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.

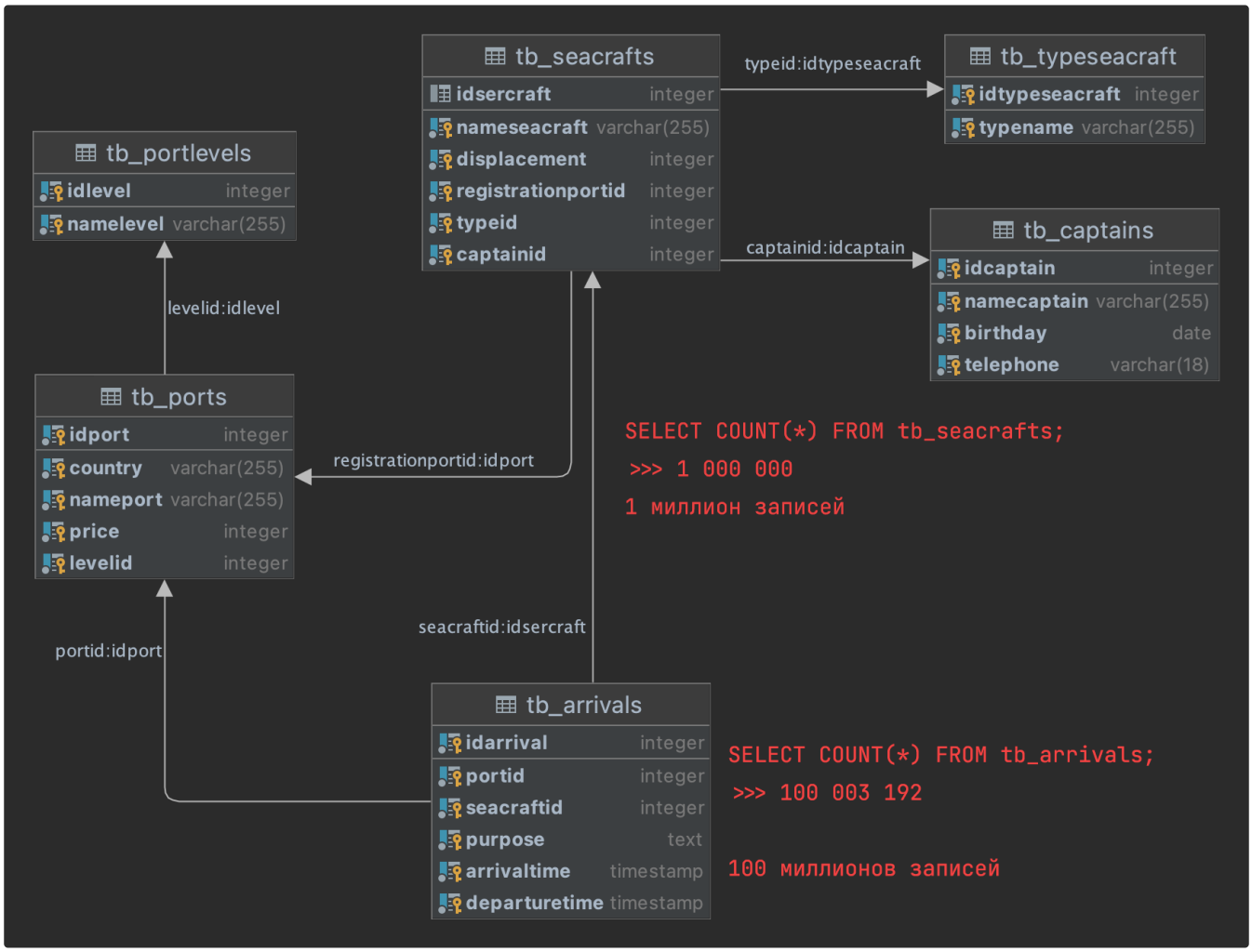
- Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.
- Оценить эффективность выполнения оптимизированного запроса.

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

```
---------------------------------------------------------------------------------------------
-- 向 tb_seacraft 中插入 1000000 条随机数据记录
-- Вставьте 1 000 000 случайных записей данных в tb_arrivals
CREATE OR REPLACE FUNCTION fc_InsertSeacrafts()
 RETURNS TEXT
 AS $$
import random
import datetime


COL_NUM = 1000000


def get_random_name():
    """
    生成一段随机 `name`
    Сгенерируйте случайный `name`.

    :return: text -> string
    """
    len_word = random.randint(1, 10)
    word = ''.join(random.sample('zyxwvutsrqponmlkjihgfedcba', len_word))

    return word


if __name__ == '__main__':
    try:
        num_captain = plpy.execute("SELECT COUNT(*) FROM tb_captains")[0]['count']
        num_port = plpy.execute("SELECT COUNT(*) FROM tb_ports")[0]['count']
        num_type = plpy.execute("SELECT COUNT(*) FROM tb_portlevels")[0]['count']

        for i in range(COL_NUM):
            regportid = random.randint(1, num_port)
            captainid = random.randint(1, num_captain)
            typeid = random.randint(1, num_type)
            displacement = random.randint(10000, 1000000)
            nameseacraft = get_random_word()

            command = 'INSERT INTO tb_seacraft(NameSeacraft, Displacement, RegPortID, TypeID, CaptainID)
VALUES ('
            command = command + str(nameseacraft) + ', ' + str(displacement) + ', ' + str(regportid) + ', ' + str(typeid) +
str(captainid) + ');'
            # return command
            plpy.execute(command)

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

```
    else:
        return "[Info] Successful insert"

$$ LANGUAGE plpython3u;

-------------------------------------------------------------
```

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

PostgreSQL имеет следующие индексы:

Имя индекса	Сценарии использования
B-Tree	Подходит для индексов, которые можно хранить последовательно (по умолчанию)
Hash	Могут обрабатывать только простые сравнения "равно".
GiST	ЭТО архитектура индексирования, которая позволяет нам настраивать индексы в соответствии с потребностями и сценариями
GIN	Инвертированные индексы, которые работают с ключами, содержащими несколько значений

[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).

