POSTGRESQL QUERY PERFORMANCE INSIGHTS

Hamid Quddus Akhtar

Percona

Keeping Open Source Open

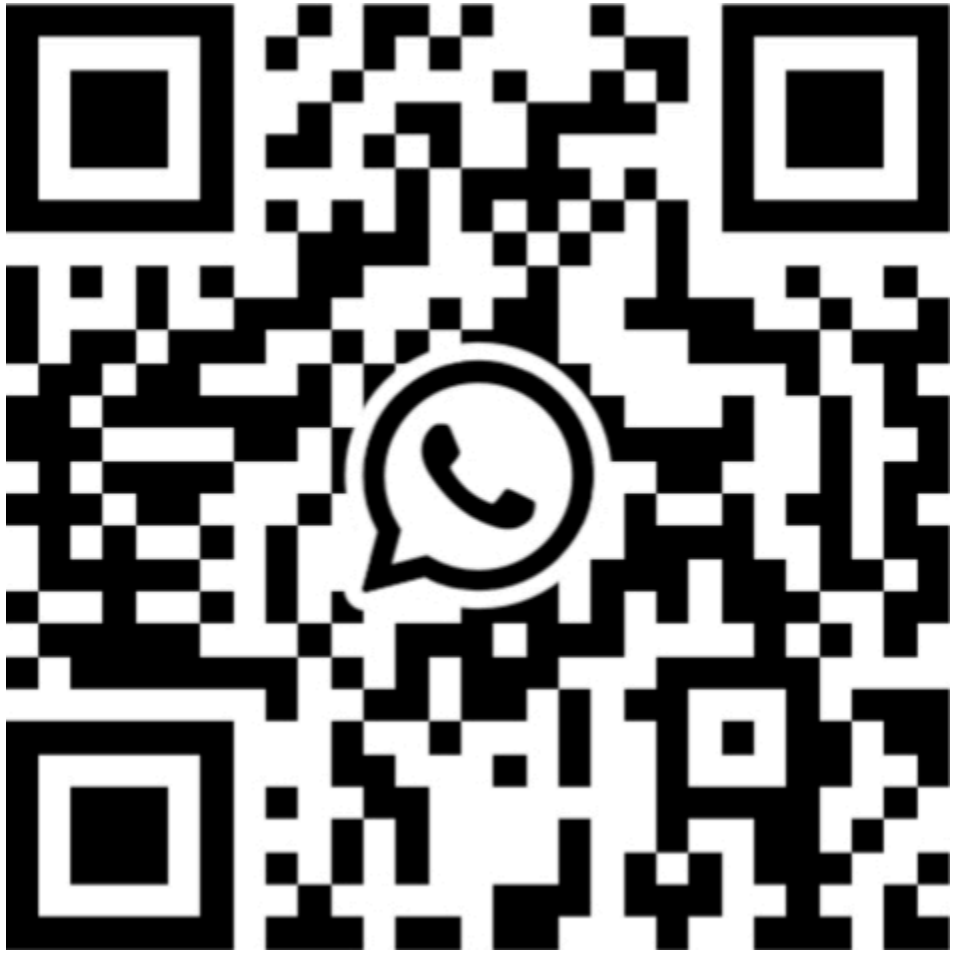
PERCONA

About Myself

- More than two decades of professional software development.
  - Actually developed a couple of games back in 1993!
- I'm part of Percona:
  - Previously as Technical Product Manager for PostgreSQL.
  - Now as the Senior Software Engineer
  - Percona has amazing culture.
- Prior to joining Percona, I had worked with EnterpriseDB where I lead the configuration management team that delivers the official PostgreSQL installers for Windows and MacOS.

Contacts

- Email:
  - hamid.akhtar@percona.com
- Skype:
  - EngineeredVirus
- WhatsApp



PERCONA

Presentation Outline

- Understanding query plan
- The tools for observing query performance
  - pg_stat_activity
  - pg_stat_statements
  - auto_explain
  - pgbadger
  - pg_stat_monitor

Initial Thoughts

Observing Query Performance - Overview

- Application/Connection Information
- Query Text
  - With parameter values
- Execution Plan
- Planning and Execution Timing Statistics
- Block Read/Write Statistics
- Wait Events and Locks

EXPLAIN and ANALYZE…

Components of Query Processing

- There are 5 components of query processing:
  - Parser

    Analyzer

    Rewriter
    - Parser and Analyzer ensure that the query is written correctly, and rewriter may perform some transformations.
  - Planner
    - The transformed query tree is passed on to the Planner which defines the execution steps for the executor.
  - Executor
    - Executes steps defined by the planner.
- To understand query performance, we must first understand the EXPLAIN command output; i.e. the query execution plan.

