



СУБД Postgres Pro

МОНИТОРИНГ
И КОНТРОЛЬ ПРОИЗВОДИТЕЛЬНОСТИ

Борис Пищик

b.pischik@postgrespro.ru

Типы мониторинга

- Оперативный

- Текущая картина производительности основанная на динамической и накопительной статистике (PPEM, pgpro_stats, pg_stats_statements, pg_query_state)

- Стратегический

- Анализ истории наблюдений для выработки методов оптимизации и расследования инцидентов (pgpro_pwr, pg_wait_sampling, ...)

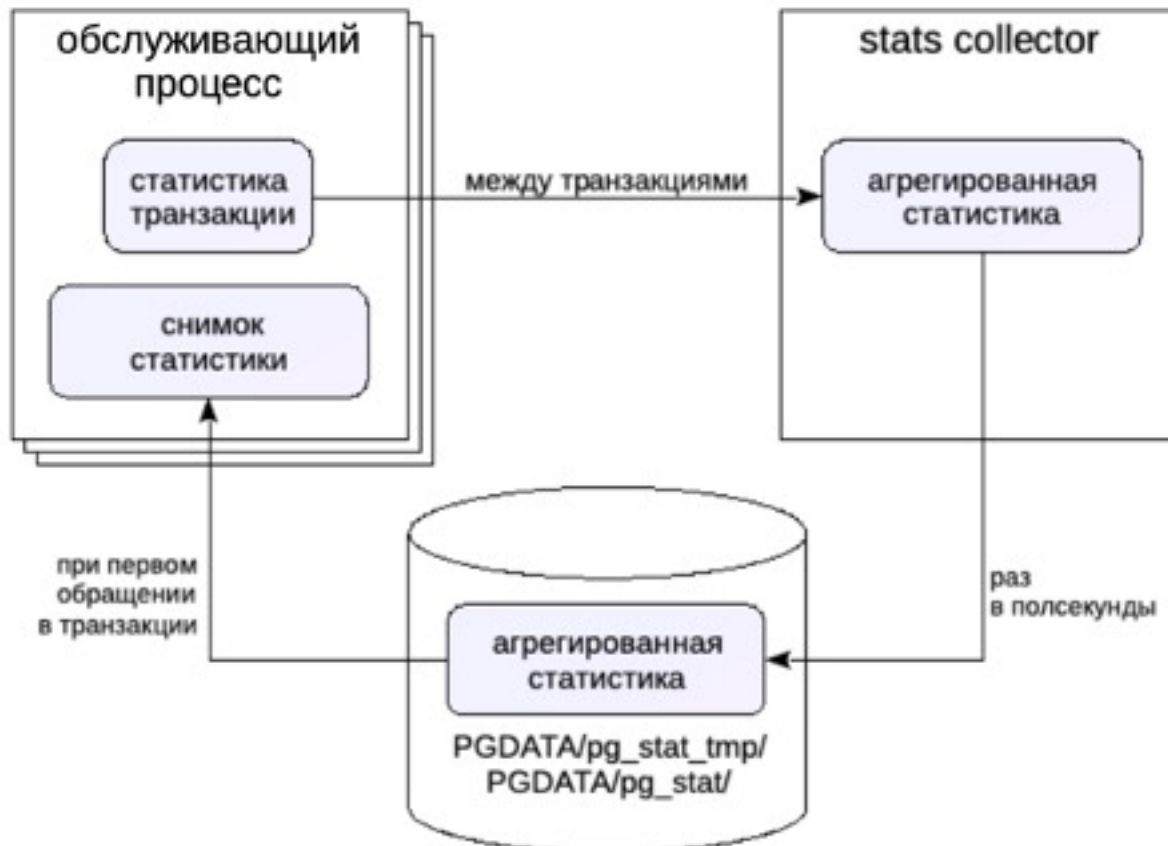
Общая методика

- Анализ и профилирование нагрузки на сервере (atop, perf, flamegraph..)
- Анализ журналов/дампов СУБД на предмет ошибок и аномалий
- Анализ ожиданий активной сессии и идентификация «тяжелых» запросов/планов
- Анализ параметров экземпляра (планировщик, autovacuum, логирование..)
- Снятие диагностического отчета за периоды пиковой нагрузки (pgpro_pwr)
- Выявление первопричины и коррекция

Источник – статистическая информация о работе БД

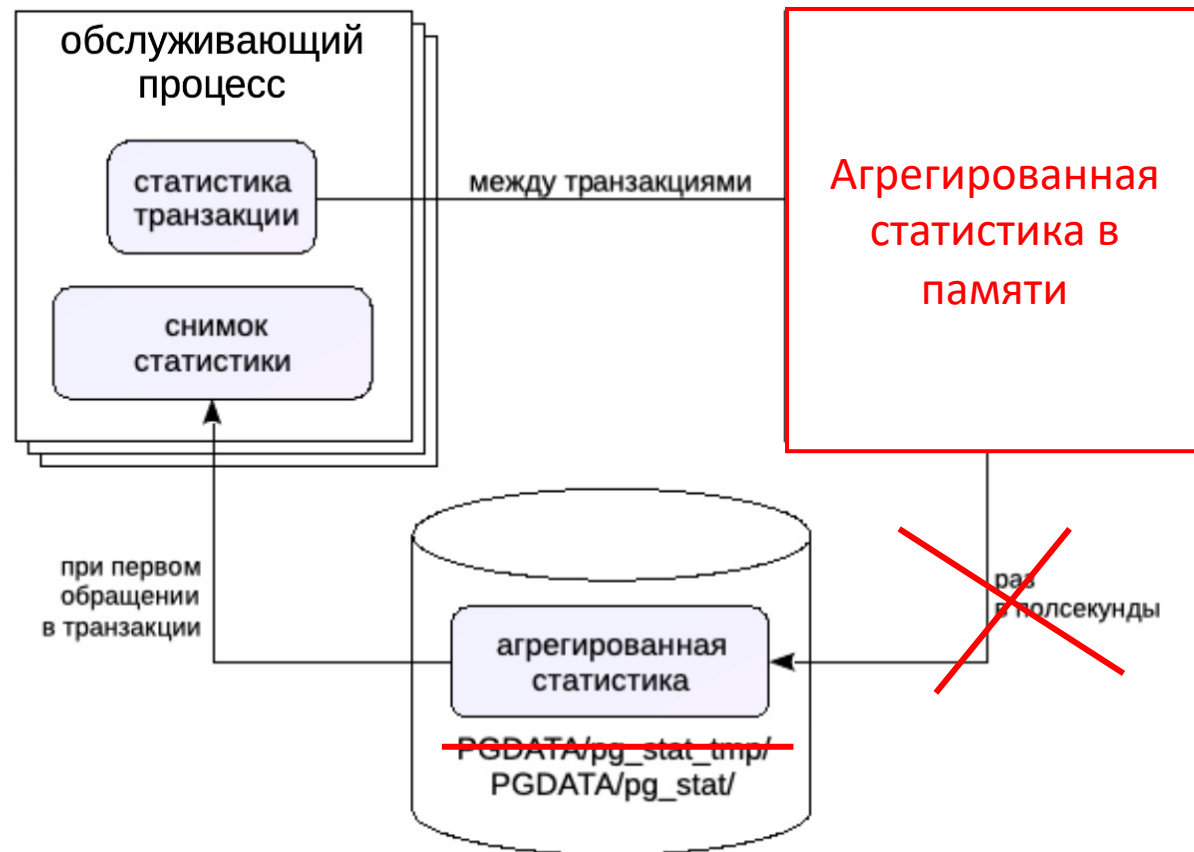
- Доступна через представления **pg_stat***, **pg_statio***, **pg_wait***, **pg_locks** и др.
- Контролируется конфигурационными параметрами **track_***
- Аккумулируется в общей памяти
- Сохраняется при нормальном перезапуске в **\$PGDATA/pg_stat**

Подсистема сбора статистики производительности PostgreSQL



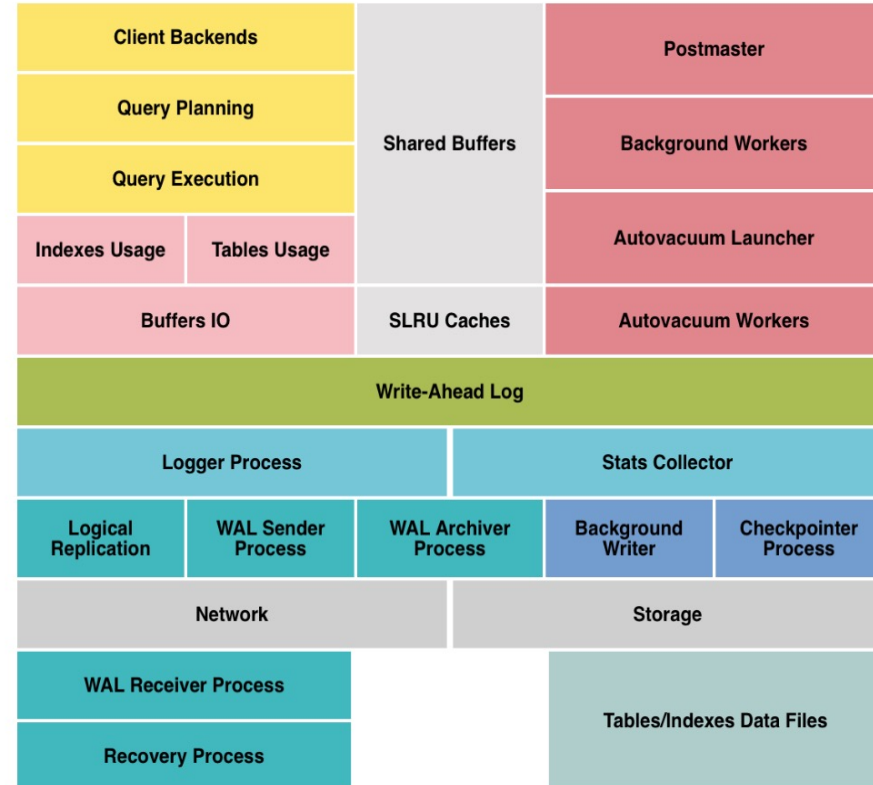
pg_stat_activity
pg_pool_backends
pg_client_session_info
pgpro_stat_wal_activity
pg_stat_replication
pg_stat_replication_slots
pg_stat_wal_receiver
pg_stat_recovery_prefetch
pg_stat_subscription
pg_stat_subscription_stats
pg_stat_ssl
pg_stat_gssapi
pg_stat_archiver
pg_stat_bgwriter
pg_stat_wal
pg_stat_database
pg_stat_database_conflicts
pg_stat_all_tables
pg_stat_all_indexes
pg_statio_all_tables
pg_statio_all_indexes
pg_statio_all_sequences
pg_stat_user_functions
pg_stat_slru
...