```
SELECT * FROM tb_arrivals WHERE ArrivalTime BETWEEN '2000-01-01 00:00:00'::TIMESTAMP AND
'2001-01-01 00:00:00' AND PortID > 1000;

DROP INDEX IX_ArriverTime;
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)
         Recheck Cond: (purpose = 'sfrht'::text)
```

```
        Rows Removed by Index Recheck: 23
        Heap Blocks: exact=23
        ->  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)
              Index Cond: (purpose = 'sfrht'::text)
   ->  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)
        Filter: (purpose = 'sfrht'::text)
 Planning Time: 14.311 ms
 Execution Time: 657.908 ms
(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/

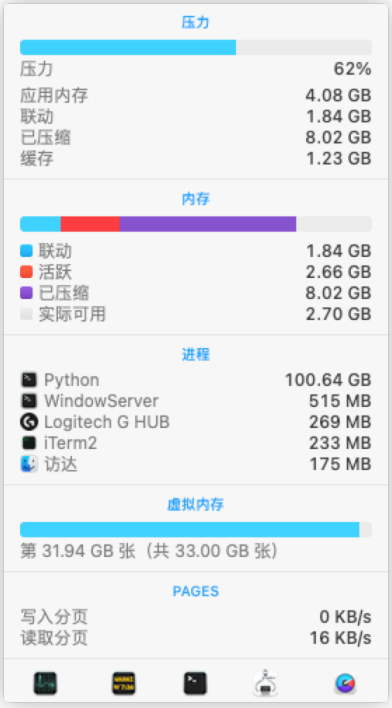
1.2.2.1 IMDB-JSONB

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())
```

```
            """series name - Название серии：  {}"""
            series_name = re.search(r'\{(.*?)\}', title_mix)
            if series_name is not None:
                series_name = str(series_name.group()[1:-1])

            """character name - Имя персонажа"""
            character_name = re.search(r'\[(.*?)\]', title_mix)
            if character_name is not None:
                character_name = str(character_name.group()[1:-1])
            # name_title.append([this_name, title, series_name, year, character_name])
            # name_title.append([this_name, [title, series_name, year, character_name]])
            rols = pd.DataFrame([[title, series_name, year, character_name]],
                                columns=['title', 'series name', 'year', 'character name'])

            list_name_title.append([this_name, rols])  # this_name 是 str, rols 是DataFrame

        if i % 10000 == 0:
            use_sec = (datetime.datetime.now() - time_start).seconds
            print('[INFO] Обработано ', i, ' строк | ', (i / source_data.shape[0]) * 100, '% | Затраченное время：',
use_sec, ' s（', use_sec / 60,
                  '）min')
        i += 1

    use_sec = (datetime.datetime.now() - time_start).seconds
    print('[INFO] Окончание извлечения данных, время：', use_sec, ' s（', use_sec / 60, '）min')

    """Получить результат df_name_title в виде DataFrame с MutIndex"""
    print('>>' * 50)
    print('[INFO] Получить результат df_name_title в виде DataFrame с MutIndex')
    time_start = datetime.datetime.now()
    df_name_title = pd.DataFrame(list_name_title, columns=['name', 'rols'])
    list_name_title = []
    df_name_title.sort_values(by='name', inplace=True)
    df_name_title.reset_index(drop=True, inplace=True)
    use_sec = (datetime.datetime.now() - time_start).seconds
    print('[INFO] DataFrame Конец преобразования, во времени：', use_sec, ' s（', use_sec / 60, '）min')

    # print('>>' * 50)
    # print('[INFO] 开始序列化（备份）df_name_title')
    # time_start = datetime.datetime.now()
    # f = open(dump_name_title, 'wb')
    # pickle.dump(obj=df_name_title, file=f)
    # f.close()
    # use_sec = (datetime.datetime.now() - time_start).seconds
    # print('[INFO] 序列化（备份）结束，用时：', use_sec, ' s（', use_sec / 60, '）min')

    """Объединить дубликаты имен, чтобы сделать их уникальными. добавить информацию об одном и
том же актере в rols"""
    print('>>' * 50)
    print('[INFO] Начните объединять дубликаты имен')
    time_start = datetime.datetime.now()
    name = None
    tmp_df_rols = []
```

```
    res_list_name_rols = []
    for i in range(df_name_title.shape[0]):

        """Если это один и тот же человек, работы объединяются в DataFrame"""
        if name == df_name_title.loc[i]['name']:
            tmp_df_rols = pd.concat([tmp_df_rols, df_name_title.loc[i]['rols']])

        else:
            """Приступая к следующему человеку
               сначала записать информацию о предыдущем человеке в новый список, затем сбросить имя и
tmp_df_rols на содержимое текущего ряда
            """
            if name is not None:
                res_list_name_rols.append([name, tmp_df_rols])

                # if name == '$haniqua':
                #     print(tmp_df_rols)

            name = df_name_title.loc[i]['name']
            tmp_df_rols = df_name_title.loc[i]['rols']

        if i % 10000 == 0:
            use_sec = (datetime.datetime.now() - time_start).seconds
            print('[INFO] Обработано ', i, ' строка | ', (i / df_name_title.shape[0]) * 100, '% | Затраченное
время：', use_sec, ' s（',
                  use_sec / 60, '）min')

    res_list_name_rols = pd.DataFrame(data=res_list_name_rols, columns=['name', 'rols'])

    # print('>>' * 50)
    # print('[INFO] 开始序列化（备份）res_list_name_rols')
    # time_start = datetime.datetime.now()
    # f = open(dump_df_res, 'wb')
    # pickle.dump(obj=res_list_name_rols, file=f)
    # f.close()
    # use_sec = (datetime.datetime.now() - time_start).seconds
    # print('[INFO] 序列化（备份）res_list_name_rols 结束，用时：', use_sec, ' s（', use_sec / 60, '）min')

    return res_list_name_rols


if __name__ == '__main__':
    print('>>' * 50)
    print('[INFO] Начало реализации')


 ##############################################################################################
#########################
    # 演员信息表
    print('>>' * 50)
    print('[INFO] Начните читать `name.basics.tsv`')
    time_start = datetime.datetime.now()
    df_name_info = pd.read_csv(
```

