

Онлайн образование



Проверить, идет ли
запись

Меня хорошо видно && слышно?



Тема вебинара

Оптимизация производительности. Профилирование. Мониторинг

Курочкин Константин

Ведущий администратор БД

«Medindex»

Telegram https://t.me/konstantin_kurochkin

Правила вебинара



Активно участвуем



Off-topic обсуждаем в telegram



Задаем вопрос
в чат или ГОЛОСОМ



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом

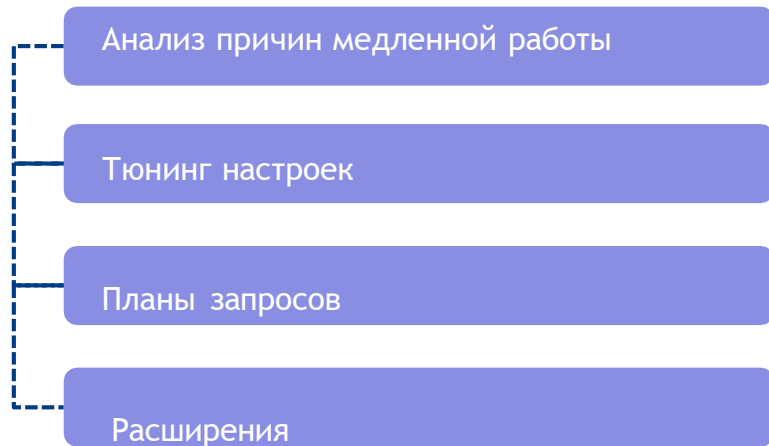


Документ



Ответьте себе или
задайте вопрос

Маршрут вебинара



Цели вебинара

После занятия вы сможете

1. Определять почему все медленно работает
2. Вносить изменения в структуру БД для улучшения производительности
3. Вносить изменения в настройки СУБД для улучшения производительности



Смысл

Зачем вам это уметь

1. Понимать узкие места и учитывать их при проектировании изначально. Ну или пытаться 😊
2. Знать когда это вопрос DBA, а когда разработчиков
3. Уметь «творчески» подходить к вопросу производительности

Итак оно «тупит», с чего начать?



Итак оно «тупит», с чего начать?

Запросы?

Настройки?

Индексы?

WAL? VACUM?

Checkpoint?

Блокировки?

Статистика?



Off Topic! Сперва попробуйте очевидное

Прежде чем заниматься оптимизацией

1. Посмотрите последний Commit в Git-е
2. Если есть ORM – последнюю миграцию
3. Или изменение postgresql.conf



Естественно для этого весь код должен быть в git-е.
Postgresql.conf должен быть в git-е и накатываться на production сервера автоматически (если не управляется внешней системой).
Все изменения в хранимые процедуры и структуру БД должны вноситься скриптами которые версионизируется, если это не делает ORM.

Прежде чем заниматься оптимизацией

Убедитесь что PostgreSQL установлен на Linux системе

- Работа со множеством процессов не принята в Windows
- NTFS не самая оптимальная ФС для работы Postgres, используется другой способ доступа к диску – другие планы запросов
- Антивирусная защита Windows

Но если надо – можно и на Windows.

Только учитывать разницу в планах запросов: dev, test и prod контуры должны быть на одной и той же ОС.



Прежде чем заниматься оптимизацией

Проверьте оборудование

CPU – для OLTP больше быстродействующих ядер, для DWH процессоры с большим L3 кэшем (полезно для параллельных запросов).

RAM – чем больше, тем лучше.))

Disk – лучше SSD.

Рекомендации:

1. Сохранять WAL файлы и данные на отдельных дисках.
2. Использовать отдельные табличные пространства и диски для индексов и данных (особенно для SATA дисков).
3. Отключить atime – время последнего доступа к файлу.

<https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization>



Прежде чем заниматься оптимизацией

Проверьте загрузку оборудование

top/htop – нет смысла браться за оптимизацию, если загрузка процессора в среднем составляет 70% и выше при этом ядра загружены более менее равномерно и загрузка процессора не выростала скачкообразно.

iostat – утилита для использования дисковых разделов (входит в пакет sysstat) Ключи:

- c - отобразить только информацию об использовании процессора;
- d - отобразить только информацию об использовании устройств;
- h - выводить данные в отчёте в удобном для чтения формате;
- k - выводить статистику в килобайтах;
- m - выводить статистику в мегабайтах;
- o **JSON** - выводить статистику в формате JSON;

...

