Авторские права
© Postgres Professional, 2023 год.
Авторы: Алексей Береснев, Илья Баштанов, Павел Толмачев

Использование материалов курса
Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Отказ от ответственности
Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Темы

Обзор pgpro_pwr

Архитектура pgpro_pwr

Установка pgpro_pwr

Выборки

Политика хранения

Выборочные линии

Размеры отношений

Отчеты

Обзор pgpro_pwr

Расширение для выявления ресурсоемких операций в БД

- поиск запросов, потребляющих много ресурсов
- определение характера и интенсивности нагрузки
- анализ исторической информации о нагрузке
- сравнение профилей нагрузки
- помощь в расследовании инцидентов

Расширение pgpro_pwr предназначено для выявления наиболее ресурсоемких операций в базе данных. PWR произносится как «пауэр» (power) — это сокращение от Postgres Pro Workload Reporting.

Расширение основано на pg_profile — расширении с открытым исходным кодом для PostgreSQL:

https://github.com/zubkov-andrei/pg_profile

Предназначение:

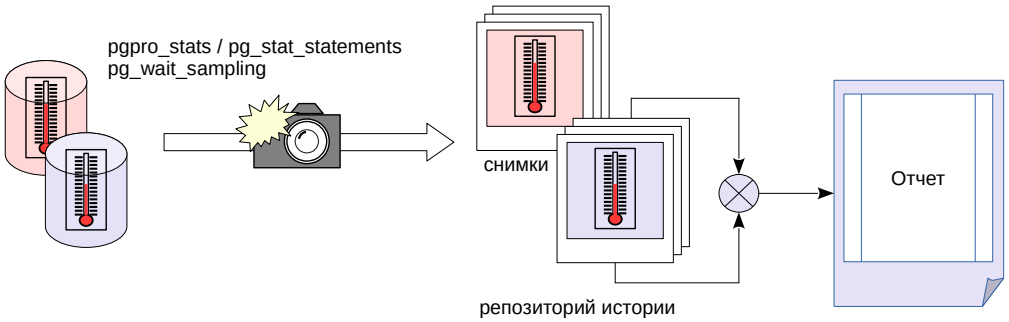
- поиск запросов, потребляющих больше всего ресурсов СУБД;
- получение информации о типе и интенсивности нагрузки;
- исследование нагрузки на «горячие» объекты;
- сохранение и анализ нагрузки в историческом разрезе;
- сравнение профилей нагрузки, имевших место в разное время;
- облегчение расследования инцидентов;
- проведение анализа нагрузочного тестирования.

Основная идея расширения — сохранение снимков собранных данных для последующего анализа.

Документация:

https://postgrespro.ru/docs/enterprise/13/pgpro-pwr

Архитектура pgpro_pwr



Статистика собирается с целевых серверов расширениями pgpro_stats или pg_stat_statements

Выборки-снимки помещаются в репозиторий истории

По выборкам создаются отчеты об изменениях

Исходные данные собираются с целевых серверов с помощью представлений сборщика статистики и расширения pgpro_stats или pg_stat_statements. Рекомендуется использовать расширение pgpro_stats, так как оно выдает планы операторов, информацию о событиях ожидания и нагрузке по отдельным базам данных, ролям, клиентам и приложениям. Сводная статистика ожиданий поступает от расширения pg_wait_sampling. Если нет возможности использовать pgpro_stats, можно использовать pg_stat_kcache как источник информации об использовании ресурсов процессора и о нагрузке на файловую систему (rusage).

Работа модуля pgpro_pwr заключается в обработке накапливаемых статистических выборок. Выборка содержит статистику о наиболее активных объектах и запросах, собранную с момента получения предыдущей выборки. По выборкам строят отчеты.

Расширение pgpro_pwr состоит из следующих частей:

- репозиторий истории — хранилище собранных данных;
- механизм управления выборками — предоставляет функции для получения выборок и поддержки репозитория в актуальном состоянии посредством удаления устаревших данных;
- механизм отчетов — представляет функции построения отчетов по данным из репозитория;
- административные функции для создания серверов, линий выборок и для управления ими.