```
        filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt'
                           '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 семестр/СУБД/资
料/DataSet/name.basics.tsv',
        header=0,
        sep='\t'
    )
    df_name_info = df_name_info.iloc[:, :-1]
    df_name_info.columns = ['nconst', 'name', 'birthYear', 'deathYear', 'profession']
    time_end = datetime.datetime.now()
    print('[INFO] Конец чтения `name.basics.tsv`, время：  ', (time_end - time_start).seconds, ' s')

 ##############################################################################################
#########################


 ##############################################################################################
#########################
    # Обработка отсутствующих значений как None для простого преобразования в JSON
    print('>>' * 50)
    print('[INFO] Начните обрабатывать отсутствующие значения как None и удалите дублирующиеся
значения для легкого преобразования в JSON')
    time_start = datetime.datetime.now()
    df_name_info.replace(to_replace=['\\N', np.nan], value=None, inplace=True)
    df_name_info.drop_duplicates(subset='name', keep='first', inplace=True)
    time_end = datetime.datetime.now()
    print('[INFO] Окончание обработки отсутствующих значений и удаления дублирующих значений,
время：  ', (time_end - time_start).seconds, ' s')

 ##############################################################################################
#########################


 ##############################################################################################
#########################
    print('>>' * 50)
    print('[INFO] Начните чтение файла', '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt'
                           '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 '
                           'семестр/СУБД/资料/DataSet/data_actors.list.txt')
    time_start = datetime.datetime.now()
    # source_data = pd.read_csv(
    #     filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-
PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
    #                'курс/6 семестр/СУБД/资料/DataSet/data_actresses.list.txt',
    #     header=0,
    #     sep='\t'
    # )

    # Список мужчин-актеров DataFrame
    source_data = pd.read_csv(
        filepath_or_buffer='/Users/fox/Library/CloudStorage/OneDrive-
PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
                   'курс/6 семестр/СУБД/资料/DataSet/data_actors.list.txt',
```

```
        header=0,
        sep='\t'
    )
    time_end = datetime.datetime.now()
    print('[INFO] Конец чтения файла, время：', (time_end - time_start).seconds, ' s')



    print('>>' * 50)
    print('[INFO] Начните предварительную обработку данных')
    time_start = datetime.datetime.now()
    source_data.columns = ['name', 't1', 't2', 't3']
    source_data['t1'].fillna(value='', inplace=True)
    source_data['t2'].fillna(value='', inplace=True)
    source_data['t3'].fillna(value='', inplace=True)

    movie_list = source_data['t1'] + source_data['t2'] + source_data['t3']
    source_data = pd.concat([source_data['name'], movie_list], axis=1)
    source_data.columns = ['name', 'title_mix']

    # source_data = source_data.iloc[:1000, :]

    time_end = datetime.datetime.now()
    print('[INFO] Окончание предварительной обработки данных, время：', (time_end - time_start).seconds,
' s')


    start_index = 0
    step_index = 1000000
    end_index = start_index + step_index
    times = 31



    for i in range(1, times):
        print('\n\n')
        print('>>' * 20, ' Запустите цикл ', i, ' <<' * 20)

        tmp_source_data = source_data.iloc[start_index:end_index, :]


 ##############################################################################################
##########################
        # # Список актрис df
        # print('[INFO]  main -> Начать обработку `data_actresses.list.txt`')
        # df_actresses = normal_actor(
        #     source_data=tmp_source_data,
        #     dump_name_title='./result/dump_df_actresses_name_title.bits',
        #     dump_df_res='./result/dump_df_actresses.bits'
        # )
        # tmp_source_data = None
        # print('[INFO] main <- Окончание обработки `data_actresses.list.txt`')
```

```
 ##############################################################################################################
##########################



 ##############################################################################################################
##########################
    # # 男演员列表df
    # print('[INFO] Start handle `data_actors.list.txt`')
    # df_actors = normal_actor(
    #     file_path='/Users/fox/Library/CloudStorage/OneDrive-
PetertheGreatSt.PetersburgPolytechnicalUniversity/СПБПУ/3 '
    #               'курс/6 семестр/СУБД/资料/DataSet/data_actors.list.txt'
    # )
    print('[INFO]  main -> Начать обработку `data_actors.list.txt`')
    df_actresses = normal_actor(
        source_data=tmp_source_data,
        dump_name_title='./result/dump_df_actors_name_title.bits',
        dump_df_res='./result/dump_df_actors.bits'
    )
    tmp_source_data = None
    print('[INFO] main <- Окончание обработки `data_actresses.list.txt`')

 ##############################################################################################################
##########################



 ##############################################################################################################
##########################
    print('>>' * 50)
    print('[INFO] Начните объединение двух больших таблиц для обработки в качестве конечного
результата')
    time_start = datetime.datetime.now()
    df_all = pd.merge(left=df_name_info,
                      right=df_actresses,
                      how='inner',
                      on='name')
    time_end = datetime.datetime.now()
    print('[INFO] Конец слияния, во времени：  ', (time_end - time_start).seconds, ' s')

    df_actresses = None
    print('[INFO] Освобождение памяти')

 ##############################################################################################################
##########################

    print('>>' * 50)
    print('[INFO] Начните сериализацию (резервное копирование) конечного результата слияния')
    time_start = datetime.datetime.now()
    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'
```

```
        f = open(dump_df_res, 'wb')
        pickle.dump(obj=df_all, file=f)
        f.close()
        use_sec = (datetime.datetime.now() - time_start).seconds
        print('[INFO] Окончание сериализации (резервного копирования) объединенного конечного
результата в срок：', use_sec, ' s（', use_sec / 60, '）min')



 ##############################################################################################
#########################
        print('[INFO] Start write to JSON file')
        df_all.to_json(
            path_or_buf='/Users/fox/Library/CloudStorage/OneDrive-
PetertheGreatSt.PetersburgPolytechnicalUniversity'
                        '/СПБПУ/3 курс/6 семестр/СУБД/资料/DataSet/result_json_actors/df_final_actors_' + str(i) +
'.json',
            orient='records',
            lines=True)
        print('[INFO] JSON Вписать полностью')
        df_all = None
        print('[INFO] Освобождение памяти')

