tb_arrivals  (cost=0.00..2106986.33 rows=4147 width=97) (actual time=2.798..35181.150 rows=48538 loops=3)
         Filter: (purpose ~~ '% at %'::text)
         Rows Removed by Filter: 33284795
 Planning Time: 0.337 ms
 Execution Time: 35212.464 ms
(8 rows)
```

3. Создать нужные индексы, позволяющие ускорить запрос.

```
CREATE INDEX IX_Purpose ON tb_arrivals(Purpose);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE Purpose LIKE '% at %';
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..2108981.63 rows=9953 width=97) (actual time=3.036..36736.283 rows=145614 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on tb_arrivals  (cost=0.00..2106986.33 rows=4147 width=97) (actual time=0.982..36713.617 rows=48538 loops=3)
         Filter: (purpose ~~ '% at %'::text)
         Rows Removed by Filter: 33284795
 Planning Time: 5.599 ms
 Execution Time: 36741.591 ms
(8 rows)
```

4. Создать нужные индексы (GIN), позволяющие ускорить запрос.

```
CREATE INDEX IX_Purpose ON tb_arrivals USING GIN(Purpose gin_trgm_ops);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE Purpose LIKE '% at %';
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=1277.14..38806.13 rows=9960 width=97) (actual time=711.662..21840.648 rows=145614 loops=1)
   ->  Bitmap Heap Scan on tb_arrivals tb_arrivals_1  (cost=1277.14..38735.33 rows=9953 width=97) (actual time=711.661..21829.443 rows=145614 loops=1)
         Recheck Cond: (purpose ~~ '% at %'::text)
         Rows Removed by Index Recheck: 8520120
         Heap Blocks: exact=33655 lossy=137020
         ->  Bitmap Index Scan on ix_purpose  (cost=0.00..1274.65 rows=9953 width=0) (actual time=706.322..706.322 rows=180530 loops=1)
               Index Cond: (purpose ~~ '% at %'::text)
   ->  Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..21.00 rows=7 width=60) (actual time=0.062..0.062 rows=0 loops=1)
         Filter: (purpose ~~ '% at %'::text)
 Planning Time: 27.951 ms
 Execution Time: 21846.879 ms
(11 rows)
```

5. Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.

	cost	actual time	Planning Time	Execution Time	Разница Execution Time с неиспользуемыми индексами
без использования индексов	1000.00..2108981.63	2.708..35207.244	0.337 ms	35212.464 ms	
с использованием индексов	1000.00..2108981.63	3.032..36313.774	4.645 ms	36319.114 ms	+1106.65 ms
с использованием индексов(GIN)	1277.14..38806.13	711.662..21840.648	27.951 ms	21846.879 ms	-13365.585 ms

6. Оценить эффективность выполнения оптимизированного запроса.

   Несмотря на то, что для текстового поля создан индекс, база данных **не использует его для запросов**.

   При использовании индекса GIN скорость выполнения запросов значительно выше.

[4] **Также необходимо продемонстрировать полезность индексов для организации полнотекстового поиска.**

1. Получить план выполнения запроса **без использования индексов** (удаление индекса или отключение его использования в плане запроса).

```
SELECT * FROM tb_arrivals WHERE char_length(Purpose) > 20;

DROP INDEX IX_Purpose;
```

2. Получить статистику (IO и Time) выполнения запроса **без использования индексов**.

```
db_port=*# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE char_length(Purpose) > 20;
                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..3252844.33 rows=33333626 width=97) (actual time=0.522..22543.607 rows=86235207 loops=1)
   ->  Seq Scan on tb_arrivals tb_arrivals_1  (cost=0.00..3086153.00 rows=33333333 width=97) (actual time=0.521..19450.968 rows=86235207 loops=1)
         Filter: (char_length(purpose) > 20)
         Rows Removed by Filter: 13764793
   ->  Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..23.20 rows=293 width=60) (actual time=0.026..0.026 rows=0 loops=1)
         Filter: (char_length(purpose) > 20)
 Planning Time: 2.516 ms
 Execution Time: 24233.586 ms
(8 rows)
```

3. Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
CREATE INDEX IX_Purpose ON tb_arrivals USING GIN(Purpose gin_trgm_ops);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE char_length(Purpose) > 20;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..3252844.33 rows=33333626 width=97) (actual time=0.514..23314.352 rows=86235207 loops=1)
   ->  Seq Scan on tb_arrivals tb_arrivals_1  (cost=0.00..3086153.00 rows=33333333 width=97) (actual time=0.513..20128.237 rows=86235207 loops=1)
         Filter: (char_length(purpose) > 20)
         Rows Removed by Filter: 13764793
   ->  Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..23.20 rows=293 width=60) (actual time=0.042..0.042 rows=0 loops=1)
         Filter: (char_length(purpose) > 20)
 Planning Time: 0.602 ms
 Execution Time: 25058.894 ms
(8 rows)
```

4. Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.

	cost	actual time	Planning Time	Execution Time	Разница Execution Time с неиспользуемыми индексами
без использования индексов	0.00..3252844.33	0.522..22543.607	2.516 ms	24233.586 ms	
с использованием индексов(GIN)	0.00..3252844.33	0.514..23314.352	0.602 ms	25058.894 ms	+825.308 ms

5. Оценить эффективность выполнения оптимизированного запроса.

   Из полученных результатов видно, что в данном случае индекс GIN не действует (не используется)

[5] Также необходимо продемонстрировать полезность индексов для организации полнотекстового поиска.

1. Получить план выполнения запроса **без использования индексов** (удаление индекса или отключение его использования в плане запроса).

```
SELECT * FROM tb_arrivals WHERE Purpose='sfrht';

DROP INDEX IX_Purpose;
```

2. Получить статистику (IO и Time) выполнения запроса **без использования индексов**.

```
db_port=*# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE Purpose='sfrht';
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..2108005.45 rows=26 width=91) (actual time=30218.895..30220.101 rows=0 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Append  (cost=0.00..2107002.85 rows=10 width=91) (actual time=30192.606..30192.609 rows=0 loops=3)
         ->  Parallel Seq Scan on tb_arrivals tb_arrivals_1  (cost=0.00..2106986.33 rows=9 width=97) (actual time=30192.604..30192.604 rows=0 loops=3)
               Filter: (purpose = 'sfrht'::text)
               Rows Removed by Filter: 33333333
         ->  Parallel Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..16.47 rows=2 width=60) (actual time=0.002..0.002 rows=0 loops=1)
               Filter: (purpose = 'sfrht'::text)
 Planning Time: 1.095 ms
 Execution Time: 30220.227 ms
(11 rows)
```

3. Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
CREATE INDEX IX_Purpose ON tb_arrivals USING GIN(Purpose gin_trgm_ops);
```

Получить план выполнения запроса с использованием индексов и сравнить с первоначальным планом.

```
db_port=# EXPLAIN ANALYZE SELECT * FROM tb_arrivals WHERE Purpose='sfrht';
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=2396.17..2505.33 rows=26 width=91) (actual time=657.419..657.419 rows=0 loops=1)
   ->  Bitmap Heap Scan on tb_arrivals tb_arrivals_1  (cost=2396.17..2484.20 rows=22 width=97) (actual time=657.409..657.409 rows=0 loops=1)
```

```
6         Recheck Cond: (purpose = 'sfrht'::text)
7         Rows Removed by Index Recheck: 23
8         Heap Blocks: exact=23
9         ->  Bitmap Index Scan on ix_purpose  (cost=0.00..2396.17 rows=22 width=0) (actual
  time=642.080..642.080 rows=23 loops=1)
10              Index Cond: (purpose = 'sfrht'::text)
11    ->  Seq Scan on tb_child_arrivals tb_arrivals_2  (cost=0.00..21.00 rows=4 width=60) (actual
  time=0.008..0.008 rows=0 loops=1)
12        Filter: (purpose = 'sfrht'::text)
13   Planning Time: 14.311 ms
14   Execution Time: 657.908 ms
15  (11 rows)
```

4. Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.

	cost	actual time	Planning Time	Execution Time	Разница Execution Time с неиспользуемыми индексами
без использования индексов	1000.00..2108005.45	30218.895..30220.101	1.095 ms	30220.227 ms	
с использованием индексов(GIN)	2396.17..2505.33	657.419..657.419	14.311 ms	657.908 ms	-29562.319 ms

5. Оценить эффективность выполнения оптимизированного запроса.

   В этом случае индекс GIN очень полезен, и скорость выполнения запросов значительно ускоряется

1.2 Использование документно-ориентированных объектов типа Json

1.2.1 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:

- создать БД IMDB test, использующую стандартные атрибуты и атрибут jsonb. Ссылка на интефейсы. Таблицы находятся на я.диске в папке DataSet. Описание атрибута jsonb:

