PostgresPro

PGCONF.RUSSIA 2024

Инструменты диагностики и примеры оптимизации запросов

Пётр Петров (p.petrov@postgrespro.ru)

PostgresPro

Темы

1. Средства отслеживания состояния работы СУБД PostgreSQL.
2. Модули отслеживания долгих запросов.
3. Примеры оптимизации запросов.
4. Особенности применения настроечных параметров СУБД.

PostgresPro

Zabbix-агент Mamonsu

Mamonsu – активный агент по сбору метрик для Zabbix:

- Работает в том числе с отечественными ОС (Alt, Astra Linux).
- 1 агент = 1 экземпляр СУБД.
- Версия PostgreSQL, начиная с 9.5.
- Дополнительный набор метрик для Postgres Pro Enterprise.

PostgresPro

postgres_exporter

- Собирает метрики, запрашиваемые prometheus.
- Для отображения метрик grafana запрашивает данные с prometheus.
- Заявляется, что 1 агент может обслуживать несколько экземпляров СУБД, но пока является бета-особенностью.
- Минимальная версия PostgreSQL 11.

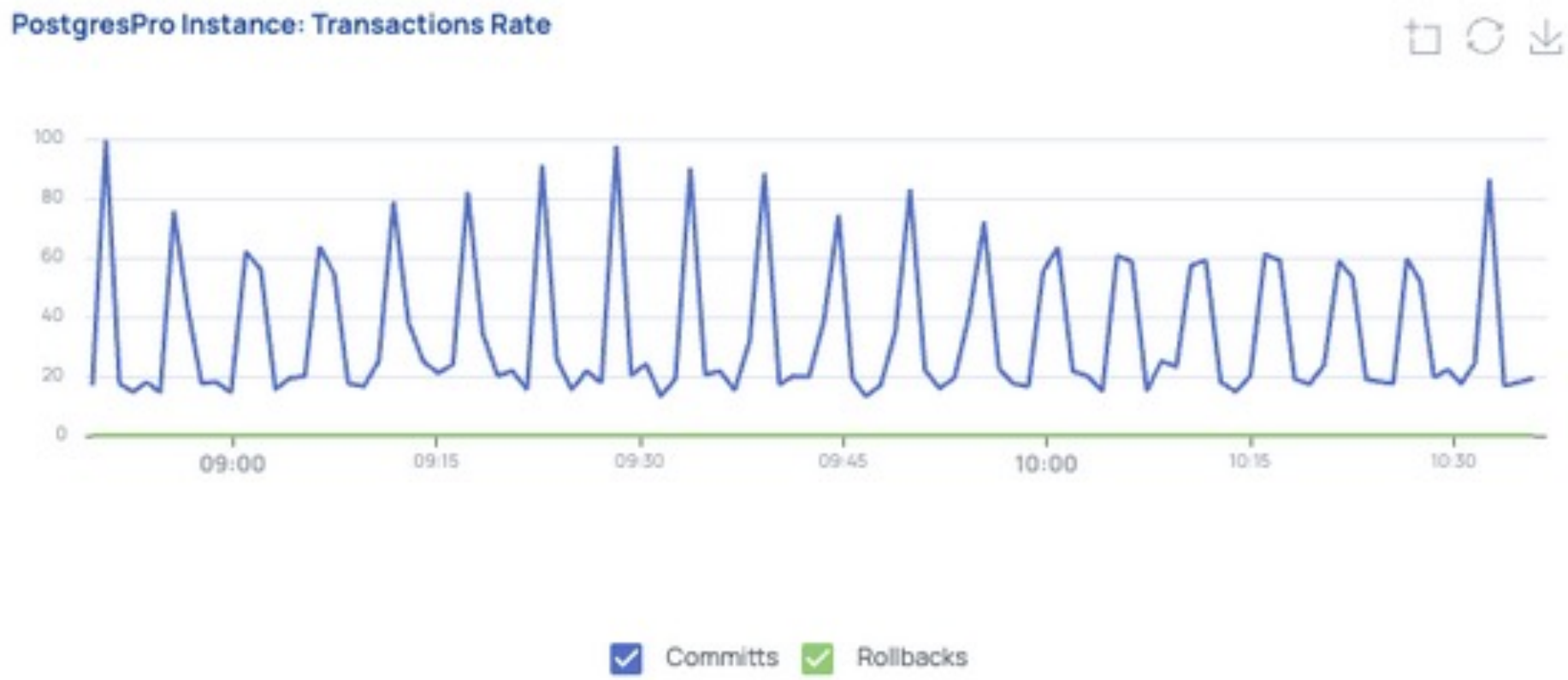
PostgresPro

Postgres Pro Enterprise Manager (PPEM)

