PROFESSIONAL Postgres

НОВЫЕ ВОЗМОЖНОСТИ

PostgreSQL 11

Oleg Bartunov - research sci. Moscow University, major PostgreSQL contributor since Postgres95

PROFESSIONAL
Postgres

JIT компиляция

- Compile time 1 (--with-llvm)
- WHERE
- Target list
- Aggregates
- Projection
- Tuple deforming
- Compile time 2
- Pluggable
- Extensions

JIT компиляция

```
# select name, setting, category from pg_settings where name like '%jit%'

          name           | setting |               category
-------------------------+---------+-------------------------------------
 jit                     | on      | Query Tuning / Other Planner Options
 jit_above_cost          | 100000  | Query Tuning / Planner Cost Constants
 jit_debugging_support   | off     | Developer Options
 jit_dump_bitcode        | off     | Developer Options
 jit_expressions         | on      | Developer Options
 jit_inline_above_cost   | 500000  | Query Tuning / Planner Cost Constants
 jit_optimize_above_cost | 500000  | Query Tuning / Planner Cost Constants
 jit_profiling_support   | off     | Developer Options
 jit_provider            | llvmjit | File Locations
 jit_tuple_deforming     | on      | Developer Options
```

Сферический пример

```
select i as x1,
       case when i % 2 = 0 then i else null end as x2,
       case when i % 3 = 0 then i else null end as x3,
       case when i % 4 = 0 then i else null end as x4,
       case when i % 5 = 0 then i else null end as x5,
       case when i % 6 = 0 then i else null end as x6,
       case when i % 7 = 0 then i else null end as x7,
       case when i % 8 = 0 then i else null end as x8,
       case when i % 9 = 0 then i else null end as x9
       into t
from generate_series(0, 10000000)  i;

vacuum t;
analyze t;
```

Сферический пример

```
set max_parallel_workers=0;
set max_parallel_workers_per_gather=0;
set jit_above_cost=0;
set jit_inline_above_cost=0;
set jit_optimize_above_cost=0;
```

Сферический пример

```
set jit=off;
explain analyze
select count(*) from t where
        sqrt(pow(x9, 2) + pow(x8,2)) < 10000;

set jit=on;
explain analyze
select count(*) from t where
         sqrt(pow(x9, 2) + pow(x8,2)) < 10000;
```

Сферический пример

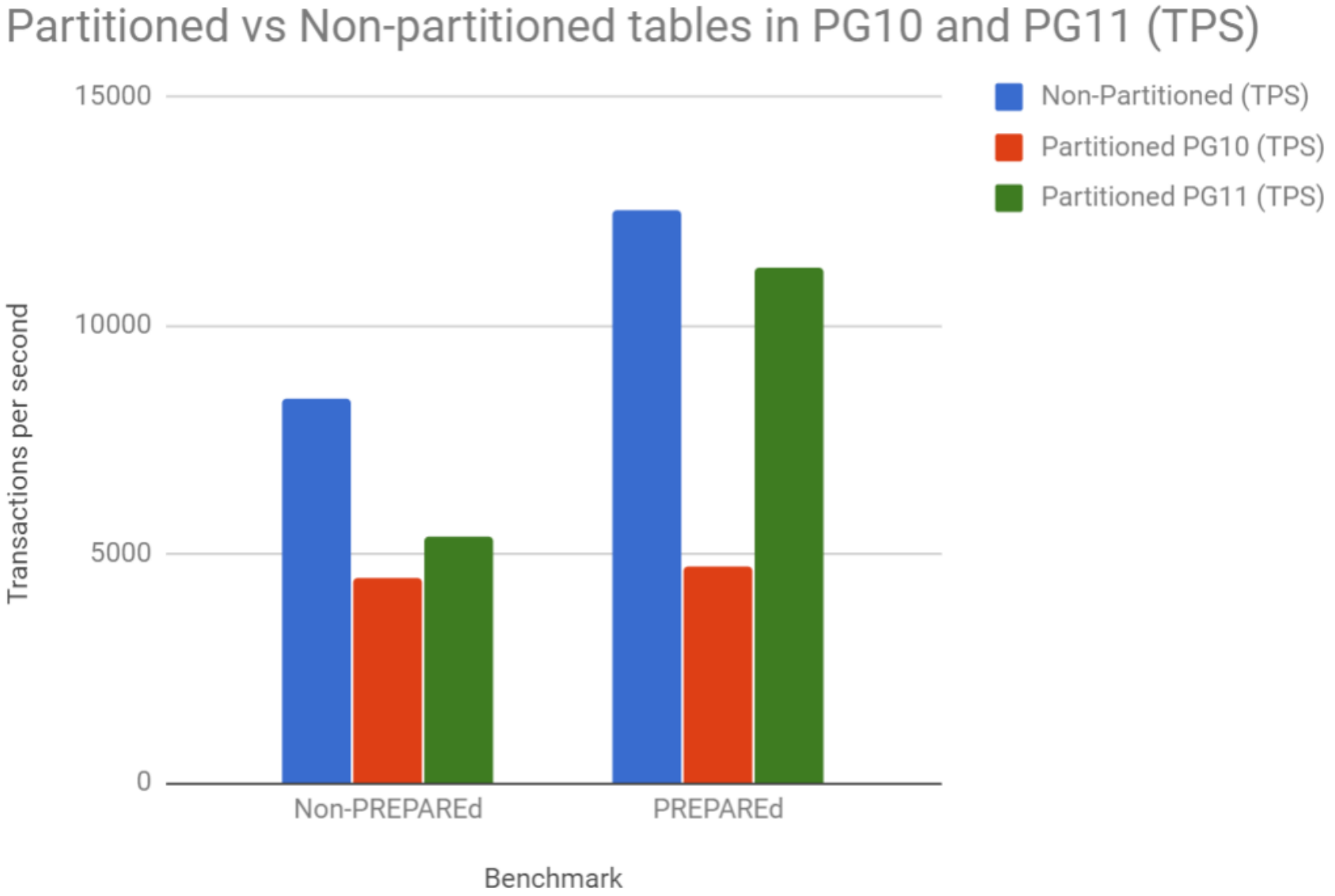
```
Planning Time: 0.71 ms
Execution Time: 1986.323 ms
                          VS
Planning Time: 0.060 ms
JIT:
  Functions: 4
  Generation Time: 0.911 ms
  Inlining: true
  Inlining Time: 23.876 ms
  Optimization: true
  Optimization Time: 41.399 ms
  Emission Time: 21.856 ms
Execution Time: 949.112 ms
```