Установка pgpro_pwr

Пакет pgpro_pwr устанавливается средствами ОС

Упрощенная установка

pgpro_pwr непосредственно на целевом сервере,
отслеживание нагрузки от имени суперпользователя

Развернутая установка

нагрузка отслеживается на одном или нескольких удаленных серверах
от имени специально зарегистрированного пользователя

Предполагается, что уже установлено расширение pgpro_stats или pg_stat_statements и настроен сборщик статистики Postgres Pro.

Само расширение pgpro_pwr собрано в отдельный установочный пакет и по умолчанию не устанавливается.

Упрощенная установка предназначена для случаев, когда pgpro_pwr устанавливается непосредственно на целевой сервер и будет отслеживать его нагрузку от имени суперпользователя. Она сводится к созданию отдельной схемы для объектов расширения и установке расширения. Так как pgpro_pwr зависит от расширения dblink, то либо в команду CREATE EXTENSION добавляют ключевое слово CASCADE, либо устанавливают dblink заранее.

Развернутая установка должна выполняться, когда планируется использовать pgpro_pwr для отслеживания нагрузки на одном или нескольких серверах с соблюдением принципа наименьших привилегий.

Описание вариантов установки в документации: https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-INSTALLATION

Установка расширения pgpro_pwr

Создадим тестовую базу данных на основе demo:

```
student$ psql
```

```
=> CREATE DATABASE pwr TEMPLATE demo;
```

```
CREATE DATABASE
```

Для удобства настроим путь поиска:

```
=> ALTER DATABASE pwr SET search_path TO bookings, public;
```

```
ALTER DATABASE
```

Пакет расширения pgpro_pwr уже установлен в виртуальной машине.

Добавим загрузку библиотек вспомогательных расширений и зададим параметры сбора статистики.

```
=> ALTER SYSTEM SET shared_preload_libraries = pg_wait_sampling, pgpro_stats;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET track_io_timing = 'on';
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET track_functions = 'all';
```

```
ALTER SYSTEM
```

Перезагрузим сервер.

```
student$ sudo systemctl restart postgrespro-ent-13.service
```

```
student$ psql -p 5432 -d pwr
```

Создадим схему для объектов расширения и добавим расширение в базу данных.

```
=> CREATE SCHEMA profile;
```

```
CREATE SCHEMA
```

```
=> CREATE EXTENSION pgpro_stats;
```

```
CREATE EXTENSION
```

```
=> CREATE EXTENSION pgpro_pwr SCHEMA profile CASCADE;
```

```
NOTICE:  installing required extension "dblink"
CREATE EXTENSION
```

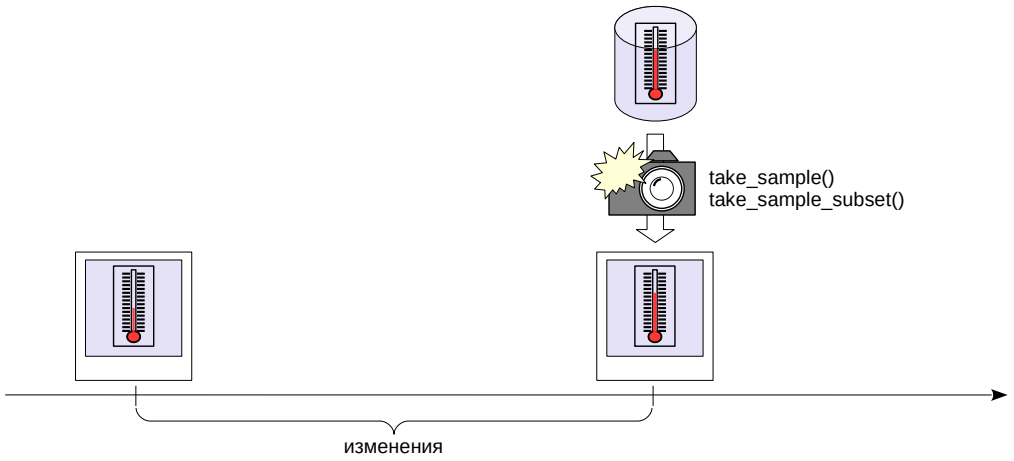
```
=> CREATE EXTENSION pg_wait_sampling;
```

```
CREATE EXTENSION
```

```
=> \dx
```

```
                                    List of installed extensions
       Name       | Version |   Schema   |                          Description
------------------+---------+------------+---------------------------------------------------------------
 dblink           | 1.2     | profile    | connect to other PostgreSQL databases from within a database
 pg_wait_sampling | 1.1     | bookings   | sampling based statistics of wait events
 pgpro_pwr        | 4.3     | profile    | PostgreSQL load profile repository and report builder
 pgpro_stats      | 1.6     | bookings   | track execution statistics of all SQL statements executed
 plpgsql          | 1.0     | pg_catalog | PL/pgSQL procedural language
(5 rows)
```