```
{
    "nconst": "nm0000151",
    "primaryName": “Morgan Freeman”,
    "roles": [
        {
            "title": "The Shawshank Redemption",
            "year": "1994",
            "character name": "Ellis Boyd 'Red' Redding"
        },
        {
            "title": "Unforgiven",
            "year": "1992",
            "character name": "Ned Logan"
        }
        {
            "title": "Through the Wormhole",
            "series name": "Are Aliens Inside Us? (#6.5)"
            "year": "2010",
            "character name": "Himself - Narrator"
        }
    ],
    "birthYear": “1937”,
    "deathYear": “\N”,
}
```

где nconst, birthYear, deathYear это записи таблицы *name.basics.tsv.gz,* roles загружать из таблицы

- Составить 3-4 запроса с использованием jsonb.
- Измерить время доступа к ключу для каждой строчки (в виде таблицы или графика). Оценить влияние длины строки на скорость доступа (линейная зависимость). Как можно это влияние уменьшить.
- Составить запрос на изменение PrimaryName у актера. Сравнить изменение объема БД для актера с малым кол-вом ролей и актера с большим количеством ролей (toasted roles).

1.2.2 Реализация

> http://www.postgres.cn/docs/14/datatype-json.html
>
> https://juejin.cn/post/6844903857009623048
>
> https://segmentfault.com/a/1190000019344353
>
> https://programmers.buzz/posts/start-exploring-database-indices/
>
> https://habr.com/ru/company/oleg-bunin/blog/646987/

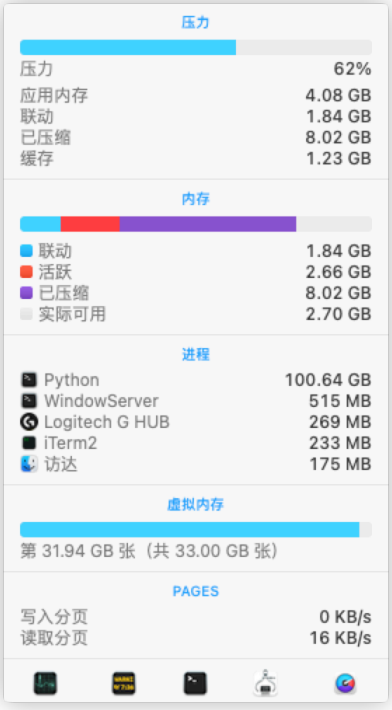
1.2.2.1 IMDB-JSONB

Были использованы все три набора данных: actors.list.txt , actresses.list.txt , name.basics.tsv .

Скрипты, используемые для обработки данных:

1. Из-за того, что одновременная обработка всех данных может занимать много памяти (используя вложенность DataFrame от pandas в качестве структуры данных). Поэтому используйте скрипт Python для разделения набора данных на более мелкие части и их последующего объединения

   Объединенный результат сохраняется в виде двоичного файла с использованием сериализации Python для уменьшения использования памяти и жесткого диска, а также для облегчения последующей обработки.



   В этом скрипте можно объединить все фильмы одного актера в один

```
# ------*------ coding: utf-8 ------*------
# @Time    : 2022/4/26 16:39
# @Author  : 冰糖雪狸 (NekoSilverfox)
# @Project : IMDB 数据处理
```

```
# @File    : 切割提取.py
# @Software: PyCharm
# @Github  :  https://github.com/NekoSilverFox
# -----------------------------------------
import pandas as pd
import numpy as np
import datetime
import pickle
import re


def normal_actor(source_data: pd.DataFrame, dump_name_title: str, dump_df_res: str) -> pd.DataFrame:
    """
    Нормализовать данные об актерах IMDB в pd.DataFrame, с отдельной строкой для каждого актера и их индивидуальных кредитов (аналогично перекрестной таблице)
    :param file_path:
    :return: pd.DataFrame
    """

    """
    Извлеките название, дату выпуска и название серии с помощью регулярных выражений и используйте их в качестве нового DataFrame
    Результатом является DataFrame с MutIndex
    """
    print('>>' * 50)
    print('[INFO] Начните извлекать информацию из строки')
    time_start = datetime.datetime.now()
    list_name_title = []
    i = 1
    name_list = []
    for col in source_data.values:
        if col[0] is not np.nan:
            names = col[0]
            name_list = names.split(', ')

        if len(name_list) == 0:
            continue

        for this_name in name_list:

            title_mix = col[1]

            # Регулярные выражения используются для извлечения
            """title - Название фильма Первая часть строки """
            title = re.search(r'^[^\(\{\[]*', title_mix)
            if title is not None:
                title = str(title.group()[:-1])

            """year - Год выпуска ()"""
            year = re.search(r'(?!=\({1})[\d]{4}(?!=\){1})', title_mix)
            if year is not None:
                year = int(year.group())
```

```
55
56          """series name - Название серии：  {}"""
57          series_name = re.search(r'\{(.*?)\}', title_mix)
58          if series_name is not None:
59              series_name = str(series_name.group()[1:-1])
60
61          """character name - Имя персонажа"""
62          character_name = re.search(r'\[(.*?)\]', title_mix)
63          if character_name is not None:
64              character_name = str(character_name.group()[1:-1])
65          # name_title.append([this_name, title, series_name, year, character_name])
66          # name_title.append([this_name, [title, series_name, year, character_name]])
67          rols = pd.DataFrame([[title, series_name, year, character_name]],
68                              columns=['title', 'series name', 'year', 'character name'])
69
70          list_name_title.append([this_name, rols])  # this_name 是 str, rols 是DataFrame
71
72      if i % 10000 == 0:
73          use_sec = (datetime.datetime.now() - time_start).seconds
74          print('[INFO] Обработано ', i, ' строк | ', (i / source_data.shape[0]) * 100, '% | Затраченное
    время：', use_sec, ' s（', use_sec / 60,
75                ' ）min')
76      i += 1
77
78  use_sec = (datetime.datetime.now() - time_start).seconds
79  print('[INFO] Окончание извлечения данных, время：', use_sec, ' s（', use_sec / 60, ' ）min')
80
81  """Получить результат df_name_title в виде DataFrame с MutIndex"""
82  print('>>' * 50)
83  print('[INFO] Получить результат df_name_title в виде DataFrame с MutIndex')
84  time_start = datetime.datetime.now()
85  df_name_title = pd.DataFrame(list_name_title, columns=['name', 'rols'])
86  list_name_title = []
87  df_name_title.sort_values(by='name', inplace=True)
88  df_name_title.reset_index(drop=True, inplace=True)
89  use_sec = (datetime.datetime.now() - time_start).seconds
90  print('[INFO] DataFrame Конец преобразования, во времени：', use_sec, ' s（', use_sec / 60, ' ）
    min')
91
92  # print('>>' * 50)
93  # print('[INFO] 开始序列化（备份）df_name_title')
94  # time_start = datetime.datetime.now()
95  # f = open(dump_name_title, 'wb')
96  # pickle.dump(obj=df_name_title, file=f)
97  # f.close()
98  # use_sec = (datetime.datetime.now() - time_start).seconds
99  # print('[INFO] 序列化（备份）结束，用时：', use_sec, ' s（', use_sec / 60, ' ）min')
100
101 """Объединить дубликаты имен, чтобы сделать их уникальными. добавить информацию об
    одном и том же актере в rols"""
102 print('>>' * 50)
103 print('[INFO] Начните объединять дубликаты имен')
104 time_start = datetime.datetime.now()
```

```
105     name = None
106     tmp_df_rols = []
107     res_list_name_rols = []
108     for i in range(df_name_title.shape[0]):
109 
110         """Если это один и тот же человек, работы объединяются в DataFrame"""
111         if name == df_name_title.loc[i]['name']:
112             tmp_df_rols = pd.concat([tmp_df_rols, df_name_title.loc[i]['rols']])
113 
114         else:
115             """Приступая к следующему человеку
116                 сначала записать информацию о предыдущем человеке в новый список, затем сбросить имя и tmp_df_rols на содержимое текущего ряда
117             """
118             if name is not None:
119                 res_list_name_rols.append([name, tmp_df_rols])
120 
121                 # if name == '$haniqua':
122                 #     print(tmp_df_rols)
123 
124             name = df_name_title.loc[i]['name']
125             tmp_df_rols = df_name_title.loc[i]['rols']
126 
127         if i % 10000 == 0:
128             use_sec = (datetime.datetime.now() - time_start).seconds
129             print('[INFO] Обработано ', i, ' строка | ', (i / df_name_title.shape[0]) * 100, '% | Затраченное время：', use_sec, ' s（',
130                   use_sec / 60, '）min')
131 
132     res_list_name_rols = pd.DataFrame(data=res_list_name_rols, columns=['name', 'rols'])
133 
134     # print('>>' * 50)
135     # print('[INFO] 开始序列化（备份）res_list_name_rols')
136     # time_start = datetime.datetime.now()
137     # f = open(dump_df_res, 'wb')
138     # pickle.dump(obj=res_list_name_rols, file=f)
139     # f.close()
140     # use_sec = (datetime.datetime.now() - time_start).seconds
141     # print('[INFO] 序列化（备份）res_list_name_rols 结束，用时：', use_sec, ' s（', use_sec / 60, '）min')
142 
143     return res_list_name_rols
144 
145 
146 if __name__ == '__main__':
147     print('>>' * 50)
148     print('[INFO] Начало реализации')
149 
150 
    ##########################################################################################################################
151     # 演员信息表
152     print('>>' * 50)
```

```
153     print('[INFO] Начните читать `name.basics.tsv`')
154     time_start = datetime.datetime.now()
155     df_name_info = pd.read_csv(
156         filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt'
157                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 семестр/СУБД/资
    料/DataSet/name.basics.tsv',
158         header=0,
159         sep='\t'
160     )
161     df_name_info = df_name_info.iloc[:, :-1]
162     df_name_info.columns = ['nconst', 'name', 'birthYear', 'deathYear', 'profession']
163     time_end = datetime.datetime.now()
164     print('[INFO] Конец чтения `name.basics.tsv`, время：  ', (time_end - time_start).seconds, ' s')
165
     ##################################################################################
    ##################################
166
167
     ##################################################################################
    ##################################
168     # Обработка отсутствующих значений как None для простого преобразования в JSON
169     print('>>' * 50)
170     print('[INFO] Начните обрабатывать отсутствующие значения как None и удалите
    дублирующиеся значения для легкого преобразования в JSON')
171     time_start = datetime.datetime.now()
172     df_name_info.replace(to_replace=['\\N', np.nan], value=None, inplace=True)
173     df_name_info.drop_duplicates(subset='name', keep='first', inplace=True)
174     time_end = datetime.datetime.now()
175     print('[INFO] Окончание обработки отсутствующих значений и удаления дублирующих
    значений, время：  ', (time_end - time_start).seconds, ' s')
176
     ##################################################################################
    ##################################
177
178
179
     ##################################################################################
    ##################################
180     print('>>' * 50)
181     print('[INFO] Начните чтение файла', '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt'
182                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 '
183                            'семестр/СУБД/资料/DataSet/data_actors.list.txt')
184     time_start = datetime.datetime.now()
185     # source_data = pd.read_csv(
186     #     filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-
    PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
187     #                'курс/6 семестр/СУБД/资料/DataSet/data_actresses.list.txt',
188     #     header=0,
189     #     sep='\t'
190     # )
191
192     # Список мужчин-актеров DataFrame
193     source_data = pd.read_csv(
```