Секционирование

- Первичный ключ!
  - Должен входить ключ секционирования
  - К сожалению, только уникальность
  - Триггеры
- По хешу
- Перемещение строк между секциями при обновлении ключа секционирования
- Секция по умолчанию — все, что не попало куда-то, попадает туда
- Оптимизация соединения секционированных таблиц, если они секционированы одинаково
- Агрегация отдельно для каждой секции и слияние результатов
- Создание локальных индексов на секциях при создании индекса на всю таблицу

Секционирование

Partition Pruning at Execution Time
Pgbench: prepared vs non-prepared



Секционирование

- Allow partition elimination during query execution (David Rowley, Beena Emerson)
  - ```
    create type sexnocc as (sex char(1), occ text);
    create table usr(id bigserial, so sexnocc);
    ```
  - ```
    insert into usr(so) select ('M','driver')::sexnocc
    from generate_series(1,1000000);
    insert into usr(so) select ('F','nurse')::sexnocc
    from generate_series(1,1000000);
    insert into usr(so)
    values(('F','driver')::sexnocc);
    ```
  - ```
    create table usr1(
     id bigint,
     sex char(1),
     occ text
    ) partition by list(occ);
    ```
  - **`alter table usr1 add primary key(id,occ)`**
    ```

Секционирование

- ```
  CREATE TABLE usr1_nurse PARTITION OF usr1
      FOR VALUES in ('nurse');
     CREATE TABLE usr2_driver PARTITION OF usr1
      FOR VALUES in ('driver');
  ```
- `alter table usr1 add primary key(id,`**`occ`**`);`
- `insert into usr1 select id, (u1.so).sex, (u1.so).occ from usr u1;`

- Как это работает?
  ```

Когда отсечение секций работает

- Литералы
  - explain analyze select * from usr1 where id between 1 and 1000001 and occ='nurse'
  - Append (cost=0.42..8.45 rows=1 width=16) (actual time=0.006..0.007 rows=1 loops=1)
    -> Index Scan using usr1_nurse_pkey on usr1_nurse (cost=0.42..8.45 rows=1 width=16) (actual time=0.006..0.006 rows=1 loops=1)
          Index Cond: ((id >= 1) AND (id <= 1000001) AND (occ = 'nurse'::text))
    Planning Time: 0.141 ms
    Execution Time: 0.019 ms