EXPLAIN - Basics

- Let’s have a look at the EXPLAIN command output.

```
regression=# EXPLAIN SELECT * FROM tenk1;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..445.00 rows=10000 width=244)
(1 row)
```

- Estimated…
  - Startup Cost
  - Total Cost
  - Number of Rows
  - Average Row Width in Bytes

EXPLAIN - Basics: Costs

- Cost is:
  - An arbitrary unit
  - Conventionally equivalent to “seq_page_cost” = 1.0
- Page Costs
  - Sequential vs Random page access
- CPU Costs
  - Cost for processing of a tuple, indexing entry, or operator
- Parallel Costs
  - Cost for setup, or tuple transfer to another parallel worker

EXPLAIN - Basics: Example

- A more complex query plan is show

```
regression=# EXPLAIN SELECT *
regression-# FROM tenk1 t1, tenk2 t2
regression-# WHERE t1.unique1 < 10 AND t2.unique2 < 10 AND t1.hundred < t2.hundred;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop  (cost=4.65..49.36 rows=33 width=488)
   Join Filter: (t1.hundred < t2.hundred)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=4.36..39.38 rows=10 width=244)
         Recheck Cond: (unique1 < 10)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.36 rows=10 width=0)
               Index Cond: (unique1 < 10)
   ->  Materialize  (cost=0.29..8.51 rows=10 width=244)
         ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.29..8.46 rows=10 width=244)
               Index Cond: (unique2 < 10)
(9 rows)
```

EXPLAIN - Basics: Nodes

- Query plan is a tree of plan nodes.
  - Nodes at the bottom level of the tree are scan nodes.
- Types of scan nodes for table access methods:
  - Sequential
  - Index
  - Bitmap Index
- For non-table row sources
  - Functions Returning Sets
    - EXPLAIN SELECT * FROM generate_series(1,10);
  - VALUES Clauses
    - EXPLAIN SELECT * FROM (VALUES (1), (2), (3)) AS t(id);

EXPLAIN ANALYZE - Basics

- Let’s have a look at the EXPLAIN ANALYZE command output.

```
regression=# EXPLAIN ANALYZE SELECT * FROM tenk1;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..445.00 rows=10000 width=244) (actual time=1.777..18.050 rows=10000 loops=1)
 Planning Time: 0.248 ms
 Execution Time: 18.853 ms
(3 rows)
```

- Additional information...
  - Actual Startup and Total Time (in ms)
  - Total Rows Returned
  - Loops
  - Planning Time (EXPLAIN with SUMMARY option gives this too)
  - Execution Time

EXPLAIN ANALYZE - Basics

- EXPLAIN ANALYZE command allows you to gauge planner's cost estimates.
  - Note! Costs are arbitrary units. So, time and cost wouldn't match.
  - In case of loops, average times and row counts are shown.
- Additional information...
  - Actual Time
  - Total Rows Returned
  - Loops
  - Planning Time (EXPLAIN with SUMMARY option gives this too)
  - Execution Time

The Tools

The Tools

- pg_stat_activity
- pg_stat_statements
- auto_explain
- pgbadger
- pg_stat_monitor

pg_stat_activity

pg_stat_activity

- A view in the pg_catalog schema.
- It tells you what's happening in the PostgreSQL right now.
  - It has one row per connection.
- Provides information about:
  - Connection, including database, username, client IP/host/port/ application/backend type,
  - Aging details in form of timestamps for transaction, connection and state, or transaction ID details.
  - State of the connection:
    - Active vs Idle
  - Wait events
    - Whether waiting on certain type of activity, and if yes, then what event!