```
194         filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-
    PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
195                 'курс/6 семестр/СУБД/资料/DataSet/data_actors.list.txt',
196         header=0,
197         sep='\t'
198     )
199     time_end = datetime.datetime.now()
200     print('[INFO] Конец чтения файла, время：', (time_end - time_start).seconds, ' s')
201
202
203
204     print('>>' * 50)
205     print('[INFO] Начните предварительную обработку данных')
206     time_start = datetime.datetime.now()
207     source_data.columns = ['name', 't1', 't2', 't3']
208     source_data['t1'].fillna(value='', inplace=True)
209     source_data['t2'].fillna(value='', inplace=True)
210     source_data['t3'].fillna(value='', inplace=True)
211
212     movie_list = source_data['t1'] + source_data['t2'] + source_data['t3']
213     source_data = pd.concat([source_data['name'], movie_list], axis=1)
214     source_data.columns = ['name', 'title_mix']
215
216     # source_data = source_data.iloc[:1000, :]
217
218     time_end = datetime.datetime.now()
219     print('[INFO] Окончание предварительной обработки данных, время：', (time_end -
    time_start).seconds, ' s')
220
221
222     start_index = 0
223     step_index = 1000000
224     end_index = start_index + step_index
225     times = 31
226
227
228
229     for i in range(1, times):
230         print('\n\n')
231         print('>>' * 20, ' Запустите цикл ', i, ' <<' * 20)
232
233         tmp_source_data = source_data.iloc[start_index:end_index, :]
234
235
     ##########################################################################################
    ##################################
236         # # Список актрис df
237         # print('[INFO]  main -> Начать обработку `data_actresses.list.txt`')
238         # df_actresses = normal_actor(
239         #     source_data=tmp_source_data,
240         #     dump_name_title='./result/dump_df_actresses_name_title.bits',
241         #     dump_df_res='./result/dump_df_actresses.bits'
242         # )
```

```
243         # tmp_source_data = None
244         # print('[INFO] main <- Окончание обработки `data_actresses.list.txt`')
245
 ##################################################################################################################################
246
247
248
 ##################################################################################################################################
249         # # 男演员列表df
250         # print('[INFO] Start handle `data_actors.list.txt`')
251         # df_actors = normal_actor(
252         #     file_path='/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
253         #               'курс/6 семестр/СУБД/资料/DataSet/data_actors.list.txt'
254         # )
255         print('[INFO]  main -> Начать обработку `data_actors.list.txt`')
256         df_actresses = normal_actor(
257             source_data=tmp_source_data,
258             dump_name_title='./result/dump_df_actors_name_title.bits',
259             dump_df_res='./result/dump_df_actors.bits'
260         )
261         tmp_source_data = None
262         print('[INFO] main <- Окончание обработки `data_actresses.list.txt`')
263
 ##################################################################################################################################
264
265
 ##################################################################################################################################
266         print('>>' * 50)
267         print('[INFO] Начните объединение двух больших таблиц для обработки в качестве конечного результата')
268         time_start = datetime.datetime.now()
269         df_all = pd.merge(left=df_name_info,
270                           right=df_actresses,
271                           how='inner',
272                           on='name')
273         time_end = datetime.datetime.now()
274         print('[INFO] Конец слияния, во времени：', (time_end - time_start).seconds, ' s')
275
276         df_actresses = None
277         print('[INFO] Освобождение памяти')
278
 ##################################################################################################################################
279
280         print('>>' * 50)
281         print('[INFO] Начните сериализацию (резервное копирование) конечного результата слияния')
282         time_start = datetime.datetime.now()
```

```
283         dump_df_res = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 семестр/СУБД/资料/DataSet/result_dump_actors/dump_actors_' + str(i) +'.bits'
284         f = open(dump_df_res, 'wb')
285         pickle.dump(obj=df_all, file=f)
286         f.close()
287         use_sec = (datetime.datetime.now() - time_start).seconds
288         print('[INFO] Окончание сериализации (резервного копирования) объединенного конечного результата в срок：', use_sec, ' s（', use_sec / 60, '） min')
289
290
291
             ##########################################################################################################################
292         print('[INFO] Start write to JSON file')
293         df_all.to_json(
294             path_or_buf='/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt.PetersburgPolytechnicalUniversity'
295                         '/СПБПУ/3 курс/6 семестр/СУБД/资料/DataSet/result_json_actors/df_final_actors_' + str(i) + '.json',
296             orient='records',
297             lines=True)
298         print('[INFO] JSON Вписать полностью')
299         df_all = None
300         print('[INFO] Освобождение памяти')
301
             ##########################################################################################################################
302
303         start_index += step_index
304         end_index += step_index
305
306     pass
307
```

2. Десериализовать все полученные малые наборы данных и сшить их вместе в большой набор данных, а также удалить дублирующиеся значения

```
1  # ------*------ coding: utf-8 ------*------
2  # @Time    : 2022/4/27 14:37
3  # @Author  : 冰糖雪狸 (NekoSilverfox)
4  # @Project : IMDB 数据处理
5  # @File    : 反序列化数据.py.py
6  # @Software: PyCharm
7  # @Github  ： https://github.com/NekoSilverFox
8  # -----------------------------------------
9  import pickle
10 import datetime
11 import pandas as pd
12
13
```

```
14  def concat_df(bits_file_path_header: str,
15                max_index: int,
16                path_result_bits_save: str) -> pd.DataFrame:
17      """
18      Десериализовать из многих сериализованных файлов и соединить их вместе
19      :param bits_file_path_header: Сериализация [заголовка] файла
20      :param max_index: Максимальный индекс заголовка документа
21      :param path_result_bits_save: Где сохранить десериализацию результата слияния
22      :return: DataFrame после десериализации
23      """
24      df_result = None
25
26      for i in range(1, max_index + 1):
27          print('-' * 50)
28          print('[INFO] Начните читать первый файл ', i)
29          time_start = datetime.datetime.now()
30          bits_file_path = bits_file_path_header + str(i) + '.bits'
31          f = open(bits_file_path, 'rb')
32          df_obj = pickle.load(file=f)
33          f.close()
34          time_end = datetime.datetime.now()
35          print('[INFO] Прочитайте конец файла ', i, ' | Затраченное время：  ', (time_end -
    time_start).seconds, ' s\n')
36
37          if i == 1:
38              df_result = df_obj
39              continue
40
41          print('[INFO] Начните сращивание файла ', i)
42          time_start = datetime.datetime.now()
43          df_result = pd.concat([df_result, df_obj])
44          time_end = datetime.datetime.now()
45          print('[INFO] Конец ', i, 'сращивания документов，  Затраченное время：  ', (time_end -
    time_start).seconds, ' s\n')
46
47      # Завершение слияния, сохранение результатов с помощью сериализации
48      print('-' * 50)
49      print('[INFO] Завершение слияния, сохранение результатов с помощью сериализации')
50      time_start = datetime.datetime.now()
51      f = open(path_result_bits_save, 'wb')
52      pickle.dump(obj=df_result, file=f)
53      f.close()
54      time_end = datetime.datetime.now()
55      print('[INFO] Конец сериализации для сохранения результатов, во времени：  ', (time_end -
    time_start).seconds, ' s\n')
56
57      return df_result
58
59
60  def merge_duplicates(df_source: pd.DataFrame) -> pd.DataFrame:
61      """
62      Повторное дублирование конечного результата
63      :param df_source: Слияние мужских и женских актеров - это массив
```

```
    :return: Объединенные и де-дуплицированные DataFrame
    """
    df_source.sort_values(by='nconst', inplace=True)
    df_source.reset_index(drop=True, inplace=True)

    i_current = 0
    i_next = i_current + 1
    stop_index = df_source.shape[0]
    while i_next <= stop_index:
        while df_source.loc[i_current]['nconst'] == df_source.loc[i_next]['nconst']:
            df_source.loc[i_current]['rols'] = pd.concat([df_source.loc[i_current]['rols'],
df_source.loc[i_next]['rols']])
            df_source.drop(index=i_next, inplace=True)
            i_next += 1

            if i_next == stop_index:
                df_source.reset_index(drop=True, inplace=True)
                return df_source

        print('[INFO] объединенный ', i_current, ' ряд | ', round(i_current / df_source.shape[0] * 100, 4),
'%')
        i_current = i_next
        i_next += 1

    df_source.reset_index(drop=True, inplace=True)
    return df_source


if __name__ == '__main__':

 ##################################################################################################################################
    # Объединить всех мужчин-актеров (актеров)

 ##################################################################################################################################
    print('>>' * 50)
    bits_file_path_header = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
                            'семестр/СУБД/资料/DataSet/result_dump_actors/dump_actors_'

    path_result_bits_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
                            'семестр/СУБД/资料/DataSet/result_dump_actors/dump_actors_ALL.bits'

    df_all_actors = concat_df(bits_file_path_header=bits_file_path_header,
                              max_index=20,
                              path_result_bits_save=path_result_bits_save)
    print('[INFO] Слияние и сериализация вывода успешно! \n выход на：  ', path_result_bits_save)
```

```
109
    ################################################################################################################################
110     # Объединить всех актрис （actresses）
111
    ################################################################################################################################
112     print('>>' * 50)
113     bits_file_path_header = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
114                             '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
115                             'семестр/СУБД/资料/DataSet/result_dump_actresses/dump_actresses_'
116
117     path_result_bits_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
118                             '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
119                             'семестр/СУБД/资料/DataSet/result_dump_actresses/dump_actresses_ALL.bits'
120
121     df_all_actresses = concat_df(bits_file_path_header=bits_file_path_header,
122                                  max_index=13,
123                                  path_result_bits_save=path_result_bits_save)
124     print('[INFO] Слияние и сериализация вывода успешно! \n выход на：', path_result_bits_save)
125
126
127
    ################################################################################################################################
128     # Объединить всех мужчин-актеров (актеров) и женщин-актеров (актрис)
129
    ################################################################################################################################
130     print('>>' * 50)
131     print('[INFO] Объединить всех мужчин-актеров (актеров) и женщин-актеров (актрис)')
132     time_start = datetime.datetime.now()
133
134     df_result_all = pd.concat([df_all_actors, df_all_actresses])
135
136     time_end = datetime.datetime.now()
137     print('[INFO] Объедините всех мужчин-актеров (актеров) и женщин-актеров (актрис) до конца, в.', (time_end - time_start).seconds, ' s\n')
138
139
    ################################################################################################################################
140     df_all_actors = None
141     df_all_actresses = None
142     print('[INFO] Освобождение памяти')
143
    ################################################################################################################################
144
145
146
    ################################################################################################################################
```

