**PostgreSQL tablespace size**

SELECT pg\_size\_pretty ( pg\_tablespace\_size ('pg\_default') );

**Query to Find all active sessions and queries:**

SELECT

pid

,datname

,usename

,application\_name

,client\_hostname

,client\_port

,backend\_start

,query\_start

,query

,state

FROM pg\_stat\_activity

WHERE state = 'active';

**Queries to Find slow, long-running, and Blocked Queries**

SELECT

pid,

user,

pg\_stat\_activity.query\_start,

now() - pg\_stat\_activity.query\_start AS query\_time,

query,

state,

wait\_event\_type,

wait\_event

FROM pg\_stat\_activity

WHERE (now() - pg\_stat\_activity.query\_start) > interval '10 minutes';

**Query to find blocking session**

SELECT

activity.pid,

activity.usename,

activity.query,

blocking.pid AS blocking\_id,

blocking.query AS blocking\_query

FROM pg\_stat\_activity AS activity

JOIN pg\_stat\_activity AS blocking ON blocking.pid = ANY(pg\_blocking\_pids(activity.pid));

**Viewing locks with table names and queries**

select

relname as relation\_name,

query,

pg\_locks.\*

from pg\_locks

join pg\_class on pg\_locks.relation = pg\_class.oid

join pg\_stat\_activity on pg\_locks.pid = pg\_stat\_activity.pid;

**Killing/cancelling a long running Postgres query:**

You can find the pid using pg\_stat\_activity

Option #1 (graceful):

SELECT pg\_cancel\_backend(<PID>);

Option #2 (forceful):

SELECT pg\_terminate\_backend(<PID>);

**Terminate all queries**

If you want to terminate all running queries, the following statement can be executed:

SELECT pg\_cancel\_backend(pid) FROM pg\_stat\_activity WHERE state = 'active' and pid <> pg\_backend\_pid();

**Query to find high cpu usage and which query is causing it.**

select

(total\_time / 1000 / 3600) as total\_hours,

(total\_time / 1000) as total\_seconds,

(total\_time / calls) as avg\_millis,

calls num\_calls,

query

from pg\_stat\_statements

order by 1 desc limit 10;

You can also use pg\_activity\_utility to monitor the cpu usage.

**Monitoring CPU and memory usage from Postgres**

We'll create two database tables that will allow you to query CPU and memory usage from within the database connection. This way your applications can monitor the health of the servers without needing to worry about another connection or another protocol.

You can run these commands on the master database and they will propagate to all the slave databases as well.

First, load the file foreign data wrapper and create the foreign data server:

CREATE EXTENSION file\_fdw;

CREATE SERVER fileserver FOREIGN DATA WRAPPER file\_fdw;

Then we'll create the table that loads CPU loadavg from the /proc/loadavg file:

CREATE FOREIGN TABLE loadavg

(one text, five text, fifteen text, scheduled text, pid text)

SERVER fileserver

OPTIONS (filename '/proc/loadavg', format 'text', delimiter ' ');

Creating the table that will let you query memory info is similar:

CREATE FOREIGN TABLE meminfo

(stat text, value text)

SERVER fileserver

OPTIONS (filename '/proc/meminfo', format 'csv', delimiter ':');

Now you can run SELECT queries to see the info!

postgres=# SELECT \* FROM loadavg;

you can also query

SELECT \* FROM meminfo;

**All databases and their sizes**

select \* from pg\_user;

**All tables and their size, with/without indexes**

select datname, pg\_size\_pretty(pg\_database\_size(datname))

from pg\_database

order by pg\_database\_size(datname) desc;

**Cache hit rates (should not be less than 0.99)**

SELECT sum(heap\_blks\_read) as heap\_read, sum(heap\_blks\_hit) as heap\_hit, (sum(heap\_blks\_hit) - sum(heap\_blks\_read)) / sum(heap\_blks\_hit) as ratio

FROM pg\_statio\_user\_tables;

**Table index usage rates (should not be less than 0.99)**

SELECT relname, 100 \* idx\_scan / (seq\_scan + idx\_scan) percent\_of\_times\_index\_used, n\_live\_tup rows\_in\_table

FROM pg\_stat\_user\_tables

ORDER BY n\_live\_tup DESC;

**How many indexes are in cache**

SELECT sum(idx\_blks\_read) as idx\_read, sum(idx\_blks\_hit) as idx\_hit, (sum(idx\_blks\_hit) - sum(idx\_blks\_read)) / sum(idx\_blks\_hit) as ratio

FROM pg\_statio\_user\_indexes;

