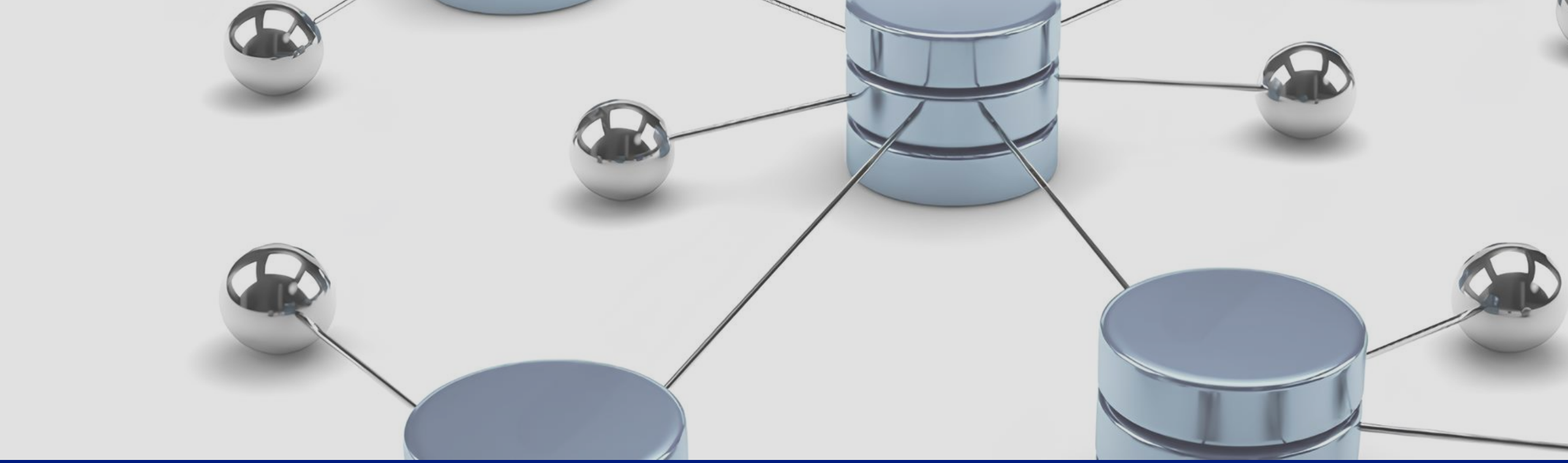
PostgresPro

Инструменты управления планами запросов

Борис Пищик
Александр Котин

28 января 2025

PGProDay



Что такое
Plan Management

Автоматическое
выявление деградации
производительности

Исправление проблем в
реальном времени

Контроль над
оптимизатором

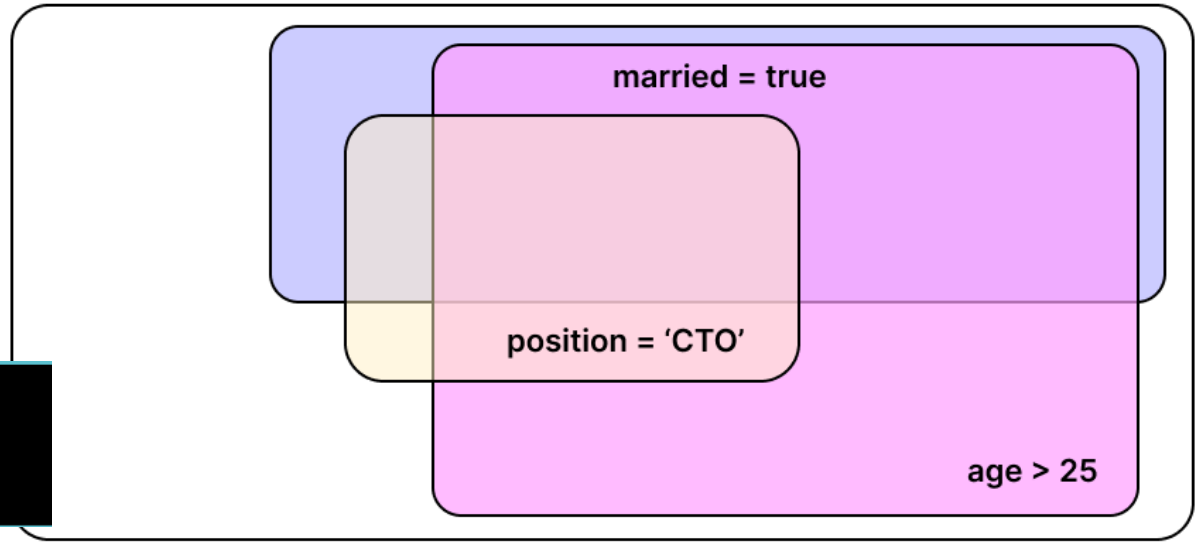
Что такое «план запроса» ?

```
EXPLAIN SELECT *
FROM users AS u1, messages AS m, users AS u2
WHERE u1.id = m.sender_id AND m.receiver_id = u2.id;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=540.00..439429.44 rows=10003825 width=27)
   Hash Cond: (m.receiver_id = u2.id)
   ->  Hash Join  (cost=270.00..301606.84 rows=10003825 width=23)
         Hash Cond: (m.sender_id = u1.id)
         ->  Seq Scan on messages m  (cost=0.00..163784.25 rows=10003825 width=19)
         ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
               ->  Seq Scan on users u1  (cost=0.00..145.00 rows=10000 width=4)
   ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
         ->  Seq Scan on users u2  (cost=0.00..145.00 rows=10000 width=4)
(9 rows)
```