Выборки



Данные собираются с зарегистрированных серверов

Выборка содержит данные с момента предыдущей выборки

Расширение pgpro_pwr собирает данные с зарегистрированных серверов. По умолчанию создается активное определение сервера с именем local, соответствующее текущему кластеру.

Данные с активных серверов собираются функцией take_sample; неактивный сервер можно указывать явно. Если имеется несколько серверов, можно получить выборки с их подмножества функцией take_sample_subset.

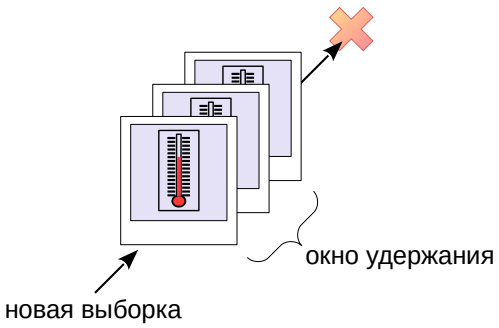
Выборка содержит информацию об изменениях в измеряемых параметрах СУБД, происшедших с момента предыдущей выборки. Аккумулируемые регулярные выборки позволяют строить отчеты о нагрузке базы данных за предыдущее время. Обычно достаточно получать одну-две выборки в час.

При формировании выборки предыдущая статистика сбрасывается функцией pgpro_stats_statements_reset или pg_stat_statements_reset в зависимости от того, какое расширение поставляет данные.

https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-MANAGING-SERVERS

https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-SAMPLES

Политика хранения



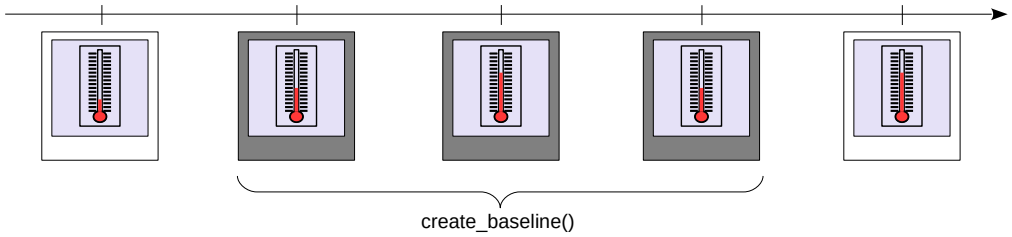
Настройка

pgpro_pwr.max_sample_age = 7	срок хранения в днях, глобально
set_server_max_sample_age()	для конкретного сервера

Для выборок можно определять политики хранения:

- глобально с помощью параметра конфигурации *pgpro_pwr.max_sample_age*;
- для конкретного сервера с помощью вызова функции set_server_max_sample_age.

Выборочные линии



Именованная последовательность выборок

- можно использовать для построения отчета
- можно задать срок хранения (имеет приоритет над политикой)

Для построения отчета по нагрузке требуется две выборки; они задают период, за который были собраны данные для отчета.

Другой способ задания периода для отчета — использование выборочной линии. Выборочные линии — удобный инструмент для работы с выборками. При создании выборочной линии задают ее имя и либо указывают начальную и конечную выборки, либо задают их с помощью диапазона времени.

Период хранения выборки задается в днях и определяется либо глобально, либо на уровне сервера. Если необходимо переопределить политику хранения для каких-либо выборок, следует использовать выборочные линии.