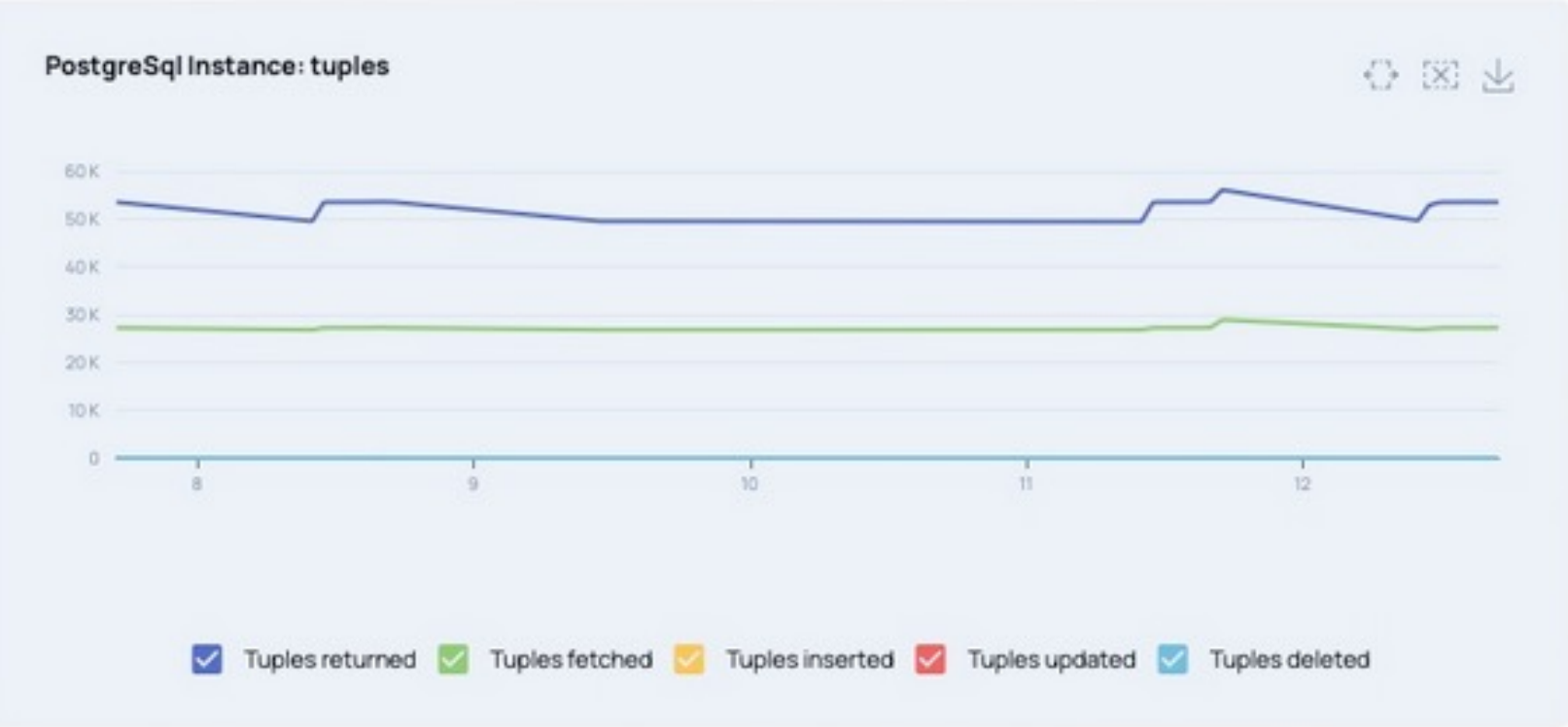
Продукт компании Postgres Professional, упрощающий выполнение задач администрирования:

1. Создание, запуск и остановка экземпляров СУБД.
2. Резервное копирование и восстановление.
3. Отображение основных метрик производительности в виде графиков с возможностью создания собственных.

Отображение статистики по транзакциям в PPEM



Статистика по строкам в PPEM



PostgresPro

Модули отслеживания долгих запросов

1. pg_stat_statements.
2. pg_stat_kcache.
3. auto_explain.
4. pg_wait_sampling.
5. plprofiler.

PostgresPro

Недостатки pg_stat_statements

1. Нет информации по планам выполнения.
2. Нет привязки к событиям ожидания.
3. Отсутствие информации потребления CPU при планировании и выполнении запросов.
4. Отсутствие статистики аннулирования кэша.

PostgresPro

Модуль pgpro_stats

- Сбор статистики по userid, dbid, queryid, planid.
- Вывод обобщённого плана выполнения помимо текста запроса.
- Сбор статистики событий ожидания с привязкой к конкретному запросу и плану.
- Расчёт суммарной статистики по запросам.
- Подсчёт статистики аннулирования кэша.

PostgresPro

Модули pg_profile и pgpro_pwr

Позволяют собирать и отображать статистику:

1. SQL-запросов.
2. Команд, изменяющих данные СУБД.
3. Использования пользовательских функций и хранимых процедур.
4. Работы процесса очистки, фонового процесса записи и многое другое.

PostgresPro

Top SQL по времени выполнения в pgpro_pwr

Query ID	Database	Exec (s)	%Total	I/O time (s)		Rows	Execution times (ms)				Executions
				Read	Write		Mean	Min	Max	StdErr	
04ccad2749 [f287fe6aee8206ff]	data_db	568435.73	50.16	46357.14	0.00	56251	10105.344	5853.911	52907.023	3149.484	56251
16844de544 [99e0a6c6d0250ae7]	data_db	275047.09	24.27	0.46		56266	4888.336	4296.680	7265.277	458.831	56266
d2972b4cd4 [9e2fc21c4af82b3]	data_db	98871.86	8.73			55416	1784.175	1322.458	7172.350	585.637	55416
20d4a6bbe1 [ae253f2950531042]	data_db	60491.75	5.34			38757	1530.352	1329.667	4263.860	143.886	39528
680db037fd [3c0842cdf1cd90ec]	data_db	56346.51	4.97	0.18		11504	4897.993	4302.192	7420.182	461.443	11504
ae4d21e89c [4d144b46c513d3ba]	data_db	7312.75	0.65	673.43		796	9186.868	6154.602	17678.778	2227.804	796
ae4d21e89c [66ac855538c3d306]	data_db	7217.26	0.64	663.60		790	9135.771	6168.327	16583.076	2273.547	790
a722875b7d [15ec70c4c8f53ad1]	data_db	6026.51	0.53	0.02		1210	4980.590	4343.231	7167.028	514.164	1210