Параметры

- prepare sth1(int,int,text) as select * from usr1 where id between $1 and $2 and occ=$3
  explain analyze execute sth1(1,10000001,'nurse')
- ```
  Append  (cost=0.00..27906.00 rows=1000000 width=16) (actual time=0.008..154.041 rows=1000000 loops=1)
    ->  Seq Scan on usr1_nurse  (cost=0.00..22906.00 rows=1000000 width=16) (actual time=0.007..111.349 rows=1000000 loops=1)
          Filter: ((id >= 1) AND (id <= 10000001) AND (occ = 'nurse'::text))
  Planning Time: 0.178 ms
  Execution Time: 177.993 ms
  ```
  ```

InitPlan

- prepare sth(int,int,text) as select * from usr1 where id between $1 and $2 and occ=(select $3 from usr limit 1) limit 10
  explain analyze execute sth(1,10000001,'nurse')
- Limit (cost=0.02..0.30 rows=10 width=17) (actual time=0.036..0.042 rows=10 loops=1)
  InitPlan 1 (returns $0)
  -> Limit (cost=0.00..0.02 rows=1 width=32) (actual time=0.013..0.013 rows=1 loops=1)
  -> Seq Scan on usr (cost=0.00..36667.01 rows=2000001 width=32) (actual time=0.012..0.012 rows=1 loops=1)
  -> Append (cost=0.00..56776.02 rows=2000001 width=17) (actual time=0.035..0.039 rows=10 loops=1)
  -> Seq Scan on usr2_driver (cost=0.00..23870.02 rows=1000001 width=17) (**never executed**)
  Filter: ((id >= 1) AND (id <= 10000001) AND (occ = $0))
  -> Seq Scan on usr1_nurse (cost=0.00..22906.00 rows=1000000 width=16) (actual time=0.013..0.016 rows=10 loops=1)
  Filter: ((id >= 1) AND (id <= 10000001) AND (occ = $0))
  Planning Time: 0.495 ms
  Execution Time: 0.086 ms

Секционирование — не работает

- explain(analyze,verbose,buffers)
  select u1.* from usr u join usr1 u1 on u1.id=u.id
  where u.id between 1 and 10000 and (u.id%2)=0 and u1.occ=case when u.id%2=1 then 'nurse' else 'driver' end
- -> Append (cost=0.43..16.91 rows=2 width=17) (actual time=0.001..0.003 rows=1 loops=5000)
  Buffers: shared hit=35038
  -> Index Scan using usr2_driver_pkey on public.usr2_driver u1 (cost=0.43..8.45 rows=1 width=17) (actual time=0.001..0.001 rows=1 loops=5000)
  Output: u1.id, u1.sex, u1.occ
  Index Cond: ((u1.id = u.id) AND (u1.occ = CASE WHEN ((u.id % '2'::bigint) = 1) THEN 'fireman'::text ELSE 'driver'::text END))
  Buffers: shared hit=20038
  -> Index Scan using usr1_nurse_pkey on public.usr1_nurse u1_1 (cost=0.43..8.45 rows=1 width=16) (actual time=0.001..0.001 rows=0 loops=5000)
  Output: u1_1.id, u1_1.sex, u1_1.occ
  Index Cond: ((u1_1.id = u.id) AND (u1_1.occ = CASE WHEN ((u.id % '2'::bigint) = 1) THEN 'fireman'::text ELSE 'driver'::text END))
  Buffers: shared hit=15000

Секционирование

- Заметный шаг вперед
- Что еще надо:
  - Отсечение секций во время выполнения
    - Это уже не первый год умеет pg_pathman
  - Полноценные первичные ключи
- В PostgresPro Enterpise в модуле pg_pathman работают как литералы, параметры и InitPlan, так и join

Индексы

- Оптимизация вставки в монотонном порядке
- Сканирование страницы HASH индекса
- Predicate locking HASH, GiST and GIN (SERIALIZIBLE)
- HOT для функциональных индексов

Покрывающие индексы (B-Tree)

```
CREATE UNIQUE INDEX idx ON TBL
                (a, b) INCLUDE (c, d)
                   VS
CREATE UNIQUE INDEX idx1 ON tbl
                (a, b)
CREATE UNIQUE INDEX idx1 ON tbl
                (a, b, c, d)

