PostgreSQL-Consulting

data egret

PGDAY'17 RUSSIA

Учим слона танцевать рок-н-ролл

Maxim Boguk

Как научить слона танцевать Рок-н-ролл

Что? Серьезно?

- Некоторые типы запросов выполняются базой неоптимально
- Альтернативный подход
- Ручная реализация быстрого алгорима – серьезный рост производительности запроса

Цели

- Показать некоторые практически полезные альтернативые алгоритмы и приемы
- Кратко показать методы перевода из PL/PGSQL в SQL

Не цели

- Основы оптимизации запросов
- Ограничения планировщика запросов PostgreSQL
- Сравнение PL/PgSQL производительности с SQL

(Pl/PgSQL примеры написаны максимально простым для понимания образом и не обязательно являются максимально эффективной реализацией)

Рекомендуемые начальные навыки

- Знание PL/PgSQL
- Возможности PostgreSQL SQL:
- WITH [RECURSIVE]
- [JOIN] LATERAL
- UNNEST [WITH ORDINALITY]

О версии PostgreSQL

- PostgreSQL версия 9.6
- Будет работать на 9.5 и 9.4 без (серьезных) переделок
- Портирование на 9.3 и более ранние версии возможно, но потребует workaround реализации отсутствующих функций

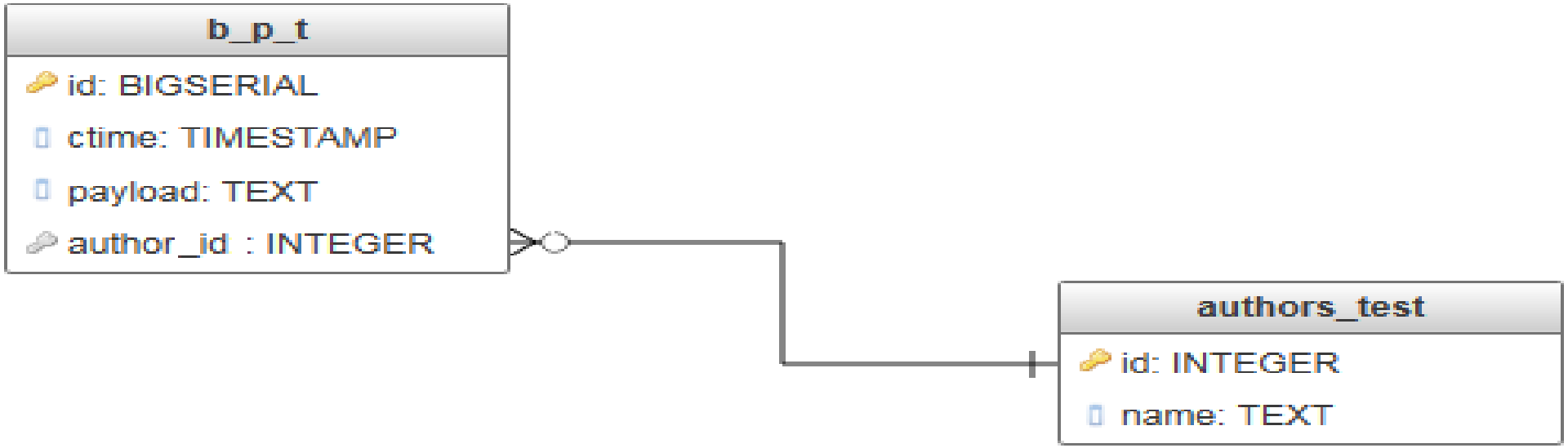
Структура презентации

- Описание проблемы
- Классическое решение - простой SQL запрос
- EXPLAIN ANALYZE
- Альтернативный алгоритм на PL/PgSQL
- Этот же алгоритм на SQL
- EXPLAIN ANALYZE
- Сравнение производительности

01

Подготовка тестовых данных

01 Подготовка тестовых данных Schema




```
CREATE UNIQUE INDEX blog_post_test_author_id_ctime_ukey
ON b_p_t USING btree (author_id, ctime);
```