PostgresPro

Top SQL по количеству чтений блоков из кэша СУБД в pgpro_pwr

Query ID	Database	blks fetched	%Total	Hits(%)	Elapsed(s)	Rows	Executions
04ccad2749 [f287fe6aee8206ff]	data_db	87023201012	39.99	85.16	568435.7	56251	56251
16844de544 [99e0a6c6d0250ae7]	data_db	40645632675	18.68	100.00	275047.1	56266	56266
d2972b4cd4 [9e2fc21c4af82b3]	data_db	40029581352	18.40	100.00	98871.9	55416	55416
20d4a6bbe1 [ae253f2950531042]	data_db	28552001009	13.12	100.00	60491.7	38757	39528
680db037fd [3c0842cdf1cd90ec]	data_db	8310018767	3.82	100.00	56346.5	11504	11504

Модули отслеживания долгих запросов

PostgresPro

Top SQL с наибольшим временем ожидания в pgpro_pwr

Query ID	Database	IO(s)	R(s)	W(s)	%Total	Reads			Writes			Elapsed(s)	Executions
						Shr	Loc	Tmp	Shr	Loc	Tmp		
04ccad2749 [f287fe6aee8206ff]	data_db	46357.139	46357.137	0.002	83.28	12911683448			24			568435.7	56251
f5500f7865 [1885229ba51e31d6]	data_db	1712.090	1712.013	0.077	3.08	84503980			5348			3338.5	1
8314e8a8a3 [76f909ceca8a56f9]	data_db	1665.391	1665.391		2.99	421918322						3334.5	1123
18b27b21fb [96449e23cb054092]	data_db	1264.005	1264.005		2.27	11953330						3158.0	181225
b448d1417b [265a69f6be6b326b]	data_db	1101.802	1101.802		1.98	6605417						1710.5	650058
8b90892d3f [76f909ceca8a56f9]	data_db	869.443	869.443		1.56	228886445						1278.4	473
ae4d21e89c [4d144b46c513d3ba]	data_db	673.434	673.434		1.21	195397547						7312.7	796
ae4d21e89c [66ac855538c3d306]	data_db	663.598	663.598		1.19	192704550						7217.3	790

Модули отслеживания долгих запросов

PostgresPro

Пример статистики ожиданий по базам данных в pgpro_pwr

Wait statistics by database

Database	Wait event type	Waited (s)	%Total
db2	Client	207.30	90.49
db2	IO	21.76	9.50
db2	*	229.06	99.99
pg_profile	LWLock	0.03	0.01
pg_profile	*	0.03	0.01
Total		229.09	

Top wait events

Database	Wait event type	Wait event	Waited (s)	%Total
db2	Client	ClientWrite	207.30	90.49
db2	IO	BufFileWrite	15.30	6.68
db2	IO	BufFileRead	6.46	2.82
pg_profile	LWLock	BufferMapping	0.02	0.01
pg_profile	LWLock	LockManager	0.01	0.00

Модули отслеживания долгих запросов

PostgresPro

Пример статистики ожиданий по связке “запрос – план выполнения” в pgpro_pwr

All wait event types

Query ID	Plan ID	Database	Waited (s)	%Total	Details
15e052870b [d1937b74a857ee1b]	e4fa4a8c1185fed0	db2	105.11	45.88	Client: 105.11
fc9e858f09 [7a77ed62d2679a51]	3e3f4c16c408f623	db2	38.55	16.83	Client: 38.55
28397ca62a [480c7b3b4837b2c]	4c2069bda720277	db2	31.19	13.61	Client: 31.19
34464d840e [54d4e146036bdb4e]	111b80468c1a2489	db2	21.85	9.54	Client: 21.85
0babcd5300 [d281f4e1cf9ce40f]	e93e0d3c9a364d37	db2	20.62	9.00	IO: 20.62
bb70911186 [42126e815669f880]	93c68b8ab0931dee	pg_profile	0.03	0.01	LWLock: 0.03

LWLock wait event type

Query ID	Plan ID	Database	Waited (s)	%Total	Details
bb70911186 [42126e815669f880]	93c68b8ab0931dee	pg_profile	0.03	0.01	BufferMapping: 0.02 LockManager: 0.01

IO wait event type

Query ID	Plan ID	Database	Waited (s)	%Total	Details
0babcd5300 [d281f4e1cf9ce40f]	e93e0d3c9a364d37	db2	20.62	9.00	BufFileWrite: 14.58 BufFileRead: 6.04

Модули отслеживания долгих запросов

PostgresPro

Пример статистики потребления CPU и чтения/записи из файловой системы в pgpro_pwr

Query ID	Plan ID	Database	User Time		System Time	
			Exec (s)	%Total	Exec (s)	%Total
28397ca62a [480c7b3b4837b2c]	4c2069bda720277	db2	254.32	29.55	1.02	4.47
34464d840e [54d4e146036bdb4e]	111b80468c1a2489	db2	252.51	29.34	0.88	3.85
0babcd5300 [d281f4e1cf9ce40f]	e93e0d3c9a364d37	db2	193.59	22.49	14.13	61.61
15e052870b [d1937b74a857ee1b]	e4fa4a8c1185fed0	db2	109.36	12.71	5.22	22.77
fc9e858f09 [7a77ed62d2679a51]	3e3f4c16c408f623	db2	32.08	3.73	1.05	4.59