pg_stat_activity: Take Aways

- Connection and application details.
- Rows of interest could be with values:
  - state = idle_in_transaction
    - idle_in_transaction (aborted) also if the transaction has savepoint(s).
  - state_change = <define threshold based on your use case>
  - wait_event = ClientRead | ClientWrite
- Aging transactions
- Wait event type is not NULL may require further investigation.
- Caveat:
  - Only currently connected server processes are shown.

pg_stat_statements

pg_stat_statements

- pg_stat_statement is one of the most commonly used extension.
- It tracks planning and execution statistics for all SQL queries executed by the server.
- It must be loaded by adding “pg_stat_statements” to “shared_preload_libraries” in postgresql.conf file.
  - Expect ~4% performance degradation when you do that with default options.
  - Uses an internal JumbleQuery function to calculate a query ID before PG14.
  - Only works in PG14 if compute_query_id is on (that’s the default).

pg_stat_statements View

- The pg_stat_statements view has 33 columns in PG 14.
- By default, track_planning is off. So, you'll see 0s for planning related columns.
  - Turning this on will have a detrimental effect on query performance!
- It provides statistics (total/min/max/mean/stddev) for:
  - Query planning times
  - Query execution times

pg_stat_statements View

- For shared, local, temp, disk blocks, it tracks reads and writes.
- For shared and local blocks, it also tracks hit and dirtied counts.
  - The same information for a specific SQL query can also be seen with “EXPLAIN [ANALYZE] (BUFFERS)” command.
- And then there are basic WAL statistics like records, fpi and bytes.

pg_stat_statements_info

- New view in PG14.
- It is a single row view that provides information about:
  - When pg_stat_statements was last reset,
  - The number of times pg_stat_statements had to deallocate least executed entry to make room for a new entry.

pg_stat_statements View Example

```
bench=# \x
bench=# SELECT query, calls, total_exec_time, rows, 100.0 * shared_blks_hit /
             nullif(shared_blks_hit + shared_blks_read, 0) AS hit_percent
        FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 2;
-[ RECORD 1 ]---+----------------------------------------------------------------
query           | UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2
calls           | 3000
total_exec_time | 25565.855387
rows            | 3000
hit_percent     | 100.0000000000000000
-[ RECORD 2 ]---+----------------------------------------------------------------
query           | UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2
calls           | 3000
total_exec_time | 271.232977
rows            | 3000
hit_percent     | 98.8454011741682975
```

* Query and data copied from PostgreSQL documentation for pg_stat_statements and adopted for the example.

- Hit percentage is 100 * (blocks hit)/(blocks hit + read).
- Sort by the required stat to identify any potential areas for optimization!

pg_stat_statements - Challenges

- If one needs to analyze time of day based (or time window based) statistics, there is no way that can be done (unless you reset statistics).
- Provides basic planning statistics, but not the plan.
- Does not provide information from relations/views perspective.
- Does not provide actual parameter values used in the SQL queries.
- It does not provide information about spread of timing data, as shown in the table on the next slide.

pg_stat_statements - Challenges

- Link to the spreadsheet: https://tinyurl.com/2p9axvxk
  - Identical:
    - Min/Max/StdDev
  - Different:
    - Mean/Total
  - Top 3 slowest queries:
    - Table 1:
      - 3000, 1000, 1000
    - Table 2:
      - 300, 500, 60

Query Timings	Min	Max	Mean	Sum Variance	Standard Deviation
1	1	1	1	0.00	0.00
50	1	50	25.5	1200.50	24.50
40	1	50	30.33333333	1340.67	21.14
189	1	189	70	20222.00	71.10
1000	1	1000	256	712142.00	377.40
1000	1	1000	380	1173422.00	442.23
3000	1	3000	754.2857143	7057193.43	1004.08
60	1	3000	667.5	7478972.00	966.89