```
147     # 合并后的最终结果再次去重
148 
        ##########################################################################################################################
149     print('>>' * 50)
150     print('[INFO] Конечный результат слияния снова дедуплицируется')
151     time_start = datetime.datetime.now()
152     df_result_all = merge_duplicates(df_source=df_result_all)
153     time_end = datetime.datetime.now()
154     print('[INFO] Конечный результат слияния снова дедуплицируется для завершения：', (time_end - time_start).seconds, ' s\n')  # 9714 s
155 
156 
        ##########################################################################################################################
157     # Окончательный результат слияния затем сериализуется и сохраняется в виде JSON-файла
158 
        ##########################################################################################################################
159     path_result_bits_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
160                             '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
161                             'семестр/СУБД/资料/DataSet/result_ALL/dump_ALL.bits'
162     print('>>' * 50)
163     print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью сериализации')
164     time_start = datetime.datetime.now()
165     f = open(path_result_bits_save, 'wb')
166     pickle.dump(obj=df_result_all, file=f)
167     f.close()
168     time_end = datetime.datetime.now()
169     print('[INFO] Конец сериализации для сохранения результатов, во времени：', (time_end - time_start).seconds, ' s\n')
170 
171 
172     path_result_json_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
173                             '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
174                             'семестр/СУБД/资料/DataSet/result_ALL/dump_ALL.json'
175     print('>>' * 50)
176     print('[INFO] Сохраните [окончательный] результат в формате JSON')
177     time_start = datetime.datetime.now()
178     df_result_all.to_json(path_or_buf=path_result_json_save,
179                           orient='records',
180                           lines=True)
181     time_end = datetime.datetime.now()
182     print('[INFO] Сохраните [окончательный] результат в формате JSON в формате：', (time_end - time_start).seconds, ' s\n')
183 
        ##########################################################################################################################
184     pass
185 
```

3. Выполнение тестов на скорость, сериализация и построение результатов в виде изображений

```python
# ------*------ coding: utf-8 ------*------
# @Time    : 2022/4/28 14:22
# @Author  : 冰糖雪狸 (NekoSilverfox)
# @Project : JSON 速度测试
# @File    : 速度测试.py
# @Software: PyCharm
# @Github  :  https://github.com/NekoSilverFox
# -----------------------------------------
import psycopg2 as pg
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import pickle


def get_id_len_json_df():
    """
    Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
    :return:
    """
    # Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
    print('>>' * 50)
    print('[INFO] Start connect database')
    conn = pg.connect(database="db_imdb",
                      user="postgres",
                      password="postgres",
                      host="localhost",
                      port="5432")
    cur = conn.cursor()
    print('[INFO] Connect database successfully')

    # Получить количество строк
    cur.execute("SELECT COUNT(*) FROM tb_json;")
    count_row = cur.fetchall()[0][0]
    print('количество строк：', count_row)

    # DataFrame для статистики
    df_counter = pd.DataFrame([[0, 0]], columns=['id', 'len_json'])

    # Выполните запрос и засеките время
    for i in range(1, count_row):
        print('[INFO] тестирует строку ' + str(i) + 'строка | ' + str(round(i / count_row * 100, 4)) + '%')

        comm_sql = 'SELECT imdata FROM tb_json WHERE iddata=' + str(i) + ';'
        cur.execute(comm_sql)
        len_row_json = len(str(cur.fetchall()[0]))  # длина JSON(B)

        df_tmp = pd.DataFrame([[i, len_row_json]], columns=['id', 'len_json'])
        df_counter = pd.concat([df_counter, df_tmp])

    conn.close()
```

```
52
53      df_counter = df_counter.iloc[1:, :]
54      df_counter.sort_values(by='len_json', inplace=True)
55
56      print('>>' * 50)
57      print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью
    сериализации')
58      time_start = datetime.datetime.now()
59      f = open('./result/df_id_json_len.bits', 'wb')
60      pickle.dump(obj=df_counter, file=f)
61      f.close()
62      time_end = datetime.datetime.now()
63      print('[INFO] Конец сериализации для сохранения результатов, во времени：  ', (time_end -
    time_start).seconds, ' s\n')
64
65
66  def json_by_id_every_col_test():
67      # Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
68      print('>>' * 50)
69      print('[INFO] Start connect database')
70      conn = pg.connect(database="db_imdb",
71                        user="postgres",
72                        password="postgres",
73                        host="localhost",
74                        port="5432")
75      cur = conn.cursor()
76      print('[INFO] Connect database successfully')
77
78      # Получить количество строк
79      cur.execute("SELECT COUNT(*) FROM tb_json;")
80      count_row = cur.fetchall()[0][0]
81      print('Количество строк：  ', count_row)
82
83      # DataFrame для статистики
84      df_counter = pd.DataFrame([[0, 0, 0, 0, 0, 0, 0]],
85                                columns=['len_row', 'full_ms', 'nconst_ms', 'name_ms', 'birthYear_ms',
    'profession_ms', 'rols_ms'])
86
87      # Выполните запрос и засеките время
88      for i in range(1, count_row):
89          print('[INFO] тестирует строку ' + str(i) + 'строка | ' + str(round(i / count_row * 100, 4)) + '%')
90
91          # Все поля во всей строке данных
92          comm_sql = 'SELECT imdata FROM tb_json WHERE iddata=' + str(i) + ';'
93          start_time = datetime.datetime.now()
94          cur.execute(comm_sql)
95          end_time = datetime.datetime.now()
96          full_use_time_ms = (end_time - start_time).microseconds
97          row = cur.fetchall()[0]
98          len_row_json = len(str(row))  # длина JSON(B)
99
100         # Поля во всей строке данных -  nconst
101         comm_sql = "SELECT imdata->>'nconst' FROM tb_json WHERE iddata=" + str(i) + ';'
```

```
102         start_time = datetime.datetime.now()
103         cur.execute(comm_sql)
104         end_time = datetime.datetime.now()
105         nconst_use_time_ms = (end_time - start_time).microseconds
106 
107         # Поля во всей строке данных -  name
108         comm_sql = "SELECT imdata->>'name' FROM tb_json WHERE iddata=" + str(i) + ';'
109         start_time = datetime.datetime.now()
110         cur.execute(comm_sql)
111         end_time = datetime.datetime.now()
112         name_use_time_ms = (end_time - start_time).microseconds
113 
114         # Поля во всей строке данных -  birthYear
115         comm_sql = "SELECT imdata->>'birthYear' FROM tb_json WHERE iddata=" + str(i) + ';'
116         start_time = datetime.datetime.now()
117         cur.execute(comm_sql)
118         end_time = datetime.datetime.now()
119         birthYear_use_time_ms = (end_time - start_time).microseconds
120 
121         # Поля во всей строке данных -  profession
122         comm_sql = "SELECT imdata->>'profession' FROM tb_json WHERE iddata=" + str(i) + ';'
123         start_time = datetime.datetime.now()
124         cur.execute(comm_sql)
125         end_time = datetime.datetime.now()
126         profession_use_time_ms = (end_time - start_time).microseconds
127 
128         # Поля во всей строке данных -  rols
129         comm_sql = "SELECT imdata->>'rols' FROM tb_json WHERE iddata=" + str(i) + ';'
130         start_time = datetime.datetime.now()
131         cur.execute(comm_sql)
132         end_time = datetime.datetime.now()
133         rols_use_time_ms = (end_time - start_time).microseconds
134 
135         df_tmp = pd.DataFrame([[len_row_json, full_use_time_ms, nconst_use_time_ms,
    name_use_time_ms, birthYear_use_time_ms, profession_use_time_ms, rols_use_time_ms]],
136                               columns=['len_row', 'full_ms', 'nconst_ms', 'name_ms', 'birthYear_ms',
    'profession_ms', 'rols_ms'])
137         df_counter = pd.concat([df_counter, df_tmp])
138 
139     conn.close()
140 
141     df_counter = df_counter.iloc[1:, :]
142     df_counter.sort_values(by='len_row', inplace=True)
143 
144     print('>>' * 50)
145     print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью
    сериализации')
146     time_start = datetime.datetime.now()
147     f = open('./result/json/res_id_ix_every_col.bits', 'wb')
148     pickle.dump(obj=df_counter, file=f)
149     f.close()
150     time_end = datetime.datetime.now()
```

```
    print('[INFO] Конец сериализации для сохранения результатов, во времени：', (time_end - time_start).seconds, ' s\n')

    # print(df_counter)

    """Построение графиков результатов"""
    plt.figure(figsize=(20, 10), dpi=100)
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['full_ms'].values,
                label='full row')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['nconst_ms'].values,
                label='nconst')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['name_ms'].values,
                label='name')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['birthYear_ms'].values,
                label='birthYear')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['profession_ms'].values,
                label='profession')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['rols_ms'].values,
                label='rols')
    plt.legend()
    plt.title('Query by key `ID` in tb_json')
    plt.xlabel('Length of JSON')
    plt.ylabel('Query time (milliseconds)')
    plt.savefig('./result/json/res_id_ix_every_col.png')
    # plt.show()


def jsonb_by_id_every_col_test():
    # Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
    print('>>' * 50)
    print('[INFO] Start connect database')
    conn = pg.connect(database="db_imdb",
                      user="postgres",
                      password="postgres",
                      host="localhost",
                      port="5432")
    cur = conn.cursor()
    print('[INFO] Connect database successfully')

    # Получить количество строк
    cur.execute("SELECT COUNT(*) FROM tb_jsonb;")
    count_row = cur.fetchall()[0][0]
    print('Количество строк：', count_row)

    # DataFrame для статистики
    df_counter = pd.DataFrame([[0, 0, 0, 0, 0, 0, 0]],
```

```
                    columns=['len_row', 'full_ms', 'nconst_ms', 'name_ms', 'birthYear_ms',
'profession_ms', 'rols_ms'])

  # Выполните запрос и засеките время
  for i in range(1, count_row):
      print('[INFO] тестирует строку ' + str(i) + 'строка | ' + str(round(i / count_row * 100, 4)) + '%')