01 Подготовка тестовых данных Часть 1

Create blog posts table:

```
DROP TABLE IF EXISTS b_p_t;

CREATE TABLE b_p_t (
  id BIGSERIAL PRIMARY KEY,
  ctime TIMESTAMP NOT NULL,
  author_id INTEGER NOT NULL,
  payload text);
```

01 Подготовка тестовых данных Часть 2

Populate the table with test data:

```
-- generate 10.000.000 blog posts from 1000 authors average 10000 post
-- per author from last 5 years, expect few hours run time
INSERT INTO b_p_t (ctime, author_id, payload)
SELECT
-- random in last 5 years
now()-(random()*365*24*3600*5)*'1 second'::interval AS ctime,
-- 1001 author
(random()*1000)::int AS author_id,
-- random text-like payload 100-2100 bytes long
(SELECT
   string_agg(substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0
   123456789           ', (random() * 72)::integer + 1, 1), '')
FROM generate_series(1, 100+i%10+(random() * 2000)::integer)) AS payload
-- 10M posts
FROM generate_series(1, 10000000) AS g(i);
```

01 Подготовка тестовых данных Часть 3

Populate the table with test data (continue):

```
--delete generated duplicates

DELETE FROM b_p_t where (author_id, ctime) IN (select author_id, ctime from b_p_t
   group by author_id, ctime having count(*)>1);

CREATE INDEX bpt_ctime_key on b_p_t(ctime);

CREATE UNIQUE INDEX bpt_a_id_ctime_ukey on b_p_t(author_id, ctime);

--create authors table

DROP TABLE IF EXISTS a_t;

CREATE TABLE a_t AS select distinct on (author_id) author_id AS id, 'author_'||
   author_id AS name FROM b_p_t;

ALTER TABLE a_t ADD PRIMARY KEY (id);

ALTER TABLE b_p_t ADD  CONSTRAINT author_id_fk FOREIGN KEY (author_id) REFERENCE
   a_t(id);

ANALYZE a_t;
```

02

IOS для запросов с OFFSET

02 IOS для запросов с OFFSET

- Запросы с большими offset – всегда медленные
- Чтобы получить 1.000.001’st row, база должна сначала пройти первые 1.000.000 строк
- Альтернатива: использование быстрого Index Only Scan чтобы пропустить первые 1.000.000 строк

02 IOS для запросов с OFFSET простой SQL

```
SELECT
*
FROM b_p_t
ORDER BY id
OFFSET 1000000
LIMIT 10
```

02 IOS для запросов с OFFSET простой SQL EXPLAIN

```
Limit (actual time=503..503 rows=10 loops=1)
   ->  Index Scan using b_p_t_pkey on b_p_t
   (actual time=0..386 rows=1000010 loops=1)
```

02 IOS для запросов с OFFSET PL/PgSQL

```
CREATE OR REPLACE FUNCTION ios_offset_test
  (a_offset BIGINT, a_limit BIGINT) RETURNS SETOF b_p_t LANGUAGE plpgsql
AS $function$
  DECLARE start_id b_p_t.id%TYPE;
BEGIN

  --find a starting id using IOS to skip OFFSET rows
  SELECT id INTO start_id FROM b_p_t
  ORDER BY id OFFSET a_offset LIMIT 1;
  --return result using normal index scan
  RETURN QUERY SELECT * FROM b_p_t WHERE id>=start_id
  ORDER BY ID LIMIT a_limit;

END;
$function$;
```

02 IOS для запросов с OFFSET
альтернативный SQL

```
SELECT bpt.* FROM
(
  --find a starting id using IOS to skip OFFSET rows
  SELECT id FROM b_p_t
  ORDER BY id OFFSET 1000000 LIMIT 1
) AS t,
LATERAL (
  --return result using normal index scan
  SELECT * FROM b_p_t WHERE id>=t.id
  ORDER BY id LIMIT 10
) AS bpt;
```

02 IOS для запросов с OFFSET
альтернативный SQL EXPLAIN

```
->  Index Only Scan using blog_post_test_pkey on b_p_t
(actual time=0..236 rows=1000001 loops=1)

 ->  Index Scan using blog_post_test_pkey on b_p_t
(actual time=0.016..0.026 rows=10 loops=1)
     Index Cond: (id >= b_p_t.id)
```