<https://losst.ru/opisanie-iostat-linux>

Прежде чем заниматься оптимизацией

Проверьте память

huge pages – подключить (при работе с большими объемами данных).

```
grep HUGETLB /boot/config -$(uname -r)  
CONFIG_CGROUP_HUGETLB=y CONFIG_HUGETLBFS=y CONFIG_HUGETLB_PAGE=y  
grep Huge /proc/meminfo
```

```
# head -1 /var/lib/postgresql/14/main/postmaster.pid      3076  
# grep ^VmPeak /proc/3076/status                          VmPeak: 4742563 kB  
# echo $((4742563 / 2048 + 1))                            2316  
# sysctl -w vm.hugepages=2316  
в postgresql.conf параметр huge_page = try
```

transparent_hugepage (THP прозрачные огромные страницы) - отключить.

```
cat /sys/kernel/mm/transparent_hugepage/enabled  
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Прежде чем заниматься оптимизацией

Проверьте память

swappiness – определяет частоту сброса данных из RAM в SWAP (значение от 0 до 100). Для PostgreSQL рекомендуется от 1 до 5.

```
cat /proc/sys/vm/swappiness sysctl vm.swappiness=5  
echo 'vm.swappiness=5' >> /etc/sysctl.conf
```

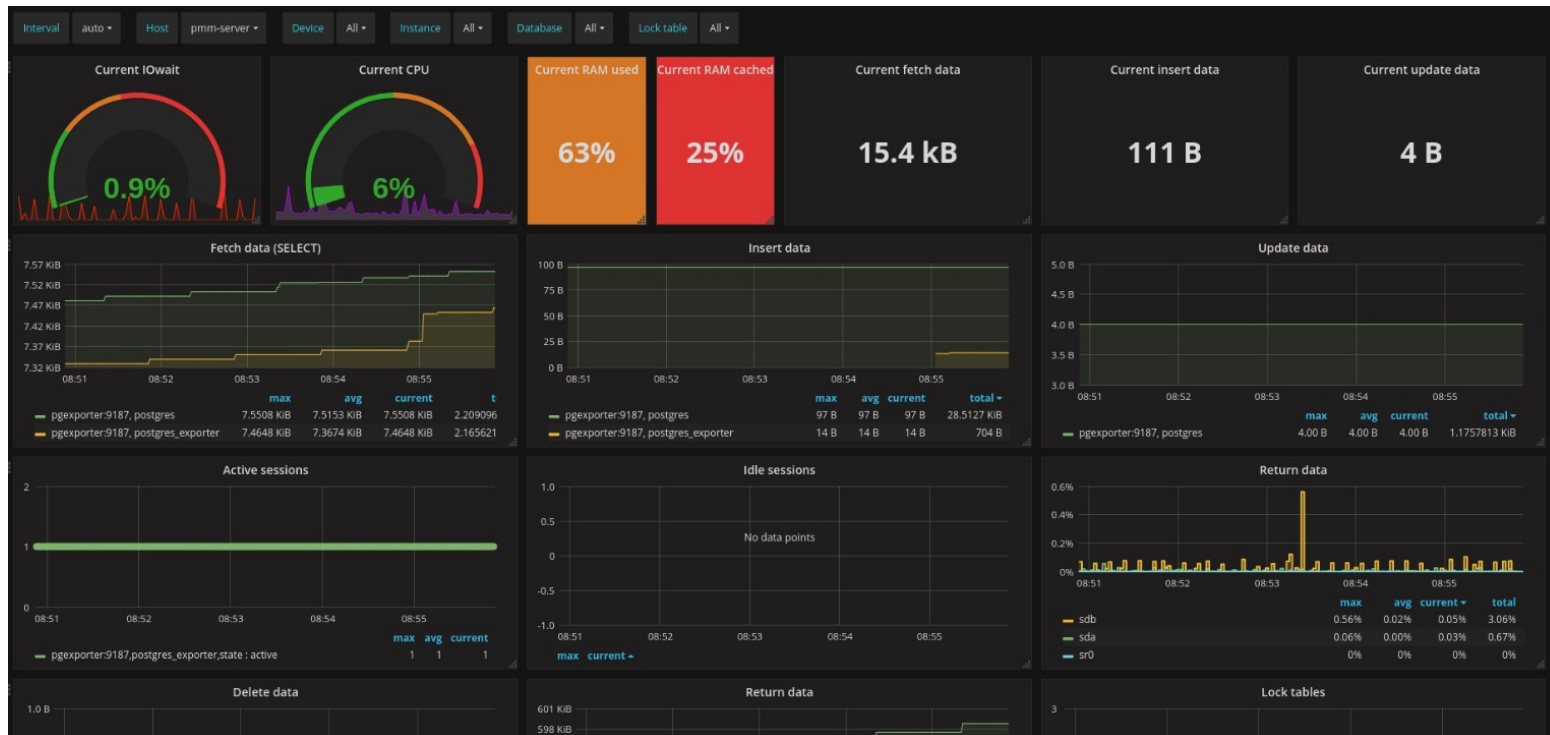
Прежде чем заниматься оптимизацией

Воспользуйтесь Prometheus + Grafana



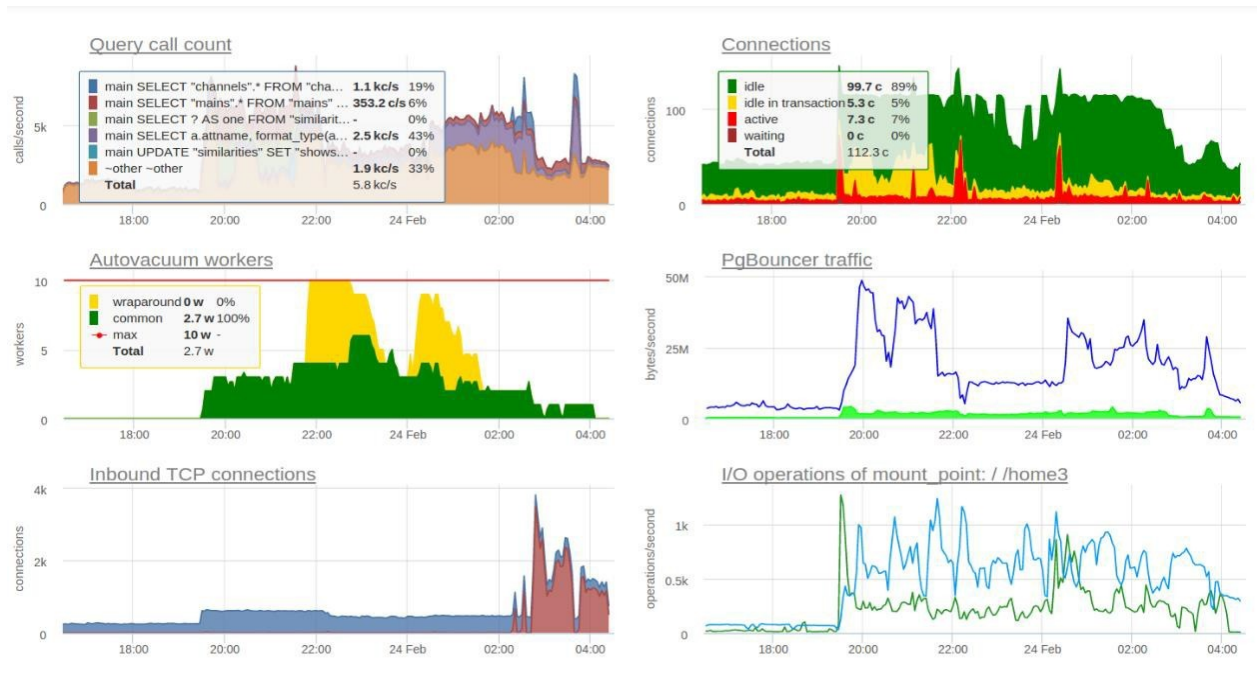
Прежде чем заниматься оптимизацией

или Percona



Прежде чем заниматься оптимизацией

или Okmetr

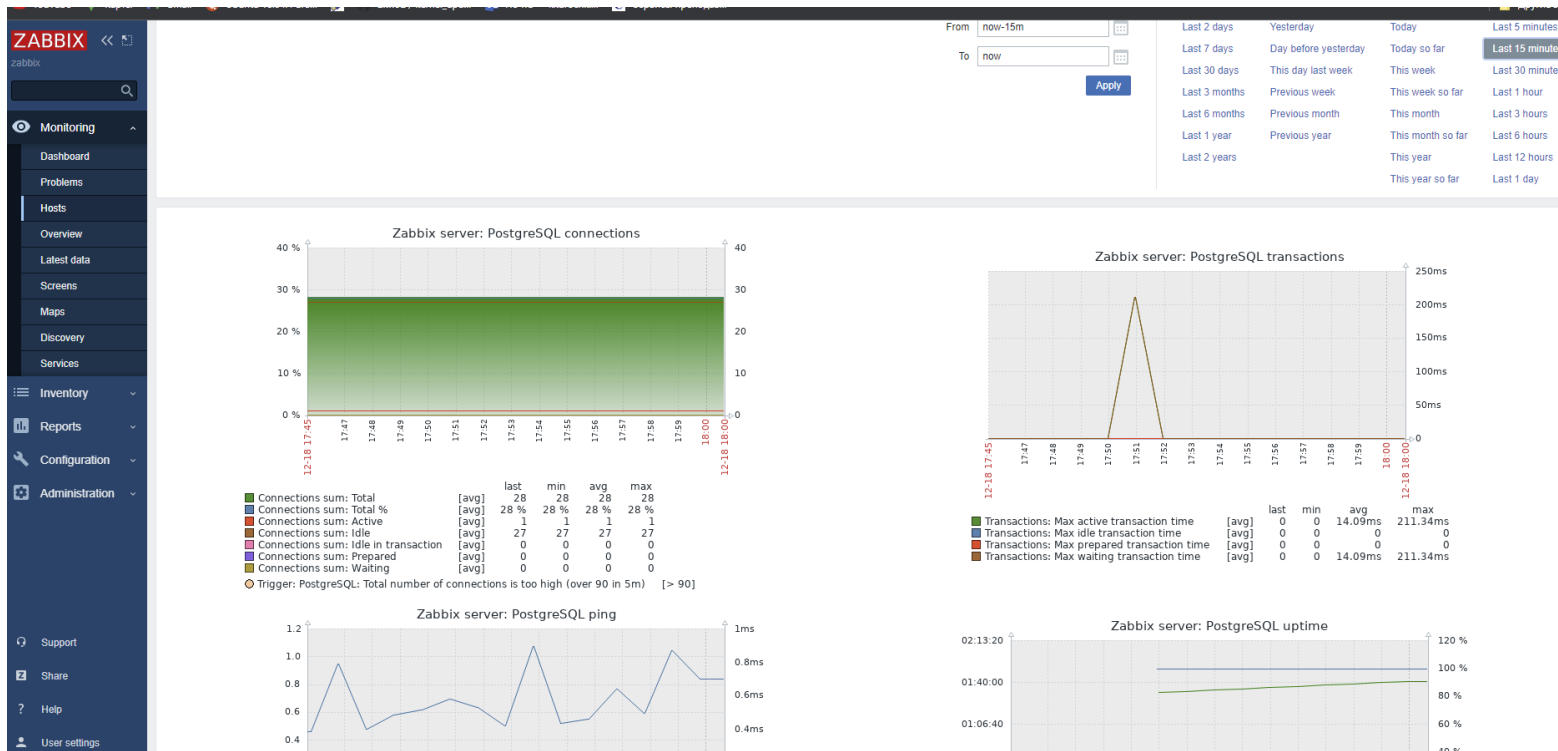


Zabbix

<https://www.zabbix.com/download>

<https://www.zabbix.com/integrations/postgresql>

<https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates/db/postgresql?at=refs%2Fheads%2Frelease%2F5.0>



Представление pg_stat_activity

Самое популярное – «найти большого и убить» если кто-то написал что-то вроде