Подсистема сбора статистики производительности PostgreSQL



Postgres Observability

- pg_stat_ssl
- pg_stat_activity
- pg_log_backend_memory_contexts()
- pg_backend_memory_contexts
- EXPLAIN
- pg_stat_monitor
- pg_stat_statements
- pg_stat_kcache
- pg_stat_user_functions
- pg_wait_sampling
- pg_blocking_pids()
- pg_prepared_xacts
- pg_stat_progress_cluster
- pg_stat_progress_copy
- pg_locks
- pg_stat_progress_create_index
- pg_stat_all_indexes
- pg_stat_all_tables
- pg_statio_all_indexes
- pg_statio_all_tables
- pg_statio_all_sequences
- pg_is_wal_replay_paused()
- pg_get_wal_replay_pause_state()
- pg_current_wal_lsn()
- pg_wal_lsn_diff()
- pg_ls_logdir()
- pg_current_logfile()
- pg_replication_slots
- pg_stat_replication_slots
- pg_stat_replication
- pg_stat_subscription
- pg_stat_wal_receiver
- pg_stat_archiver
- pg_ls_archive_statusdir()
- pg_stat_database_conflicts



- pg_buffercache
- pg_shmem_allocations
- pg_stat_slru
- pg_stat_activity
- pg_stat_database
- pg_stat_progress_vacuum
- pg_stat_progress_analyze
- pg_stat_all_tables
- pg_stat_wal
- pg_ls_waldir()
- pg_walfile_name()
- pg_current_wal_insert_lsn()
- pg_walfile_name_offset()
- pg_current_wal_flush_lsn()
- pg_last_wal_receive_lsn()
- pg_last_wal_replay_lsn()
- pg_last_xact_replay_timestamp()
- pg_stat_bgwriter
- pg_stat_progress_basebackup
- pgstattuple
- pg_tablespace_size()
- pg_database_size()
- pg_total_relation_size()
- pg_relation_size()
- pg_table_size()
- pg_indexes_size()
- pg_ls_tmpdir()
- pg_ls_dir()
- pg_relation_filename()
- pg_relation_filepath()
- pg_filename_relation()

nicstat

iostat

Экземпляры / PgPro Manager repository / Метрики

[СОЗДАТЬ ГРАФИК](#)

ГРАФИКИ (5) Пресет Производительность ▼ Период Последние 15 минут ▼

Одновременное масштабирование

[← НАЗАД](#)

ДЕТАЛИ ЭКЗЕМПЛЯРА

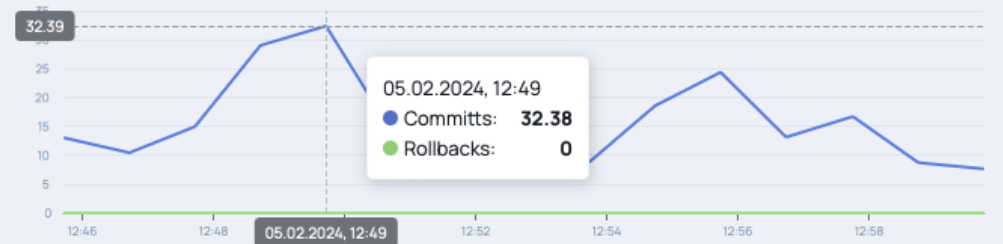
[Обзор](#)[Табличные пространства](#)[Базы данных](#)[Метрики](#)[Сессии](#)[SQL статистика](#)[Журнал сообщений](#)[Резервные копии](#)[Параметры](#)[Профайлер](#)[Аутентификация](#)[Роли](#)

PostgresPro Connections: Overview



- max_connections
- Active
- Disabled
- Fastpath
- Idle
- Idle in Transaction
- Idle in Transaction (aborted)
- Total
- Waiting
- Other

PostgresPro Instance: Transactions Rate



- Commits
- Rollbacks

System: Load Average



- 1 min
- 5 min
- 15 min

System: Processes Overview



- Running
- Blocked
- Forkrate

Диагностика и мониторинг производительности

- `pg_stat_activity`

- `pg_query_state`

- `pgpro_stats`



- `auto_explain`

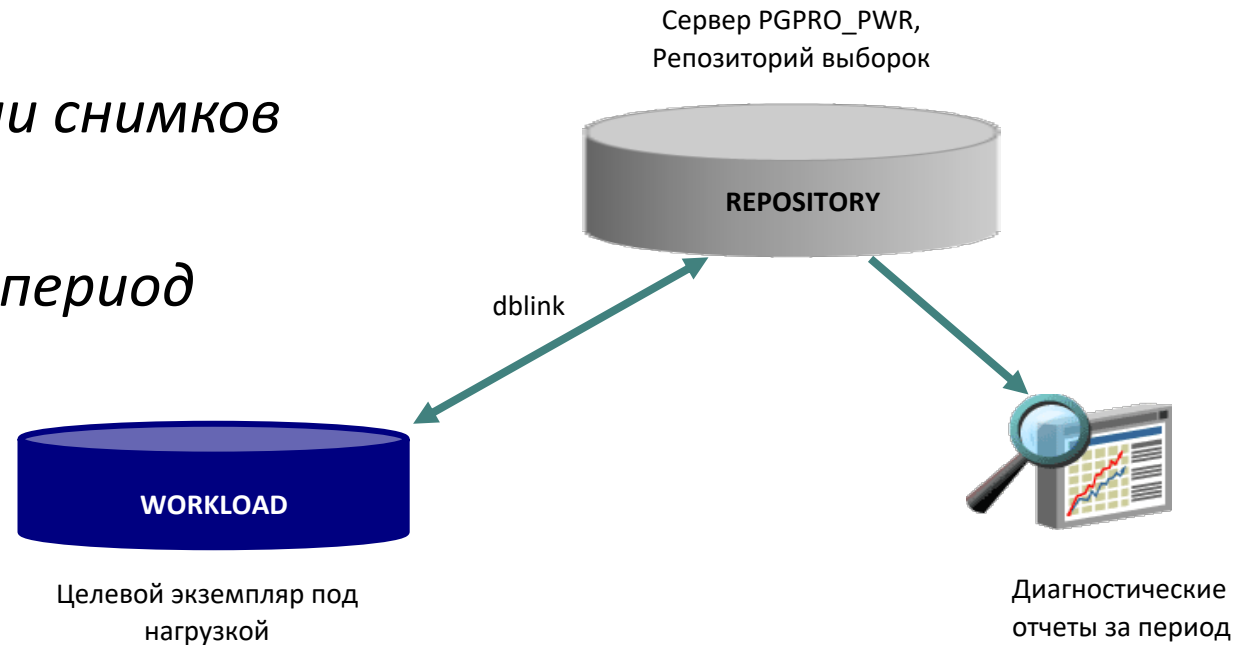
- `pg_stat_progress_*`

- `pgpro_pwr`

Сценарии - 1

- «Плавающая» проблема производительности СУБД
- «Работало хорошо – теперь работает плохо»
- Как идентифицировать «тяжелый» запрос и посмотреть статистику по нему
- Как «перехватывать» запросы в отдельный трассировочный файл
- Как посмотреть историю ожиданий по сессии

- «*Диагностическая карта*» производительности базы данных
- *Периодические снимки статистики - samples*
- *Репозиторий – хранилище истории снимков*
- *Детальный отчет за выбранный период*



Server statistics

Database statistics

Database	Transactions	Block reads	Block writes	Temp files	Temp files	Temp files	Temp files	Temp files	Temp files
Database	Count	Count	Count	Count	Count	Count	Count	Count	Count
dbf	1000	1000	1000	1000	1000	1000	1000	1000	1000
postgres	1000	1000	1000	1000	1000	1000	1000	1000	1000
total	2000	2000	2000	2000	2000	2000	2000	2000	2000

Database statistics by database

Database	Count	Count	Count	Count	Count	Count	Count	Count	Count
dbf	1000	1000	1000	1000	1000	1000	1000	1000	1000
postgres	1000	1000	1000	1000	1000	1000	1000	1000	1000
total	2000	2000	2000	2000	2000	2000	2000	2000	2000