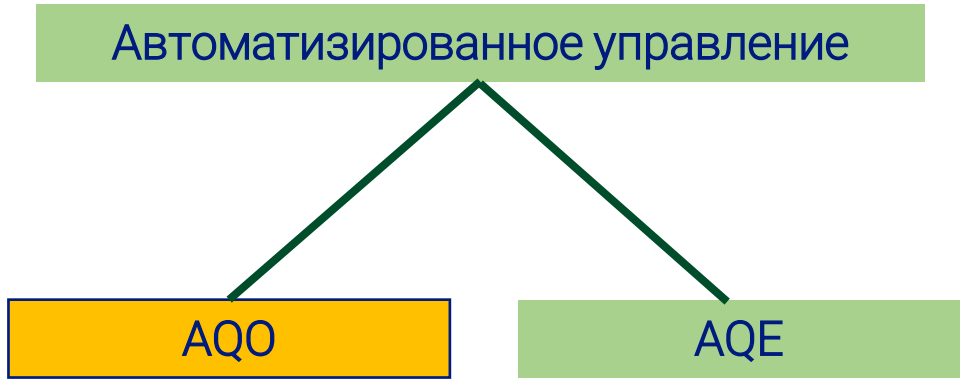
Зачем нужно управлять планами запросов ?

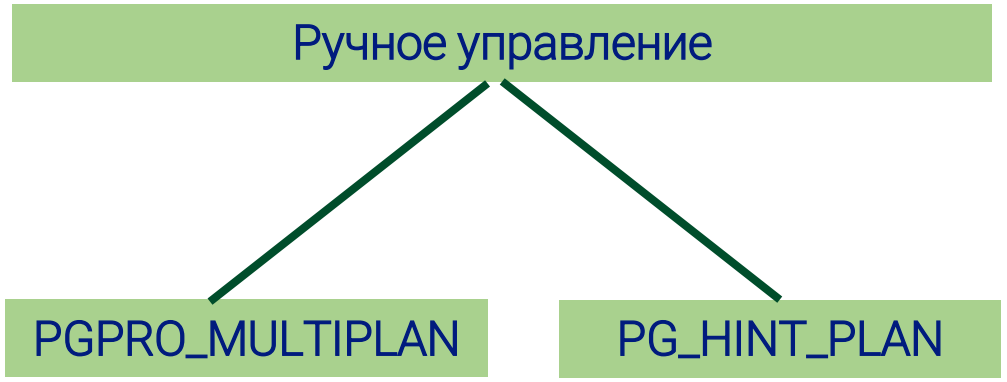
- Контроль за ключевыми запросами
- Стабилизация производительности
- Разные версии СУБД могут работать по-разному
- Статистика отстает от данных
- Оптимизатор ошибается

```
SELECT * FROM users
WHERE age > 25 AND married = true
AND position = 'CTO';
```