... PRIMARY KEY (a, b) INCLUDE (c, d)
```

SP-GiST

- Компрессия или трансформация
- Полигоны
- ^@ - префиксный поиск, индексная поддержка
  SELECT * FROM table WHERE c ^@ "abc"

Производительность

- Bitmap index only scan
- Обновление FSM во время вакуума
- Возможность пропуска сканирования индексов во время вакуума
- Одновременный коммит конкурентных транзакций
- postgres_fdw - «пропихивание» соединений для UPDATE/DELETE
- toast_tuple_target

WAL

- Размер WAL сегмента в initdb
- Хранение WAL сегментов только с последнего checkpoint

Бэкап и репликация

- TRUNCATE в логической репликации
- PREPARE в логической репликации
- Исключение временных и UNLOGGED таблиц из base backup
- Контроль чексумм в потоковой репликации (для таблиц)
- Промотка позиции replication slot

Postgres PROFESSIONAL

Для DBA

- ALTER TABLE .. ADD COLUMN .. NOT NULL DEFAULT X
- VACUUM/ANALYZE сразу нескольких таблиц!

Параллелизм

- Параллельное построение btree-индексов
  - Потребуется хорошая дисковая подсистема
- Параллельное соединение по хешу с использованием общей хеш-таблицы для исполнителей
- Параллелизация некоторых UNION, CREATE TABLE AS, CREATE MATERIALIZED VIEW
- Передача LIMIT в параллельные исполнители

Параллельное создание индексов

```
alter table usr reset (parallel_workers)
create index on usr(lower((so).occ)) — 2 сек
alter table usr set (parallel_workers=2)
create index on usr(upper((so).occ)) — 1.8 сек
```

Запрос

- ```
  explain analyze
  select u1.* from usr u, usr1 u1 where
  u.id=u1.id+0
  ```
  ```

Hash join/Append
Последовательный вариант

```
Hash Join  (cost=61667.02..138443.05 rows=2000001 width=17) (actual
time=510.891..1373.250 rows=2000001 loops=1)
  Output: u1.id, u1.sex, u1.occ
  Hash Cond: ((u1.id + 0) = u.id)
  ->  Append  (cost=0.00..41776.01 rows=2000001 width=17) (actual
time=0.006..196.783 rows=2000001 loops=1)
        ->  Seq Scan on public.usr2_driver u1  (cost=0.00..16370.01
rows=1000001 width=17) (actual time=0.006..59.224 rows=1000001 loops=1)
              Output: u1.id, u1.sex, u1.occ
        ->  Seq Scan on public.usr1_nurse u1_1  (cost=0.00..15406.00
rows=1000000 width=16) (actual time=0.007..53.778 rows=1000000 loops=1)
              Output: u1_1.id, u1_1.sex, u1_1.occ
  ->  Hash  (cost=36667.01..36667.01 rows=2000001 width=8) (actual
time=506.897..506.897 rows=2000001 loops=1)
        Output: u.id
        Buckets: 2097152  Batches: 1  Memory Usage: 94510kB
        ->  Seq Scan on public.usr u  (cost=0.00..36667.01 rows=2000001
width=8) (actual time=0.022..181.269 rows=2000001 loops=1)
              Output: u.id
Planning Time: 0.121 ms
Execution Time: 1425.336 ms
```

Parallel hash join/parallel append

```
Gather  (cost=36417.01..88331.93 rows=2000001 width=17) (actual time=211.514..702.699
rows=2000001 loops=1)
  Output: u1.id, u1.sex, u1.occ
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Hash Join  (cost=35417.01..67331.92 rows=833334 width=17) (actual
time=198.436..531.482 rows=666667 loops=3)
        Output: u1.id, u1.sex, u1.occ
        Hash Cond: ((u1.id + 0) = u.id)
        Worker 0: actual time=191.990..581.212 rows=776591 loops=1
        Worker 1: actual time=191.995..580.015 rows=774769 loops=1
        ->  Parallel Append  (cost=0.00..24276.01 rows=833334 width=17) (actual
time=0.016..72.361 rows=666667 loops=3)
              Worker 0: actual time=0.020..84.256 rows=776591 loops=1
              Worker 1: actual time=0.021..83.558 rows=774769 loops=1
              ->  Parallel Seq Scan on public.usr2_driver u1  (cost=0.00..10536.67 rows=416667
width=17) (actual time=0.010..21.786 rows=333334 loops=3)
                    Output: u1.id, u1.sex, u1.occ
                    Worker 0: actual time=0.019..50.035 rows=776591 loops=1
                    Worker 1: actual time=0.008..10.019 rows=140514 loops=1
              ->  Parallel Seq Scan on public.usr1_nurse u1_1  (cost=0.00..9572.67 rows=416667
width=16) (actual time=0.014..31.811 rows=500000 loops=2)
                    Output: u1_1.id, u1_1.sex, u1_1.occ
                    Worker 1: actual time=0.020..39.448 rows=634255 loops=1
        ->  Parallel Hash  (cost=25000.34..25000.34 rows=833334 width=8) (actual
time=196.108..196.108 rows=666667 loops=3)
              Output: u.id
              Buckets: 2097152  Batches: 1  Memory Usage: 94656kB
              Worker 0: actual time=191.877..191.877 rows=647361 loops=1
              Worker 1: actual time=191.877..191.877 rows=649440 loops=1
              ->  Parallel Seq Scan on public.usr u  (cost=0.00..25000.34 rows=833334 width=8)
(actual time=0.025..64.161 rows=666667 loops=3)
                    Output: u.id
                    Worker 0: actual time=0.018..63.181 rows=647361 loops=1
                    Worker 1: actual time=0.017..63.209 rows=649440 loops=1
Planning Time: 0.125 ms
Execution Time: 763.410 ms
```