Top SQL by reads/writes done by filesystem layer

Query ID	Plan ID	Database	Read Bytes		Write Bytes	
			Exec	%Total	Exec	%Total
0babcd5300 [d281f4e1cf9ce40f]	e93e0d3c9a364d37	db2			3446 MB	99.49
bb70911186 [42126e815669f880]	93c68b8ab0931dee	pg_profile			4112 kB	0.12

Модули отслеживания долгих запросов

PostgresPro

Примеры обобщённых планов выполнения в pgpro_pwr

fc9e858f09	SELECT p.prod_id , p.category , p.title , p.actor , p.price , p.special , p.common_prod_id FROM products p WHERE p.title LIKE $1 AND p.price BETWEEN $2 AND $3 ORDER BY p.prod_id, p

3e3f4c16c408f623	Sort
	  Output: prod_id, category, title, actor, price, special, common_prod_id
	  Sort Key: p.prod_id, p.category DESC, p.title DESC, p.actor, p.price DESC, p.special, p.common_prod_id
	  -> Bitmap Heap Scan on public.products p
	  Output: prod_id, category, title, actor, price, special, common_prod_id
	  Filter: ((p.title ~~ $1) AND (p.price >= $4) AND (p.price <= $5))
	  -> Bitmap Index Scan on prod_title_cat_prod_id
	  Index Cond: ((p.title ~>=~ $6) AND (p.title ~<~ $7))

7c56f6b3ea	SELECT extname, extnamespace::regnamespace::name AS extnamespace, extversion FROM pg_catalog.pg_extension WHERE extname IN ($1,$2,$3)

18558adc1f4d44fa	Seq Scan on pg_catalog.pg_extension
	  Output: extname, ((extnamespace)::regnamespace)::name, extversion
	  Filter: (pg_extension.extname = ANY ($4))

f47e5c907c	select count($1) as pgpro_fxs from pg_catalog.pg_proc where proname IN ($2,$3,$4)

a0f5bbbbe9d25819	Aggregate
	  Output: count($1)
	  -> Index Only Scan using pg_proc_proname_args_nsp_index on pg_catalog.pg_proc
	  Output: proname, proargtypes, pronamespace
	  Index Cond: (pg_proc.proname = ANY ($5))

Модули отслеживания долгих запросов

PostgresPro

Долгий UPDATE

На основе информации из pgpro_pwr выявлен один из наиболее долгих запросов:

```
UPDATE contract.request_data
   SET status_code = $1
     , request_date = $2
     , response_date = $3
     , model_version = $4
     , contract_id = $5
 WHERE id = $6;
```

PostgresPro

План выполнения UPDATE

```
UPDATE ON request_data (ROWS=91465 width=946)
```

```
-> Seq Scan ON request_data (ROWS=91465
width=946)
    FILTER: ((id)::NUMERIC = '18310725'::NUMERIC)
```

Общее время выполнения – более одной секунды.

В таблице тип id bigint, из приложения передан тип java.math.BigInteger

PostgresPro

Сопоставление типов данных в postgreSQL и Java

https://github.com/pgjdbc/pgjdbc/blob/REL42.7.3/pgjdbc/src/main/java/org/postgresql/jdbc/TypeInfoCache.java#L84

```
{"int8", Oid.INT8, Types.BIGINT, "java.lang.Long",
 Oid.INT8_ARRAY},
```

```
{"numeric", Oid.NUMERIC, Types.NUMERIC,
 "java.math.BigDecimal", Oid.NUMERIC_ARRAY}
```

PostgresPro

Выводы

1. long -> java.lang.Long.
2. numeric -> java.math.BigDecimal.
3. money -> java.lang.Double.
4. float4 -> java.lang.Float.
5. float8 -> java.lang.Double.
6. int4 -> java.lang.Integer.

PostgresPro

Поиск данных по списку значений в виде строки

```
EXPLAIN (ANALYZE)
WITH statuses AS (
SELECT v.status::BIGINT
  FROM regexp_split_to_table('10,30,20', ',') AS v(STATUS)
)
SELECT id
  FROM req.lot l
 WHERE STATUS IN (SELECT STATUS FROM statuses s);
```

PostgresPro

План первоначального запроса

```
Hash JOIN (ROWS=140490) (ROWS=118 loops=1)
  Hash Cond: (lot.status = (v.status)::BIGINT)
    -> Seq Scan ON lot (ROWS=280979) (ROWS=280979 loops=1)
    -> Hash (ROWS=200) (ROWS=3 loops=1)
      -> HashAggregate (ROWS=200 width=32) (ROWS=3 loops=1)
          GROUP KEY: (v.status)::BIGINT
            -> FUNCTION Scan ON regexp_split_to_table v
(ROWS=1000 width=32) (ROWS=3 loops=1)
```

PostgresPro

План запроса поиска строк по условию IN

```
EXPLAIN (ANALYZE)
SELECT l.id
  FROM req.lot l
 WHERE l.status IN (10, 20, 30);
```

```
INDEX Scan USING ixf__lot__status__is_active ON lot l
  (COST=0.42..221.04 ROWS=112 width=8) (actual TIME=0.310..5.448 ROWS=118
loops=1)
```

```
 INDEX Cond: (status = ANY ('{10,20,30}'::BIGINT[]))
```

```
Planning TIME: 1.334 ms
Execution TIME: 5.554 ms
```

Поиск данных по списку значений в виде

PostgresPro

Вариант запроса с использованием regexp_split_to_array

```
EXPLAIN (ANALYZE)
SELECT l.id
  FROM req.lot l
 WHERE l.status = ANY(regexp_split_to_array('10,30,20',
',')::BIGINT[]);

                          QUERY PLAN
INDEX Scan USING ixf__lot__status__is_active ON lot l (ROWS=112)
(actual TIME=0.310..5.448 ROWS=118 loops=1)
  INDEX Cond: (status = ANY ('{10,20,30}'::BIGINT[]))
```