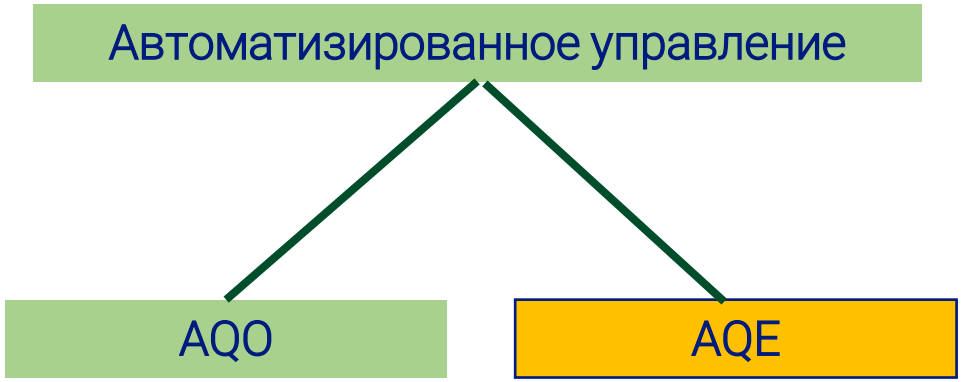


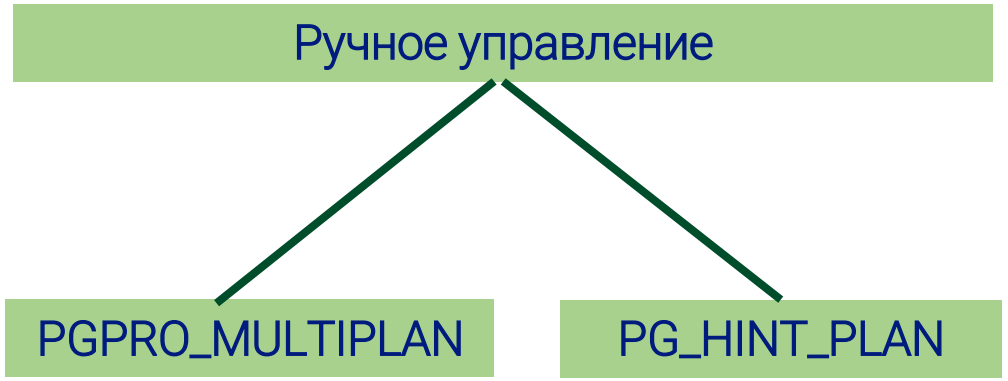
Инструменты Postgres Pro по управлению планами запросов





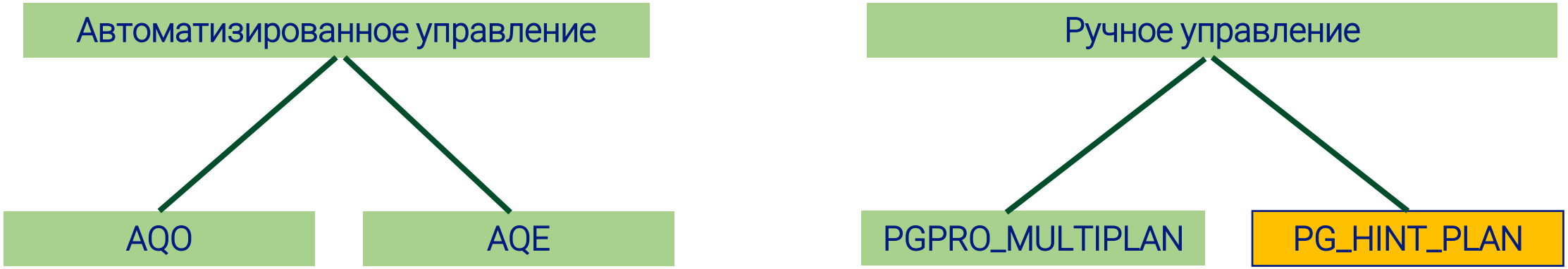
Инструменты Postgres Pro по управлению планами запросов





Инструменты Postgres Pro по управлению планами запросов

Инструменты Postgres Pro по управлению планами запросов



PostgresPro

AQO

Адаптивная оптимизация запросов

AQO – принцип работы

AQO – пример работы

PostgresPro

```
demo=# EXPLAIN (ANALYZE, TIMING OFF) SELECT t.ticket_no
  FROM flights f
         JOIN ticket_flights tf ON f.flight_id = tf.flight_id
         JOIN tickets t ON tf.ticket_no = t.ticket_no
 WHERE f.scheduled_departure > '1999-01-01'::timestamptz
   AND f.actual_arrival < f.scheduled_arrival + interval '1 hour'
   AND tf.fare_conditions = 'Business';
```

```
 Hash Join  (cost=9486.05..211054.58 rows=769494 width=14) (actual rows=771441 loops=1)
   AQO: rows=769494, error=-0%
   Hash Cond: (tf.flight_id = f.flight_id)
   ->  Nested Loop  (cost=0.86..188501.75 rows=859656 width=18) (actual rows=859656 loops=1)
         AQO: rows=859656, error=0%
         ->  Index Scan using ticket_flights_fare_conditions_idx on ticket_flights tf  (cost=0.43..102778.94 rows=73356 width=18) (act
ual rows=859656 lo
ops=1)
               AQO: rows=73356, error=-1072%
               Index Cond: ((fare_conditions)::text = 'Business'::text)
```

```
 Planning Time: 3.705 ms
 Execution Time: 1482.285 ms
 Using aqo: true
 AQO mode: AUTO
 AQO advanced: OFF
 Query hash: -4140461232606493427
 JOINS: 2
```