Для каждой выборочной линии можно задавать время хранения в днях. Если для линии время хранения не задано, то относящиеся к ней выборки вообще не будут удаляться.

Для управления выборочными линиями предназначены функции:

- create_baseline — создает выборочную линию;
- drop_baseline — удаляет выборочную линию;
- keep_baseline — задает срок хранения;
- show_baselines — выдает информацию об имеющихся выборочных линиях.

https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-BASELINES

Получение выборки и создание выборочной линии

Откроем второй сеанс.

```
student$ psql -p 5432 -d pwr
```

Получим первую выборку. Предыдущая статистика сбрасывается.

```
=> SELECT * FROM profile.take_sample();
```

```
 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.23
(1 row)
```

Используем пример соединения хешированием из учебного курса QPT.

Во втором сеансе выполним транзакцию, выделив недостаточный объем оперативной памяти; двухпроходное хеш-соединение будет использовать временные файлы:

```
=> BEGIN;
```

```
BEGIN
```

```
=> SET LOCAL work_mem = '12MB';
```

```
SET
```

```
=> SET LOCAL hash_mem_multiplier = 1;
```

```
SET
```

```
=> EXPLAIN (analyze, buffers, costs off, timing off, summary off)
SELECT * FROM bookings b
  JOIN tickets t ON b.book_ref = t.book_ref;
```

```
                           QUERY PLAN
-----------------------------------------------------------------
 Hash Join (actual rows=366733 loops=1)
   Hash Cond: (t.book_ref = b.book_ref)
   Buffers: shared hit=3 read=7844, temp read=3491 written=3491
   I/O Timings: read=31.161
   ->  Seq Scan on tickets t (actual rows=366733 loops=1)
         Buffers: shared read=6159
         I/O Timings: read=24.691
   ->  Hash (actual rows=262788 loops=1)
         Buckets: 262144  Batches: 2  Memory Usage: 9071kB
         Buffers: shared read=1685, temp written=690
         I/O Timings: read=6.470
         ->  Seq Scan on bookings b (actual rows=262788 loops=1)
               Buffers: shared read=1685
               I/O Timings: read=6.470
 Planning:
   Buffers: shared hit=186 read=11 dirtied=1
   I/O Timings: read=0.069
(17 rows)
```

```
=> COMMIT;
```

```
COMMIT
```

Получим вторую выборку — в нее попадут данные о нагрузке после первой выборки:

```
=> SELECT * FROM profile.take_sample();
```

```
 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.25
(1 row)
```

Теперь во втором сеансе выполним транзакцию, выделив достаточно рабочей памяти для хеш-соединения:

```
=> BEGIN;
```

```
BEGIN
```

```
=> SET LOCAL work_mem = '48MB';
```

```
SET
```

```
=> SET LOCAL hash_mem_multiplier = 3;
```

```
SET
```

```
=> EXPLAIN (analyze, buffers, costs off, timing off, summary off)
SELECT * FROM bookings b
  JOIN tickets t ON b.book_ref = t.book_ref;
```

```
                           QUERY PLAN
-----------------------------------------------------------------
 Hash Join (actual rows=366733 loops=1)
   Hash Cond: (t.book_ref = b.book_ref)
   Buffers: shared hit=7844
   ->  Seq Scan on tickets t (actual rows=366733 loops=1)
         Buffers: shared hit=6159
   ->  Hash (actual rows=262788 loops=1)
         Buckets: 524288  Batches: 1  Memory Usage: 18190kB
         Buffers: shared hit=1685
         ->  Seq Scan on bookings b (actual rows=262788 loops=1)
               Buffers: shared hit=1685
 Planning:
   Buffers: shared hit=8
(12 rows)
```

```
=> COMMIT;
```

```
COMMIT
```

А сейчас получим третью выборку — в нее попадут данные о нагрузке после второй выборки:

```
=> SELECT * FROM profile.take_sample();
```

```
 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.22
(1 row)
```