```
select * from * 😊
```

Получаем активные запросы длительностью более 5 секунд:

```
SELECT now() - query_start as "runtime", username, datname, wait_event_type,  
state, query FROM pg_stat_activity WHERE now() - query_start > '5  
seconds'::interval and state='active' ORDER BY runtime DESC;
```

State = 'idle' тоже собственно вызывают подозрения.
Но хуже всего – idle in transaction!

Далее убиваем:

```
SELECT pg_cancel_backend(procpid); для active
```

```
SELECT pg_terminate_backend(procpid); для idle
```

	column_name character varying	data_type character varying
1	datid	oid
2	datname	name
3	pid	integer
4	usesysid	oid
5	username	name
6	application_name	text
7	client_addr	inet
8	client_hostname	text
9	client_port	integer
10	backend_start	timestamp with time...
11	xact_start	timestamp with time...
12	query_start	timestamp with time...
13	state_change	timestamp with time...
14	waiting	boolean
15	state	text
16	query	text

Представление pg_stat_activity

«Повисшие транзакции» - зло

```
SELECT pid, xact_start, now() - xact_start AS duration FROM pg_stat_activity  
WHERE state LIKE '%transaction%' ORDER BY duration DESC;
```

В общем случае транзакции должны выполняться как можно меньшее время.
Если что-то висит несколько часов, то это что-то не нормальное.
Открытые транзакции влияют на работу VACUUM, WAL, репликацию.

Представление pg_stat_statements

```
create extension pg_stat_statements;
```

```
shared_preload_libraries = 'pg_stat_statements'
```

По умолчанию чаще всего не включено, но очень нужно.

По этому представлению можно увидеть не только выполняемые в данный момент запросы, но и уже выполненные, кроме того – с анализом их воздействия на сервер.

	column_name name	data_type character varying
1	userid	oid
2	dbid	oid
3	queryid	bigint
4	query	text
5	calls	bigint
6	total_time	double precision
7	min_time	double precision
8	max_time	double precision
9	mean_time	double precision
10	stddev_time	double precision
11	rows	bigint
12	shared_blks_hit	bigint
13	shared_blks_read	bigint
14	shared_blks_dirtied	bigint
15	shared_blks_written	bigint
16	local_blks_hit	bigint
17	local_blks_read	bigint
18	local_blks_dirtied	bigint
19	local_blks_written	bigint
20	temp_blks_read	bigint
21	temp_blks_written	bigint
22	blk_read_time	double precision
23	blk_write_time	double precision

Представление pg_stat_statements

ТОП по загрузке CPU