AQO – пример работы

PostgresPro

```
Gather  (cost=128055.06..286646.12 rows=771441 width=14) (actual time=839.764..1134.926 rows=771441 loops=1)
  AQO: rows=771441, error=0%, fss=-1169867161196126870
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Hash Join  (cost=127055.06..208502.02 rows=321434 width=14) (actual time=831.767..1070.611 rows=257147 loops=3)
        AQO: rows=771441, error=0%, fss=-1169867161196126870
        Hash Cond: (t.ticket_no = tf.ticket_no)
        ->  Parallel Seq Scan on tickets t  (cost=0.00..61924.68 rows=1229107 width=14) (actual time=0.044..158.490 rows=983286 loops=3)
              AQO: rows=2949857, error=0%, fss=-782071859942470067
        ->  Parallel Hash  (cost=121467.14..121467.14 rows=321434 width=14) (actual time=479.928..479.935 rows=257147 loops=3)
              Buckets: 262144  Batches: 8  Memory Usage: 6592kB
              ->  Parallel Hash Join  (cost=6286.35..121467.14 rows=321434 width=14) (actual time=34.990..419.595 rows=257147 loops=3)
                    AQO: rows=771441, error=0%, fss=-1710832237659048278
                    Hash Cond: (tf.flight_id = f.flight_id)
                    ->  Parallel Seq Scan on ticket_flights tf  (cost=0.00..114240.51 rows=358190 width=18) (actual time=0.065..298.964 rows=286552 loops=3)
                          AQO: rows=859656, error=0%, fss=4723136667746782336
                          Filter: ((fare_conditions)::text = 'Business'::text)
                          Rows Removed by Filter: 2510732
                    ->  Parallel Hash  (cost=4899.87..4899.87 rows=110919 width=4) (actual time=34.068..34.071 rows=62854 loops=3)
                          Buckets: 262144  Batches: 1  Memory Usage: 9504kB
                          ->  Parallel Seq Scan on flights f  (cost=0.00..4899.87 rows=110919 width=4) (actual time=0.038..17.278 rows=62854 loops=3)
                                AQO: rows=188563, error=0%, fss=-1465870163991728921
                                Filter: ((scheduled_departure > '1970-09-01 00:00:00+03'::timestamp with time zone) AND (actual_arrival < (scheduled_arrival
:interval)))
                                Rows Removed by Filter: 8768
Planning Time: 1.456 ms
Execution Time: 1162.346 ms
Using aqo: true
AQO mode: AUTO
AQO advanced: OFF
Query hash: -4140461232606493427
JOINS: 2
```