      # Все поля во всей строке данных
      comm_sql = 'SELECT imdata FROM tb_jsonb WHERE iddata=' + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      full_use_time_ms = (end_time - start_time).microseconds
      row = cur.fetchall()[0]
      len_row_jsonb = len(str(row))  # длина JSON(B)

      # Поля во всей строке данных -  nconst
      comm_sql = "SELECT imdata->>'nconst' FROM tb_jsonb WHERE iddata=" + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      nconst_use_time_ms = (end_time - start_time).microseconds

      # Поля во всей строке данных -  name
      comm_sql = "SELECT imdata->>'name' FROM tb_jsonb WHERE iddata=" + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      name_use_time_ms = (end_time - start_time).microseconds

      # Поля во всей строке данных -  birthYear
      comm_sql = "SELECT imdata->>'birthYear' FROM tb_jsonb WHERE iddata=" + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      birthYear_use_time_ms = (end_time - start_time).microseconds

      # Поля во всей строке данных -  profession
      comm_sql = "SELECT imdata->>'profession' FROM tb_jsonb WHERE iddata=" + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      profession_use_time_ms = (end_time - start_time).microseconds

      # Поля во всей строке данных -  rols
      comm_sql = "SELECT imdata->>'rols' FROM tb_jsonb WHERE iddata=" + str(i) + ';'
      start_time = datetime.datetime.now()
      cur.execute(comm_sql)
      end_time = datetime.datetime.now()
      rols_use_time_ms = (end_time - start_time).microseconds

      df_tmp = pd.DataFrame([[len_row_jsonb, full_use_time_ms, nconst_use_time_ms,
name_use_time_ms, birthYear_use_time_ms, profession_use_time_ms, rols_use_time_ms]],
```

```
                    columns=['len_row', 'full_ms', 'nconst_ms', 'name_ms', 'birthYear_ms',
'profession_ms', 'rols_ms'])
        df_counter = pd.concat([df_counter, df_tmp])

    conn.close()

    df_counter = df_counter.iloc[1:, :]
    df_counter.sort_values(by='len_row', inplace=True)

    print('>>' * 50)
    print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью
сериализации')
    time_start = datetime.datetime.now()
    f = open('./result/jsonb/res_id_ix_every_col.bits', 'wb')
    pickle.dump(obj=df_counter, file=f)
    f.close()
    time_end = datetime.datetime.now()
    print('[INFO] Конец сериализации для сохранения результатов, во времени：  ', (time_end -
time_start).seconds, ' s\n')

    # print(df_counter)

    """Построение графиков результатов"""
    plt.figure(figsize=(20, 10), dpi=100)
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['full_ms'].values,
                label='full row')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['nconst_ms'].values,
                label='nconst')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['name_ms'].values,
                label='name')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['birthYear_ms'].values,
                label='birthYear')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['profession_ms'].values,
                label='profession')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['rols_ms'].values,
                label='rols')
    plt.legend()
    plt.title('Query by key `ID` in tb_jsonb')
    plt.xlabel('Length of JSONB')
    plt.ylabel('Query time (milliseconds)')
    plt.savefig('./result/jsonb/res_id_ix_every_col.png')
    # plt.show()


def json_update_by_id_every_col_test():
    # Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
    print('>>' * 50)
```

```
303     print('[INFO] Start connect database')
304     conn = pg.connect(database="db_imdb",
305                       user="postgres",
306                       password="postgres",
307                       host="localhost",
308                       port="5432")
309     cur = conn.cursor()
310     print('[INFO] Connect database successfully')
311
312     # Получить количество строк
313     cur.execute("SELECT COUNT(*) FROM tb_json;")
314     count_row = cur.fetchall()[0][0]
315     print('Количество строк：', count_row)
316
317     # DataFrame для статистики
318     df_counter = pd.DataFrame([[0, 0, 0, 0, 0]],
319                               columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])
320
321     # Выполните запрос и засеките время
322     for i in range(1, count_row):
323         print('\n[INFO] тестирует строку ' + str(i) + 'строка | ' + str(round(i / count_row * 100, 4)) + '%')
324
325         try:
326             # Все поля во всей строке данных
327             comm_sql = 'SELECT imdata FROM tb_json WHERE iddata=' + str(i) + ';'
328             cur.execute(comm_sql)
329             len_row_json = len(str(cur.fetchall()[0]))  # длина JSON(B)
330
331             cur.execute('BEGIN;')
332
333             print('\tBEGIN;')
334
335             # Поля во всей строке данных -  nconst
336             comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{nconst}',
    '\"tt0000009\"'::jsonb) WHERE iddata=" + str(i) + ';'
337             start_time = datetime.datetime.now()
338             cur.execute(comm_sql)
339             end_time = datetime.datetime.now()
340             nconst_use_time_ms = (end_time - start_time).microseconds
341             print('\tnconst Конец испытания, время：', nconst_use_time_ms, ' ms')
342
343             # Поля во всей строке данных -  name
344             comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{name}',
    '\"tt_name\"'::jsonb) WHERE iddata=" + str(i) + ';'
345             start_time = datetime.datetime.now()
346             cur.execute(comm_sql)
347             end_time = datetime.datetime.now()
348             name_use_time_ms = (end_time - start_time).microseconds
349             print('\tname Конец испытания, время：', name_use_time_ms, ' ms')
350
351             # Поля во всей строке данных -  birthYear
352             comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{birthYear}',
    '\"2222\"'::jsonb) WHERE iddata=" + str(i) + ';'
```

```
353             start_time = datetime.datetime.now()
354             cur.execute(comm_sql)
355             end_time = datetime.datetime.now()
356             birthYear_use_time_ms = (end_time - start_time).microseconds
357             print('\tbirthYear Конец испытания, время：  ', birthYear_use_time_ms, ' ms')
358
359             # Поля во всей строке данных -  rols
360             comm_sql = 'UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, \'{rols}\', \'[{"year": 2000, "title": "t_title", "series name": "t_series", "character name": "t_character_name"}]\'::jsonb) WHERE iddata=' + str(i) + ';'
361             start_time = datetime.datetime.now()
362             cur.execute(comm_sql)
363             end_time = datetime.datetime.now()
364             rols_use_time_ms = (end_time - start_time).microseconds
365             print('\trols Конец испытания, время：  ', rols_use_time_ms, ' ms')
366
367         except:
368             cur.execute('ROLLBACK;')
369             continue
370
371         cur.execute('ROLLBACK;')
372         df_tmp = pd.DataFrame([[len_row_json, nconst_use_time_ms, name_use_time_ms, birthYear_use_time_ms, rols_use_time_ms]],
373                               columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])
374         df_counter = pd.concat([df_counter, df_tmp])
375
376     conn.close()
377
378     df_counter = df_counter.iloc[1:, :]
379     df_counter.sort_values(by='len_row', inplace=True)
380
381     print('>>' * 50)
382     print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью сериализации')
383     time_start = datetime.datetime.now()
384     f = open('./result/json/res_update_id_ix_every_col.bits', 'wb')
385     pickle.dump(obj=df_counter, file=f)
386     f.close()
387     time_end = datetime.datetime.now()
388     print('[INFO] Конец сериализации для сохранения результатов, во времени：  ', (time_end - time_start).seconds, ' s\n')
389
390     # print(df_counter)
391
392     """Построение графиков результатов"""
393     plt.figure(figsize=(20, 10), dpi=100)
394     plt.scatter(x=df_counter['len_row'].values,
395                 y=df_counter['nconst_ms'].values,
396                 label='nconst')
397     plt.scatter(x=df_counter['len_row'].values,
398                 y=df_counter['name_ms'].values,
399                 label='name')
400     plt.scatter(x=df_counter['len_row'].values,
```

```
401             y=df_counter['birthYear_ms'].values,
402             label='birthYear')
403     plt.scatter(x=df_counter['len_row'].values,
404             y=df_counter['rols_ms'].values,
405             label='rols')
406     plt.legend()
407     plt.title('Test UPDATE, query by key `ID` in tb_json')
408     plt.xlabel('Length of JSON')
409     plt.ylabel('Query time (milliseconds)')
410     plt.savefig('./result/json/res_update_id_ix_every_col.png')
411     # plt.show()
412
413
414 def jsonb_update_by_id_every_col_test():
415     # Получение длины данных JSON, соответствующих каждому идентификатору в базе данных
416     print('>>' * 50)
417     print('[INFO] Start connect database')
418     conn = pg.connect(database="db_imdb",
419                       user="postgres",
420                       password="postgres",
421                       host="localhost",
422                       port="5432")
423     cur = conn.cursor()
424     print('[INFO] Connect database successfully')
425
426     # Получить количество строк
427     cur.execute("SELECT COUNT(*) FROM tb_jsonb;")
428     count_row = cur.fetchall()[0][0]
429     print('Количество строк：', count_row)
430
431     # DataFrame для статистики
432     df_counter = pd.DataFrame([[0, 0, 0, 0, 0]],
433                               columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])
434
435     # Выполните запрос и засеките время
436     for i in range(1, count_row):
437         print('\n[INFO] тестирует строку ' + str(i) + 'строка | ' + str(round(i / count_row * 100, 4)) + '%')
438
439         try:
440             # Все поля во всей строке данных
441             comm_sql = 'SELECT imdata FROM tb_jsonb WHERE iddata=' + str(i) + ';'
442             cur.execute(comm_sql)
443             len_row_jsonb = len(str(cur.fetchall()[0]))  # длина JSON(B)
444
445             cur.execute('BEGIN;')
446
447             print('\tBEGIN;')
448
449             # Поля во всей строке данных -  nconst
450             comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{nconst}',
    '\"tt0000009\"'::jsonb) WHERE iddata=" + str(i) + ';'
451             # comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, \'{nconst}\',
    \'\"tt0000009\"\'::jsonb) WHERE iddata=" + str(i) + ';'
```