Query Timings	Min	Max	Mean	Sum Variance	Standard Deviation
1	1	1	1	0.00	0.00
500	1	3000	250.5	124500.50	249.50
40	1	3000	180.3333333	154040.67	226.60
60	1	3000	150.25	164900.75	203.04
60	1	3000	132.2	171416.80	185.16
60	1	3000	120.1666667	175760.83	171.15
3000	1	3000	531.5714286	7284423.71	1020.11
60	1	3000	472.625	7479005.88	966.89

auto_explain

auto_explain Extension

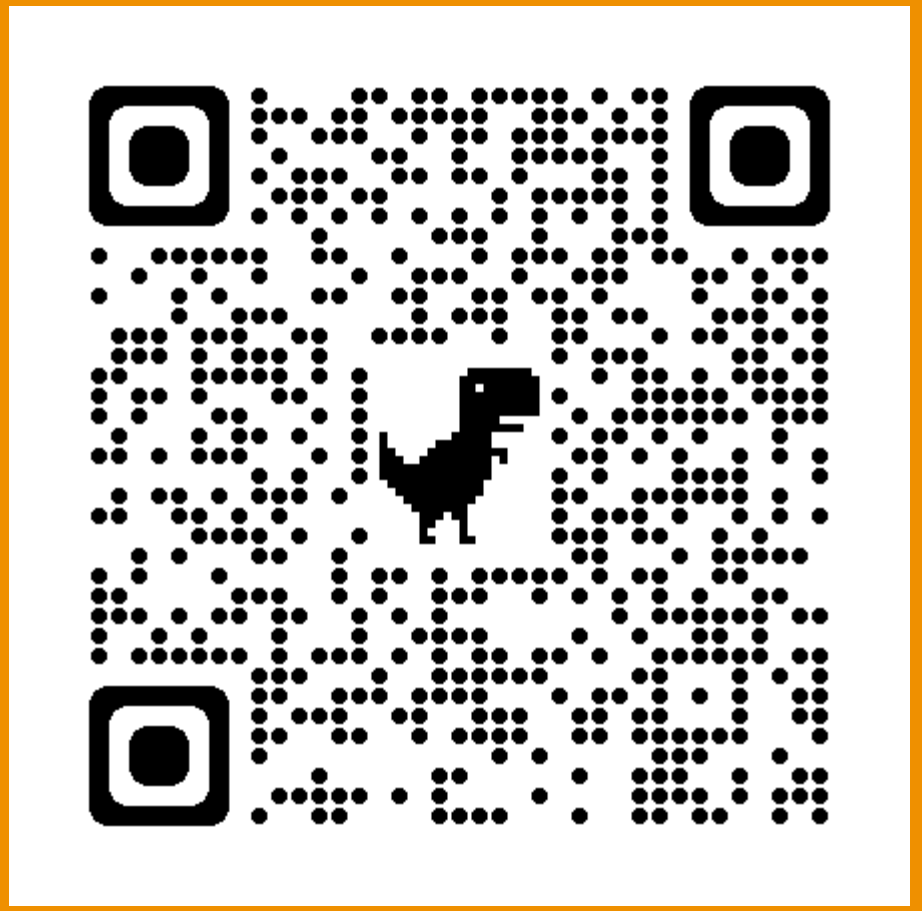
- It is a no SQL extension for PostgreSQL. So, all you have is a shared object that can be loaded in session or as part of preload libraries.
- Provides the option to log execution plans automatically.
  - auto_explain.log_min_duration GUC defines time threshold for logging SQL queries. Setting this to zero will log all queries.
  - So you don’t have to run EXPLAIN command manually.
- Configuration options include similar options to the EXPLAIN command:
  - WAL, buffers, timings, etc.
- Additionally, you can set to log triggers, or track deeper than top level query in a function call.

auto_explain Extension

- Allows access to the execution plan being used for a client application.
  - Why that matters? An application may be doing some innocuous change to the SQL query causing planner to choose a different plan.
- The challenge:
  - No SQL interface means that access to log files is required, or additional of another tool that can provide that access.

pgbadger

https://github.com/darold/pgbadger



PERCONA

pgbadger Tool

- A perl based, fast PostgreSQL server log file parser, analyzer and report generator.
- Support multiple report formats including HTML, JSON and text.
- Relies on proper log configuration in postgresql.conf file; e.g.
  - log_min_duration_statement
  - log_temp_files
  - etc.
- Beware to not overcook the log files.
  - For me, a 15 minute pgbench and make installcheck-world yielded a 4GB of log file!