Было 650.235 мс, стало 5.554 мс

PostgresPro

Подзапросы

```
EXPLAIN (ANALYZE)
SELECT l.id
     , (SELECT string_agg(doc_number, '; ')
          FROM buy.purchase_result
         WHERE lot_id = l.id
       ) AS pur_result
     , (SELECT COUNT(*)
          FROM buy.purchase_result pr
         WHERE pr.lot_id = l.id
           AND pr.is_active
       ) AS pr_count
     , (SELECT string_agg(DISTINCT sup.name_full, ';')
          FROM buy.purchase_result pr
          JOIN req.supplier sup
            ON pr.supplier_id = sup.id
           AND sup.is_active
         WHERE pr.lot_id = l.id
           AND pr.is_active
       ) AS sup_info
  FROM req.lot l
 WHERE l.organization_id = 964;
```

Общее время = 4 минуты

PostgresPro

План выполнения первого подзапроса

```
SubPlan 1
  -> Aggregate (ROWS=1) (actual TIME=32.387..32.388
ROWS=1 loops=7495)
      -> Seq Scan ON purchase_result (ROWS=2) (actual
      TIME=29.084..32.361 ROWS=0 loops=7495)
          ROWS Removed BY FILTER: 147909
```

Общее время = 32.388 * 7495 = 4 минуты, поиск по purchase_result

PostgresPro

План выполнения второго подзапроса

```
SubPlan 2
  -> Aggregate (ROWS=1) (actual TIME=0.044..0.044
ROWS=1 loops=7495)
   -> INDEX ONLY Scan USING ipr_lot_id ON purchase_result
   pr (ROWS=2) (actual TIME=0.032..0.033 ROWS=0 loops=7495)

      INDEX Cond: (lot_id = l.id)
         Heap Fetches: 0
```

Общее время = 0.044 * 7495 = 329.780 мсек, поиск по purchase_result

PostgresPro

План выполнения третьего подзапроса

```
SubPlan 3
  -> Aggregate (ROWS=1) (actual TIME=0.064..0.064 ROWS=1 loops=7495)
    -> Nested Loop (ROWS=1) (actual TIME=0.018..0.022 ROWS=0 loops=7495)
          -> INDEX Scan USING ixf_pt_lot_id ON purchase_result pr_1
          (ROWS=2) (actual TIME=0.005..0.006 ROWS=0 loops=7495)
                INDEX Cond: (lot_id = l.id)
    -> INDEX Scan USING pk_sup ON supplier sup (ROWS=1) (actual
TIME=0.023..0.023 ROWS=1 loops=3419)
          INDEX Cond: (id = pr_1.supplier_id)
          FILTER: is_active
          ROWS Removed BY FILTER: 0
```

Общее время = 0.064 * 7495 = 479.680 мсек, поиск по purchase_result и supplier

Подзапрос

PostgresPro

Объединение трёх подзапросов

```
SELECT string_agg(pr.doc_number, '; ') AS doc_numbers
     , COUNT(1) FILTER(WHERE pr.is_active) AS pr_count
     , string_agg(DISTINCT sup.name_full, ';') FILTER(WHERE pr.is_active) AS sup_info
  FROM buy.purchase_result pr
  LEFT JOIN req.supplier sup
    ON sup.id = pr.supplier_id
   AND sup.is_active
 WHERE pr.lot_id = l.id


CREATE INDEX pr_lot_id_doc_number_ix
    ON buy.purchase_result(lot_id, is_active, supplier_id, doc_number);

CREATE UNIQUE INDEX sup_info_ux
    ON req.supplier(id, is_active, name_full);
```

PostgresPro

Итоговый запрос

```
SELECT l.id
     , pr.doc_numbers
     , pr.pr_count
     , pr.sup_info
  FROM req.lot l
  LEFT JOIN LATERAL (SELECT string_agg(pr.doc_number, '; ') AS doc_numbers
                         , COUNT(*) FILTER(WHERE pr.is_active) AS pr_count
                         , string_agg(DISTINCT sup.name_full, ';')
                           FILTER(WHERE pr.is_active) AS sup_info
                      FROM buy.purchase_result pr
                      LEFT JOIN req.supplier sup
                        ON sup.id = pr.supplier_id
                       AND sup.is_active
                     WHERE pr.lot_id = l.id
                    ) pr
    ON (1 = 1)
 WHERE l.organization_id = 964;
```

PostgresPro

План выполнения нового подзапроса

```
-> AGGREGATE (ROWS=1) (actual TIME=0.014..0.014 ROWS=1 loops=7495)
  -> Nested LOOP LEFT JOIN (ROWS=2) (actual TIME=0.006..0.008 ROWS=0
loops=7495)
    -> INDEX ONLY Scan USING pr_lot_id_doc_number_ix ON purchase_result pr
(ROWS=2) (actual TIME=0.004..0.004 ROWS=0 loops=7495)
         INDEX Cond: (lot_id = l.id)
         Heap Fetches: 0
    -> INDEX ONLY Scan USING sup_info_ux ON supplier sup (ROWS=1 width=105)
(actual TIME=0.005..0.005 ROWS=1 loops=3419)
         INDEX Cond: ((id = pr.supplier_id) AND (is_active = TRUE))
         Heap Fetches: 0
```