02 IOS для запросов с OFFSET
Сравнение производительности

OFFSET	Native	PL/PgSQL	Advanced
1000	0.7ms	0.5ms	0.4ms
10000	3.0ms	1.2ms	1.1ms
100000	26.2ms	7.7ms	7.4ms
1000000	273.0ms	71.0ms	70.9ms

03

Внесение LIMIT под (LEFT) JOIN

03 Внесение LIMIT под (LEFT) JOIN

- Комбинация JOIN, ORDER BY и LIMIT
- База данных делает JOIN для всех строк

  (а не только для LIMIT строк)
- Часто это лишняя работа
- Альтернатива: внести LIMIT+ORDER BY под JOIN

03 Внесение LIMIT под (LEFT) JOIN простой SQL

```
SELECT
*
FROM b_p_t
JOIN a_t ON a_t.id=author_id
WHERE author_id IN (1,2,3,4,5)
ORDER BY ctime
LIMIT 10
```

03 Внесение LIMIT под (LEFT) JOIN простой SQL EXPLAIN

```
->  Sort (actual time=345..345 rows=10 loops=1)

   ->  Nested Loop (actual time=0.061..295.832 rows=50194
  loops=1)

        ->  Index Scan using b_p_t_author_id_ctime_ukey on
  b_p_t (actual time=0..78 rows=50194 loops=1)
                     Index Cond: (author_id = ANY
  ('{1,2,3,4,5}'::integer[]))

                ->  Index Scan using a_t_pkey on a_t (actual
  time=0.002 rows=1 loops=50194)
```

03 Внесение LIMIT под (LEFT) JOIN PL/PgSQL

```
CREATE OR REPLACE FUNCTION join_limit_pulldown_test (a_authors BIGINT[], a_limit BIGINT)
  RETURNS TABLE (id BIGINT, ctime TIMESTAMP, author_id INT, payload TEXT, name TEXT)
   LANGUAGE plpgsql
AS $function$
  DECLARE t record;
BEGIN
  FOR t IN (
    -- find ONLY required rows first
    SELECT * FROM b_p_t WHERE b_p_t.author_id=ANY(a_authors)
    ORDER BY ctime LIMIT a_limit
  ) LOOP
    -- and only after join with authors
    RETURN QUERY SELECT t.*, a_t.name FROM a_t WHERE a_t.id=t.author_id;
  END LOOP;
END;
$function$;
```

03 Внесение LIMIT под (LEFT) JOIN альтернативный SQL

```
SELECT bpt_with_a_name.* FROM
-- find ONLY required rows first
(
  SELECT * FROM b_p_t WHERE author_id IN (1,2,3,4,5)
  ORDER BY ctime LIMIT 10
) AS t, LATERAL (
  -- and only after join with the authors
  SELECT t.*,a_t.name FROM a_t WHERE a_t.id=t.author_id
) AS bpt_with_a_name
-- second ORDER BY required
ORDER BY ctime LIMIT 10;
```

03 Pull down LIMIT under JOIN альтернативный SQL EXPLAIN

```
->  Nested Loop (actual time=68..68 rows=10 loops=1)

    ->  Sort (actual time=68..68 rows=10 loops=1)

        ->  Index Scan using b_p_t_author_id_ctime_ukey on b_p_t
  (actual time=0..49 rows=50194 loops=1)
              Index Cond: (author_id = ANY ('{1,2,3,4,5}'::integer[]))


    ->  Index Scan using a_t_pkey on a_t (actual time=0.002..0.002
  rows=1 loops=10)
                    Index Cond: (id = b_p_t.author_id)
```

03 Pull down LIMIT under JOIN сравнение производительности

```
author_id IN (1,2,3,4,5)  / LIMIT 10
```

простой SQL: 155ms

PL/PgSQL: 52ms

альтернативный SQL: 51ms

04

DISTINCT

04 DISTINCT

- DISTINCT on a large table always slow
- It scans whole table
- Even if the amount of distinct values low
- Alternative: creative index use to perform DISTINCT
- Technique known as LOOSE INDEX SCAN:

  https://wiki.postgresql.org/wiki/Loose_indexscan

04 DISTINCT простой SQL

```
select
distinct author_id
from b_p_t
```

04 DISTINCT простой SQL EXPLAIN

```
Unique (actual time=0..5235 rows=1001 loops=1)
   ->  Index Only Scan using b_p_t_author_id_ctime_ukey on
  b_p_t
  (actual time=0..3767 rows=9999964 loops=1)
```