**Show unused indexes:**

SELECT relname AS table\_name, indexrelname AS index\_name, idx\_scan, idx\_tup\_read, idx\_tup\_fetch, pg\_size\_pretty(pg\_relation\_size(indexrelname::regclass))

FROM pg\_stat\_all\_indexes

WHERE schemaname = 'public'

AND idx\_scan = 0

AND idx\_tup\_read = 0

AND idx\_tup\_fetch = 0

ORDER BY pg\_relation\_size(indexrelname::regclass) DESC;

**Find cardinality of index:**

SELECT relname, relkind, reltuples as cardinality, relpages

FROM pg\_class

WHERE relname LIKE 'tableprefix%';

or

Find cardinality of index

SELECT schema\_name,

object\_name,

object\_type,

cardinality,

pages

FROM (

SELECT pg\_catalog.pg\_namespace.nspname AS schema\_name,

relname as object\_name,

relkind as object\_type,

reltuples as cardinality,

relpages as pages

FROM pg\_catalog.pg\_class

JOIN pg\_catalog.pg\_namespace ON relnamespace = pg\_catalog.pg\_namespace.oid

) t

WHERE schema\_name NOT LIKE 'pg\_%'

and schema\_name <> 'information\_schema'

--and schema\_name = '$schema\_name'

--and object\_name = '$object\_name'

ORDER BY pages DESC, schema\_name, object\_name

;

**Show Table Bloats**

with foo as (

SELECT

schemaname, tablename, hdr, ma, bs,

SUM((1-null\_frac)\*avg\_width) AS datawidth,

MAX(null\_frac) AS maxfracsum,

hdr+(

SELECT 1+COUNT(\*)/8

FROM pg\_stats s2

WHERE null\_frac<>0 AND s2.schemaname = s.schemaname AND s2.tablename = s.tablename

) AS nullhdr

FROM pg\_stats s, (

SELECT

(SELECT current\_setting('block\_size')::NUMERIC) AS bs,

CASE WHEN SUBSTRING(v,12,3) IN ('8.0','8.1','8.2') THEN 27 ELSE 23 END AS hdr,

CASE WHEN v ~ 'mingw32' THEN 8 ELSE 4 END AS ma

FROM (SELECT version() AS v) AS foo

) AS constants

GROUP BY 1,2,3,4,5

), rs as (

SELECT

ma,bs,schemaname,tablename,

(datawidth+(hdr+ma-(CASE WHEN hdr%ma=0 THEN ma ELSE hdr%ma END)))::NUMERIC AS datahdr,

(maxfracsum\*(nullhdr+ma-(CASE WHEN nullhdr%ma=0 THEN ma ELSE nullhdr%ma END))) AS nullhdr2

FROM foo

), sml as (

SELECT

schemaname, tablename, cc.reltuples, cc.relpages, bs,

CEIL((cc.reltuples\*((datahdr+ma-

(CASE WHEN datahdr%ma=0 THEN ma ELSE datahdr%ma END))+nullhdr2+4))/(bs-20::FLOAT)) AS otta,

COALESCE(c2.relname,'?') AS iname, COALESCE(c2.reltuples,0) AS ituples, COALESCE(c2.relpages,0) AS ipages,

COALESCE(CEIL((c2.reltuples\*(datahdr-12))/(bs-20::FLOAT)),0) AS iotta -- very rough approximation, assumes all cols

FROM rs

JOIN pg\_class cc ON cc.relname = rs.tablename

JOIN pg\_namespace nn ON cc.relnamespace = nn.oid AND nn.nspname = rs.schemaname AND nn.nspname <> 'information\_schema'

LEFT JOIN pg\_index i ON indrelid = cc.oid

LEFT JOIN pg\_class c2 ON c2.oid = i.indexrelid

)

SELECT

current\_database(), schemaname, tablename, /\*reltuples::bigint, relpages::bigint, otta,\*/

ROUND((CASE WHEN otta=0 THEN 0.0 ELSE sml.relpages::FLOAT/otta END)::NUMERIC,1) AS tbloat,

CASE WHEN relpages < otta THEN 0 ELSE bs\*(sml.relpages-otta)::BIGINT END AS wastedbytes,

iname, /\*ituples::bigint, ipages::bigint, iotta,\*/

ROUND((CASE WHEN iotta=0 OR ipages=0 THEN 0.0 ELSE ipages::FLOAT/iotta END)::NUMERIC,1) AS ibloat,

CASE WHEN ipages < iotta THEN 0 ELSE bs\*(ipages-iotta) END AS wastedibytes

FROM sml

ORDER BY wastedbytes DESC