Общее время = 0.014 * 7495 = 104.930 мсек, было 4 минуты

PostgresPro

Вычисляемое выражение по двум столбцам таблицы

```
EXPLAIN (ANALYZE)
WITH ds AS (
SELECT l.id
     , CASE
         WHEN EXTRACT(YEAR FROM l.date_planned) = 2019 AND
              EXTRACT(YEAR FROM l.date_delivery_from) = 2019 THEN 1
         WHEN EXTRACT(YEAR FROM l.date_planned) = 2019 AND
              EXTRACT(YEAR FROM l.date_delivery_from) > 2019 THEN 21
         ELSE 0
       END AS razdel
  FROM req.lot l
 WHERE l.year < 2019
)
SELECT *
  FROM ds
 WHERE razdel != 0;
```

PostgresPro

План выполнения первоначального запроса

```
                                QUERY PLAN
-----------------------------------------------------------------------------------
Bitmap Heap Scan ON lot l (ROWS=179325 width=12) (actual TIME=143.985..523.901 ROWS=1445
loops=1)
  Recheck Cond: (YEAR < 2019)
  FILTER:
  ROWS Removed BY FILTER: 178781
  Heap Blocks: exact=20196
    -> Bitmap INDEX Scan ON ix_lot (ROWS=180227) (actual TIME=14.598..14.599 ROWS=180226
loops=1)
       INDEX Cond: (YEAR < 2019)
Planning TIME: 4.737 ms
Execution TIME: 524.258 ms
```

Вычисляемое выражение по двум столбцам

PostgresPro

Замена вычисляемого столбца на два дополнительных условия фильтрации к столбцам таблицы

У любой даты из отрезка 2019-01-01 и 2019-12-31 год = 2019.

У любой даты >= 2019-01-01 год >= 2019.

```
EXPLAIN (ANALYZE)
SELECT l.id
  FROM req.lot l
 WHERE l.year < 2019
   AND l.date_planned BETWEEN make_date(2019, 1, 1) AND make_date(2019, 12, 31)
   AND l.date_delivery_from >= make_date(2019, 1, 1);
```

Вычисляемое выражение по двум столбцам

PostgresPro

Изменчивая функция

1. Может изменять данные в БД.
2. Может возвращать различные результаты с одинаковыми аргументами.
3. Вычисляется для каждой строки множества.
4. Планировщик не делает никаких предположений о поведении такой функции.

PostgresPro

Стабильная функция

1. Не может изменять данные в БД.
2. Возвращает одинаковые результаты с одинаковыми аргументами в рамках одного запроса.
3. Множество вызовов функции заменяется одним.

PostgresPro

Постоянная функция

1. Не ищет и не меняет данные в БД.
2. Не использует данные временных зон и локалях сервера.
3. Возвращает одинаковые результаты с одинаковыми аргументами.
4. Может быть выполнена на этапе планирования.

PostgresPro

План выполнения запроса после замены вычисляемого выражения

```
                                        QUERY PLAN
----------------------------------------------------------------------------------------------
Bitmap Heap Scan ON lot l (ROWS=9215) (actual TIME=110.011..346.557 ROWS=1445 loops=1)
  Recheck Cond: (YEAR < 2019)
  FILTER: ((date_planned >= '2019-01-01'::DATE) AND
           (date_planned <= '2019-12-31'::DATE) AND
           (date_delivery_from >= '2019-01-01'::DATE)
          )
  ROWS Removed BY FILTER: 178781
  Heap Blocks: exact=20196
    -> Bitmap INDEX Scan ON ix__lot (ROWS=180227 width=0) (actual TIME=16.352..16.353
ROWS=180226 loops=1)
         INDEX Cond: (YEAR < 2019)
```

Расчётное количество строк больше фактического в 9 раз, было в 124 раза

Вычисляемое выражение по двум столбцам

PostgresPro

Использование расширенной статистики для устранения ошибки в расчётном количестве строк

Нужна статистика по комбинации полей year и date_planned.

```
CREATE STATISTICS lot_year_date_planned(mcv) ON YEAR,
date_planned FROM req.lot;

ALTER TABLE req.lot ALTER COLUMN YEAR SET STATISTICS 1250;

ALTER TABLE req.lot ALTER COLUMN date_planned SET
STATISTICS 1250;

ANALYZE req.lot;

CREATE INDEX req_dp_ddf_year_ix ON req.lot(date_planned,
date_delivery_from, YEAR);
```

Вычисляемое выражение по двум столбцам

PostgresPro

План выполнения запроса после сбора расширенной статистики и создания индекса

```
                                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
Bitmap Heap Scan ON lot l (ROWS=1397 width=8) (actual TIME=5.102..7.942 ROWS=1445 loops=1)
  Recheck Cond: ((date_planned >= '2019-01-01'::DATE) AND
                 (date_planned <= '2019-12-31'::DATE) AND
                 (date_delivery_from >= '2019-01-01'::DATE) AND (YEAR < 2019)
                )
  Heap Blocks: exact=936
    -> Bitmap INDEX Scan ON req_dp_ddf_year_ix (ROWS=1397) (actual TIME=4.970..4.970
ROWS=1445 loops=1)
         INDEX Cond: ((date_planned >= '2019-01-01'::DATE) AND
                      (date_planned <= '2019-12-31'::DATE) AND
                      (date_delivery_from >= '2019-01-01'::DATE) AND
                      (YEAR < 2019)
                     )
```

Было 523.901 мс, стало 7.942 мс, расхождение в количествах строк сократилось до минимума.