04 DISTINCT PL/PgSQL

```
CREATE OR REPLACE FUNCTION fast_distinct_test () RETURNS SETOF INTEGER
  LANGUAGE plpgsql
AS $function$
  DECLARE _author_id b_p_t.author_id%TYPE;
BEGIN
  --start from least author_id
  SELECT min(author_id) INTO _author_id FROM b_p_t;
  LOOP
    --finish if nothing found
    EXIT WHEN _author_id IS NULL;
    --return found value
    RETURN NEXT _author_id;
    --find the next author_id > current author_id
    SELECT author_id INTO _author_id FROM b_p_t WHERE
    author_id>_author_id ORDER BY author_id LIMIT 1;
  END LOOP;
END;
$function$;
```

04 DISTINCT альтернативный SQL

```
WITH RECURSIVE t AS (
 --start from least author_id
 (SELECT author_id AS _author_id FROM b_p_t ORDER BY author_id LIMIT 1
 )
 UNION ALL
 SELECT author_id AS _author_id FROM t, LATERAL (
   --find the next author_id > current author_id
   SELECT author_id FROM b_p_t WHERE author_id>t._author_id
   ORDER BY author_id LIMIT 1
 ) AS a_id
)
--return found values
SELECT _author_id FROM t;
```

04 DISTINCT Advanced SQL EXPLAIN

```
  ->  Index Only Scan using b_p_t_author_id_ctime_ukey
on b_p_t b_p_t_1 (actual time=0.015..0.015 rows=1
loops=1)

  ->  Index Only Scan using b_p_t_author_id_ctime_ukey
on b_p_t
(actual time=0.007..0.007 rows=1 loops=1001)
```

04 DISTINCT
сравнение производительности

Native SQL: 2660ms

PL/PgSQL: 18ms

Advanced SQL: 10ms

05

DISTINCT ON

06 DISTINCT ON

- DISTINCT ON used when an application require fetch the latest data for ALL authors in single query
- DISTINCT ON on a large table always is performance killer
- Alternative: again, creative use of indexes and SQL saves the day

06 DISTINCT ON простой SQL

```
SELECT
DISTINCT ON (author_id)
*
FROM b_p_t
ORDER BY author_id, ctime DESC
```

06 DISTINCT ON простой SQL EXPLAIN

```
Unique (actual time=29938..39450 rows=1001 loops=1)
   ->  Sort (actual time=29938..37845 rows=9999964 loops=1)
         Sort Key: author_id, ctime DESC
         Sort Method: external merge  Disk: 10002168kB
         ->  Seq Scan on b_p_t (actual time=0.004..2472
  rows=9999964 loops=1)
```

06 DISTINCT ON PL/PgSQL

```
CREATE OR REPLACE FUNCTION fast_distinct_on_test() RETURNS SETOF b_p_t LANGUAGE plpgsql
AS $function$
  DECLARE _b_p_t record;
BEGIN
  --start from greatest author_id
  SELECT * INTO _b_p_t FROM b_p_t ORDER BY author_id DESC, ctime DESC LIMIT 1;
  LOOP
    --finish if nothing found
    EXIT WHEN _b_p_t IS NULL;
    --return found value
    RETURN NEXT _b_p_t;
    --latest post from next author_id < current author_id
    SELECT * FROM b_p_t INTO _b_p_t WHERE author_id<_b_p_t.author_id
    ORDER BY author_id DESC, ctime DESC LIMIT 1;
  END LOOP;
END;
$function$;
```

06 DISTINCT ON альтернативный SQL

```
WITH RECURSIVE t AS (
  --start from greatest author_id
  (SELECT * FROM b_p_t ORDER BY author_id DESC, ctime DESC LIMIT 1)
  UNION ALL
  SELECT bpt.* FROM t, LATERAL (
    --latest post from the next author_id < current author_id
    SELECT * FROM b_p_t WHERE author_id<t.author_id
    ORDER BY author_id DESC, ctime DESC LIMIT 1
  ) AS bpt
)
--return found values
SELECT * FROM t;
```