Database server statistics

DB	Database	Count	Count	Count	Count	Count	Count	Count	Count
dbf	1000	1000	1000	1000	1000	1000	1000	1000	1000
postgres	1000	1000	1000	1000	1000	1000	1000	1000	1000
total	2000	2000	2000	2000	2000	2000	2000	2000	2000

* дополнительно: <https://postgrespro.ru/video/how-to>

PGPRO_PWR – пример конфигурации

- ```
select profile.create_server('boris2', 'host=192.168.26.xx
dbname=postgres password=postgres port=5433');
```
- ```
select profile.take_sample('boris2');
```
- ```
select profile.show_samples('boris2');
```
- ```
psql -Aqtc "select
profile.get_report('boris2', 2, 6)" -o
report_boris2_2_6.html
```

```
postgres=# select profile.show_samples('boris2');
show_samples
```

```
-----
(1,"2022-10-12 13:21:23+03",t,,)
(2,"2022-10-12 13:46:44+03",t,,)
(3,"2022-10-13 11:23:25+03",t,,)
(4,"2022-10-13 11:28:08+03",t,,)
(5,"2022-10-13 11:53:28+03",t,,)
(6,"2022-10-13 12:06:47+03",t,,)
(7,"2022-10-19 12:16:40+03",t,,)
(7 rows)
```

Server statistics

Database statistics

Database	Transactions			Block statistics			Block I/O times		Tuples					Temp files		Size	Growth
	Commits	Rollbacks	Deadlocks	Hit(%)	Read	Hit	Read	Write	Ret	Fet	Ins	Upd	Del	Size	Files		
db5	3377			100.00		185864			2129743	39349						9317 kB	
demo	17183			100.00		193323			2298257	43985						707 MB	
mamonsu_database	61593			100.00		359089			6689577	48937		1416		6998 MB	1337	9517 kB	
postgres	3764	5		99.93	1096	1554516	0.10		12364299	647192	21928	12584	14362	2306 kB	9	86 MB	4744 kB
Total	85917	5		99.95	1096	2292792	0.10		23481876	779463	21928	14000	14362	7000 MB	1346	811 MB	4744 kB

Session statistics by database

Database	Timings (s)			Sessions			
	Total	Active	Idle(T)	Established	Abandoned	Fatal	Killed
db5	80463.05	4.36	0.36	8			
demo	123699.00	225.80	0.33	11	3		
mamonsu_database	80411.25	192.43	0.34	8			
postgres	126511.75	10.96	4.11	25	1		
Total	411085.04	433.54	5.13	52	4		

Database vacuum statistics

DB	DB blocks statistics				VM marks		Dead tuples			WAL size	I/O time			Vacuum time		CPU time		Interrupts
	Fetchd	%Total	Read	%Total	Frozen	Visible	Deleted	Left	%Eff		Read	Write	%Total	Total	Delay	User	System	
...																		

Top SQL by shared blocks dirtied

Query ID	Plan ID	Database	User	Dirtied	% Total	Hits(%)	Dirtied	WAL	% Total	Elapsed(s)	Rows	Executions
----------	---------	----------	------	---------	---------	---------	---------	-----	---------	------------	------	------------

Top SQL by shared blocks written

Query ID	Plan ID	Database	User	Written	% Total	% BackendW	Hits(%)	Elapsed(s)	Rows	Executions
----------	---------	----------	------	---------	---------	------------	---------	------------	------	------------

Top SQL by WAL size

Query ID	Plan ID	Database	User	WAL	% Total	Dirtied	WAL FPI	WAL records
d275cb5060881acb [90ccf98e26]	213a97861e7fc6d1	postgres	postgres	102 bytes				1
3eb29788a00e77d8 [32b010c3ee]	c0202e2850158989	postgres	postgres	74 bytes				2
f98a9e62cfa447cc [baafa5cb06]	83e01ba124f277dc	postgres	postgres	28 bytes				1

rusage statistics

Top SQL by system and user time

Query ID	Plan ID	Database	User	User Time		System Time	
				Exec (s)	% Total	Exec (s)	% Total
f5a41aeb8ac80a11 [0f0595a0c4]	164966c1bac4b0c2	postgres	postgres	23.09	20.11	1.19	7.97
41b1b8bc4f294a7a [066b0aed97]	c4dd16d9a304e445	postgres	postgres	8.42	7.34	1.92	12.91
e199fc56ea356630 [f2e9e36a25]	79e505b049bd71c7	postgres	postgres	7.38	6.43	1.91	12.85
eaba21ef6af3a9bb [6208431af0]	8cf4b20416a7b658	testdb1	postgres	8.1	7.05	0.22	1.49
eaba21ef6af3a9bb [d5fcdd0d45]	8cf4b20416a7b658	tst1	postgres	7.9	6.88	0.17	1.16
eaba21ef6af3a9bb [a13a5219d6]	8cf4b20416a7b658	tst2	postgres	7.87	6.86	0.19	1.26

content

- [Server statistics](#)
 - [Database statistics](#)
 - [Cluster SLRU statistics](#)
 - [Session statistics by database](#)
 - [Statement statistics by database](#)
 - [Cluster statistics](#)
 - [WAL statistics](#)
 - [Tablespace statistics](#)
 - [Wait statistics by database](#)
 - [Top wait events](#)
- [Load distribution](#)
 - [Load distribution among heavily loaded databases](#)
 - [Load distribution among heavily loaded applications](#)
 - [Load distribution among heavily loaded hosts](#)
 - [Load distribution among heavily loaded users](#)
- [SQL query statistics](#)
 - [Top SQL by execution time](#)
 - [Top SQL by executions](#)
 - [Top SQL by shared blocks fetched](#)
 - [Top SQL by shared blocks read](#)
 - [Top SQL by shared blocks dirtied](#)
 - [Top SQL by shared blocks written](#)
 - [Top SQL by WAL size](#)
 - [rusage statistics](#)
 - [Top SQL by system and user time](#)
 - [Top SQL by reads/writes done by filesystem layer](#)
 - [SQL query wait statistics](#)
 - [Complete list of SQL texts](#)
- [Schema object statistics](#)
 - [Top tables by estimated sequentially scanned volume](#)
 - [Top tables by blocks fetched](#)
 - [Top tables by blocks read](#)
 - [Top DML tables](#)
 - [Top tables by updated/deleted tuples](#)
 - [Top growing tables](#)
 - [Top indexes by blocks fetched](#)
 - [Top indexes by blocks read](#)
 - [Top growing indexes](#)
 - [Unused indexes](#)
- [Vacuum-related statistics](#)
 - [Top tables by vacuum operations](#)

PGPRO_PWR – дифференциальный отчет

SQL query statistics

Top SQL by execution time

Query ID	Plan ID	Database	User	I	Exec (s)	% Total	CPU time (s)		Rows	Execution times (ms)				Executions
							Usr	Sys		Mean	Min	Max	StdErr	
f5a41aeb8ac80a11 [0f0595a0c4]	164966c1bac4b0c2	postgres	postgres	1	13.91	18.59	13.08	0.85	29062	0.48	0.38	0.95	0.09	29062
				2	9.74	18.53	9.44	0.31	20587	0.47	0.39	1.83	0.09	20587
41b1b8bc4f294a7a [066b0aed97]	c4dd16d9a304e445	postgres	postgres	1	5.94	7.94	4.78	1.11	1399	4.25	3.91	19.79	0.67	1399
				2	4.25	8.08	3.44	0.77	976	4.35	4.01	7.89	0.53	976
e199fc56ea356630 [f2e9e36a25]	79e505b049bd71c7	postgres	postgres	1	5.34	7.14	4.23	1.08	287	18.61	17.88	27	0.73	287
				2	3.79	7.2	2.97	0.79	203	18.65	18.01	24.84	0.79	203
eaba21ef6af3a9bb [6208431af0]	8cf4b20416a7b658	testdb1	postgres	1	4.78	6.39	4.62	0.13	21238	5.56	0.33	18.12	7.31	861
				2	3.39	6.44	3.28	0.08	15022	5.56	0.35	19.73	7.32	609
eaba21ef6af3a9bb [d5fcdd0d45]	8cf4b20416a7b658	tst1	postgres	1	4.64	6.2	4.51	0.1	20090	5.39	0.04	17.09	7.32	861
				2	3.29	6.25	3.2	0.07	14210	5.4	0.04	17.86	7.34	609
eaba21ef6af3a9bb [ef03420ab3]	8cf4b20416a7b658	tst4	postgres	1	4.64	6.2	4.51	0.1	20090	5.39	0.04	17.33	7.32	861
				2	3.28	6.24	3.18	0.07	14210	5.39	0.04	18.44	7.33	609
eaba21ef6af3a9bb [a13a5219d6]	8cf4b20416a7b658	tst2	postgres	1	4.64	6.19	4.5	0.11	20090	5.38	0.04	21.07	7.32	861
				2	3.28	6.23	3.18	0.07	14210	5.38	0.04	17.53	7.32	609
eaba21ef6af3a9bb [88271abcdf]	8cf4b20416a7b658	tst3	postgres	1	4.63	6.19	4.49	0.11	20090	5.38	0.04	18.42	7.31	861
				2	3.27	6.23	3.17	0.08	14210	5.37	0.04	16.6	7.31	609
e199fc56ea356630 [9d085f8dd7]	79e505b049bd71c7	testdb1	postgres	1	4.17	5.57	3.38	0.76	287	14.53	14.11	19.19	0.41	287
				2	2.96	5.64	2.39	0.55	203	14.6	14.08	24.18	0.79	203
e199fc56ea356630 [9679e6c955]	79e505b049bd71c7	tst2	postgres	1	4	5.35	3.22	0.75	287	13.94	13.28	20.3	0.48	287
				2	2.84	5.4	2.31	0.51	203	13.98	13.52	18.72	0.55	203
e199fc56ea356630 [3f49f2a9ff]	79e505b049bd71c7	tst1	postgres	1	4	5.35	3.22	0.75	287	13.94	13.41	17.28	0.36	287
				2	2.83	5.38	2.28	0.53	203	13.92	13.43	16.1	0.28	203