Список выборок:

```
=> SELECT * FROM profile.show_samples();
```

```
 sample |      sample_time       | sizes_collected | dbstats_reset | clustats_reset | archstats_reset
--------+------------------------+-----------------+---------------+----------------+-----------------
      1 | 2024-01-16 21:34:40+03 | t               |               |                |
      2 | 2024-01-16 21:34:41+03 | t               |               |                |
      3 | 2024-01-16 21:34:41+03 | t               |               |                |
(3 rows)
```

Сформируем из имеющихся выборок две выборочные линии.

```
=> SELECT profile.create_baseline(baseline=>'BaseLine1', start_id=>1, end_id=>2);
```

```
 create_baseline
-----------------
               1
(1 row)
```

```
=> SELECT profile.create_baseline(baseline=>'BaseLine2', start_id=>2, end_id=>3);
```

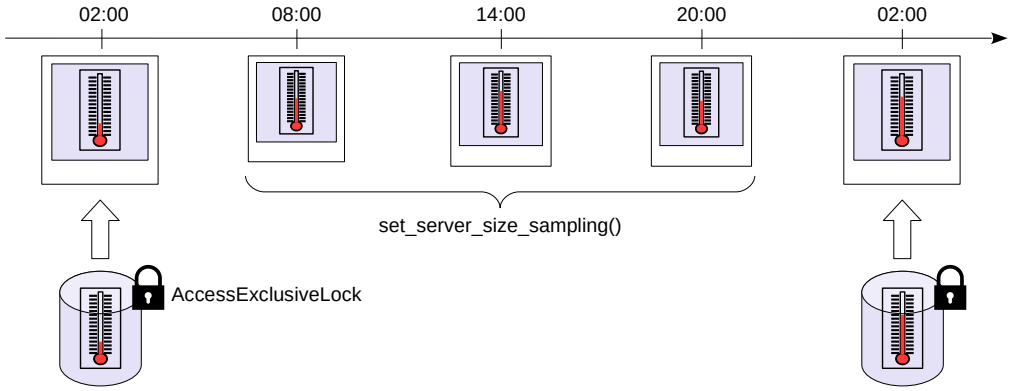
```
 create_baseline
-----------------
               2
(1 row)
```

Список выборочных линий:

```
=> SELECT * FROM profile.show_baselines();
```

```
 baseline  | min_sample | max_sample | keep_until_time
-----------+------------+------------+-----------------
 BaseLine1 |          1 |          2 |
 BaseLine2 |          2 |          3 |
(2 rows)
```

Размеры отношений



Политика анализа размеров

разрешенное окно в течение суток для сбора данных о размерах

минимальный промежуток между выборками, содержащими размеры

11

Сбор сведений о размерах всех отношений в базе данных занимает продолжительное время, при этом на каждое анализируемое отношение устанавливается исключительная блокировка AccessExclusiveLock.

Однако чаще всего достаточно собирать информацию о размерах один раз в сутки, для этого можно определить политику анализа размеров для серверов. Политика устанавливает:

- окно в течение суток, когда разрешается сбор сведений о размерах отношений;
- минимальный промежуток между двумя выборками со сведениями о размерах.

Политику можно задать с помощью функции set_server_size_sampling.

Просмотреть политики для серверов можно, вызвав функцию show_servers_size_sampling.

https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-RELATION-SIZES-COLLECTION

Отчеты

Отчеты pgpro_pwr формируются в формате HTML

get_report() — обычные отчеты, предоставляющие статистику по нагрузке для заданного интервала

get_diffreport() — разностные отчеты со статистикой по одинаковым объектам в двух сравниваемых интервалах

Выбранный элемент отчета автоматически выделяется в других разделах отчета

Настройка

pgpro_pwr.max = 7 — число первых по некоторому критерию объектов, выбираемых для каждой таблицы отчета

Функции построения отчетов позволяют определить интервалы разными способами, принимая идентификаторы выборок, выборочные линии или диапазоны времени. В последнем случае выбирается минимальный интервал, охватывающий указанный диапазон.

Почти любой элемент в отчете можно выделить одним щелчком мыши. Выбранный элемент сразу будет выделен во всех разделах отчета, что облегчит его поиск.