06 DISTINCT ON альтернативный SQL EXPLAIN

```
->  Index Scan Backward using b_p_t_author_id_ctime_ukey on b_p_t
(actual time=0.008..0.008 rows=1 loops=1)

      ->  Index Scan Backward using b_p_t_author_id_ctime_ukey on
  b_p_t
      (actual time=0.007..0.007 rows=1 loops=1001)
      Index Cond: (author_id < t_1.author_id)
```

06 DISTINCT ON альтернативный SQL (вариант 2)

```
--fast distinct(author_id) implementation (from part 4)
WITH RECURSIVE t AS (
  --start from least author_id
  (SELECT author_id AS _author_id FROM b_p_t ORDER BY author_id LIMIT 1)
  UNION ALL
  SELECT author_id AS _author_id
  FROM t, LATERAL (
    --find the next author_id > current author_id
    SELECT author_id FROM b_p_t WHERE author_id>t._author_id ORDER BY author_id LIMIT 1
  ) AS a_id
)
SELECT bpt.*
-- loop over authors list (from part 5)
FROM t, LATERAL (
   -- return the latest post for each author
   SELECT * FROM  b_p_t WHERE b_p_t.author_id=t._author_id ORDER BY ctime DESC LIMIT 1
) AS bpt;
```

06 DISTINCT ON over all table
Сравнение производительности

простой SQL:	3755ms
PL/PgSQL:	27ms
Альтернативный SQL:	13ms
Альтернативный(2) SQL:	18ms

06

А вот теперь будет весело: лента новостей

07 Лента новостей

Идея простейшей ленты новостей:

- Найти N самых свежих постов по списку авторов
- Может быть с OFFSET (страница)
- Легко реализовать
- Сложно сделать быстро

07 Лента новостей классический SQL

```
SELECT *
FROM b_p_t
WHERE
author_id IN (1,2,3,4,5)
ORDER BY ctime DESC
LIMIT 20 OFFSET 20
```

07 Лента новостей EXPLAIN

```
Limit (actual time=146..146 rows=20 loops=1)

  ->  Sort (actual time=146..146 rows=40 loops=1)
        Sort Key: ctime DESC
        Sort Method: top-N heapsort  Memory: 84kB

        ->  Index Scan using bpt_a_id_ctime_ukey on b_p_t
  (actual time=0.015..105 rows=50194 loops=1)
              Index Cond: (author_id = ANY
  ('{1,2,3,4,5}'::integer[]))
```

07 Лента новостей
Проблемы простейшей реализации

- Запрос получает все посты за все время по списку авторов
- Больше постов – медленнее запрос
- Длиннее список авторов – медленнее запрос

07 Лента новостей желаемые особенности

- Идеальный запрос не должен требовать более чем OFFSET+LIMIT обращений к индексу на b_p_t
- Должен быть быстрым для разумных значений OFFSET/LIMIT (100-1000)

07 Лента новостей альтернативная идея (входные данные)

pos	author_id
1	20
2	10
3	30
4	100
5	16

Начинаем с массива author_id (**_a_ids**):

```
'{20,10,30,100,16}'::int[]
```

07 Лента новостей альтернативная идея (начало)

pos	ctime
1	01-02-2017
2	03-02-2017
3	10-02-2017
4	28-02-2017
5	21-02-2017

Теперь используя идеи из предыдущих серий заполняем массив ctime самых новых постов этих авторов (**_a_ctimes**):

```
SELECT
array_agg((
  SELECT ctime
  FROM b_p_t WHERE author_id=_a_id
  ORDER BY ctime DESC LIMIT 1
) ORDER BY _pos)
FROM UNNEST('{20,10,30,100,16}'::int[])
WITH ORDINALITY AS u(_a_id, _pos)
```