Вычисляемое выражение по двум столбцам

PostgresPro

Исключение условий фильтрации во время планирования запроса

```
WITH params AS NOT MATERIALIZED (
SELECT :version_cond AS version_cond
)
SELECT l.id
  FROM req.lot l
  JOIN params p
    ON (1 = 1)
 WHERE l.year = 2019
   AND ((p.version_cond = 1 AND l.status = 50 AND l.type_correct = 0) OR
        (p.version_cond = 2 AND l.status = 50 AND l.is_last_version)
       );
```

PostgresPro

План выполнения запроса при version_cond = 1

```
                                 QUERY PLAN
-------------------------------------------------------------------------
-----
Bitmap Heap Scan ON lot l (ROWS=14580 width=8) (actual
TIME=1.716..14.347 ROWS=18576 loops=1)
  Recheck Cond: ((YEAR = 2019) AND (type_correct = 0) AND
(STATUS = 50))
  Heap Blocks: exact=3262
    -> Bitmap INDEX Scan ON year_type_cor_status_ix
(ROWS=14580) (actual TIME=1.243..1.244 ROWS=18576 loops=1)
        INDEX Cond: ((YEAR = 2019) AND (type_correct = 0) AND
(STATUS = 50))
```

PostgresPro

План выполнения запроса при version_cond = 2

```
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
Bitmap Heap Scan ON lot l (ROWS=14580 width=8) (actual TIME=3.067..36.650 ROWS=18576
loops=1)
  Recheck Cond: ((YEAR = 2019) AND (STATUS = 50))
  FILTER: is_last_version
  Heap Blocks: exact=3262
    -> Bitmap INDEX Scan ON year_type_cor_lv_ix (ROWS=14580 width=0) (actual
TIME=2.612..2.612 ROWS=18576 loops=1)
         INDEX Cond: ((YEAR = 2019) AND (is_last_version = TRUE) AND (STATUS = 50))
```

PostgresPro

План выполнения запроса при version_cond = 3

```
                                  QUERY PLAN
-------------------------------------------------------------------------------
RESULT (ROWS=0) (actual TIME=0.002..0.002 ROWS=0 loops=1)
  One-TIME FILTER: FALSE
Planning TIME: 0.429 ms
Execution TIME: 0.034 ms
```

При version_cond = 3 запрос вернёт пустое множество ещё на этапе планирования

PostgresPro

Параметры join_collapse_limit и from_collapse_limit

1. Тормозят аналитические запросы
2. join_collapse_limit = 8 -> 30
3. from_collapse_limit = 8 -> 30