 ##############################################################################################
#########################

        start_index += step_index
        end_index += step_index

    pass
```

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

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


def concat_df(bits_file_path_header: str,
              max_index: int,
              path_result_bits_save: str) -> pd.DataFrame:
```

```
    """
    Десериализовать из многих сериализованных файлов и соединить их вместе
    :param bits_file_path_header: Сериализация [заголовка] файла
    :param max_index: Максимальный индекс заголовка документа
    :param path_result_bits_save: Где сохранить десериализацию результата слияния
    :return: DataFrame после десериализации
    """
    df_result = None

    for i in range(1, max_index + 1):
        print('-' * 50)
        print('[INFO] Начните читать первый файл ', i)
        time_start = datetime.datetime.now()
        bits_file_path = bits_file_path_header + str(i) + '.bits'
        f = open(bits_file_path, 'rb')
        df_obj = pickle.load(file=f)
        f.close()
        time_end = datetime.datetime.now()
        print('[INFO] Прочитайте конец файла ', i, ' | Затраченное время：', (time_end - time_start).seconds, '
s\n')

        if i == 1:
            df_result = df_obj
            continue

        print('[INFO] Начните сращивание файла ', i)
        time_start = datetime.datetime.now()
        df_result = pd.concat([df_result, df_obj])
        time_end = datetime.datetime.now()
        print('[INFO] Конец ', i, 'сращивания документов，Затраченное время：', (time_end -
time_start).seconds, ' s\n')

    # Завершение слияния, сохранение результатов с помощью сериализации
    print('-' * 50)
    print('[INFO] Завершение слияния, сохранение результатов с помощью сериализации')
    time_start = datetime.datetime.now()
    f = open(path_result_bits_save, 'wb')
    pickle.dump(obj=df_result, 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')

    return df_result


def merge_duplicates(df_source: pd.DataFrame) -> pd.DataFrame:
    """
    Повторное дублирование конечного результата
    :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)



 ##############################################################################################
#######################
    # Объединить всех актрис （actresses）
```

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

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

    df_all_actresses = concat_df(bits_file_path_header=bits_file_path_header,
                                 max_index=13,
                                 path_result_bits_save=path_result_bits_save)
    print('[INFO] Слияние и сериализация вывода успешно! \n выход на：  ', path_result_bits_save)



 ##################################################################################################
#########################
    # Объединить всех мужчин-актеров (актеров) и женщин-актеров (актрис)

 ##################################################################################################
#########################
    print('>>' * 50)
    print('[INFO] Объединить всех мужчин-актеров (актеров) и женщин-актеров (актрис)')
    time_start = datetime.datetime.now()

    df_result_all = pd.concat([df_all_actors, df_all_actresses])

    time_end = datetime.datetime.now()
    print('[INFO] Объедините всех мужчин-актеров (актеров) и женщин-актеров (актрис) до конца, в.',
(time_end - time_start).seconds, ' s\n')


 ##################################################################################################
#########################
    df_all_actors = None
    df_all_actresses = None
    print('[INFO] Освобождение памяти')

 ##################################################################################################
#########################



 ##################################################################################################
#########################
    # 合并后的最终结果再次去重

 ##################################################################################################
#########################
```

```
    print('>>' * 50)
    print('[INFO] Конечный результат слияния снова дедуплицируется')
    time_start = datetime.datetime.now()
    df_result_all = merge_duplicates(df_source=df_result_all)
    time_end = datetime.datetime.now()
    print('[INFO] Конечный результат слияния снова дедуплицируется для завершения：', (time_end - time_start).seconds, ' s\n')  #  9714  s


    ##################################################################################################################
    # Окончательный результат слияния затем сериализуется и сохраняется в виде JSON-файла

    ##################################################################################################################
    path_result_bits_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
                            'семестр/СУБД/资料/DataSet/result_ALL/dump_ALL.bits'
    print('>>' * 50)
    print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью сериализации')
    time_start = datetime.datetime.now()
    f = open(path_result_bits_save, 'wb')
    pickle.dump(obj=df_result_all, 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')


    path_result_json_save = '/Users/fox/Library/CloudStorage/OneDrive-PetertheGreatSt' \
                            '.PetersburgPolytechnicalUniversity/СПБПУ/3 курс/6 ' \
                            'семестр/СУБД/资料/DataSet/result_ALL/dump_ALL.json'
    print('>>' * 50)
    print('[INFO] Сохраните [окончательный] результат в формате JSON')
    time_start = datetime.datetime.now()
    df_result_all.to_json(path_or_buf=path_result_json_save,
                          orient='records',
                          lines=True)
    time_end = datetime.datetime.now()
    print('[INFO] Сохраните [окончательный] результат в формате JSON в формате：', (time_end - time_start).seconds, ' s\n')

    ##################################################################################################################
    pass
```

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

```
# ------*------ 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()