```
SELECT substring(query, 1, 50) AS short_query,  
round(total_time::numeric, 2) AS total_time, calls, rows,  
round(total_time::numeric / calls, 2) AS avg_time,  
round((100 * total_time/ sum(total_time::numeric) OVER ()):::numeric, 2) AS percentage_cpu  
FROM pg_stat_statementsORDER BY total_timeDESC LIMIT 20;
```

ТОП по времени выполнения

```
SELECT substring(query, 1, 100) AS short_query,  
round(total_time::numeric, 2) AS total_time, calls, rows,  
round(total_time::numeric / calls, 2) AS avg_time,  
round((100 * total_time/ sum(total_time::numeric) OVER ()):::numeric, 2) AS percentage_cpu  
FROM pg_stat_statementsORDER BY avg_timeDESC LIMIT 20;
```

Представление pg_stat_user_tables

Системное представление содержит данные о таблицах.

Большое зло —«последовательное чтение» больших таблиц.

```
SELECT schemaname, relname, seq_scan, seq_tup_read,  
seq_tup_read/ seq_scan AS avg, idx_scan  
FROM pg_stat_user_tables WHERE seq_scan > 0  
ORDER BY seq_tup_read DESC LIMIT 25;
```

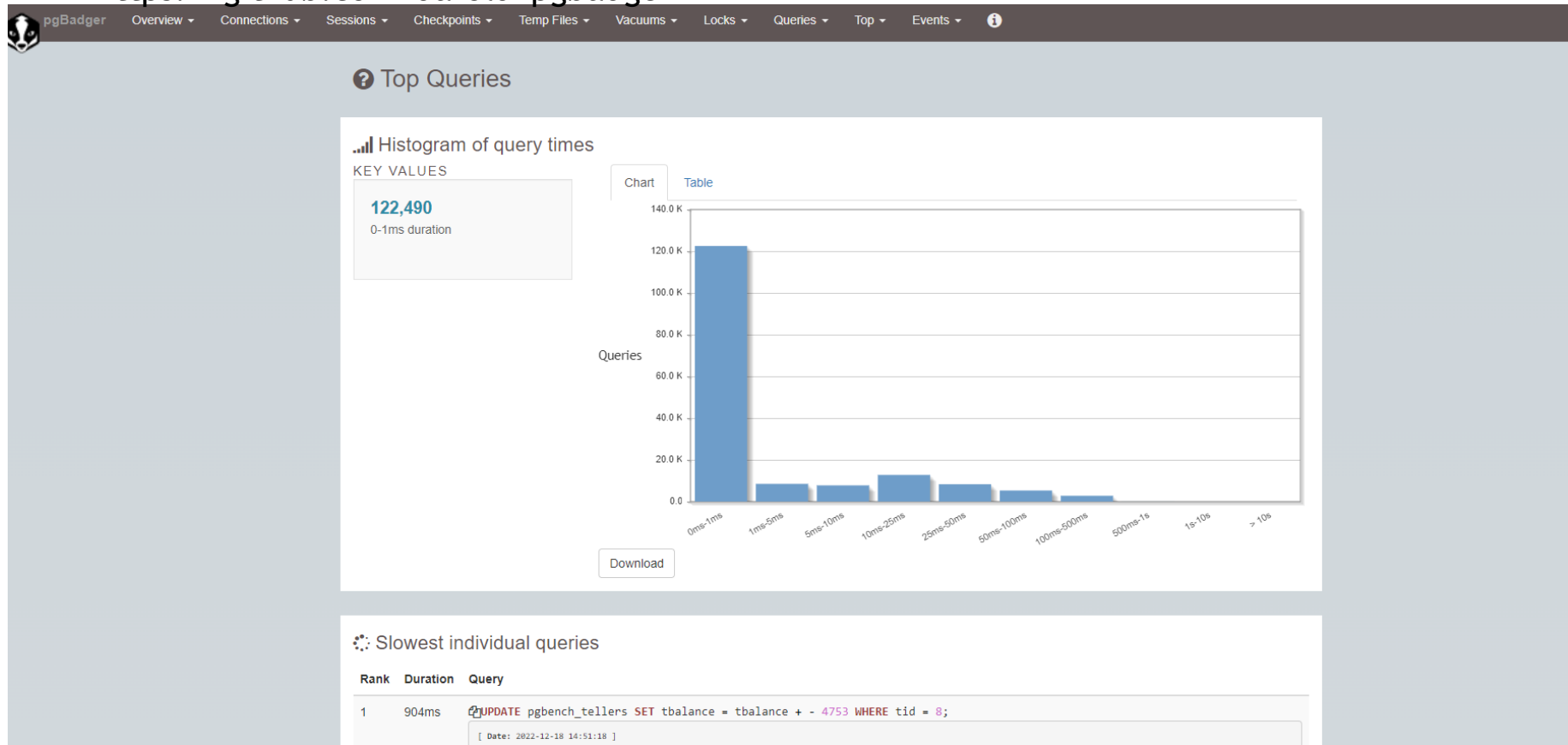
Сверху того запроса будут таблицы в которых больше всего операций последовательного чтения. Они и являются подозрительными для анализа причин отсутствия индексов.

Можно ещё посмотреть кэширование этих таблиц по представлению pg_statio_user_tables:
колонки heap_blks...и idx_blks...

	column_name name	data_type character varying
1	relid	oid
2	schemaname	name
3	relname	name
4	seq_scan	bigint
5	seq_tup_read	bigint
6	idx_scan	bigint
7	idx_tup_fetch	bigint
8	n_tup_ins	bigint
9	n_tup_upd	bigint
10	n_tup_del	bigint
11	n_tup_hot_upd	bigint
12	n_live_tup	bigint
13	n_dead_tup	bigint
14	n_mod_since_analy...	bigint
15	last_vacuum	timestamp with time z...
16	last_autovacuum	timestamp with time z...
17	last_analyze	timestamp with time z...
18	last_autoanalyze	timestamp with time z...
19	vacuum_count	bigint
20	autovacuum_count	bigint
21	analyze_count	bigint
22	autoanalyze_count	bigint

Pgbadger

<https://github.com/darold/pgbadger>



Практика



Тюнинг настроек

2

Тюнинг настроек

1. Настройки памяти
 2. Настройки дискового пространства
 3. Настройки оптимизатора
-
-
-

Настройки памяти Postgres

С чего начать
настройки памяти
для Postgres?



Настройки памяти Postgres

Настройки памяти в зависимости от памяти сервера должны быть примерно такими:

`effective_cache_size = 2/3 RAM`
`shared_buffers = RAM/4`
`temp_buffers = 256MB` `work_mem`
`= RAM/32`
`maintenance_work_mem = RAM/16`

Но проще воспользоваться конфигураторами, которые посчитают и порекомендуют ещё ряд полезных настроек:

<http://pgconfigurator.cybertec.at/> - продвинутый конфигуратор от Cybertec (которые умные книжки пишут)

<https://pgtune.leopard.in.ua/> - онлайн версия классического конфигуратора pgtune

Настройки дисковой подсистемы

fsync – данные журнала принудительно сбрасываются на диск с кэша ОС.

synchronous_commit – транзакция завершается только когда данные фактически сброшены на диск

checkpoint_completion_target – чем ближе к единице тем менее резкими будут скачки I/O при операциях checkpoint

effective_io_concurrency – число параллельных операций ввода/вывода (по количеству дисков, для SSD – несколько сотен)

random_page_cost – отношение рандомного чтения к последовательному.

Настройки оптимизатора

join_collapse_limit – сколько перестановок имеет смысл делать для поиска оптимального плана запроса

default_statistics_target – число записей просматриваемых при сборе статистики по таблицам. Чем больше тем тяжелее собрать статистику.

track_activity_query_size – по умолчанию 1024, в современных системах этого в большинстве случаев недостаточно.

`enable_bitmapscan = on`

`enable_hashjoin = on`

`enable_indexscan = on`

`enable_indexonlyscan = on`

`enable_mergejoin = on`

`enable_nestloop = on`

`enable_seqscan = on`

`enable_sort = on`

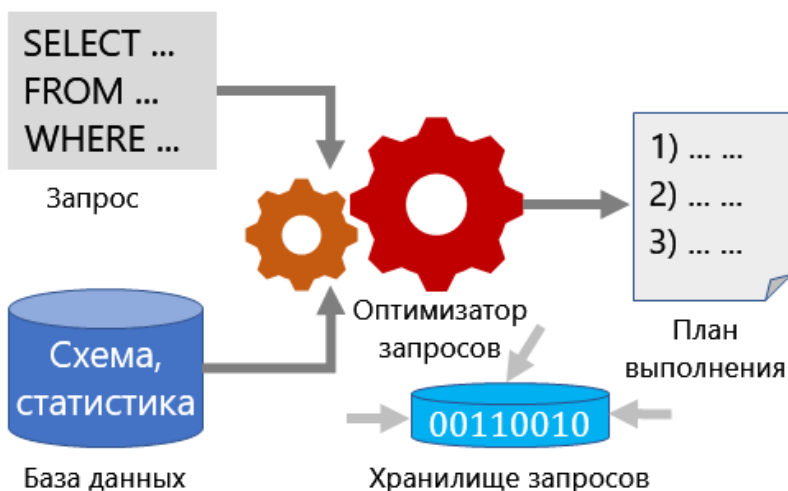
Оптимизация запросов

Планы запросов

**У кого есть все нужные контуры
окружения: DEV/TEST/STAGE/PROD?**