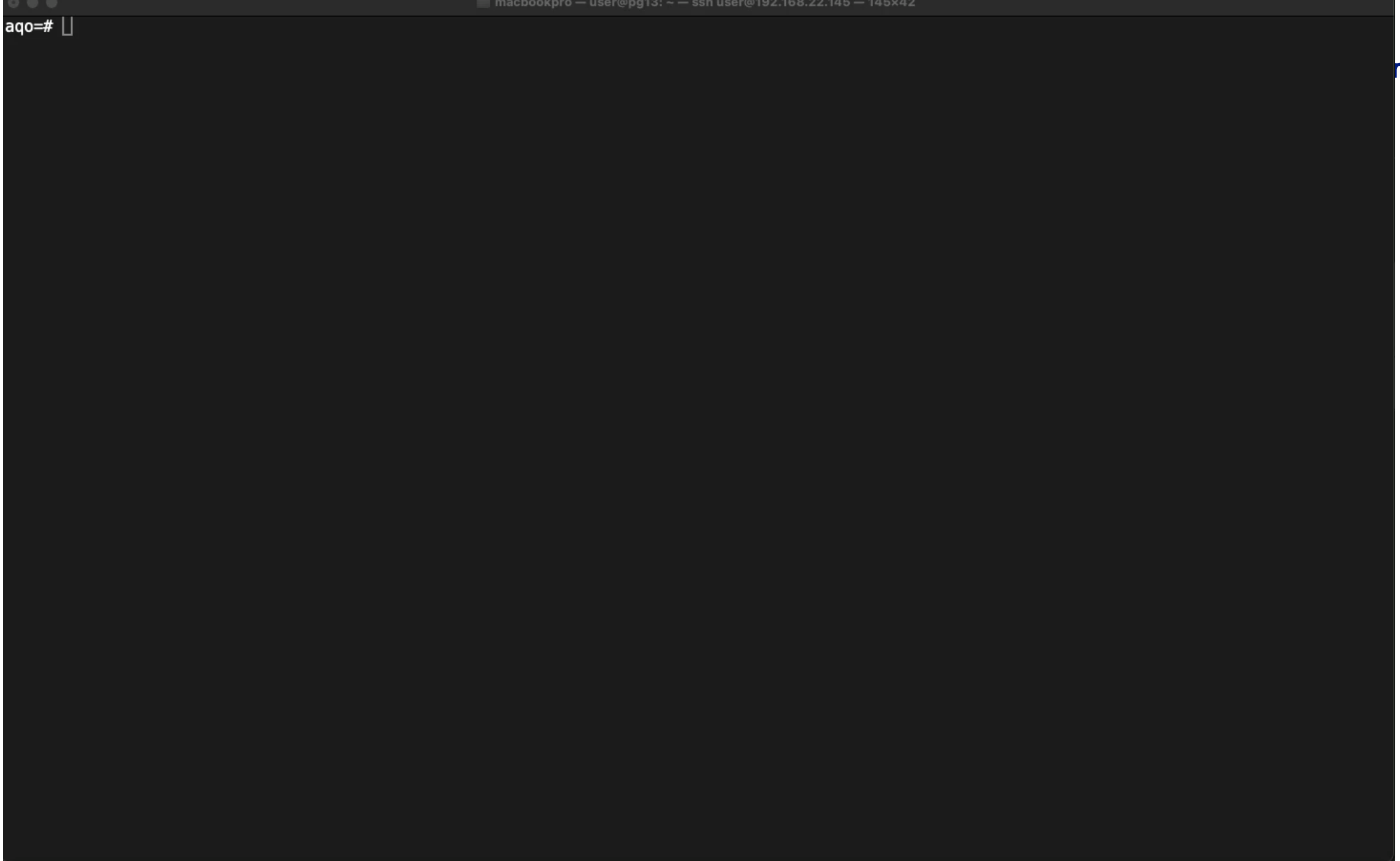
PostgresPro

AQO Enterprise

Новые возможности адаптивной оптимизации

AQO Enterprise – что нового ?

- Полностью автоматический режим постоянного обучения – autonomous mode
- Новая технология предсказания ошибок планировщика – delta rows
- Повышение стабильности работы



ro

PostgresPro

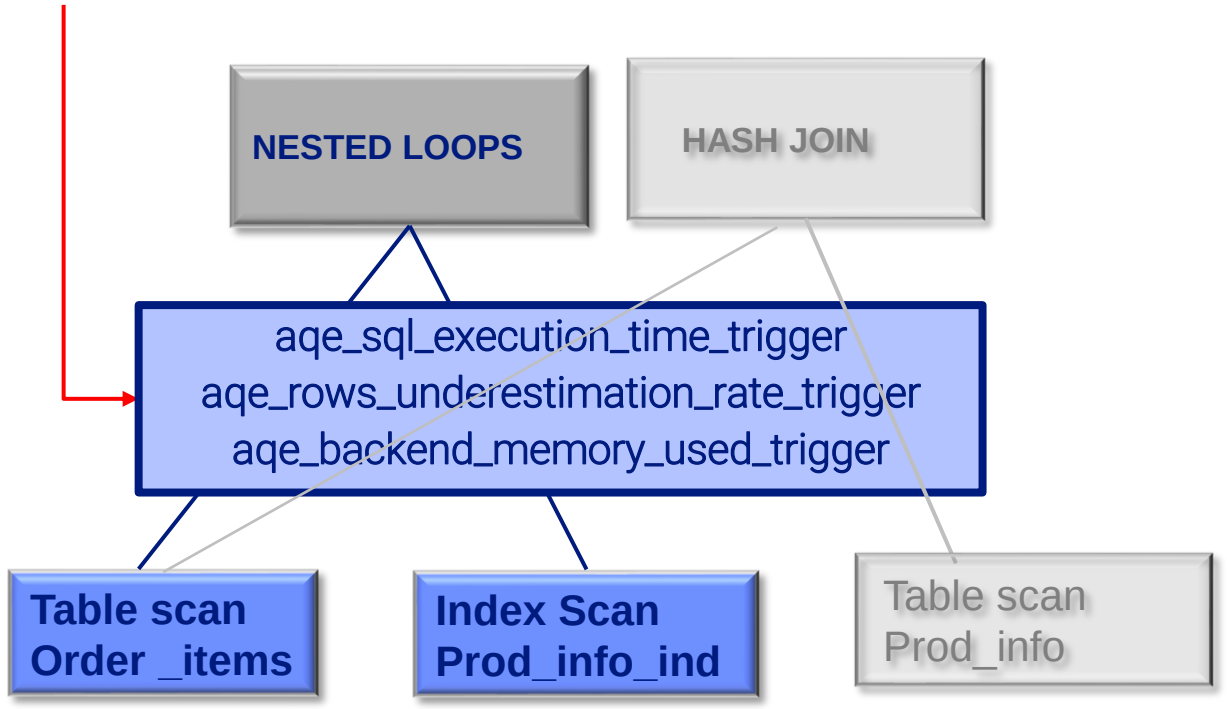
Adaptive Query Executor

Динамическое перестроение планов

Adaptive Query Executor (AQE) – механизм

- «Ошибки» оптимизатора исправляются прямо во время выполнения запроса!
- По установленному триггеру выполнение прерывается и запрос возвращается на повторную оптимизацию
- При генерации нового плана учитывается статистика выполнения
- В новом плане могут поменяться методы доступа к данным, типы и порядок соединений и т.д.