07 Лента новостей альтернативная идея (продолжение 1)

pos	author_id
1	20
2	10
3	30
4	100
5	16

pos	ctime
1	01-02-2017
2	03-02-2017
3	10-02-2017
4	28-02-2017
5	21-02-2017

Находим позицию самого нового поста из **_a_ctimes**.
Это очевидно самый свежий пост из всех и будет первым постом возвращенным запросом.

```
SELECT pos FROM
UNNEST(_a_ctimes) WITH ORDINALITY AS u(a_ctime, pos)
ORDER BY a_ctime DESC NULLS LAST LIMIT 1
```

07 Лента новостей альтернативная идея (продолжение 2)

pos	author_id
1	20
2	10
3	30
4	100
5	16

pos	ctime
1	01-02-2017
2	03-02-2017
3	10-02-2017
4	28-02-2017
5	21-02-2017

Found	author_id	ctime
1	4	28-02-2017

Заменяем строку 4 в **_a_ctimes** значением ctime предыдущего поста того же автора.

```
SELECT ctime AS _a_ctime FROM b_p_t WHERE author_id=_a_ids[pos] AND
ctime<_a_ctimes[pos] ORDER BY ctime DESC LIMIT 1
```

07 Лента новостей альтернативная идея (окончание)

1	20
2	10
3	30
4	100
5	16

pos	ctime
1	01-02-2017
2	03-02-2017
3	10-02-2017
4	20-02-2017
5	21-02-2017

Found	author_id	ctime
1	4	28-02-2017
2	5	21-02-2017

“Rinse and repeat” шаги 2 и 3 собирая найденные строки, до получения LIMIT строк.

07 Лента новостей PL/PgSQL (начало)

```
CREATE OR REPLACE FUNCTION feed_test(a_authors INT[], a_limit INT, a_offset INT) RETURNS SETOF b_p_t
LANGUAGE plpgsql AS $function$
  DECLARE _a_ids INT[] := a_authors;
  DECLARE _a_ctimes TIMESTAMP[];
  DECLARE _rows_found INT := 0;
  DECLARE _pos INT;
  DECLARE _a_ctime TIMESTAMP;
  DECLARE _a_id INT;
BEGIN
  -- loop over authors list
  FOR _pos IN SELECT generate_subscripts(a_authors, 1) LOOP
    --populate the latest post ctime for every author
    SELECT ctime INTO _a_ctime FROM b_p_t WHERE author_id=_a_ids[_pos] ORDER
BY ctime DESC LIMIT 1;
    _a_ctimes[_pos] := _a_ctime;
  END LOOP;
```

07 Лента новостей PL/PgSQL (продолжение)

```
WHILE _rows_found<a_limit+a_offset LOOP
  --seek position of the latest post in ctime array
  SELECT pos INTO _pos
  FROM UNNEST(_a_ctimes) WITH ORDINALITY AS u(a_ctime, pos)
  ORDER BY a_ctime DESC NULLS LAST LIMIT 1;
  --get ctime of previous post of the same author
  SELECT ctime INTO _a_ctime FROM b_p_t
  WHERE author_id=_a_ids[_pos] AND ctime<_a_ctimes[_pos]
  ORDER BY ctime DESC LIMIT 1;
```

Лента новостей PL/PgSQL (окончание)

```
    --offset rows done, start return results
    IF _rows_found>=a_offset THEN
        RETURN QUERY SELECT * FROM b_p_t
        WHERE author_id=_a_ids[_pos] AND ctime=_a_ctimes[_pos];
    END IF;
    --increase found rows count
    _rows_found := _rows_found+1;
    --replace ctime for author with previous message ctime
    _a_ctimes[_pos] := _a_ctime;
  END LOOP;
END;
$function$;
```