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

```
    print('>>' * 50)
    print('[INFO] Конец слияния, сохранение [окончательного] результата с помощью сериализации')
    time_start = datetime.datetime.now()
    f = open('./result/df_id_json_len.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')


def json_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_json;")
    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_json 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_json = len(str(row))  # длина JSON(B)

        # Поля во всей строке данных -  nconst
        comm_sql = "SELECT imdata->>'nconst' FROM tb_json 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_json 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_json 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_json 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_json 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_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]],
                              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/json/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_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)
    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, 0, 0, 0]],
                              columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])

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

        try:
            # Все поля во всей строке данных
            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)

            cur.execute('BEGIN;')

            print('\tBEGIN;')

            # Поля во всей строке данных -  nconst
            comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{nconst}', '\"tt0000009\"'::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
            print('\tnconst Конец испытания, время:  ', nconst_use_time_ms, ' ms')

            # Поля во всей строке данных -  name
            comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{name}', '\"tt_name\"'::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
            print('\tname Конец испытания, время:  ', name_use_time_ms, ' ms')

            # Поля во всей строке данных -  birthYear
            comm_sql = "UPDATE tb_json SET imdata=jsonb_set(imdata::jsonb, '{birthYear}', '\"2222\"'::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
            print('\tbirthYear Конец испытания, время:  ', birthYear_use_time_ms, ' ms')

            # Поля во всей строке данных -  rols
```

```
            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) +
';'
            start_time = datetime.datetime.now()
            cur.execute(comm_sql)
            end_time = datetime.datetime.now()
            rols_use_time_ms = (end_time - start_time).microseconds
            print('\trols Конец испытания, время：  ', rols_use_time_ms, ' ms')

        except:
            cur.execute('ROLLBACK;')
            continue

        cur.execute('ROLLBACK;')
        df_tmp = pd.DataFrame([[len_row_json, nconst_use_time_ms, name_use_time_ms,
birthYear_use_time_ms, rols_use_time_ms]],
                              columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_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/json/res_update_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['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['rols_ms'].values,
                label='rols')
    plt.legend()
    plt.title('Test UPDATE, query by key `ID` in tb_json')
    plt.xlabel('Length of JSON')
```

```
    plt.ylabel('Query time (milliseconds)')
    plt.savefig('./result/json/res_update_id_ix_every_col.png')
    # plt.show()


def jsonb_update_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]],
                              columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_ms', 'rols_ms'])

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

        try:
            # Все поля во всей строке данных
            comm_sql = 'SELECT imdata FROM tb_jsonb WHERE iddata=' + str(i) + ';'
            cur.execute(comm_sql)
            len_row_jsonb = len(str(cur.fetchall()[0]))  # длина JSON(B)

            cur.execute('BEGIN;')

            print('\tBEGIN;')

            # Поля во всей строке данных -  nconst
            comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{nconst}',
'\"tt0000009\"'::jsonb) WHERE iddata=" + str(i) + ';'
            # comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, \'{nconst}\',
\'\"tt0000009\"\'::jsonb) WHERE iddata=" + str(i) + ';'
            # print(comm_sql)
            start_time = datetime.datetime.now()
            cur.execute(comm_sql)
            end_time = datetime.datetime.now()
            nconst_use_time_ms = (end_time - start_time).microseconds
            print('\tnconst Конец испытания, время：  ', nconst_use_time_ms, ' ms')

            # Поля во всей строке данных -  name
```

```
            comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{name}', \"tt_name\"::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
            print('\tname Конец испытания, время：  ', name_use_time_ms, ' ms')

            # Поля во всей строке данных -  birthYear
            comm_sql = "UPDATE tb_jsonb SET imdata=jsonb_set(imdata::jsonb, '{birthYear}', \"2222\"::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
            print('\tbirthYear Конец испытания, время：  ', birthYear_use_time_ms, ' ms')

            # Поля во всей строке данных -  rols
            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) +
';'
            start_time = datetime.datetime.now()
            cur.execute(comm_sql)
            end_time = datetime.datetime.now()
            rols_use_time_ms = (end_time - start_time).microseconds
            print('\trols Конец испытания, время：  ', rols_use_time_ms, ' ms')

        except:
            cur.execute('ROLLBACK;')
            continue

        cur.execute('ROLLBACK;')
        df_tmp = pd.DataFrame([[len_row_jsonb, nconst_use_time_ms, name_use_time_ms,
birthYear_use_time_ms, rols_use_time_ms]],
                              columns=['len_row', 'nconst_ms', 'name_ms', 'birthYear_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_update_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)
```

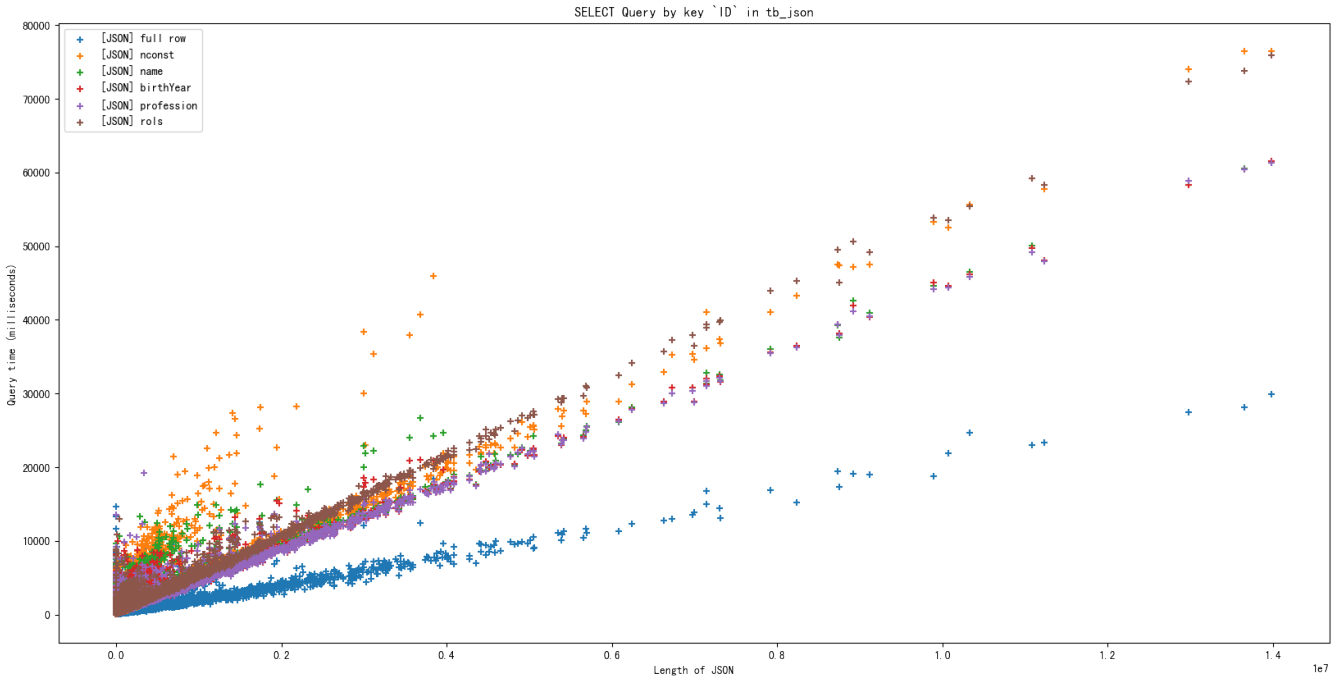
```
    """Построение графиков результатов"""
    plt.figure(figsize=(20, 10), dpi=100)
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['nconst_ms'].values,
                label='nconst')
    plt.scatter(x=df_counter['len_row'].values,
                y=df_counter['name_ms'].values,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


if __name__ == '__main__':
    # get_id_len_json_df()

    json_by_id_every_col_test()

    jsonb_by_id_every_col_test()

    json_update_by_id_every_col_test()

    jsonb_update_by_id_every_col_test()
```