PGPRO_PWR – дополнительная статистика

- *Планы запросов*
- *Статистика по VACUUM (top tables, wal generated by vacuum ..)*
- *Top wait events на уровне statements*
- *Load distribution by databases/hosts/users/applications*
- *Invalidations by database*

PostgresPro ENTERPRISE MANAGER

Поиск

Admin User

Экземпляры / PgPro Manager repository / Профайлер

Активность экземпляра

БД: postgres

Сервер: test-long-data

Период: 31.03.2022, 00:00 - 01.06.2022, 00:00

ВЫБРАТЬ

Создание отчета

Построить отчет за период 03.04.2022, 17:05:07 - 06.04.2022, 11:41:21?

ОТМЕНА ВЫБРАТЬ ЕЩЕ ПЕРИОД СФОРМИРОВАТЬ ОТЧЕТ

PostgreSQL Instance: tuples

Legend: Tuples returned, Tuples fetched, Tuples inserted, Tuples updated, Tuples deleted

PostgreSQL bgwriter write/sync

PostgreSQL checkpoints count

Legend: Checkpoints buffers written, Background buffers written, Backend buffers written, Number of buffers allocated

ENTERPRISE MANAGER

🌙
🔔
AU Admin User

▾ Server statistics Load distribution SQL query statistics Schema object statistics User function statistics Vacuum-related statistics Cluster settings during the rep

LOAD DISTRIBUTION AMONG HEAVILY LOADED USERS

Resource	Load distribution
Total time (sec.)	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Executed count	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
I/O time (sec.)	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue, darkblue);"></div>
Blocks fetched	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Shared blocks read	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Shared blocks dirtied	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Shared blocks written	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
WAL generated	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Temp and Local blocks written	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue, darkblue);"></div>
Temp and Local blocks read	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue, darkblue);"></div>
Invalidation messages sent	<div style="width: 100%; height: 10px; background: linear-gradient(to right, lightblue, blue);"></div>
Cache resets	

SQL query statistics

TOP SQL BY EXECUTION TIME

Query ID	Plan ID	Database	User	Exec (s)	%Total	I/O time (s)		CPU time (s)		Rows	Execution times (ms)				Executions
						Read	Write	Usr	Sys		Mean	Min	Max	StdErr	
						Запущенный процесс (1)									

← НАЗАД
ПРОФИЛЕР
🔍 Обзор
📄 Отчеты
📷 Снимки статистики
🕒 Расписание
🖥 Серверы

- *Статистика по всем выполненным SQL*

pgpro_stats_statements

pgpro_stats_totals

pgpro_stats_metrics

pgpro_stats_vacuum_tables

pgpro_stats_vacuum_indexes

- *Дополнительные возможности:*

- *Планы запросов*
- *Возможность изменения частоты сбора статистики*
- *Статистика по событиям ожидания*
- *Статистика использования ресурсов при планировании и выполнении*
- *Статистика очистки таблиц и индексов (VACUUM)*
- *Добавление собственных метрик*
- *Трассировка сессий*

PGPRO_STATS – пример статистики

```

SELECT queryid, query, planid, plan, wait_stats FROM pgpro_stats_statements WHERE query LIKE 'select * from test where%';
-[ RECORD 1 ]-----
queryid | 1109491335754870054
query   | select * from test where x >= $1 and x <= $2
planid  | 8287793242828473388
plan    | Gather
        |   Output: i, x
        |   Workers Planned: 2
        |   -> Parallel Seq Scan on public.test
        |       Output: i, x
        |       Filter: ((test.x >= $3) AND (test.x <= $4))
wait_stats | {"IO": {"DataFileRead": 10}, "IPC": {"BgWorkerShutdown": 10}, "Total": {"IO": 10, "IPC": 10, "Total": 20}}
-[ RECORD 2 ]-----
queryid | 1109491335754870054
query   | select * from test where x >= $1 and x <= $2
planid  | -9045072158333552619
plan    | Bitmap Heap Scan on public.test
        |   Output: i, x
        |   Recheck Cond: ((test.x >= $3) AND (test.x <= $4))
        |   -> Bitmap Index Scan on test_x_idx
        |       Index Cond: ((test.x >= $5) AND (test.x <= $6))

```

PGPRO_STATS - трассировка сессий в отдельные файлы

- *Гибкая система фильтров для трассировки:*
 - по *username, client_addr, database_name, pid, application_name*
 - по ресурсам: *duration, plan_time, exec_time, shared_blks_read* и т.д.

- *Управление детализацией вывода explain:*
 - *costs, analyze, verbose, buffers, wal, timing*

PGPRO_STATS – трассировка сессий (фильтры)

```
postgres=# select pgpro_stats_trace_insert('alias', 'filter1','pid', 858795, 'database_name', current_database(), 'explain_analyze',
true,'tracefile','mytrace1','explain_costs', true);
pgpro_stats_trace_insert
-----
                1
(1 row)

postgres=# \x auto
Expanded display is used automatically.
postgres=# SELECT * from pgpro_stats_trace_show();
-[ RECORD 1 ]-----
filter_id      | 1
active         | t
alias          | filter1
tracefile      | mytrace1
pid            | 858795
database_name  | postgres
client_addr    |
application_name |
username       |
queryid        |
planid         |
duration       |
plan_time      |
exec_time      |
user_time      |
system_time    |
rows           |
shared_blks_hit |
shared_blks_read |
shared_blks_fetched |
shared_blks_dirtied |
shared_blks_written |
```

PGPRO_STATS – трассировка сессий (пример файла)

```
Query Text: select * from pg_stat_activity;
Hash Left Join (cost=2.52..4.07 rows=100 width=460) (actual time=0.245..0.263 rows=13 loops=1)
  Output: s.datid, d.datname, s.pid, s.leader_pid, s.usessysid, u.rolname, s.application_name, s.client_addr, s.client_hostname, s.client_port,
s.backend_start, s.xact_start, s.query_start, s.state_change, s.wait_event_type, s.wait_event, s.state, s.backend_xid, s.backend_xmin, s.query_id, s.query, s.backend_type
  Inner Unique: true
  Hash Cond: (s.usessysid = u.oid)
  Buffers: shared hit=2
-> Hash Left Join (cost=1.11..2.39 rows=100 width=396) (actual time=0.201..0.214 rows=13 loops=1)
  Output: s.datid, s.pid, s.leader_pid, s.usessysid, s.application_name, s.client_addr, s.client_hostname, s.client_port, s.backend_start
, s.xact_start, s.query_start, s.state_change, s.wait_event_type, s.wait_event, s.state, s.backend_xid, s.backend_xmin, s.query_id, s.query, s
.backend_type, d.datname
  Inner Unique: true
  Hash Cond: (s.datid = d.oid)
  Buffers: shared hit=1
-> Function Scan on pg_catalog.pg_stat_get_activity s (cost=0.00..1.00 rows=100 width=332) (actual time=0.147..0.149 rows=13 loops=1)
  Output: s.datid, s.pid, s.usessysid, s.application_name, s.state, s.query, s.wait_event_type, s.wait_event, s.xact_start, s.query
_start, s.backend_start, s.state_change, s.client_addr, s.client_hostname, s.client_port, s.backend_xid, s.backend_xmin, s.backend_type, s.ssl
, s.sslversion, s.sslcipher, s.sslbits, s.ssl_client_dn, s.ssl_client_serial, s.ssl_issuer_dn, s.gss_auth, s.gss_princ, s.gss_enc, s.gss_deleg
ation, s.leader_pid, s.query_id
  Function Call: pg_stat_get_activity(NULL::integer)
-> Hash (cost=1.05..1.05 rows=5 width=68) (actual time=0.029..0.033 rows=5 loops=1)
  Output: d.datname, d.oid
  Buckets: 1024 Batches: 1 Memory Usage: 9kB
  Buffers: shared hit=1
-> Seq Scan on pg_catalog.pg_database d (cost=0.00..1.05 rows=5 width=68) (actual time=0.016..0.018 rows=5 loops=1)
  Output: d.datname, d.oid
  Buffers: shared hit=1
-> Hash (cost=1.18..1.18 rows=18 width=68) (actual time=0.022..0.023 rows=19 loops=1)
  Output: u.rolname, u.oid
  Buckets: 1024 Batches: 1 Memory Usage: 10kB
  Buffers: shared hit=1
-> Seq Scan on pg_catalog.pg_authid u (cost=0.00..1.18 rows=18 width=68) (actual time=0.014..0.016 rows=19 loops=1)
  Output: u.rolname, u.oid
  Buffers: shared hit=1
Settings: effective_cache_size = '5749MB', work_mem = '400kB'
```