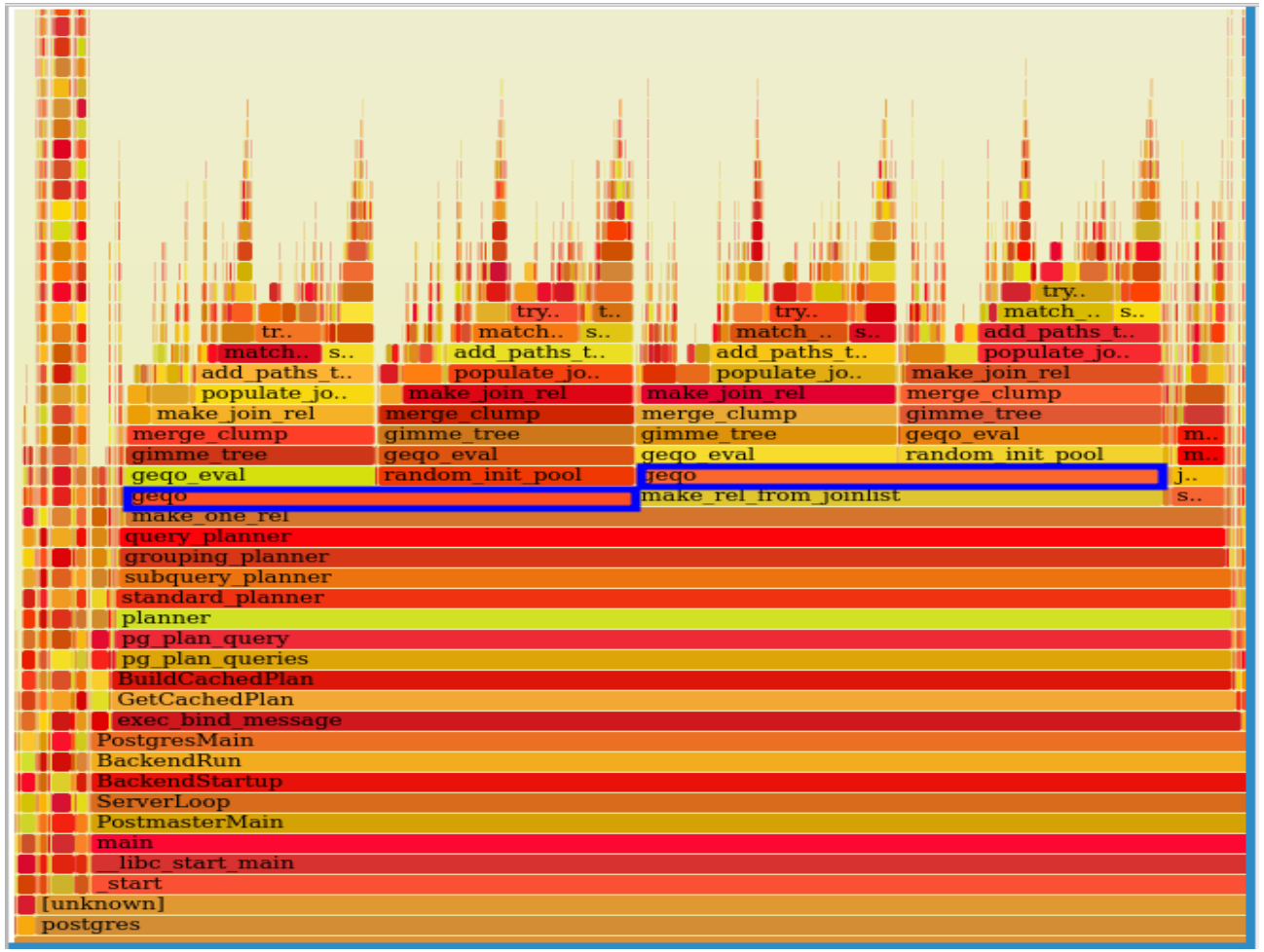
PostgresPro

Высокое потребление CPU

1. from_collapse_limit и join_collapse_limit равны 30
2. Количество соединений таблиц > 12 (geqo_threshold)
3. Использование генетического оптимизатора

PostgresPro

Результаты профилирования



Параметры join_collapse_limit и

PostgresPro

Причины высокого потребления CPU

1. Время планирования > 300 мсек.
2. Время выполнения < 5 мсек.
3. Из основной таблицы данные извлекались по первичному ключу, а затем к ним добавлялись данные из различных таблиц.

После уменьшения значений параметров до 8
время планирования стало сопоставимо с временем выполнения.

PostgresPro

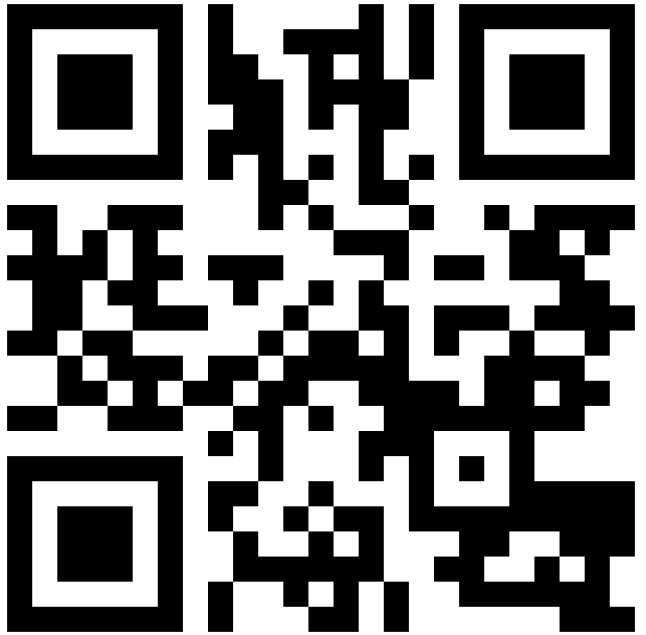
Финальные выводы

1. Оптимизация запросов – не только построение индексов.
2. Не стоит переносить код “как есть” при миграции на другую СУБД.
3. В PostgreSQL работа планировщика постоянно улучшается, нужно использовать подходящий механизм.
4. Перед применением настройки в промышленном окружении нужно проверить её влияние при нагрузочном тестировании.

PostgresPro

Ссылки на средства отслеживания состояния СУБД

- Mamonsu. https://postgrespro.ru/docs/enterprise/16/mamonsu
- postgres_exporter. https://github.com/prometheus-community/postgres_exporter
- Postgres Pro Enterprise Manager. https://postgrespro.ru/products/PPEM



PostgresPro

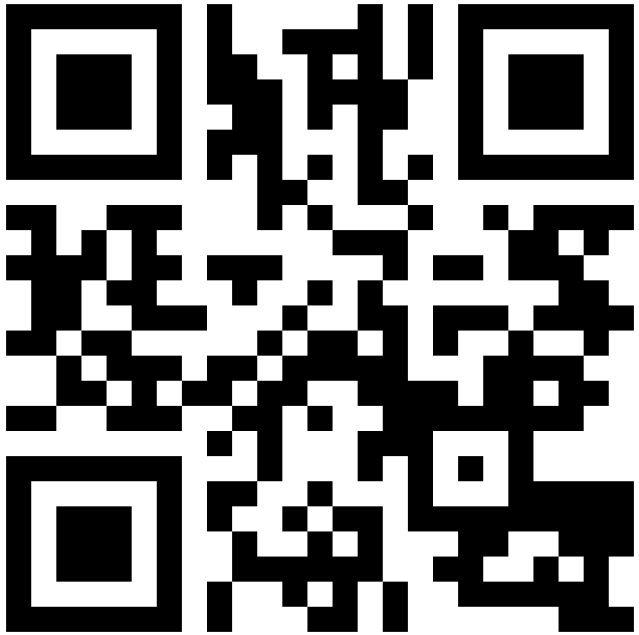
Ссылки на модули отслеживания долгих запросов

- pg_stat_statements. https://postgrespro.ru/docs/enterprise/16/pgstatstatements
- pg_stat_kcache. https://github.com/powa-team/pg_stat_kcache
- pg_wait_sampling. https://github.com/postgrespro/pg_wait_sampling
- plprofiler. https://github.com/bigsql/plprofiler
- auto_explain. https://postgrespro.ru/docs/enterprise/16/auto-explain

PostgresPro

Ссылки на дополнительные модули

- pgpro_stats. https://postgrespro.ru/docs/enterprise/16/pgpro-stats
- pg_profile. https://github.com/zubkov-andrei/pg_profile
- pgpro_pwr. https://postgrespro.ru/docs/enterprise/16/pgpro-pwr



PostgresPro

Спасибо за внимание!

Пётр Петров (p.petrov@postgrespro.ru)