pgbadger Statistics

- Some stats are borrowed from statistics tables and view. However, the log files have the ability to provide way more information.
  - That's what pgbadger utilizes.
- Generates lots of useful reports that allow you to:
  - Information about almost anything and everything in PostgreSQL server. For example:
    - See information about temporary files in general and in query context.
    - So that you can tune the work_mem

pgbadger - HTML Report

pgBadger | Overview | Connections | Sessions | Checkpoints | Temp Files | Vacuums | Locks | Queries | Top | Events

Queries generating the most temporary files (N)

Rank	Count	Total size	Min size	Max size	Avg size	Query
1	1,075	28.74 MiB	10.05 KiB	32.00 KiB	27.37 KiB	SELECT count(*) FROM join_foo LEFT JOIN (SELECT b1.id, b1.t FROM join_bar b1 JOIN join_bar b2 USING (id)) ss ON join_foo.id < ss.id + ? AND join_foo.id > ss.id - ?; Examples
2	906	23.28 MiB	24 B	64.00 KiB	26.31 KiB	SELECT count(*) FROM simple r JOIN simple s USING (id); Examples
3	630	19.72 MiB	23.37 KiB	64.00 KiB	32.05 KiB	SELECT original > ? AS initially_multibatch, final > original AS increased_batches FROM hash_join_batches ($$ SELECT count(*) FROM simple r JOIN simple s USING (id); $$); Examples
4	571	17.84 MiB	32.00 KiB	32.00 KiB	32.00 KiB	SELECT final > ? AS multibatch FROM hash_join_batches ($$ SELECT count(*) FROM join_foo LEFT JOIN (SELECT b1.id, b1.t FROM join_bar b1 JOIN join_bar b2 USING (id)) ss ON join_foo.id < ss.id + ? AND join_foo.id > ss.id - ?; $$); Examples
5	530	38.82 MiB	236 B	146.46 KiB	75.01 KiB	SELECT count(*) FROM simple r JOIN bigger_than_it_looks s USING (id); Examples
6	504	10.89 MiB	11.95 KiB	30.28 KiB	22.13 KiB	EXPLAIN (ANALYZE, format ?) SELECT count(*) FROM join_foo LEFT JOIN (SELECT b1.id, b1.t FROM join_bar b1 JOIN join_bar b2 USING (id)) ss ON join_foo.id < ss.id + ? AND join_foo.id > ss.id - ?; Examples