**И на каком контуре нужно
отлаживать запросы?**

Планы запросов



ОСНОВНЫЕ ОПЕРАТОРЫ ПЛАНА PG SQL

Оператор	Пояснение
Seq Scan	Последовательный перебор строк таблицы (возможно с отбором по условию)
Index Only Scan	Поиск по покрывающему индексу без захода в основную таблицу
Index Scan	Поиск по индексу, с заходом в основную таблицу за доп. колонками
Nested Loops	Соединение вложенными циклами
Hash Join	Соединение с помощью хеш-таблицы (Соответствия)
Merge Join	Соединение заранее отсортированных наборов с помощью алгоритма слияния
Sort	Сортировка УПОРЯДОЧИТЬ ПО
Прочие	Агрегаты СГРУППИРОВАТЬ ПО, ПЕРВЫЕ и пр.

Планы запросов

Самые частые ошибки оптимизатора:

- 1) Разница в rows – в оценочном и действительным планом – если значения отличаются в разы – это «звоночек».
- 2) Nested Loops с большим cost или большим rows – вложенными циклами должны соединяться только маленькие (до тысяч строк) таблицы.
- 3) Seq Scan по таблице где rows более нескольких тысяч, при этом есть FILTER – нужно посмотреть на поля в FILTER и найти подходящий индекс. Если не нашли – бинго! Одна из проблем решена.

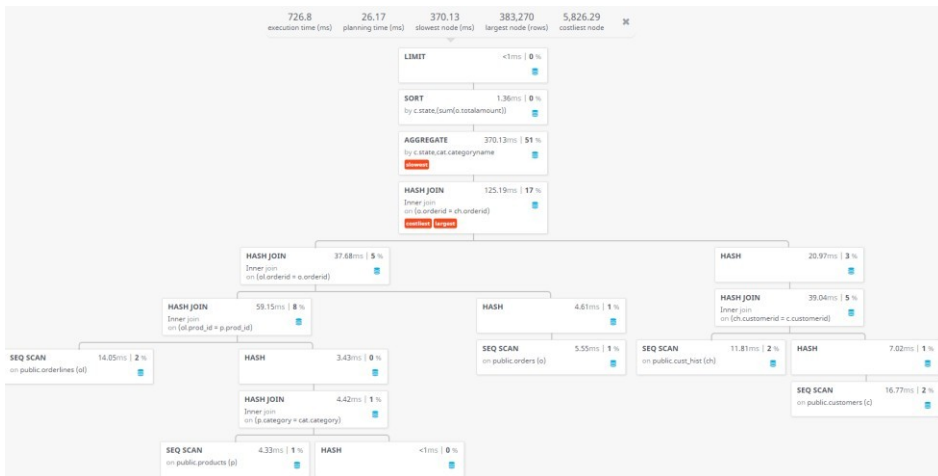
Логика чтения плана запроса:

- 1) Смотрим на самый большой cost оператора
- 2) Это Seq Scan или nested loops?
- 3) Смотрим следующий по стоимости оператор

Оптимизация чаще всего заканчивается либо добавлением индекса, либо упрощением запроса (разбиением на части, использованием вложенных таблиц и т.п.), либо обновлением статистики.

Средства визуализации планов запросов

<https://tatiyants.com/pev>



<https://explain.depesz.com/>

HTML	SOURCE	TEXT	STATS	
#	exclusive	inclusive	rows x	rows loops node
1.	0.003	725.775	↑ 1.0	10 1 → Limit (cost=17,024.84..17,024.87 rows=10 width=133) (actual time=725.773..725.775 rows=10 loops=1) Buffers: shared hit=23 read=1392
2.	1.355	725.772	↑ 74.2	11 1 → Sort (cost=17,024.84..17,026.88 rows=616 width=133) (actual time=725.771..725.772 rows=11 loops=1) Sort Key: c.state, (sum(o.totalamount)) Sort Method: top-N heapsort Memory: 25kB Buffers: shared hit=23 read=1392
3.	370.132	724.417	↓ 1.0	832 1 → HashAggregate (cost=16,994.41..17,006.65 rows=616 width=133) (actual time=723.877..724.417 rows=832 loops=1) Group Key: c.state, cat.categoryname Buffers: shared hit=13 read=1392
4.	125.191	354.265	↓ 1.2	383,270 1 → HashJoin (cost=4,966.48..13,742.65 rows=325,176 width=133) (actual time=118.314..354.265 rows=383,270 loops=1) Buffers: shared hit=13 read=1392
5.	37.678	133.484	↑ 1.0	60,350 1 → HashJoin (cost=634.86..4,539.11 rows=60,350 width=138) (actual time=22.651..133.484 rows=60,350 loops=1) Buffers: shared hit=9 read=581
6.	59.147	85.647	↑ 1.0	60,350 1 → HashJoin (cost=464.86..2,962.11 rows=60,350 width=122) (actual time=12.467..85.647 rows=60,350 loops=1) Buffers: shared hit=4 read=483
7.	14.054	14.054	↑ 1.0	60,350 1 → Seq Scan on orderlines ol (cost=0..988.5 rows=60,350 width=8) (actual time=0.005..14.054 rows=60,350 loops=1) Buffers: shared hit=2 read=383
8.	3.431	12.446	↑ 1.0	10,000 1 → Hash (cost=339.86..339.86 rows=10,000 width=122) (actual time=12.446..12.446 rows=10,000 loops=1) Buffers: shared hit=2 read=100
9.	4.420	9.015	↑ 1.0	10,000 1 → HashJoin (cost=1.36..339.86 rows=10,000 width=122) (actual time=0.283..9.015 rows=10,000 loops=1) Buffers: shared hit=2 read=100

<https://explain.tensor.ru/>

Явно повлиять на план запроса

`enable_nestloop = off` – большие таблицы нельзя соединять вложенными циклами

`enable_seqscan = off` – большие таблицы нельзя последовательно сканировать

В PostgreSQL нет HINT-ов, но если в это время «встал прод», то данные параметры могут на какое то время помочь – снизив общую производительность системы, но при этом исправив «больные» запросы.

В коммерческих версиях (например, Postgres Pro Enterprise 14) появился `pg_hint_plan`, который позволяет управлять планом выполнения запроса.

Запросы LIKE

Каждый когда-нибудь писал запрос вида:

```
SELECT * FROM Products WHERE name LIKE '%spoon%'
```

Ну или за вас это делала ORM ;)

Главная

Печатные формы

История

Система

Связанные документы

?






Наименование

Кол-во

Остаток

Зебра полосатая

Справочник

	238415 – 238415 Зебра полосатая	26
	238415 – 238415 Зебра полосатая (Большой, Белый)	0
	238415 – 238415 Зебра полосатая (Большой, Черный)	0
	238415 – 238415 Зебра полосатая (Малый, Белый)	0
	238415 – 238415 Зебра полосатая (Малый, Черный)	0
Создать новый товар «Зебра полосатая»		

Обычные индексы для подобных запросов крайне малоэффективны.

Что поделать если вы не можете позволить себе Elasticsearch :

Gin и GiST индексы – для полнотекстового поиска.

Create index on table using
`gin(column)`

Рекомендации по оптимизации запросов

- делать меньше запросов
- читать меньше данных
- обновлять меньше данных (несколько update insert в одну транзакцию)
- проанализировать передачу данных:
 - сколько данных было просканировано - сколько отослано
 - сколько было отослано - сколько использовано приложением
- не использовать UNION, когда можно UNION ALL
- использовать в выборке только нужные столбцы select *
- избегать distinct
- при использовании сложных индексов учитывать наличие в запросе первого столбца индекса
- избегать декартова произведения в джойнах
- используйте читабельный синтаксис SQL
- не используем русский язык в именах полей
- **пишите комментарии**

4 Расширения

PgMemcache - <https://github.com/ohmu/pgmemcache>

inMemory таблицы

Если нужно кэширование внутри СУБД, или временную таблицу в памяти, или нужно оперативно заменить таблицу на inmemory KV хранилище.