PGPRO_STATS – графический интерфейс RPEM

Экземпляры / PgPro Manager repository / SQL статистика

Статистика выражения

Query ID	Top level	База данных	Пользователь	Calls	Rows	Total execution time, mc	Total planning time, mc	Blocks time, mc							
Plan ID						Max	Min	Mean	Stdev	Max	Min	Mean	Stdev	Read	Write
2948794686727685322	-176418593907006535	pgpro_manager_repo	pgpro_manager	12074	13229470	8568477.46	0.00			0.00	0.00	0.00	0.00	2712993.95	31215
-4940975716380762037	-7452334651223290015	pgpro_manager_repo	pgpro_manager	4512	4506	4658594.93	0.00			0.00	0.00	0.00	0.00	0.00	0.00
5977927359041152324	-1755255332860400087	pgpro_manager_repo	pgpro_manager	14022	14022	4411282.95	0.00			0.00	0.00	0.00	0.00	100124.05	1152.03
-8345031607887110380	325442572100685824	pgpro_manager_repo	pgpro_manager	4512	4506	2319632.80	0.00			0.00	0.00	0.00	0.00	0.00	0.00
-9009610577462668850	6701048979024298807	pgpro_manager_repo	pgpro_manager	4512	4512	2315056.72	0.00			0.00	0.00	0.00	0.00	0.00	0.00
4049638393080033471	-2873485088126847906	pgpro_manager_repo	pgpro_manager	4512	4512	2283759.93	0.00			0.00	0.00	0.00	0.00	0.00	0.00
-1532875410990978629	-828980528345253256	pgpro_manager_repo	postgres	110465	3048834	1456303.19	0.00			0.00	0.00	0.00	0.00	0.92	0.00
6962627314752790933	-749883763158504123	pgpro_manager_repo	pgpro_manager	69872847	69872847	1407035.74	0.00			0.00	0.00	0.00	0.00	124.96	108

Экземпляры / PgPro Manager repository / stat_statements / Оператор

Query ID 2948794686727685322

Common

Calls: 12074

Rows: 13229470

Пользователь: pgpro_manager

База данных: pgpro_manager_repo

Query ID: 2948794686727685322

Plan ID: -176418593907006535

Plans: 0

Top level: true

Execution time, mc

Total: 8568477.46

Max: 2059.56

Min: 27711

Mean: 709.66

Stdev: 346.14

Rusage: (63074304, 21659648, 8396.62913300003111, 9.66044300000014, 81883.1868, 0.0, 0.0, 0.1832, 572801)

Planing time, mc

Total: 0.00

Max: 0.00

Min: 0.00

Mean: 0.00

Stdev: 0.00

Rusage: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Blocks time, mc

WAL

Bytes, B: 2174586882.00

Records: 13288937

Fpi: 228513

JIT

Functions: 0

Generation time, mc: 0.00

Optimization time, mc: 0.00

Emission: 0

Emission time, mc: 0.00

Inlining: 0

Inlining time, mc: 0.00

Invalid msgs: (0, 0, 0, 0, 0, 0, 0)

Shared blocks

Hit: 1067149726

Read: 14431444.7

Dirtyed: 237299

Written: 4438

Local blocks

Hit: 0

Read: 0

Dirtyed: 0

Written: 0

Temp blocks

Read count: 0

Read time, mc: 0.00

Written count: 0

Written time, mc: 0.00

Wait stats, mc

IO	Value
DataFileRead	10
DataFileWrite	10

LWLock

BufferContent	Value
BufferContent	30

Total

IO	Value
IO	20
LWLock	30
Total	50

Query

```
DELETE FROM monitoring_history
WHERE
value_set < (
SELECT
to_timestamp(
extract(
$,
FROM
current_timestamp
) - $2::int
)
AND instance_id = $3
```

Plan

```
Delete on public.monitoring_history
InitPlan | (returns $0)
-> Result
Output: to_timestamp(((EXTRACT($1 FROM CURRENT_TIMESTAMP) - $2))::double precision)
-> Seq Scan on public.monitoring_history
Output: monitoring_history.ctid
Filter: ((monitoring_history.value_set < $0) AND (monitoring_history.instance_id = $3))
```


- *Автоматическое логирование планов выполнения медленных запросов*
- *Загружаемая библиотека*
- *Настройка пороговых значений и детализации логирования*

AUTO_EXPLAIN - пример



```
postgres=# show shared_preload_libraries;
                                shared_preload_libraries
-----
auto_explain, plugin_debugger, pgpro_scheduler, pg_query_state, aqo, pg_stat_statements,
ats
(1 row)

postgres=# SET auto_explain.log_min_duration = 0;
SET
postgres=# SET auto_explain.log_analyze = true;
SET
postgres=# SELECT count(*)
postgres-#          FROM pg_class, pg_index
postgres-#          WHERE oid = indrelid AND indisunique;
count
-----
      284
(1 row)
```

```
2023-02-17 12:54:15.944 MSK [544419] СООБЩЕНИЕ: duration: 1.737 ms planning: 1.327 ms plan:
Query Text: SELECT count(*)
          FROM pg_class, pg_index
          WHERE oid = indrelid AND indisunique;
Aggregate  (cost=77.39..77.40 rows=1 width=8) (actual time=1.709..1.715 rows=1 loops=1)
-> Hash Join (cost=63.00..76.61 rows=312 width=0) (actual time=0.311..1.682 rows=284 loops=1)
    Hash Cond: (pg_index.indrelid = pg_class.oid)
-> Seq Scan on pg_index  (cost=0.00..12.79 rows=312 width=4) (actual time=0.013..1.314 rows=284 loops=1)
    Filter: indisunique
    Rows Removed by Filter: 60
-> Hash  (cost=53.00..53.00 rows=800 width=4) (actual time=0.285..0.286 rows=800 loops=1)
    Buckets: 1024  Batches: 1  Memory Usage: 37kB
-> Seq Scan on pg_class  (cost=0.00..53.00 rows=800 width=4) (actual time=0.007..0.194 rows=800 loops=1)
```

PG_WAIT_SAMPLING

- *Статистика по событиям ожиданий для процессов БД*
- *Текущие ожидания и история по каждому процессу*
- *Профили ожиданий*

PG_WAIT_SAMPLING - пример

```
postgres=# SELECT * FROM pg_wait_sampling_current;
```

pid	event_type	event	queryid
600675	Client	ClientRead	0
598447	Client	ClientRead	0
598446	Client	ClientRead	0
598442	Client	ClientRead	0
598441	Client	ClientRead	0
598428	Client	ClientRead	0
598420	Client	ClientRead	0
598413	Activity	AutoVacuumMain	0

```
postgres=# SELECT * FROM pg_wait_sampling_history
where pid=598441 order by ts;
```

ts	event_type	event
2022-11-23 13:39:45.211669+03	Client	ClientRead
2022-11-23 13:39:45.221822+03	Client	ClientRead
2022-11-23 13:39:45.231988+03	Client	ClientRead
2022-11-23 13:39:45.242139+03	Client	ClientRead
2022-11-23 13:39:45.252296+03	Client	ClientRead

```
postgres=# SELECT * FROM
pg_wait_sampling_get_current(598441);
```

pid	event_type	event	queryid
598441	Client	ClientRead	0

```
postgres=# select * from pg_wait_sampling_profile
where pid=598441;
```

pid	event_type	event	count
598410	Activity	CheckpointerMain	368674
598410	IO	ControlFileSyncUpdate	12
598410	IO	DataFileWrite	4
598410	IO	DataFileSync	9
598410	Timeout	CheckpointWriteDelay	14810
598410	IO	DataFileFlush	10
598410	IO	WALSync	11

Сценарии - 2

- Долго работает регламентная задача – сколько осталось ?
- Долго работает запрос - по какому плану и какой прогресс ?
- Как логировать «долгий» запрос ?

Мониторинг прогресса выполнения долгих операций

```
postgres=# select viewname from pg_views where viewname like '%progres%';
```

```
viewname
```

```
-----  
pg_stat_progress_analyze  
pg_stat_progress_vacuum  
pg_stat_progress_cluster  
pg_stat_progress_create_index  
pg_stat_progress_basebackup  
pg_stat_progress_copy
```

PG_QUERY_STATE

- *Статистика по выполнению запроса в реальном времени*
- *Вызов через табличную функцию*
- *Детализация статистики при помощи параметров*
- *Progress bar (в разработке)*

PG_QUERY_STATE - пример

postgres=# **select * from flights a, ticket_flights b where a.flight_id = b.flight_id;** ← *долго выполняется...*


postgres=# **select pid, query from pg_stat_activity where query like 'select * from flights%';** ← *определяем PID сессии, выполняющей запрос*

```
-----+-----
89725 | select * from flights a, ticket_flights b where a.flight_id = b.flight_id;
```

postgres=# **select * from pg_query_state(pid => 89725, costs => true, timing => true, buffers => true);** ← *смотрим статистику по активному запросу*

```
-[ RECORD 1 ]+-----
| select * from flights a, ticket_flights b where a.flight_id = b.flight_id;
| Hash Join (cost=2977.44..85590.07 rows=2360335 width=95) (Current loop: actual time=25.262..1240.195 rows=1188262, loop number=1)
| Hash Cond: (b.flight_id = a.flight_id)
| Buffers: shared hit=20629, temp read=7751 written=15856
| -> Seq Scan on ticket_flights b (cost=0.00..43438.35 rows=2360335 width=32) (Current loop: actual time=0.008..210.506 rows=2360335, loop number=1)
| Buffers: shared hit=19835
| -> Hash (cost=1450.64..1450.64 rows=65664 width=63) (Current loop: actual time=25.017..25.017 rows=65664, loop number=1)
| Buckets: 4096 Batches: 32 Memory Usage: 239kB
| Buffers: shared hit=794, temp written=678
| -> Seq Scan on flights a (cost=0.00..1450.64 rows=65664 width=63) (Current loop: actual time=0.010..7.876 rows=65664, loop number=1)
| Buffers: shared hit=794
```


PG_QUERY_STATE – графический интерфейс RPEM



ENTERPRISE MANAGER

← НАЗАД

ДЕТАЛИ ЭКЗЕМПЛЯРА

- Обзор
- Базы данных
- Метрики
- Сессии**
- SQL статистика
- Журнал событий
- Резервные копии
- Параметры
- Аутентификация

Поиск

Admin User
Администратор инстанса, П...