07 Лента новостей альтернативая реализация SQL

```
WITH RECURSIVE
r AS (
  SELECT
    --empty result
    NULL::b_p_t AS _return,
    --zero rows found yet
    0::integer AS _rows_found,
    --populate author ARRAY
    '{1,2,3,4,5}'::int[] AS _a_ids,
    --populate author ARRAY of latest blog posts
    (SELECT
      array_agg((SELECT ctime FROM b_p_t WHERE author_id=_a_id ORDER BY ctime DESC LIMIT 1) ORDER BY _pos)
      FROM UNNEST('{1,2,3,4,5}'::int[]) WITH ORDINALITY AS u(_a_id, _pos)
    ) AS _a_ctimes
  UNION ALL
  SELECT
    --return found row to the result set if we already done OFFSET or more entries
    CASE WHEN _rows_found>=100 THEN (SELECT b_p_t FROM b_p_t WHERE author_id=_a_ids[_pos] AND ctime=_a_ctimes[_pos]) ELSE NULL END,
    --increase found row count
    _rows_found+1,
    --pass through the same a_ids array
    _a_ids,
    --replace current ctime for author with previous message ctime for the same author
    _a_ctimes[:_pos-1]||_a_ctime||_a_ctimes[_pos+1:]
  FROM r,
  LATERAL (SELECT _pos FROM UNNEST(_a_ctimes) WITH ORDINALITY AS u(_a_ctime, _pos) ORDER BY _a_ctime DESC NULLS LAST LIMIT 1) AS t1,
  LATERAL (SELECT ctime AS _a_ctime FROM b_p_t WHERE author_id=_a_ids[_pos] AND ctime<_a_ctimes[_pos] ORDER BY ctime DESC LIMIT 1) AS t2
  --found the required amount of rows (offset+limit done)
  WHERE  _rows_found<105
)
SELECT (_return).* FROM r WHERE _return IS NOT NULL ORDER BY _rows_found;
```

07 Альтернативная реализация SQL общая структура запроса

```
WITH RECURSIVE
r AS (
  --initial part of recursive union
  SELECT
  ...
  UNION ALL

  --main part of recursive union
  SELECT
  ...
  --exit condition
  WHERE  _rows_found<200
)

--produce final ordered result
SELECT (_return).* FROM r WHERE _return IS NOT NULL
ORDER BY _rows_found
```

07 Альтернативная реализация SQL инициализация

```
--empty result
NULL::b_p_t AS _return,
--zero rows found so far
0::integer AS _rows_found,
--author ARRAY
'{1,2,3,4,5}'::int[] AS _a_ids,
--populate author ARRAY of latest blog posts (see part 5)
(SELECT
  array_agg((
    SELECT ctime FROM b_p_t WHERE author_id=_a_id ORDER BY ctime DESC LIMIT 1
  ) ORDER BY _pos)
  FROM UNNEST('{1,2,3,4,5}'::int[]) WITH ORDINALITY AS u(_a_id, _pos)
) AS _a_ctimes
```

07 Альтернативная реализация SQL основная часть

```
--return found row to the result set if we already done OFFSET or more entries
CASE WHEN _rows_found>=100 THEN (
  SELECT b_p_t FROM b_p_t WHERE author_id=_a_ids[_pos] AND ctime=_a_ctimes[_pos]
) ELSE NULL END,
--increase found row count
_rows_found+1,
--pass through the same a_ids array
_a_ids,
--replace current ctime for author with previous message ctime
_a_ctimes[:_pos-1]||_a_ctime||_a_ctimes[_pos+1:]
FROM r,
LATERAL (
  SELECT _pos FROM UNNEST(_a_ctimes) WITH ORDINALITY AS u(_a_ctime, _pos)
  ORDER BY _a_ctime DESC NULLS LAST LIMIT 1
) AS t1,
LATERAL (
  SELECT ctime AS _a_ctime FROM b_p_t
  WHERE author_id=_a_ids[_pos] AND ctime<_a_ctimes[_pos]
  ORDER BY ctime DESC LIMIT 1
) AS t2
```

07 Производительность (с 10 авторами)

LIMIT	OFFSET	Native	PL/PgSQL	Alternative
1	0	180ms	1.0ms	1.0ms
10	0	182ms	1.8ms	1.3ms
10	10	182ms	1.9ms	1.9ms
10	1000	187ms	53.0ms	24.7ms
1000	0	194ms	100.8ms	35.1ms
100	100	185ms	13.2ms	6.2ms

08

Заключение

Заключение

- Эффективное выполнение некоторых популярных запросов требует альтернативного алгоритма
- Отладка альтернативного алгоритма проще производися с использованием PL/PgSQL
- алгоритм реализованный на SQL запросе – работает быстрее
- Процесс:

1. Реализация и отладка на PL/PgSQL
2. Конвертирование итога в SQL

Вопросы?

Совсем вопросов нет?

Really?

Спасибо!