Разделы отчета подробно описаны в документации: https://postgrespro.ru/docs/enterprise/13/pgpro-pwr#PGPRO-PWR-REPORT-GENERATION

Отчеты

Получим отчет. Для построения обычного отчета достаточно указать две выборки или одну выборочную линию.

В плане запроса, выполненного в условиях недостатка рабочей памяти, видно, что используются временные файлы для двухпроходного хеш-соединения.

Раздел отчета "Top SQL by temp usage" должен содержать сведения об этом запросе.

```
student$ psql -d pwr -Aqtc 'SELECT profile.get_report(1,2)' -o /tmp/report_1_2.html
```

Открываем файл /tmp/report_1_2.html...

```
student$ xdg-open /tmp/report_1_2.html
```

Получим разностный отчет. Используем две выборочные линии.

В разностном отчете должно быть заметно, что во втором запросе рабочей памяти хватило для выполнения однопроходного хеш-соединения и временные файлы не использовались.

В отчете просмотрите разделы:

- Top SQL by temp usage;
- Load distribution among heavily loaded databases;
- Load distribution.

```
student$ psql -d pwr -Aqtc "SELECT profile.get_diffreport('BaseLine1','BaseLine2')" -o /tmp/report_diff.html
```

Открываем файл /tmp/report_diff.html...

```
student$ xdg-open /tmp/report_diff.html
```

Итоги

Расширение pgpro_pwr предназначено для выявления ресурсоемких операций в БД

Статистика собирается расширениями pgpro_stats или pg_stat_statements

Поддерживается политика хранения выборок

Можно получать обычные и разностные отчеты в HTML

Практика

1. Подготовьте СУБД для работы с расширением pgpro_pwr.
2. Создайте базу данных и установите расширение pgpro_pwr.
3. Проведите нагрузочное тестирование с помощью pgbench, получив две выборки — до и после тестирования.
4. Создайте новое табличное пространство и сделайте выборку.
5. Снова проведите тестирование с помощью pgbench так, чтобы индексы были в новом табличном пространстве, после чего сделайте еще одну выборку.
6. Получите разностный отчет для первого и второго тестов.

1. Подключите разделяемые библиотеки pg_wait_sampling и pgpro_stats.
2. В базе данных создайте схему для объектов расширения pgpro_pwr.
3. Выполните обычную инициализацию таблиц pgbench и обычный тест нагрузки, например, на 60 секунд.
4. Предварительно создайте каталог для табличного пространства, например, в домашнем каталоге пользователя postgres.
5. Заново проведите инициализацию таблиц pgbench, указав для табличного пространства ключ --index-tablespace. Сделайте новую выборку.
6. Получите разностный отчет с двумя промежутками.

1. Подготовка СУБД для работы с расширением pgpro_pwr

```
student$ psql
```

Подключим разделяемые библиотеки.

Расширение pgpro_pwr получает сводную статистику ожиданий от расширения pg_wait_sampling:

```
=> ALTER SYSTEM SET shared_preload_libraries = pg_wait_sampling, pgpro_stats;
```

```
ALTER SYSTEM
```

Параметр track_io_timing нужен для анализа времени, затраченного операторами на чтение или запись блоков:

```
=> ALTER SYSTEM SET track_io_timing = 'on';
```

```
ALTER SYSTEM
```

Параметр track_functions позволяет отслеживать вызовы пользовательских функций:

```
=> ALTER SYSTEM SET track_functions = 'all';
```

```
ALTER SYSTEM
```

Перезагрузим сервер.

```
student$ sudo systemctl restart postgrespro-ent-13.service
```

---

2. Создание базы данных и установка расширения pgpro_pwr

```
student$ psql
```

Создадим базу данных.

```
=> CREATE DATABASE pwr;
```

```
CREATE DATABASE
```

```
=> \c pwr
```

```
You are now connected to database "pwr" as user "student".
```