Экземпляры / PgPro Manager repository / Сессии

Текущая активность процессов

Обновление: 2 сек.

СБРОСИТЬ ФИЛЬТРЫ

State: active

query ↓ query_start ↓ state ↓ wait_event_type ↓ pg_q

query	query_start	state	wait_event_type	pg_q
SELECT t.ticket_no,pg_sleep(1) FROM flights f JOIN ticket_flights tf ON f.flight_id = tf.flight_id JOIN tickets t ON tf.ticket_	3.09.2023, 14:58:13.11	active	Timeout	pg_query_sta
SELECT t.ticket_no,pg_sleep(1) FROM flights f JOIN ticket_flights tf ON f.flight_id = tf.flight_id JOIN tickets t ON tf.ticket_	3.09.2023, 14:58:13.11	active	Timeout	pg_query_sta
SELECT t.ticket_no,pg_sleep(1) FROM flights f JOIN ticket_flights tf ON f.flight_id = tf.flight_id JOIN tickets t ON tf.ticket_	3.09.2023, 14:58:13.11	active	Timeout	pg_query_sta

СКРЫТЬ/ОТОБРАЗИТЬ КОЛОНКИ

План активного запроса

34

PG_QUERY_STATE – графический интерфейс PPEM

The screenshot displays the PostgresPro Enterprise Manager interface. A modal window titled "План активного запроса" (Active Query Plan) is open, showing the SQL query and its execution plan. The query is a SELECT statement with a pg_sleep(1) function. The execution plan shows a Gather operation followed by a Parallel Hash Join. A red arrow points to a button labeled "ОБНОВИТЬ СТАТИСТИКУ ВЫПОЛНЕНИЯ" (Refresh Execution Statistics).

План активного запроса

```

SELECT
  t.ticket_no,
  pg_sleep(1)
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 > '2010-09-01'
  AND f.actual_arrival < f.scheduled_arrival + interval '1 hour'
  AND tf.fare_conditions = 'Business';

```

Plan:

```

Gather (cost=110828.18..220220.57 rows=295299 width=18) (Current
loop: actual time=1606.014..215989.753 rows=214, loop number=1)
  AQP not used, fss=-313062150
  Output: t.ticket_no, (pg_sleep('1'::double precision))
  Workers Planned: 2
  Workers Launched: 2
  Buffers: shared hit=249 read=21503, temp read=2861 written=9136
  I/O Timings: shared/local read=270.391, temp read=4.149
  write=25.137
  -> Parallel Hash Join (cost=109828.18..189690.67 rows=123041
width=18) (Current loop: actual time=1605.832..215989.159 rows=214,
loop number=1)

```

ОБНОВИТЬ СТАТИСТИКУ ВЫПОЛНЕНИЯ

Управление планами запросов

- `pg_hint_plan`

- `plantuner`

- `aqo`



- `sr_plan`

- `replan`

Сценарии - 3

- Запрос от «коробочного» приложения работает медленно, что делать ?
- Как «заставить» оптимизатор исправлять собственные ошибки ?
- Как обеспечить стабильную производительность запросов ?

Диагностика

- explain analyze
- pgpro_stats – session trace
- auto_explain
- debug_print_parse / debug_print_rewritten / debug_print_plan

- *Корректировка планов выполнения запросов при помощи инструкций*
- *Инструкции по методам сканирования, методам и порядку соединений*
- *Интерактивный режим и возможность сохранения инструкций*

PG_HINT_PLAN

```
postgres=# LOAD 'pg_hint_plan';
LOAD
postgres=#
```

```
postgres=# /*+
postgres*#   HashJoin(a b)
postgres*#   SeqScan(a)
postgres*# */
postgres=# EXPLAIN SELECT *
postgres=#   FROM pgbench_branches b
postgres=#   JOIN pgbench_accounts a ON b.bid = a.bid
postgres=#   ORDER BY a.aid;
```

QUERY PLAN

```
-----
Sort (cost=31465.84..31715.84 rows=100000 width=197)
  Sort Key: a.aid
  -> Hash Join (cost=1.02..4016.02 rows=100000 width=197)
    Hash Cond: (a.bid = b.bid)
    -> Seq Scan on pgbench_accounts a (cost=0.00..2640.00 rows=100000 width=97)
    -> Hash (cost=1.01..1.01 rows=1 width=100)
      -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=100)
```

(7 rows)

```
postgres=# INSERT INTO hint_plan.hints(norm_query_string, application_name, hints)
postgres=#   VALUES (
postgres(#     'EXPLAIN (COSTS false) SELECT * FROM t1 WHERE t1.id = ?;',
postgres(#     'psql',
postgres(#     'SeqScan(t1)'
postgres(#   );
INSERT 0 1
postgres=# UPDATE hint_plan.hints
postgres=#   SET hints = 'IndexScan(t1)'
postgres=#   WHERE id = 1;
UPDATE 1
postgres=# DELETE FROM hint_plan.hints
postgres=#   WHERE id = 1;
DELETE 1
postgres=#
```

- *Управление видимостью индексов для планировщика*
- *Инструкции, позволяющие запретить или разрешить индекс*
- *Через параметры GUC*


```
=# LOAD 'plantuner';  
=# create table test(id int);  
=# create index id_idx on test(id);  
=# create index id_idx2 on test(id);
```

```
=# \d test  
Table "public.test"  
Column | Type | Modifiers  
-----+-----+-----  
id | integer |  
Indexes:  
"id_idx" btree (id)  
"id_idx2" btree (id)
```

```
=# explain select id from test where id=1;  
QUERY PLAN  
-----  
Bitmap Heap Scan on test (cost=4.34..15.03 rows=12 width=4)  
Recheck Cond: (id = 1)  
-> Bitmap Index Scan on id_idx2 (cost=0.00..4.34 rows=12 width=0)  
Index Cond: (id = 1)
```

```
(4 rows)  
=# set enable_seqscan=off;  
=# set plantuner.disable_index='id_idx2';  
=# explain select id from test where id=1;  
QUERY PLAN  
-----  
Bitmap Heap Scan on test (cost=4.34..15.03 rows=12 width=4)  
Recheck Cond: (id = 1)  
-> Bitmap Index Scan on id_idx (cost=0.00..4.34 rows=12 width=0)  
Index Cond: (id = 1)
```