```
452             # print(comm_sql)
453             start_time = datetime.datetime.now()
454             cur.execute(comm_sql)
455             end_time = datetime.datetime.now()
456             nconst_use_time_ms = (end_time - start_time).microseconds
457             print('\tnconst Конец испытания, время：  ', nconst_use_time_ms, ' ms')
458
459             # Поля во всей строке данных -  name
460             comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}',
    '\"tt_name\"'::jsonb) WHERE iddata=" + str(i) + ';'
461             start_time = datetime.datetime.now()
462             cur.execute(comm_sql)
463             end_time = datetime.datetime.now()
464             name_use_time_ms = (end_time - start_time).microseconds
465             print('\tname Конец испытания, время：  ', name_use_time_ms, ' ms')
466
467             # Поля во всей строке данных -  birthYear
468             comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{birthYear}',
    '\"2222\"'::jsonb) WHERE iddata=" + str(i) + ';'
469             start_time = datetime.datetime.now()
470             cur.execute(comm_sql)
471             end_time = datetime.datetime.now()
472             birthYear_use_time_ms = (end_time - start_time).microseconds
473             print('\tbirthYear Конец испытания, время：  ', birthYear_use_time_ms, ' ms')
474
475             # Поля во всей строке данных -  rols
476             comm_sql = 'UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, \'{rols}\', \'[{"year":
    2000, "title": "t_title", "series name": "t_series", "character name": "t_character_name"}]\'::jsonb)
    WHERE iddata=' + str(i) + ';'
477             start_time = datetime.datetime.now()
478             cur.execute(comm_sql)
479             end_time = datetime.datetime.now()
480             rols_use_time_ms = (end_time - start_time).microseconds
481             print('\trols Конец испытания, время：  ', rols_use_time_ms, ' ms')
482
483         except:
484             cur.execute('ROLLBACK;')
485             continue
486
487         cur.execute('ROLLBACK;')
488         df_tmp = pd.DataFrame([[len_row_jsonb, nconst_use_time_ms, name_use_time_ms,
    birthYear_use_time_ms, rols_use_time_ms]],
489                               columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])
490         df_counter = pd.concat([df_counter, df_tmp])
491
492     conn.close()
493
494     df_counter = df_counter.iloc[1:, :]
495     df_counter.sort_values(by='len_row', inplace=True)
496
497     print('>>' * 50)
498     print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью
    сериализации')
```

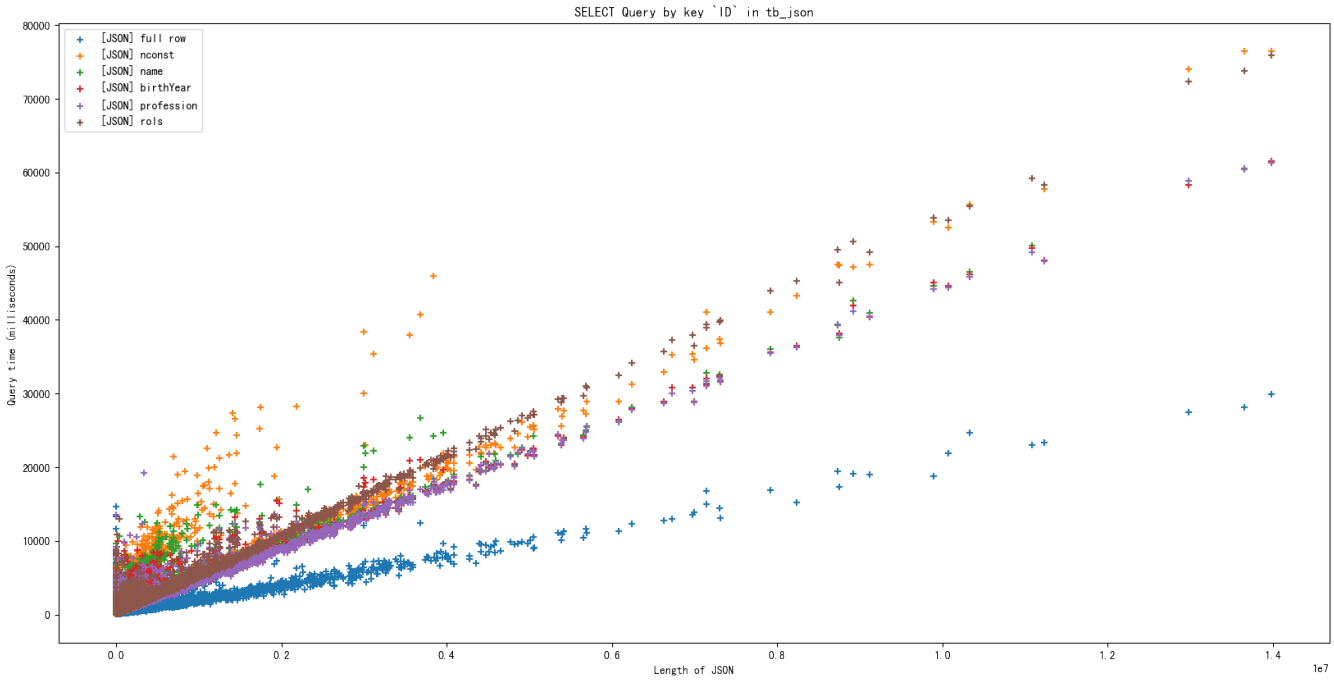
```
499     time_start = datetime.datetime.now()
500     f = open('./result/jsonb/res_update_id_ix_every_col.bits', 'wb')
501     pickle.dump(obj=df_counter, file=f)
502     f.close()
503     time_end = datetime.datetime.now()
504     print('[INFO] Конец сериализации для сохранения результатов, во времени:  ', (time_end -
    time_start).seconds, ' s\n')
505
506     # print(df_counter)
507
508     """Построение графиков результатов"""
509     plt.figure(figsize=(20, 10), dpi=100)
510     plt.scatter(x=df_counter['len_row'].values,
511                 y=df_counter['nconst_ms'].values,
512                 label='nconst')
513     plt.scatter(x=df_counter['len_row'].values,
514                 y=df_counter['name_ms'].values,
515                 label='name')
516     plt.scatter(x=df_counter['len_row'].values,
517                 y=df_counter['birthYear_ms'].values,
518                 label='birthYear')
519     plt.scatter(x=df_counter['len_row'].values,
520                 y=df_counter['rols_ms'].values,
521                 label='rols')
522     plt.legend()