```
CREATE EXTENSION pgmemcache; shared_preload_libraries = 'pgmemcache'  
memcache_server_add('hostname:port':TEXT)
```

```
memcache_add(key::TEXT, value::TEXT) newval = memcache_decr(key::TEXT)  
memcache_delete(key::TEXT)
```



cstore_fdw - https://github.com/citusdata/cstore_fdw

Колоночные хранилища

Достоинства:

Выборки из огромных таблиц
Нет необходимости в нормализации
Сжатие данных

Недостатки:

Нет Update и Delete
Insert только группами «insert into
table select * from source»

```
shared_preload_libraries = 'cstore_fdw' CREATE EXTENSION  
cstore_fdw;
```

```
CREATE SERVER cstore_server FOREIGN DATA WRAPPER cstore_fdw;  
CREATE FOREIGN TABLE table( ) SERVER cstore_server  
OPTIONS(compression 'pglz')
```

```
INSERT INTO table SELECT * FROM sourcetable
```

TimescaleDB - <https://github.com/timescale/timescaledb>

Поддержка временных рядов

Создаёт таблицу секционированную по времени.

Применяется если нужно очень много писать данных во времени: системы мониторинга, биржевые системы...

```
CREATE EXTENSION timescaledb; shared_preload_libraries = 'timescaledb'
```

```
VicCREATE TABLE table(time TIMESTAMPTZ, value TEXT);  
SELECT create_hypertable('table', 'time');
```

Рефлексия

Рефлексия



Какие варианты оптимизации запомнили ?

Рефлексия



Какие варианты оптимизации запомнили ?

А какие будете применять ???

Заполните, пожалуйста, опрос о
занятии по ссылке в чате:
<https://otus.ru/polls/57771/>

Спасибо за внимание!

Приходите на следующие вебинары

Курочкин Константин
Ведущий администратор БД
«Medindex»