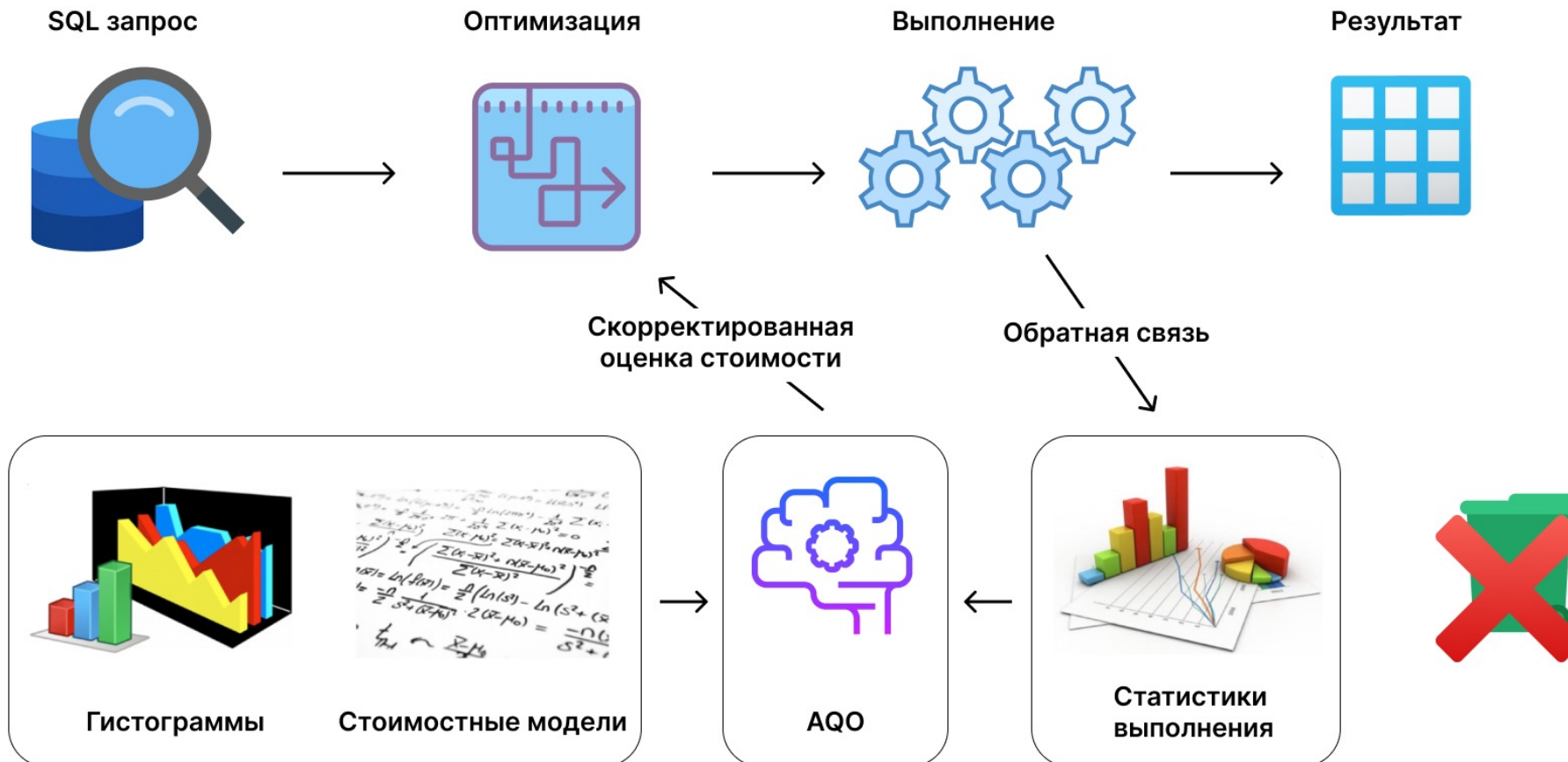
```
(4 rows)  
=# set plantuner.disable_index='id_idx2,id_idx';  
=# explain select id from test where id=1;  
QUERY PLAN  
-----  
Seq Scan on test (cost=10000000000.00..10000000040.00 rows=12 width=4)  
Filter: (id = 1)
```

```
(2 rows)  
=# set plantuner.enable_index='id_idx';  
=# explain select id from test where id=1;  
QUERY PLAN  
-----  
Bitmap Heap Scan on test (cost=4.34..15.03 rows=12 width=4)  
Recheck Cond: (id = 1)  
-> Bitmap Index Scan on id_idx (cost=0.00..4.34 rows=12 width=0)  
Index Cond: (id = 1)
```

```
(4 rows)
```

- *Адаптивная оптимизация по обратной связи – cardinality feedback*
- *Расширение возможностей встроенного оптимизатора запросов*
- *Сохраняет статистику выполнения запроса и использует ее*
- *Использует ML для оценки кардинальности*
- *Различные режимы использования*

AQO - концепция



AQO – пример использования

1) Включаем AQO в режиме обучения и информирования:

```
shared_preload_libraries='aqo'  
create extension aqo;
```

```
SET aqo.enable=on;
```

```
SELECT * FROM a, b WHERE a.id=b.id; ← выполняем запрос
```

2) Запускаем запрос в режиме EXPLAIN ANALYZE несколько раз и наблюдаем меняется ли план:

```
EXPLAIN ANALYZE SELECT * FROM a, b WHERE a.id=b.id;
```

```
...
```

```
...
```

```
EXPLAIN ANALYZE SELECT * FROM a, b WHERE a.id=b.id;
```

3) Когда видим, что AQO подобрал оптимальный план – отключаем режим обучения:

```
SET aqo.mode = frozen;
```

AQO – пример работы

```
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'::timestampz
      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) (actual rows=859656 loops=1)
        AQO: rows=73356, error=-1072%
        Index Cond: ((fare_conditions)::text = 'Business'::text)
```

```
Planning Time: 0.615 ms
Execution Time: 1779.805 ms
Using aqo: true
AQO mode: INTELLIGENT
JOINS: 1
(23 rows)
```

AQO – пример работы

```

Gather (cost=110364.34..274069.84 rows=769494 width=14) (actual rows=771441 loops=1)
  AQP: rows=769494, error=-0%
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Hash Join (cost=109364.34..196120.44 rows=320622 width=14) (actual rows=257147 loops=3)
    AQP: rows=769494, error=-0%
    Hash Cond: (tf.flight_id = f.flight_id)
    -> Parallel Hash Join (cost=102697.53..183882.36 rows=358190 width=18) (actual rows=286552 loops=3)
      AQP: rows=859656, error=0%
      Hash Cond: (t.ticket_no = tf.ticket_no)
      -> Parallel Seq Scan on tickets t (cost=0.00..61827.52 rows=1229107 width=14) (actual rows=983286 loops=3)
        AQP: rows=2949857, error=0%
      -> Parallel Hash (cost=97673.20..97673.20 rows=273626 width=18) (actual rows=286552 loops=3)
        Buckets: 8192 Batches: 128 Memory Usage: 480kB
        -> Parallel Index Scan using ticket_flights_fare_conditions_idx on ticket_flights tf (cost=0.43..97673.20 rows=273626 width=18) (
          actual rows=286552 loops=3)
            AQP: rows=656702, error=-31%
            Index Cond: ((fare_conditions)::text = 'Business'::text)
        -> Parallel Hash (cost=4850.87..4850.87 rows=110636 width=4) (actual rows=62854 loops=3)
          Buckets: 16384 Batches: 32 Memory Usage: 384kB
          -> Parallel Seq Scan on flights f (cost=0.00..4850.87 rows=110636 width=4) (actual rows=62854 loops=3)
            AQP: rows=188081, error=-0%
            Filter: ((scheduled_departure > '1999-01-01 00:00:00+03'::timestamp with time zone) AND (actual_arrival < (scheduled_arri
val + '01:
00:00'::interval)))
          Rows Removed by Filter: 8768
        Planning Time: 0.859 ms
        Execution Time: 432.285 ms
        Using aqp: true
        AQP mode: INTELLIGENT
        JOINS: 2

```

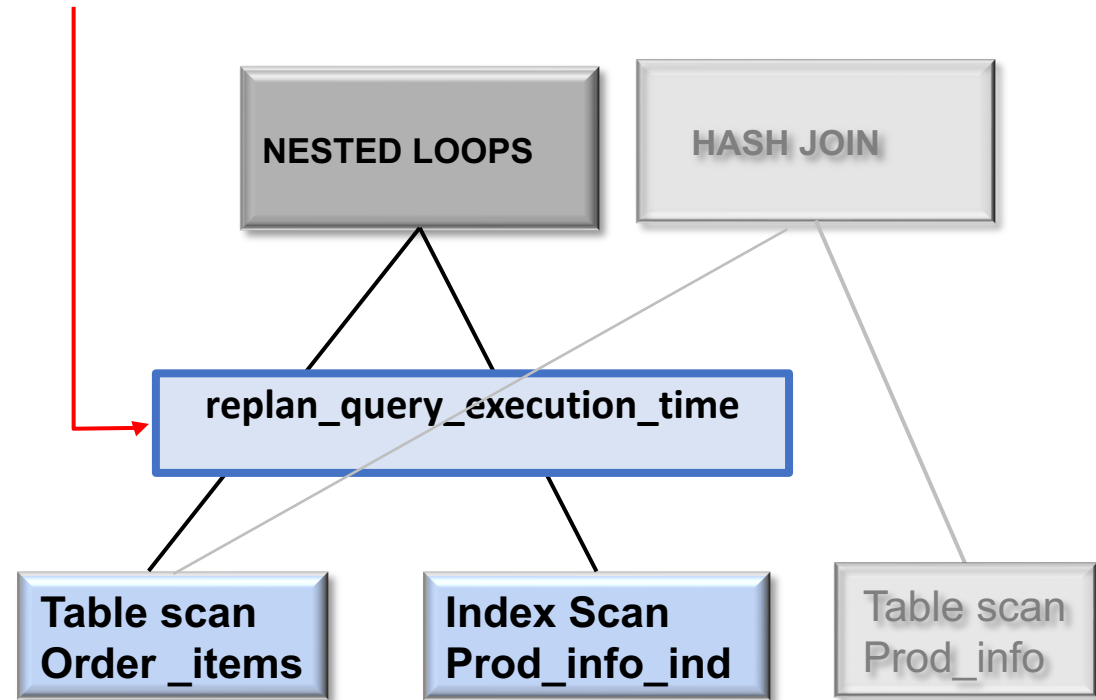
REPLAN – перепланирование запросов в реальном времени

- *Adaptive Query Executor (AQE)*
- *Re-оптимизация запроса во время выполнения*
- *Триггеры перепланирования задаются параметрами*
- *Отключено по умолчанию*
- *Детали перепланирования - в логе и выводе Explain*

REPLAN – механизм

- Если оптимизатор запросов «ошибся» при выборе плана – он может исправиться прямо во время выполнения запроса
- Если в процессе выполнения срабатывает установленный триггер (таймаут), то запрос прерывается и передается оптимизатору для генерации нового плана с учетом накопленной статистики, далее запрос уходит на повторное выполнение
- В новом плане могут поменяться методы доступа к данным, типы и порядок соединений и т.д.