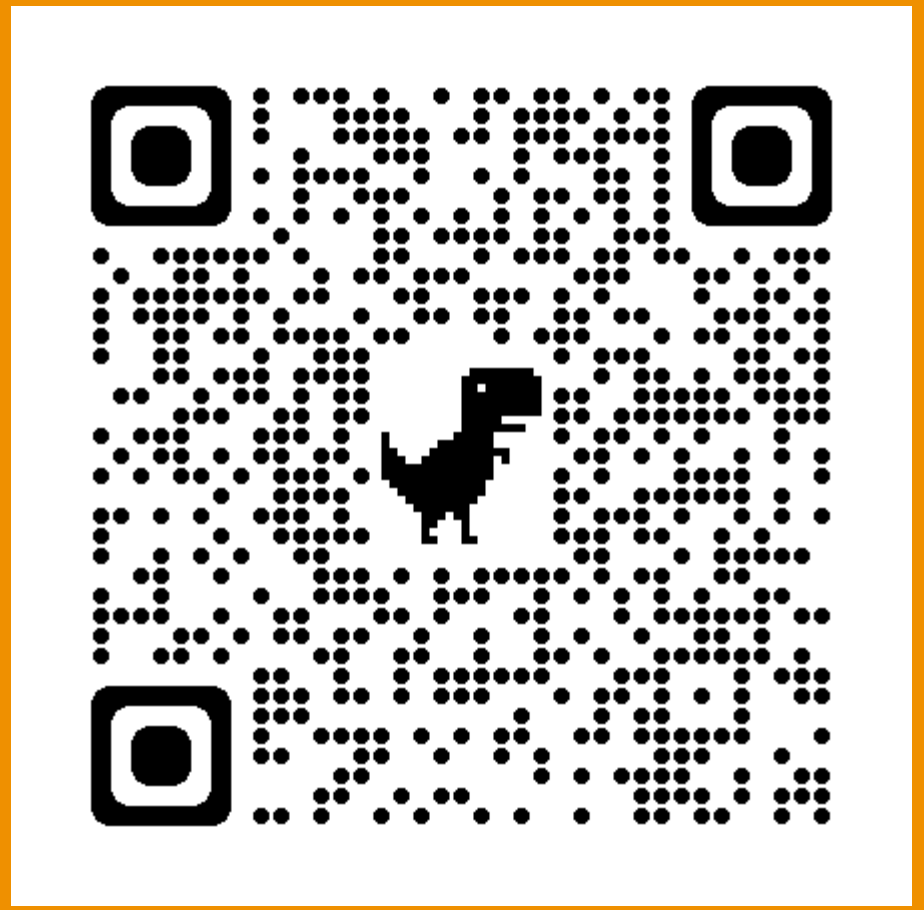
PERCONA

pgbadger Challenges

- Parsing huge log files could take time.
- You may need to setup a cronjob to run pgbadger regularly.
- Does not link queries with applications.
- The HTML interface is rather basic.
  - It’s not responsive! So viewing on a mobile phone is going to be tricky.

pg_stat_monitor

https://github.com/percona/pg_stat_monitor



PERCONA

pg_stat_monitor Extension

- pg_stat_monitor is a query performance observability extension that combines pg_stat_activity, pg_stat_statements and auto_explain to paint a wholistic picture.
- It provides:
  - Connection and application details [pg_stat_activity]
  - Query planning and execution statistics [pg_stat_statements]
  - Query execution plan [auto_explain]
  - All of this through its SQL interface!

pg_stat_monitor Extension

- It is designed to maintain query planning and execution statistics in a series of configurable time buckets.
  - Default is 10 buckets of 1 minute duration each.
- It groups queries within a time bucket based on:
  - Database
  - User
  - Client IP
  - Application Name
  - Query ID
  - Plan ID

pg_stat_monitor View

- pg_stat_monitor view has all of pg_stat_statements columns and 19 additional columns in PG14!
  - That's a total of 51 columns!
  - So, I'm not going to repeat those here.
- enable_query_plan is off. So, you'll see an empty value for plan.
  - Turning this on will have a significant detrimental effect on query performance!
- Application and client information
- Top query information including:
  - Top query ID
  - Top query text

pg_stat_monitor View

- It also provides:
  - List of relations and views impacted by a query
  - Query meta information that might be present in query test in Google Sqlcomment like syntax; key value pairs.
  - Actual parameter values that are used in a query to simplify the debugging and analysis process.
- All of this data is maintained in a fixed number of time buckets.
  - Buckets are recycled!

pg_stat_monitor - Histogram

```
SELECT resp_calls, query FROM pg_stat_monitor;
                    resp_calls                    |                query
--------------------------------------------------+---------------------------------------
{1," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 0"} | select client_ip, query from pg_stat_monitor
{3," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 1"} | select * from pg_stat_monitor_reset()
{3," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 0"," 1"} | SELECT * FROM foo

postgres=# SELECT * FROM histogram(0, 'F44CD1B4B33A47AF') AS a(range TEXT, freq INT, bar TEXT);
       range        | freq |              bar
--------------------+------+--------------------------------
  (0 - 3)}          |    2 | ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  (3 - 10)}         |    0 |
  (10 - 31)}        |    1 | ■■■■■■■■■■■■■■
  (31 - 100)}       |    0 |
  (100 - 316)}      |    0 |
  (316 - 1000)}     |    0 |
  (1000 - 3162)}    |    0 |
  (3162 - 10000)}   |    0 |
  (10000 - 31622)}  |    0 |
  (31622 - 100000)} |    0 |
(10 rows)
```

- Query-bucket-wise timing histogram to clearly show the spread of timing data.

pg_stat_monitor

- Extension is not yet available for production.
- Feel free to try out the RC release though.