При повторном планировании оптимизатор решил поменять алгоритм соединения и метод доступа к таблице

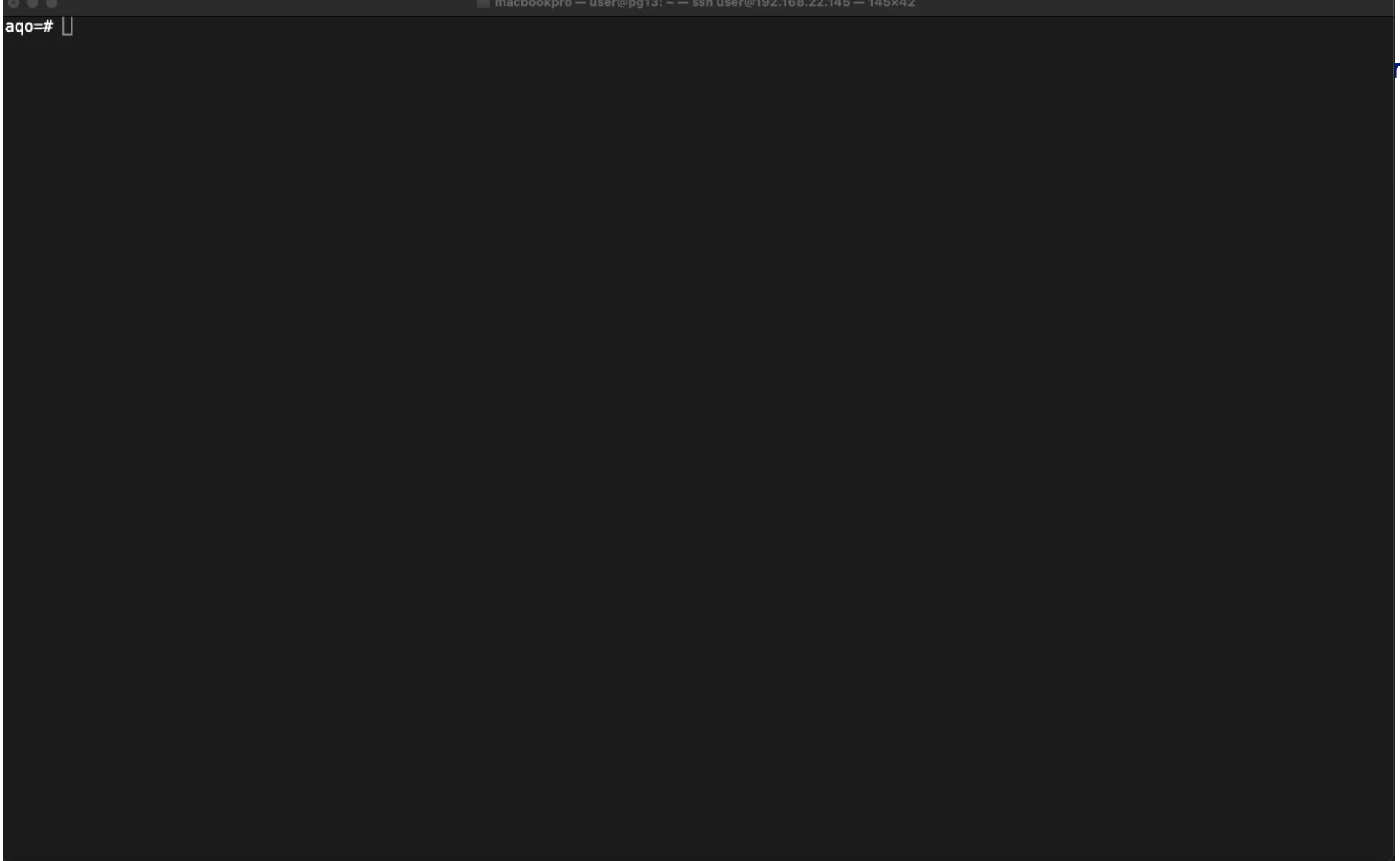


перестроение плана по событию-триггеру

Информация о перепланировании в журнале

PostgresPro

```
2024-04-01 09:05:25.319 UTC [1465035] LOG:  Replanning triggered by timeout 100 (0-th shift) ms.
        Attempt: 0.
        Duration: 101.893916 ms  plan:
        Query Text: select count(1)
        from pipeline p,
             tasks t
        where p.status = 'STARTED'
          and p.id = t.pipeline_id
          and t.status = 'NEW'
          and t.creation_time < now() - interval '5 seconds';
        Aggregate  (cost=4.08..4.09 rows=1 width=8) (actual time=101.894..101.894 rows=0 loops=1) (early terminated)
          NodeSign: 6618202469075138310
          Cardinality: -1
          Groups Number: -1
          Output: count(1)
          ->  Nested Loop  (cost=0.86..4.08 rows=1 width=0) (actual time=0.053..101.858 rows=106 loops=1) (early terminated)
                NodeSign: 13263165824905188009
                Cardinality: 106
                Groups Number: -1
                Join Filter: (p.id = t.pipeline_id)
                Rows Removed by Join Filter: 503265
                ->  Index Scan using pipeline_status_creation_time_idx on public.pipeline p  (cost=0.42..2.02 rows=1 width=8) (actual time=0.027..0.033 ro
ws=22 loops=1) (early terminated)
                      NodeSign: 11600703161693743624
                      Cardinality: 22
                      Groups Number: -1
                      Output: p.id, p.status, p.creation_time
                      Index Cond: (p.status = 'STARTED'::text)
                ->  Index Scan using tasks_status_creation_time_idx on public.tasks t  (cost=0.44..2.04 rows=1 width=8) (actual time=0.011..3.157 rows=228
80 loops=22) (early terminated)
                      NodeSign: 3773207335955725774
                      Cardinality: 22880
                      Groups Number: -1
                      Output: t.id, t.pipeline_id, t.status, t.creation_time
                      Index Cond: ((t.status = 'NEW'::text) AND (t.creation_time < (now() - '00:00:05'::interval)))
        Settings: effective_cache_size = '24659200kB', random_page_cost = '1.1', work_mem = '1MB', search_path = 'dev, public'
        Query Identifier: 2465789657660344514
```



ro

AQE – что нового ?