Например: при повторном планировании оптимизатор решил поменять метод соединения с nested loops на hash join



По умолчанию выбран **nested loops join**


```
Aggregate (cost=560.06..560.07 rows=1 width=8) (actual time=4.705..4.707 rows=1 loops=1)
NodeSign: 4127104911444856927
Cardinality: -1
Groups Number: -1
Output: count(*)
-> Hash Join (cost=232.05..533.39 rows=10667 width=0) (actual time=4.681..4.701 rows=70 loops=1)
NodeSign: 15478158356720060206
Cardinality: -1
Groups Number: -1
Hash Cond: (rt2.x = q1.x)
-> Seq Scan on public.replan_test rt2 (cost=0.00..154.67 rows=10667 width=4) (actual time=0.007..0.785 rows=10100 loops=1)
NodeSign: 17491169463296369090
Cardinality: -1
Groups Number: -1
Output: rt2.x, rt2.payload
-> Hash (cost=231.99..231.99 rows=5 width=4) (actual time=3.083..3.084 rows=70 loops=1)
NodeSign: 17790666881744499777
Output: q1.x
Buckets: 1024 Batches: 1 Memory Usage: 11kB
-> Subquery Scan on q1 (cost=230.42..231.99 rows=5 width=4) (actual time=3.026..3.073 rows=70 loops=1)
NodeSign: 17790666881744499777
Output: q1.x
-> HashAggregate (cost=230.42..231.94 rows=5 width=12) (actual time=3.025..3.065 rows=70 loops=1)
NodeSign: 13744336777062894583
Cardinality: 5
Groups Number: 101
Output: rt1.x, rt1.payload, NULL::integer
Group Key: rt1.x, rt1.payload
Filter: (avg(rt1.x) > '30'::numeric)
Batches: 1 Memory Usage: 80kB
Rows Removed by Filter: 31
-> Seq Scan on public.replan_test rt1 (cost=0.00..154.67 rows=10100 width=8) (actual time=0.004..0.865 rows=
```

```
NodeSign: 14925644762129093451
Cardinality: 10100
Groups Number: -1
Output: rt1.x, rt1.payload

Planning Time: 0.135 ms
Execution Time: 4.746 ms
Replan Active: true
Table Entries: 2
Controlled Statements: 1
Replanning Attempts: 1
Total Execution Time: 15.530 ms
```

REPLAN – информация о перепланировании в журнале

```
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 rows=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=2280 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
```

- *Возможность «фиксации» плана запроса*
- *Стабильность планов*
- *Простая настройка*

```
shared_preload_libraries = 'sr_plan'  
sr_plan.enable = 'true'
```

```
CREATE EXTENSION sr_plan;
```

```
SET sr_plan.auto_tracking = on;
```

```
EXPLAIN SELECT count(*) FROM a WHERE x = 1 OR (x > 11 AND x < 22) OR x = 22;
```

```
Custom Scan (SRScan) (cost=1.60..0.00 rows=1 width=8)
```

```
Plan is: tracked
```

```
Query ID: 5393873830515778388
```

```
Const hash: 0
```

```
-> Aggregate (cost=1.60..1.61 rows=1 width=8)
```

```
    -> Seq Scan on a (cost=0.00..1.60 rows=2 width=0)
```

```
        Filter: ((x = $1) OR ((x > $2) AND (x < $3)) OR (x = $4))
```

```
-- Укажите возвращенный 'Tracked plan ID':
```

```
SELECT sr_plan_freeze(5393873830515778388, 0);
```

```
RESET sr_plan.auto_tracking;
```

SR_PLAN - новое

- *Авто-регистрация плана: `sr_plan.auto_tracking`*
- *Повышение стабильности работы*
- *Снижение накладных расходов*

Итоги

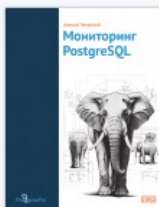
- *В СУБД Postgres Pro входит обширный набор средств мониторинга*
- *Модули поставляются в составе дистрибутива и готовы к использованию*
- *Статистика динамическая, кумулятивная, реального времени*
- *Возможность хранить историю статистики в виде снимков*
- *Интерактивное и адаптивное управление планами запросов*
- *Диагностика и трассировка*

Планы разработки

- *Управление планами запросов – аналог Oracle SPM*
- *Динамическое перепланирование запроса – дополнительные триггеры*
- *Дальнейшие улучшения механизма адаптивной оптимизации*
- *Снимки состояния активных сессий*

Мониторинг PostgreSQL

[Главная](#) > [Образование](#) > [Книги](#)



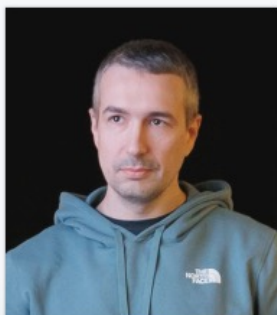
Лесовский А. В.

Мониторинг PostgreSQL. — М.: Бумба, 2024. — 247 с.

ISBN 978-5-907754-42-3

Мониторинг PostgreSQL составляет важную часть работы администратора, помогая отвечать на многие вопросы, связанные с производительностью. Эта книга всесторонне охватывает обширную тему мониторинга, соединяя в себе справочные материалы об имеющемся инструментарии, практические приемы его использования и способы интерпретации полученных данных. Знание внутреннего устройства PostgreSQL и особенностей мониторинга, почерпнутое из этой книги, поможет в долгосрочной перспективе эффективно эксплуатировать СУБД и успешно решать возникающие задачи.

Книга предназначена для администраторов баз данных, системных администраторов, специалистов по надежности.

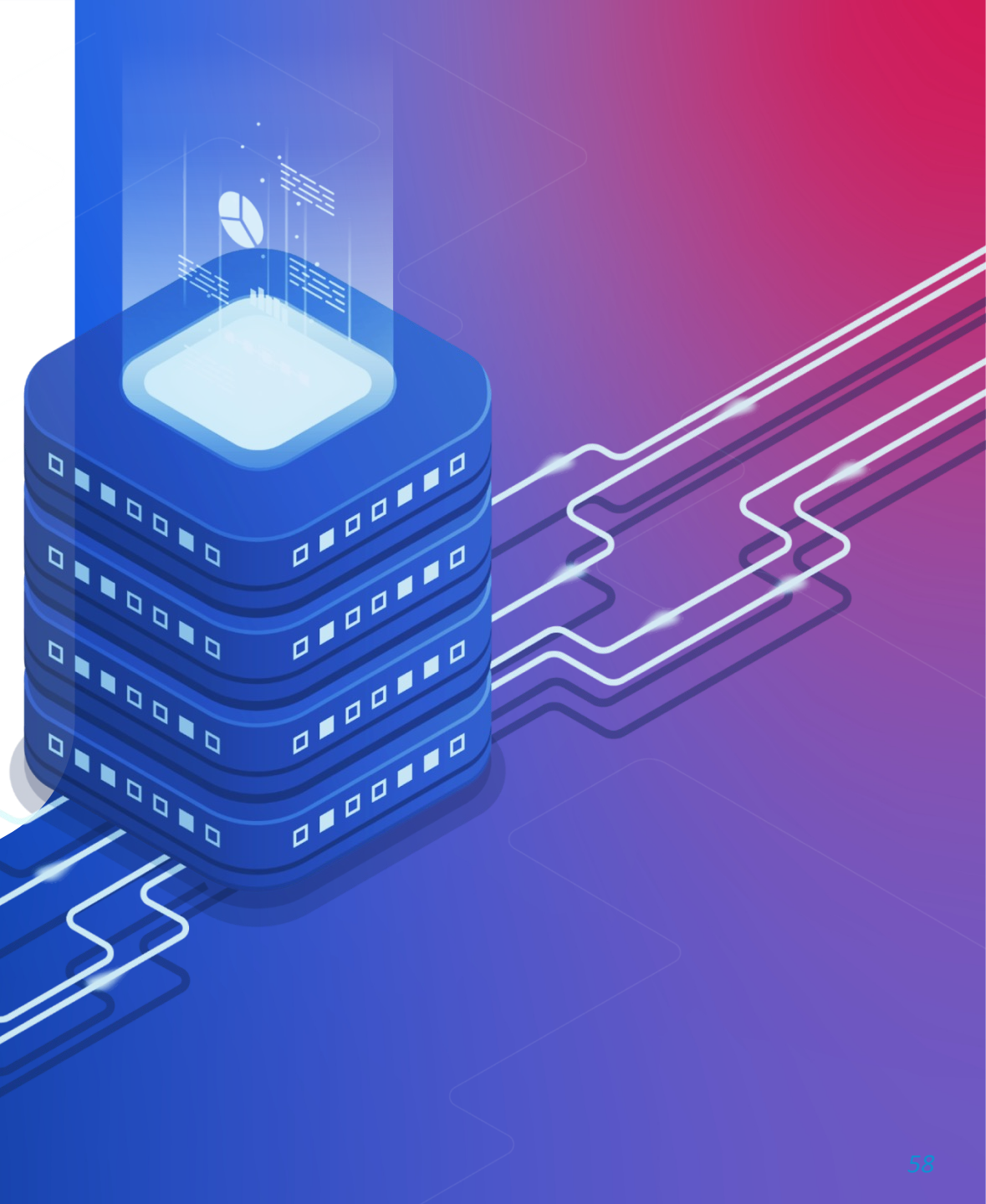


Алексей Лесовский — профессиональный администратор баз данных, системный администратор, разработчик, devops-инженер. Почти 20 лет он занимается задачами эксплуатации больших и сложных систем, проектирования и разработки программного обеспечения.



PosgresPro

Спасибо



*Протестировать
СУБД Postgres Pro:*



117036, Москва, ул. Дмитрия Ульянова, 7А



8 (495) 150-06-91



sales@postgrespro.ru

http://postgrespro.ru