Создадим схему для хранения объектов расширения и установим расширение.

```
=> CREATE SCHEMA profile;
```

```
CREATE SCHEMA
```

```
=> CREATE EXTENSION pgpro_pwr SCHEMA profile CASCADE;
```

```
NOTICE:  installing required extension "dblink"
CREATE EXTENSION
```

```
=> CREATE EXTENSION pg_wait_sampling;
```

```
CREATE EXTENSION
```

```
=> CREATE EXTENSION pgpro_stats;
```

```
CREATE EXTENSION
```

```
=> \dx
```

```
                                    List of installed extensions
       Name       | Version |   Schema   |                          Description
------------------+---------+------------+---------------------------------------------------------------
 dblink           | 1.2     | profile    | connect to other PostgreSQL databases from within a database
 pg_wait_sampling | 1.1     | public     | sampling based statistics of wait events
 pgpro_pwr        | 4.3     | profile    | PostgreSQL load profile repository and report builder
 pgpro_stats      | 1.6     | public     | track execution statistics of all SQL statements executed
 plpgsql          | 1.0     | pg_catalog | PL/pgSQL procedural language
(5 rows)
```

---

3. Нагрузочное тестирование

Сделаем выборку:

```
=> SELECT * FROM profile.take_sample();
```

```
 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.23
(1 row)
```

Инициализируем нагрузочные таблицы pgbench:

```
student$ /opt/pgpro/ent-13/bin/pgbench -i pwr

dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.05 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.14 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.06 s, vacuum 0.03 s, primary keys 0.04 s).
```

Запустим pgbench на 30 секунд:

```
student$ /opt/pgpro/ent-13/bin/pgbench -T30 pwr

starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
default transaction isolation level: read committed
transaction maximum tries number: 1
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 12263
latency average = 2.446 ms
tps = 408.755024 (including connections establishing)
tps = 408.774829 (excluding connections establishing)
```

Получим вторую выборку, в которую попадут данные о нагрузке при инициализации и нагрузочном тесте:

```
=> SELECT * FROM profile.take_sample();

 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.24
(1 row)
```

---

4. Создание нового табличного пространства

Создадим табличное пространство.

```
postgres$ mkdir /var/lib/postgresql/ts_dir

=> CREATE TABLESPACE ts_idx LOCATION '/var/lib/postgresql/ts_dir';

CREATE TABLESPACE
```

Сделаем выборку:

```
=> SELECT * FROM profile.take_sample();

 server | result |  elapsed
--------+--------+------------
 local  | OK     | 00:00:00.2
(1 row)
```

---

5. Нагрузочное тестирование при размещении индексов в отдельном табличном пространстве

Инициализируем нагрузочные таблицы pgbench:

```
student$ /opt/pgpro/ent-13/bin/pgbench -i --index-tablespace=ts_idx pwr

dropping old tables...
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.05 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.16 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.06 s, vacuum 0.04 s, primary keys 0.04 s).
```

Запустим pgbench на 30 секунд:

```
student$ /opt/pgpro/ent-13/bin/pgbench -T30 pwr

starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
default transaction isolation level: read committed
transaction maximum tries number: 1
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
```

```
duration: 30 s
number of transactions actually processed: 11704
latency average = 2.563 ms
tps = 390.125358 (including connections establishing)
tps = 390.148505 (excluding connections establishing)
```

Получим еще одну выборку:

```
=> SELECT * FROM profile.take_sample();

 server | result |   elapsed
--------+--------+-------------
 local  | OK     | 00:00:00.24
(1 row)
```

Получим список выборок:

```
=> SELECT * FROM profile.show_samples();

 sample |      sample_time       | sizes_collected | dbstats_reset | clustats_reset | archstats_reset
--------+------------------------+-----------------+---------------+----------------+-----------------
      1 | 2024-01-16 21:44:59+03 | t               |               |                |
      2 | 2024-01-16 21:45:29+03 | t               |               |                |
      3 | 2024-01-16 21:45:30+03 | t               |               |                |
      4 | 2024-01-16 21:46:00+03 | t               |               |                |
(4 rows)
```

---

5. Получение разностного отчета

Получим разностный отчет.

```
student$ psql -d pwr -Aqtc "SELECT profile.get_diffreport(1,2,3,4)" -o /tmp/report_diff.html
```

Открываем файл /tmp/report_diff.html...

```
student$ xdg-open /tmp/report_diff.html
```