523     plt.title('Test UPDATE, query by key `ID` in tb_jsonb')
524     plt.xlabel('Length of JSONB')
525     plt.ylabel('Query time (milliseconds)')
526     plt.savefig('./result/jsonb/res_update_id_ix_every_col.png')
527     # plt.show()
528
529
530 if __name__ == '__main__':
531     # get_id_len_json_df()
532
533     json_by_id_every_col_test()
534
535     jsonb_by_id_every_col_test()
536
537     json_update_by_id_every_col_test()
538
539     jsonb_update_by_id_every_col_test()
540
```

1.2.2.1.1 Результат

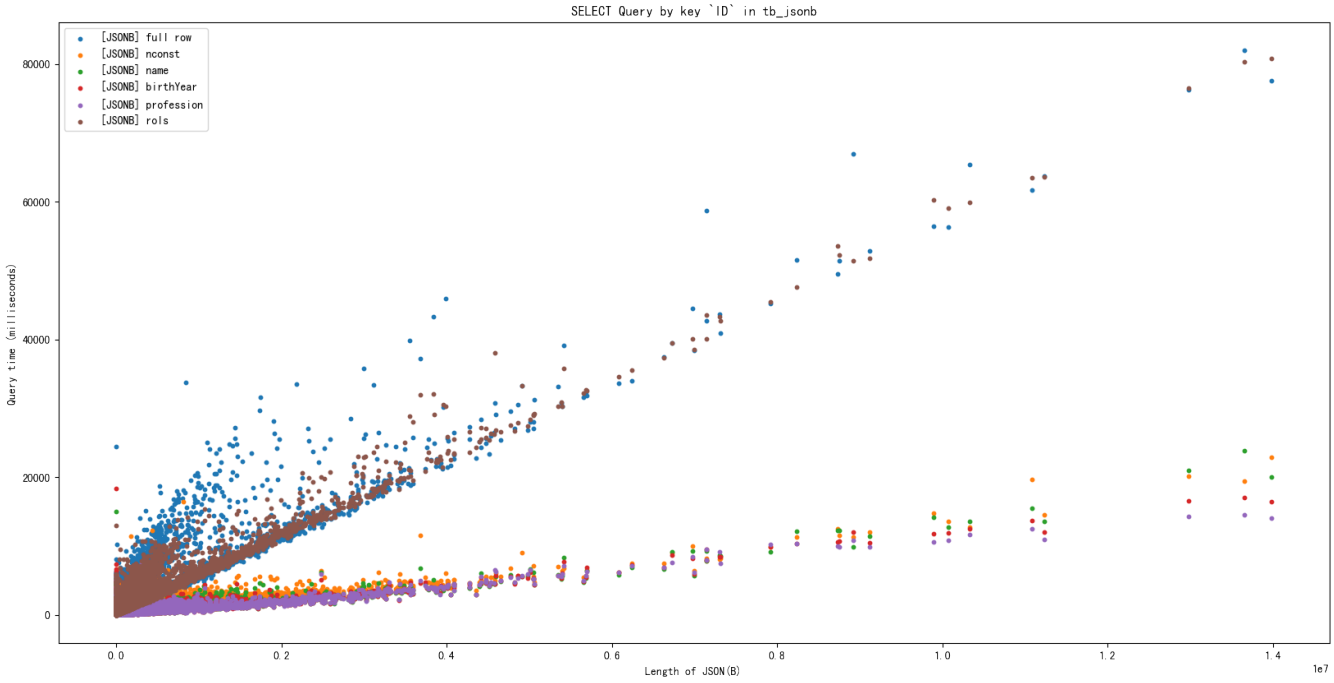
Сравнение времени, затрачиваемого JSON и JSONB при вставке данных в базу данных:

Типа	Времия
JSON	49s
JSONB	79.32s

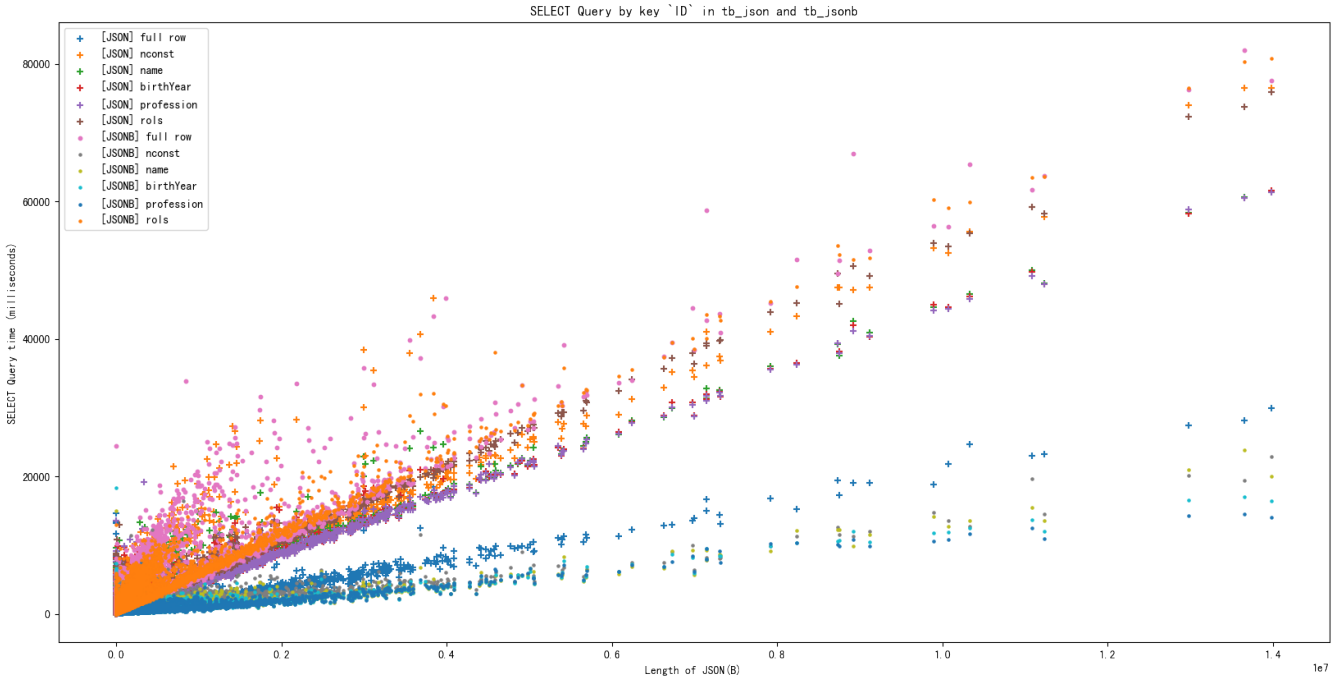
Сравнение времени SELECT для каждого поля JSON в таблице tb_json:



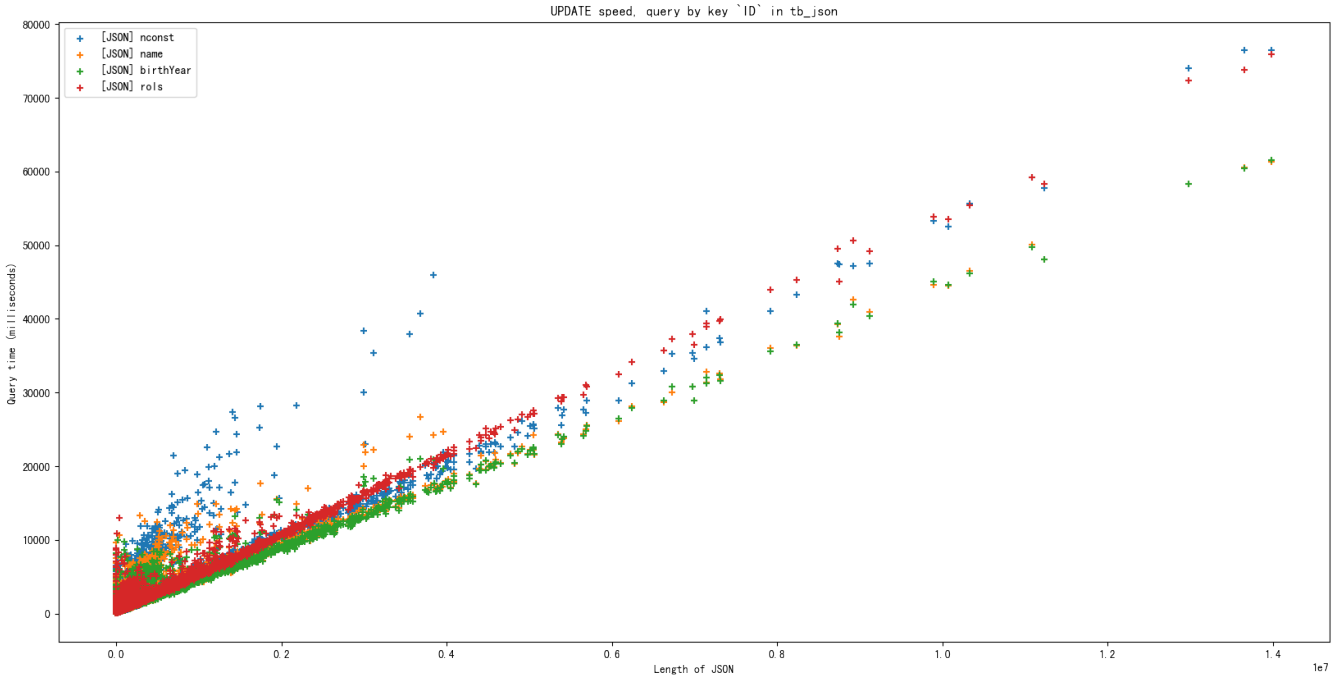
Сравнение времени SELECT для каждого поля JSONB в таблице tb_jsonb:



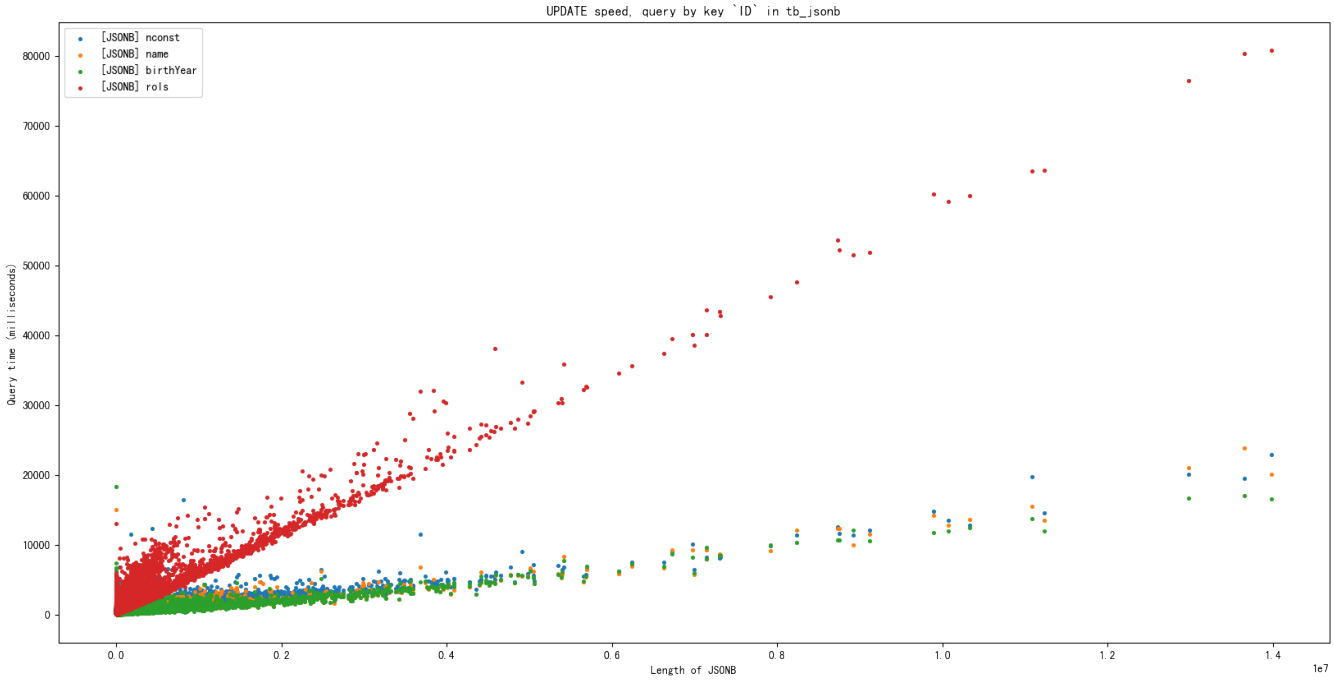
Сравнение скорости запросов к таблицам JSON и JSONB для всех полей:



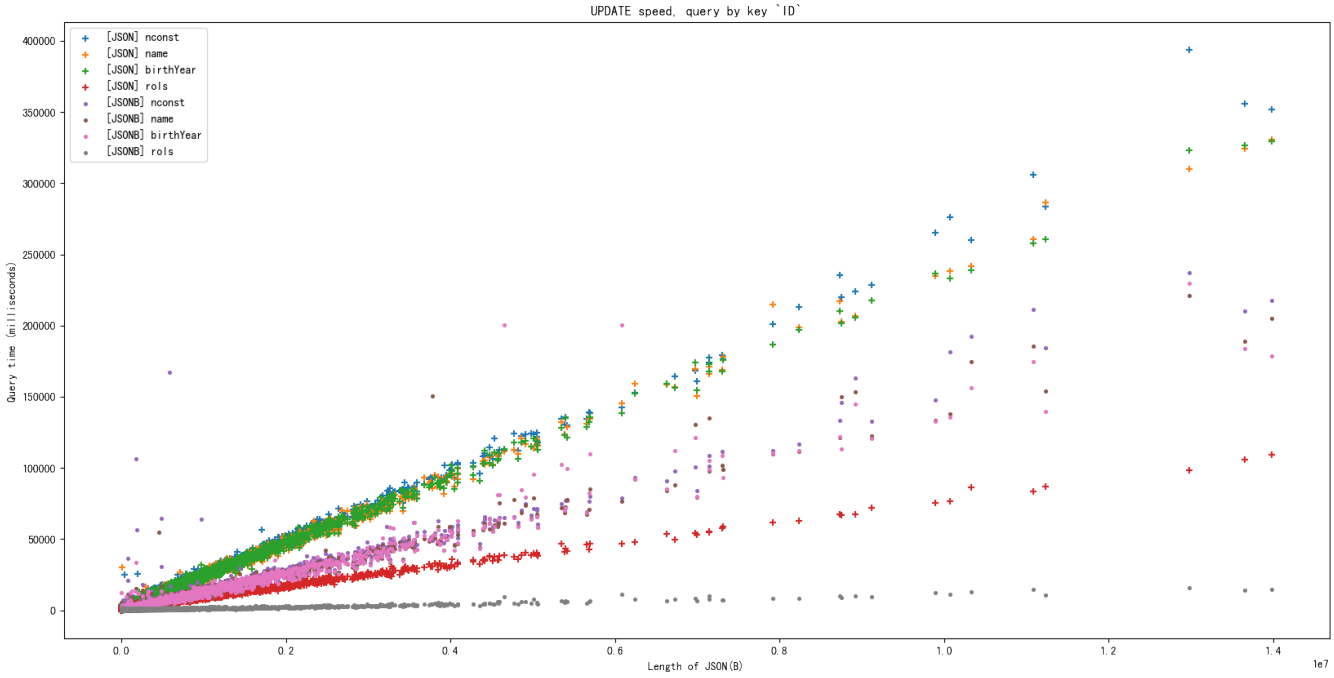
Сравнение времени UPDATE для каждого поля JSON в таблице tb_json:



Сравнение времени UPDATE для каждого поля JSONB в таблице tb_jsonb:



Сравнение скорости UPDATE к таблицам JSON и JSONB для всех полей:


UPDATE speed, query by key `ID`
[JSON] nconst
[JSON] name
[JSON] birthYear
[JSON] rols
[JSONB] nconst
[JSONB] name
[JSONB] birthYear
[JSONB] rols
Query time (milliseconds)
Length of JSON(B)
1e7

1.2.2.2 TOAST

В PG **страница является основной единицей хранения данных в файле**, ее размер фиксирован и может быть задан только во время компиляции и не может быть изменен впоследствии, **размер по умолчанию составляет 8 КБ**.

Кроме того, **PG не позволяет хранить строку данных на разных страницах**. Для очень длинных строк данных PG инициирует TOAST, который сжимает или нарезает большие поля на несколько физических строк и сохраняет их в другой системной таблице (таблица TOAST), этот тип хранения называется **внерядным хранением**.

Для каждого поля таблицы в PG существует четыре стратегии TOAST.

- PLAIN - **позволяет избежать сжатия и хранения вне ряда**. Разрешается выбирать только те типы данных, для хранения которых не требуется политика TOAST (например, типы int), в то время как для таких типов, как текст, требующих длины хранения, превышающей размер страницы, эта политика недопустима.
- EXTENDED - **позволяет сжимать и хранить вне ряда**. Как правило, сначала он сжимается, а если он все еще слишком большой, то сохраняется вне очереди. Это политика по умолчанию для большинства типов данных, которые могут быть TOASTed.
- EXTERNAL - **позволяет хранить данные вне ряда**, но без сжатия. Это значительно ускоряет операции подстроки для полей типа text и bytea. Такие поля, как строки, которые работают с частью данных, могут достичь более высокой производительности при использовании этой политики, поскольку нет необходимости считывать всю строку данных и затем распаковывать ее.
- MAIN - **позволяет сжимать, но не хранить вне линии**. На практике, однако, внепоточное хранение активируется в крайнем случае, когда других методов (например, сжатия) недостаточно для гарантированного хранения больших данных. Поэтому правильнее будет сказать, что хранение вне ряда вообще не должно использоваться.

Просмотр политики TOAST для таблицы tb_jsonb:

Таблица TOAST имеет три поля:

- chunk_id -- используется для указания OID конкретного значения TOAST, которое можно интерпретировать как все строки с одинаковым значением chunk_id, образующие строку данных в поле TOAST исходной таблицы (в данном случае, блога).
- chunk_seq -- используется для указания позиции данных ряда в общем массиве данных.
- chunk_data -- фактические данные чанка