- Полный набор триггеров для перепланирования:
  - время выполнения (aqe_sql_execution_time_trigger)
  - ошибка планировщика (aqe_rows_underestimation_rate_trigger)
  - заполнение памяти бэкенда (aqe_backend_memory_used_trigger)
- Смарт-логика срабатывания триггеров
- Поддержка Extended Protocol и Prepared Statements !

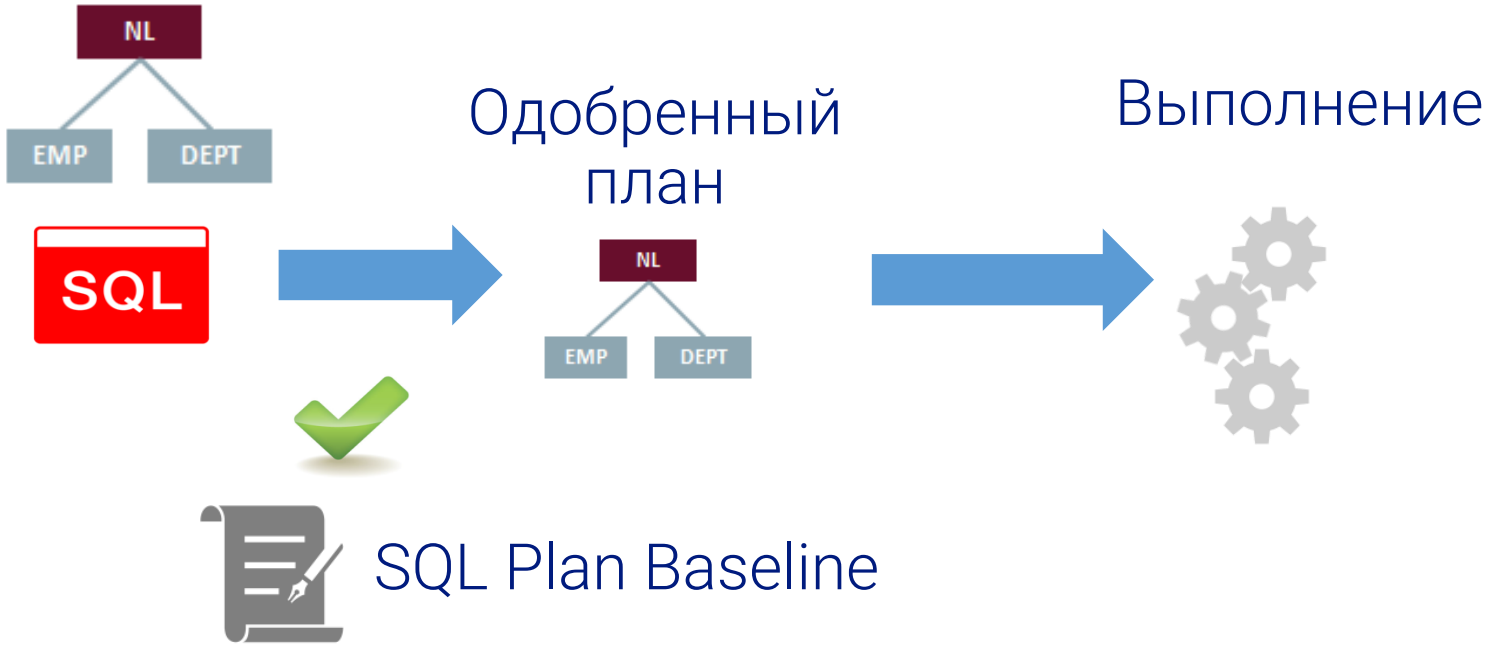
PostgresPro

PGPRO_MULTIPLAN

Развитие технологии стабилизации планов

SQL Multi-Plan Management – принцип работы

- SQL запрос поступает на обработку в Postgres Pro
- Оптимизатор генерирует план выполнения и сверяется со списком сохраненных планов (=baseline)
- Если сгенерированный план присутствует в списке и имеет флаг approved, то он идёт на выполнение
- Если плана в списке нет, к выполнению допускается самый «дешевый» approved-план из списка (в режиме захвата он также добавляется с флагом unapproved)