Оптимизатор

- Улучшен сбор статистики для сильно неравномерных распределений
- Улучшена статистика для <=, =>

Window function

- Соответствие стандарту SQL 2011

```
SELECT n,
  sum(n)over(partition BY n/3 rows between unbounded preceding and
unbounded following),
  first_value(n)over(ORDER BY n/3 groups 2 preceding)
FROM generate_series(1,12) AS gs(n)
ORDER BY n
```

```
 n  | sum | first_value
----+-----+-------------
  1 |   3 |           1
  2 |   3 |           1
  3 |  12 |           1
  4 |  12 |           1
  5 |  12 |           1
  6 |  21 |           1
  7 |  21 |           1
  8 |  21 |           1
  9 |  30 |           3
 10 |  30 |           3
 11 |  30 |           3
 12 |  12 |           6
(12 rows)
```

Полнотекст

```
# select websearch_to_tsquery('dog or cat');
----------------------
 'dog' | 'cat'

# select websearch_to_tsquery('dog -cat');
----------------------
 'dog' & !'cat'

# select websearch_to_tsquery('or cat');
----------------------
 'cat'
```

Json(b) & полнотекст

```
# select jsonb_to_tsvector
        ('{"a":"texts", "b":12}', '"string"');
------------------
 'text':1


# select jsonb_to_tsvector
        ('{"a":"texts", "b":12}', '["string", "numeric"]');
-------------------
 '12':3 'text':1
```

PL/* процедуры

```
CREATE PROCEDURE transaction_test1()
LANGUAGE plpgsql
AS $$
BEGIN
    FOR i IN 0..9 LOOP
        INSERT INTO test1 (a) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END
$$;

CALL transaction_test1();
```

pgbench

- Functions and operators
- \if \case
- --init-steps
- Random-seed, Zipfian, hashing

psql

Два новых способа выйти из psql:

exit или quit

COPY

Теперь можно вставить больше 2^32 строк за раз
(ждите во всех поддерживаемых версиях)

Postgres PROFESSIONAL

POSIX

- NaN ^ 0 = 1
- 1 ^ NaN = 1

References

1. https://www.postgresql.org/ - community site
2. https://wiki.postgresql.org/ - wiki
   https://wiki.postgresql.org/wiki/Main_Page/ru - wiki на русском, тонна информации !
   https://wiki.postgresql.org/wiki/FAQ - PostgreSQL FAQ
3. https://www.postgresql.org/list/ - mailing lists archive, https://postgrespro.com/list/
4. https://planet.postgresql.org/ - Planet PostgreSQL (blogs)
5. https://www.postgresql.org/about/events/ - Events (PGConf.eu, PGConf.ru, PGConf.asia, PGCon.org, PGConf.org)
6. https://postgrespro.ru/education - Postgres Professional education materials
7. https://pgxn.org/ - PostgreSQL Extension Network
8. https://postgres-slack.herokuapp.com/ - Slack channel
9. https://stackoverflow.com/questions/tagged/postgresql - StackOverflow
10. https://www.facebook.com/groups/postgres/ - FB (en)
11. https://www.facebook.com/groups/postgresql - FB (ru)
12. https://telegram.me/pgsql - Telegram (ru)
13. https://postgis.net/ - PostGIS
14. https://www.depesz.com/tag/pg11/ - Waiting for PG 11 by Depesz
15. https://paquier.xyz/tag/11/ - Postgres 11 highlight by Michael Paquier
16. https://habr.com/company/postgrespro/ - Блог Postgres Professional на Habr
17. https://www.youtube.com/watch?v=16Acsogh1LM - Новые возможности PG11 (Сигаев, Фролков)

СПАСИБО ЗА ВНИМАНИЕ !