```
postgres=# \d+ tb_jsonb;

                                Table "public.tb_jsonb"
 Column |  Type   | Collation | Nullable | Storage  | Compression | Stats target | Description
--------+---------+-----------+----------+----------+-------------+--------------+-------------
 iddata | integer |           | not null | plain    |             |              |
 imdata | jsonb   |           |          | extended |             |              |

Indexes:
```

```
    "ix_jsonb_iddata" btree (iddata)
Access method: heap



db_imdb=# select relname, relfilenode, reltoastrelid from pg_class where relname='tb_jsonb';
 relname  | relfilenode | reltoastrelid
----------+-------------+---------------
 tb_jsonb |      162078 |        162082
(1 row)



db_imdb=# \d+ pg_toast.pg_toast_162078;
TOAST table "pg_toast.pg_toast_162078"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.tb_jsonb"
Indexes:
    "pg_toast_162078_index" PRIMARY KEY, btree (chunk_id, chunk_seq)
Access method: heap


postgres=# select * from pg_toast.pg_toast_162078;
```

Тестирование политики TOAST в Postgresql

1. **[EXTENDED] Сравнить изменение объема БД для актера с малым кол-вом ролей**

```
BEGIN;
SELECT * FROM tb_jsonb WHERE iddata=51989;
SELECT pg_table_size('tb_jsonb');  -- 1145241600 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1092 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=51989;  -- 202 Byte

UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}', '"Bf AAAAAAAAAAAAAAAAAAAAAAAA"'::jsonb) WHERE iddata=51989;

SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=51989;  -- 230 Byte
SELECT pg_table_size('tb_jsonb');  -- 1145241600 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1092 MB
ROLLBACK;
```

2. **[EXTERNAL] Сравнить изменение объема БД для актера с малым кол-вом ролей**

Политика TOAST изменена на EXTERNAL для отключения сжатия

```
postgres=# BEGIN;
postgres=# ALTER TABLE tb_jsonb ALTER imdata SET STORAGE EXTERNAL;

db_imdb=*# \d+ tb_jsonb;
                                Table "public.tb_jsonb"
 Column |  Type   | Collation | Nullable | Storage  | Compression | Stats target | Description
--------+---------+-----------+----------+----------+-------------+--------------+-------------
 iddata | integer |           | not null | plain    |             |              |
 imdata | jsonb   |           |          | external |             |              |
Indexes:
    "ix_jsonb_iddata" btree (iddata)
Access method: heap

```

```
BEGIN;
ALTER TABLE tb_jsonb ALTER imdata SET STORAGE EXTERNAL;

SELECT * FROM tb_jsonb WHERE iddata=51989;
SELECT pg_table_size('tb_jsonb');  -- 1202307072 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1147 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=51989;  -- 202 Byte
select count(*) from pg_toast.pg_toast_162078; -- 511805

```

```
10 UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}', '"Bf
   AAAAAAAAAAAAAAAAAAAAAAAA"'::jsonb) WHERE iddata=51989;
11
12 SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE
   iddata=51989;  -- 230 Byte
13 SELECT pg_table_size('tb_jsonb');  -- 1202307072 Byte
14 SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1147 MB
15 select count(*) from pg_toast.pg_toast_162078; -- 511805
16 ROLLBACK;
```

3. [EXTENDED] **Сравнить изменение объема БД для актера с большим кол-вом ролей**

```
1  BEGIN;
2  SELECT pg_table_size('tb_jsonb');  -- 1145241600 Byte
3  SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1092 MB
4  SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE
   iddata=3789;  -- 4034997 Byte
5
6  UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}', '"David
   AAAAAAAAAAAAAAAAAAAAAAAA"'::jsonb) WHERE iddata=3789;
7
8  SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE
   iddata=3789;  -- 4035007 [+10]  Byte
9  SELECT pg_table_size('tb_jsonb');  -- 1149378560 [+4136960]  Byte
10 SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1096 MB [+3.9453125 MB]
11 ROLLBACK;
```

4. [EXTERNAL] **Сравнить изменение объема БД для актера с большим кол-вом ролей**

Политика TOAST изменена на EXTERNAL для отключения сжатия

```
1  postgres=# BEGIN;
2  postgres=# ALTER TABLE tb_jsonb ALTER imdata SET STORAGE EXTERNAL;
3
4  db_imdb=*# \d+ tb_jsonb;
5                                        Table "public.tb_jsonb"
6   Column |  Type   | Collation | Nullable | Storage  | Compression | Stats target | Description
7  --------+---------+-----------+----------+----------+-------------+--------------+-------------
8   iddata | integer |           | not null | plain    |             |              |
9   imdata | jsonb   |           |          | external |             |              |
10 Indexes:
11    "ix_jsonb_iddata" btree (iddata)
12 Access method: heap
13
```

Мы можем видеть, что объем данных значительно больше без включенного сжатия.

```
BEGIN;
ALTER TABLE tb_jsonb ALTER imdata SET STORAGE EXTERNAL;

SELECT pg_table_size('tb_jsonb');  -- 1186045952 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1131 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=3789;  -- 4034997 Byte

UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}', '"David AAAAAAAAAAAAAAAAAAAAAAAA"'::jsonb) WHERE iddata=3789;

SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=3789;  -- 15845197 [+11810200] Byte
SELECT pg_table_size('tb_jsonb');  -- 1202307072 [+16261120] Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1147 MB [+16 MB]
ROLLBACK;
```

1.2.2.2.1 Результат

- Если политика разрешает сжатие, TOAST предпочитает сжатие.
- Хранение вне ряда включается, когда объем данных превышает примерно 2 КБ, независимо от того, сжаты они или нет.
- Изменение политики TOAST **не** повлияет на способ хранения существующих данных.

2. Приложение

2.1 Адрес репозитория GitHub

Все исходные данные, выводы и код можно найти на GitHub, адрес репозитория:

https://github.com/NekoSilverFox/PostgreSQL-SPbSTU