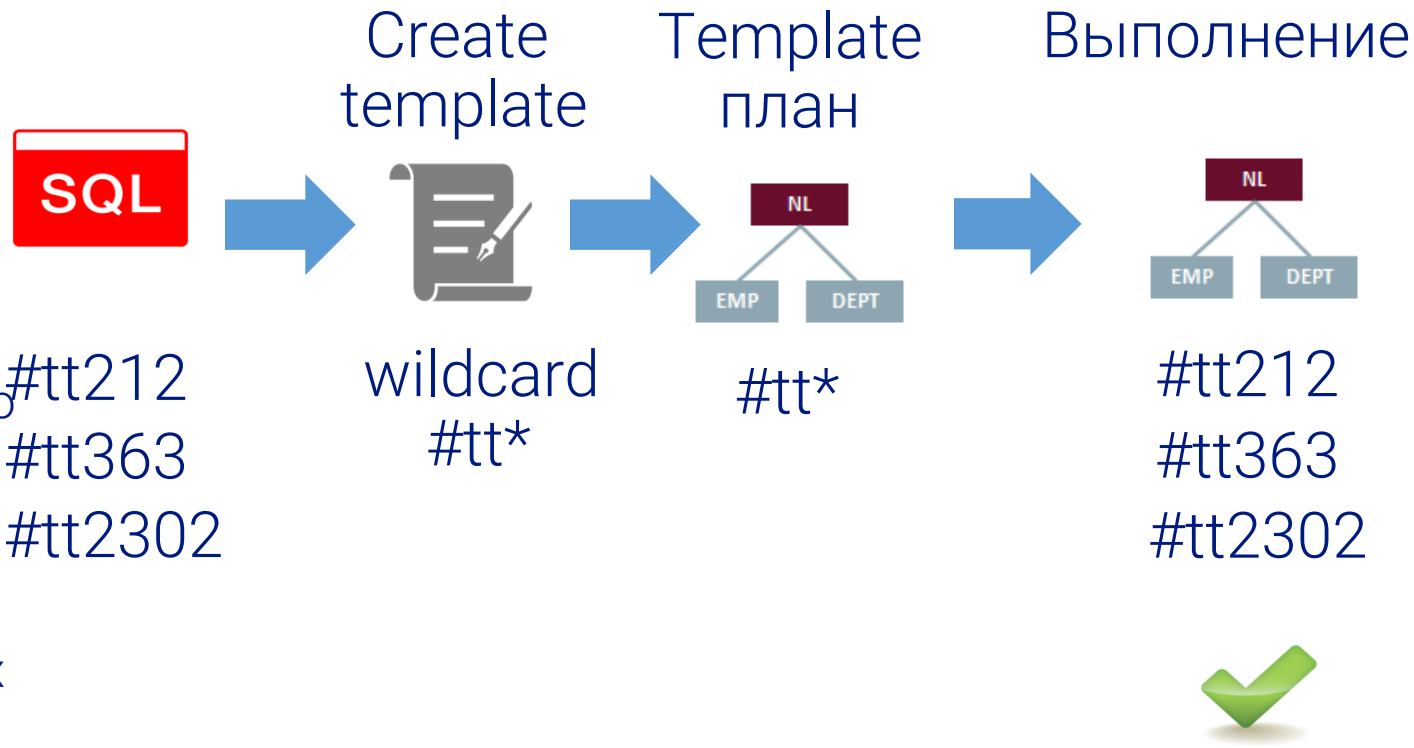


PGPRO_MULTIPLAN ДЕМО (Baselines)

- Включим режим записи планов pgpro_multiplan, выполним тестовые запросы с разными константами и сформируем историю планов
- Посмотрим на сохранённые планы и разрешим их к выполнению – сформируем baseline
- Попробуем разные варианты планов и последовательно добавим их в baseline
- Сбросим все настройки и отключим pgpro_multiplan

Wildcards для имен таблиц и шаблонные планы

- У вас есть запросы, использующие динамические имена таблиц
- Вы хотите использовать единый план запроса с разными именами таблиц
- Задаете wildcards через GUC
- Выполняете запрос и создаете из него шаблон
- Сохраняете план-шаблон
- Для всех запросов, удовлетворяющих структуре и регулярному выражению, выполняется шаблонный директивный план «template hintset plan»



PGPRO_MULTIPLAN ДЕМО (Wildcards)

- Установим шаблон имени таблицы — wildcard
- Зарегистрируем запрос, использующий имена таблиц по шаблону
- Зафиксируем план запроса в режиме template
- Выполним запрос и убедимся, что используется «замороженный» план
- Поменяем имена таблиц в запросе и режимы оптимизатора и убедимся, что шаблонный план применяется

PGPRO_MULTIPLAN – развитие технологии фиксации планов

PostgresPro

- Аналог Oracle SQL Plan Management
- История планов и управление статусом
- Списки «хороших» планов — Baselines
- Wildcards для имён объектов

PostgresPro

Планы на 2025 год *

- Real-Time Plan Management
- SQL Tuning Sets
- Интеграция Plan Management в PPEM
- AQE Observability
- Feedback !

* планы и сроки могут меняться

PostgresPro

Спасибо за внимание!