1.2.2.1.1 Результат

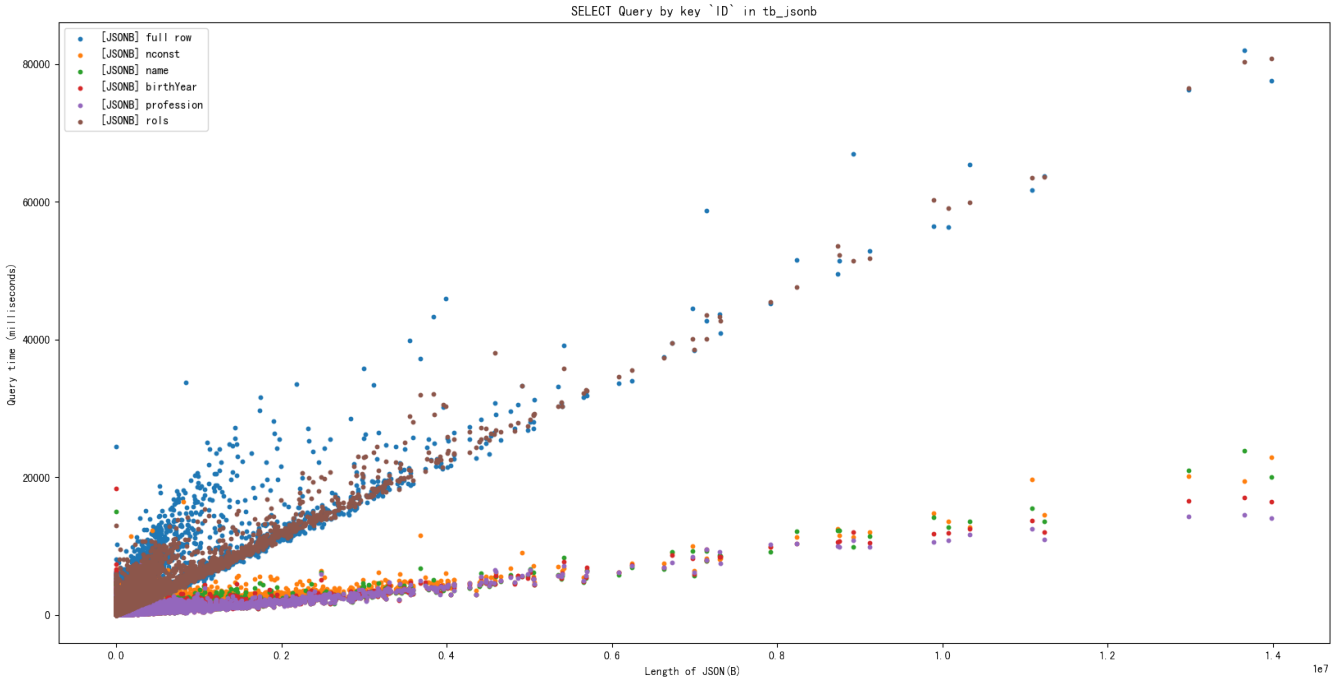
Сравнение времени, затрачиваемого JSON и JSONB при вставке данных в базу данных:

Типа	Времия
JSON	49s
JSONB	79.32s

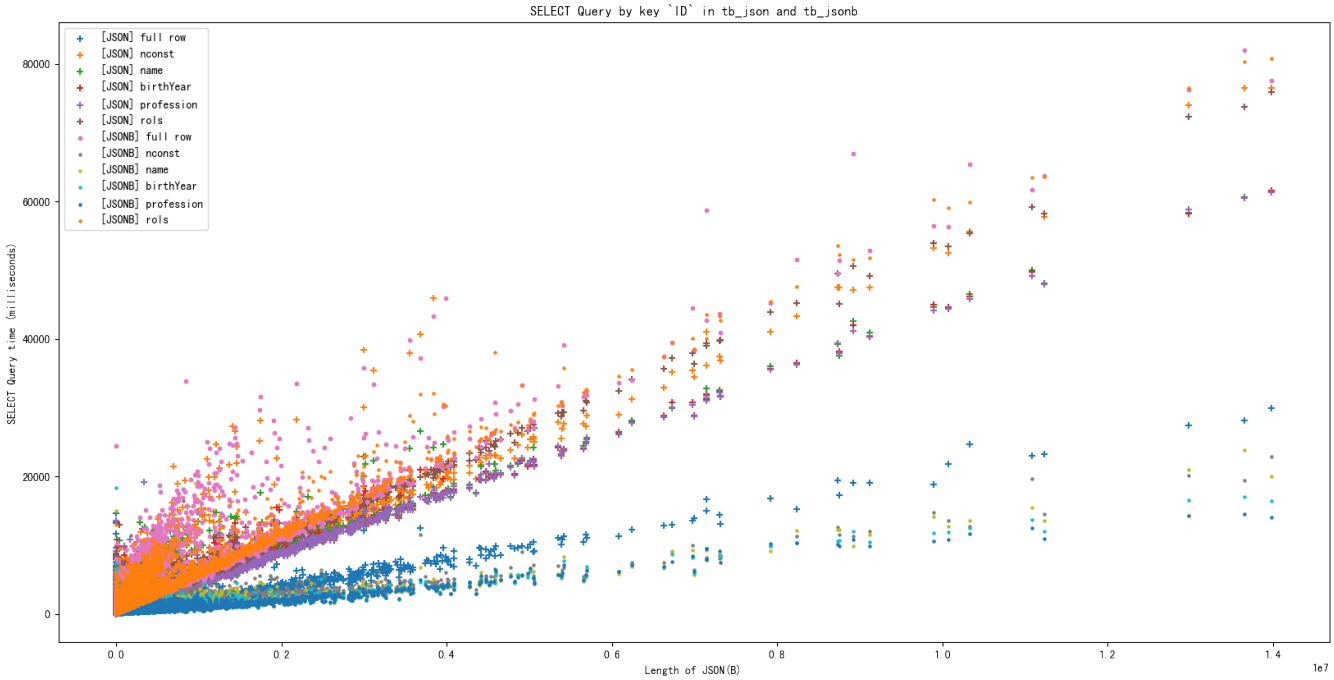
Сравнение времени SELECT для каждого поля JSON в таблице tb_json:



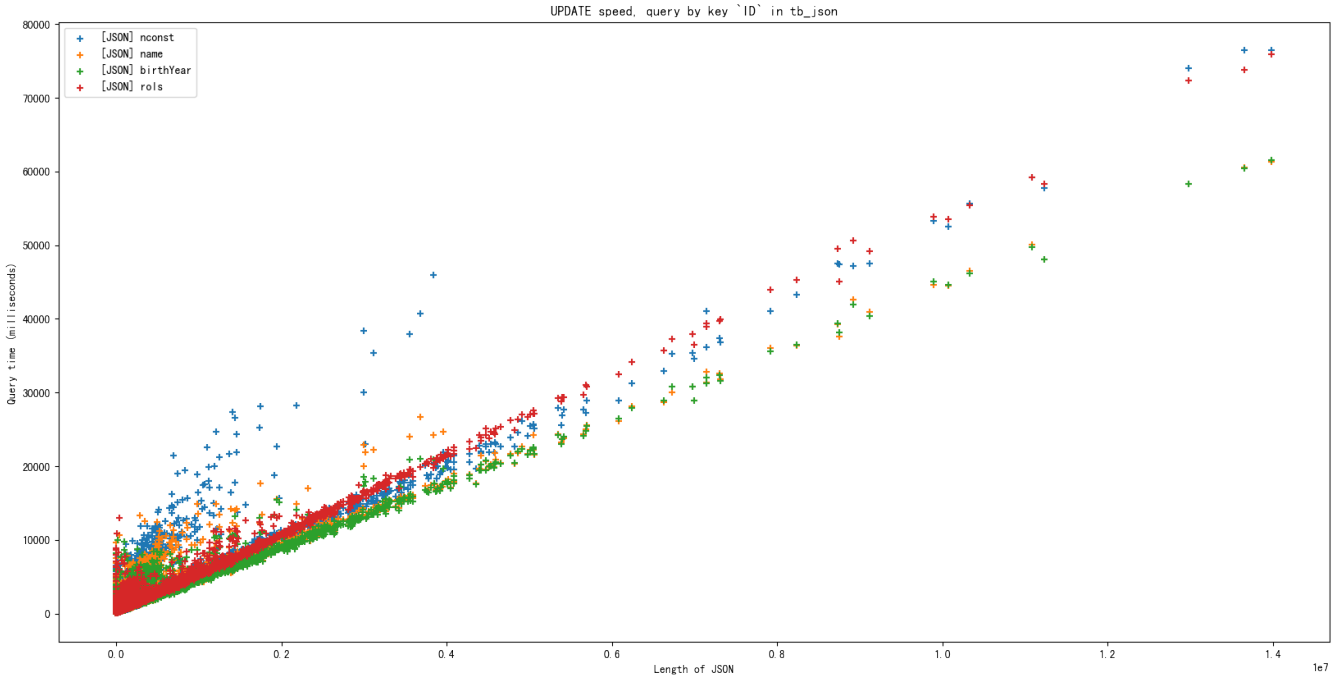
Сравнение времени SELECT для каждого поля JSONB в таблице tb_jsonb:



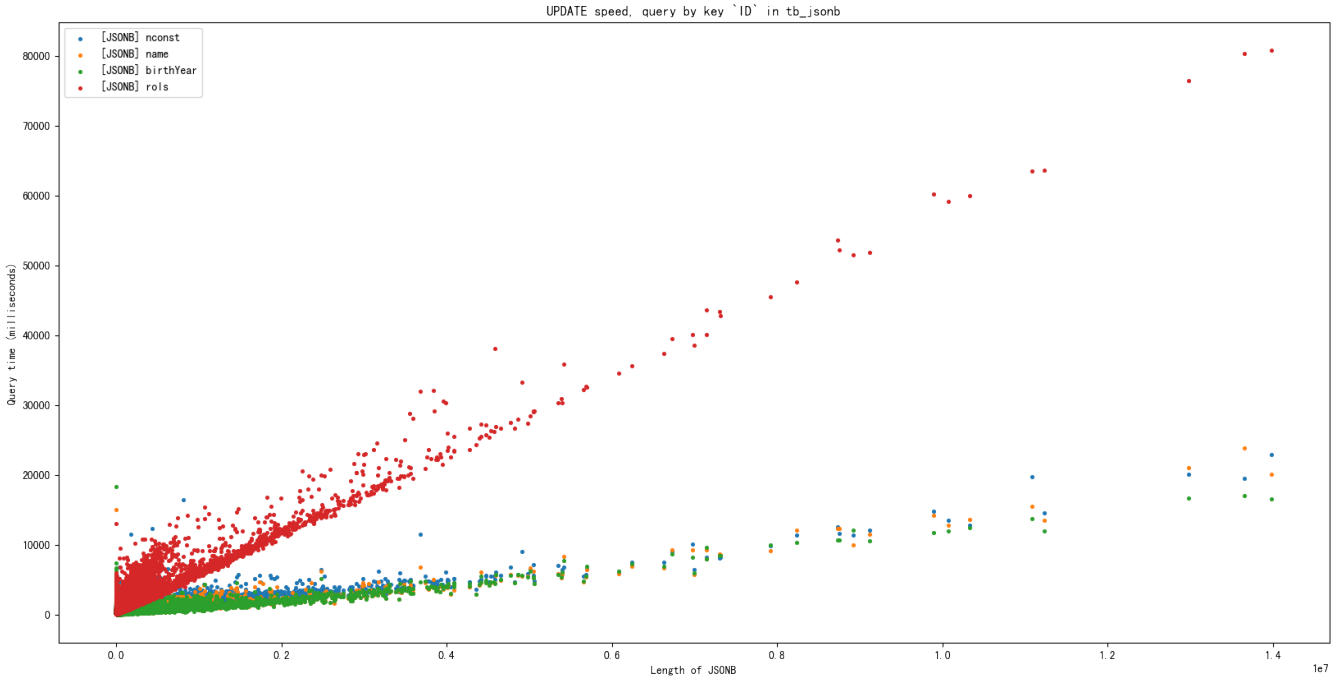
Сравнение скорости запросов к таблицам JSON и JSONB для всех полей:



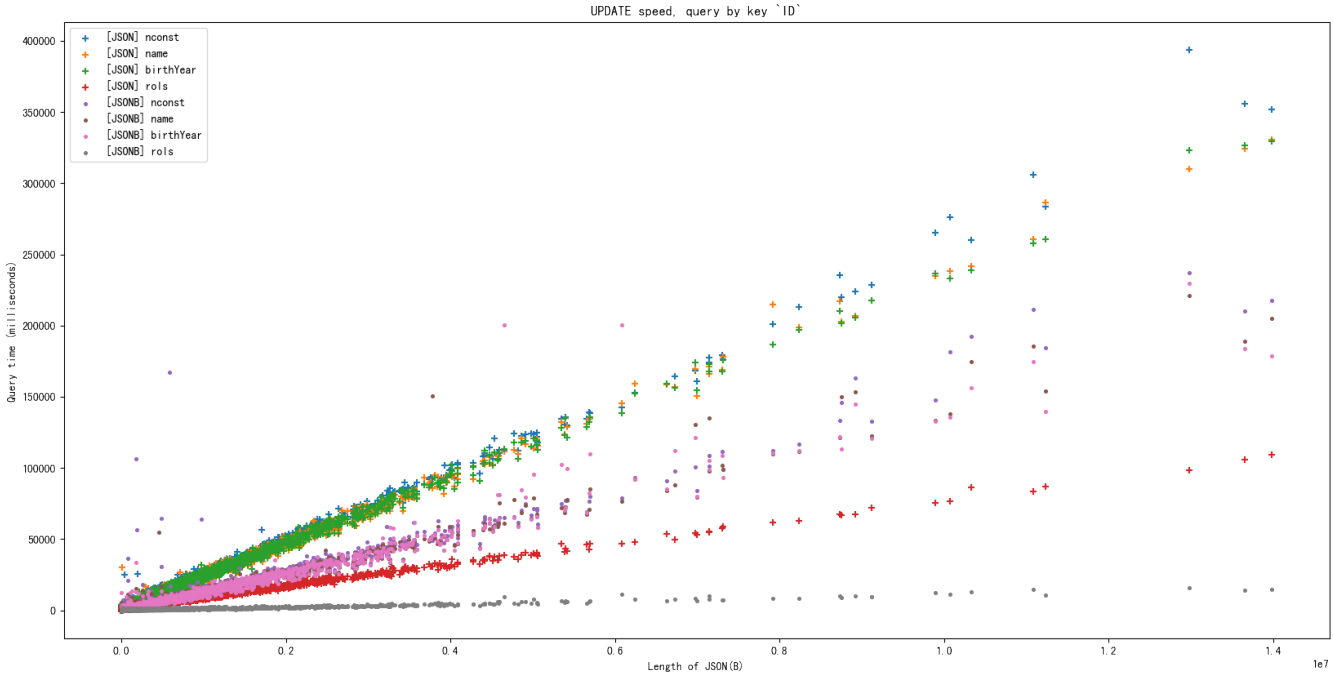
Сравнение времени UPDATE для каждого поля JSON в таблице tb_json:



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');  -- 1108811776 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1057 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE
iddata=51989;  -- 202 Byte
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB

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');  -- 1108811776 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1057 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');  -- 1108811776 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1057 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE
iddata=51989;  -- 202 Byte
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
```

```
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB

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');  -- 1108811776 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1057 MB
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB
ROLLBACK;
```

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

```
BEGIN;
SELECT pg_table_size('tb_jsonb');  -- 1108811776 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1057 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=3789;
 -- 4034997 Byte
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB

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;
 -- 4035007 [+10] Byte
SELECT pg_table_size('tb_jsonb');  -- 1112997888 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1061 MB
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB
ROLLBACK;
```

4. **[EXTERNAL] Сравнить изменение объема БД для актера с большим 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
```

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');  -- 1117184000 Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1065 MB
SELECT iddata, pg_column_size(imdata) , imdata, imdata->>'{name}' FROM tb_jsonb WHERE iddata=3789;
 -- 4034997 Byte
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB
```

```
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 [+] Byte
SELECT pg_table_size('tb_jsonb');  -- 1133617152 [+16433152] Byte
SELECT pg_size_pretty(pg_table_size('tb_jsonb'));  -- 1081 MB [+ MB]
SELECT SUM(size) FROM pg_ls_waldir(); -- 1056964608 Byte
SELECT pg_size_pretty(SUM(size)) FROM pg_ls_waldir(); -- 1008 MB
ROLLBACK;
```

1.2.2.2.1 Результат

Типа	Длина jsonb	Размер таблицы до изменеия	Размер таблицы после изменения	Прирост размера таблицы	размер WAL журнала транзакций до изменеия	размер WAL журнала транзакций после изменения
EXTENDED	202 Byte	1108811776 Byte (1057 MB)	1108811776 Byte (1057 MB)	+0 Byte	1056964608 Byte (1008 MB)	1108811776 Byte (1057 MB)
EXTERNAL	202 Byte	1108811776 Byte (1057 MB)	1108811776 Byte (1057 MB)	+0 Byte	1056964608 Byte (1008 MB)	1056964608 Byte (1008 MB)
EXTENDED	4034997 Byte	1108811776 Byte (1057 MB)	1112997888 Byte (1061 MB)	+4186112 Byte	1056964608 Byte (1008 MB)	1056964608 Byte (1008 MB)
EXTERNAL	4034997 Byte	1117184000 Byte (1065 MB)	1112997888 Byte (1081 MB)	+16433152 Byte	1056964608 Byte (1008 MB)	1056964608 Byte (1008 MB)

- 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