Summarizing

Tools for the Use Case

- Application/Connection Information
  - pg_stat_activity
  - pg_stat_monitor
- Query Text
  - pg_stat_statements
  - pgbadger
  - pg_stat_monitor
- Execution Plan
  - auto_explain
  - pg_stat_statements
  - pgbadger
  - pg_stat_monitor

Tools for the Use Case

- Planning and Execution Timing Statistics
  - pg_stat_statements
  - pgbadger
  - pg_stat_monitor
- Query Execution Timing Histogram
  - pg_stat_monitor
- Block Read/Write Statistics
  - pg_stat_statements
  - pgbadger
  - pg_stat_monitor

Tools for the Use Case

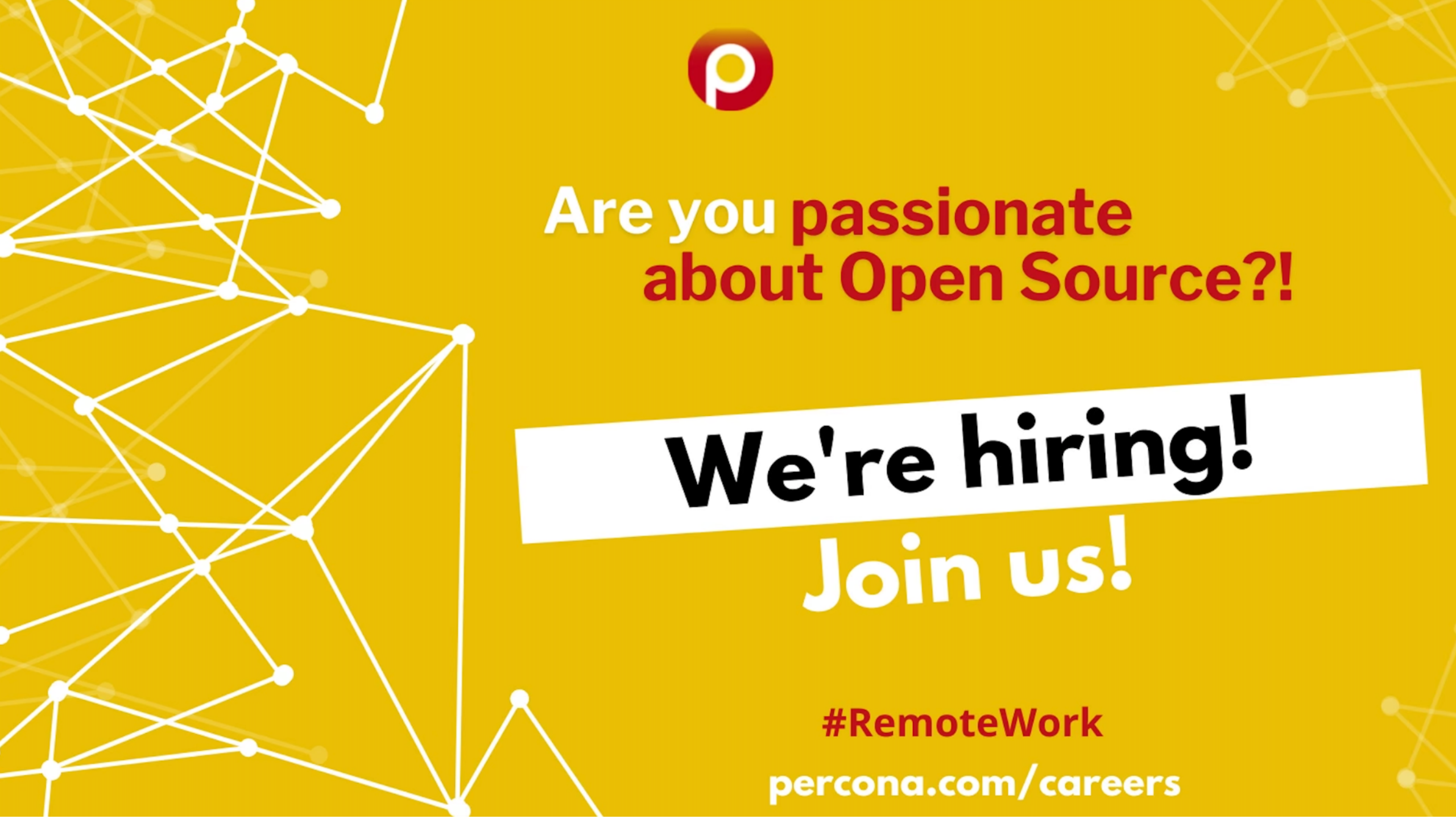
- Wait Events and Locks
  - pg_stat_activity
  - pgbadger
- SQL Interface
  - pg_stat_activity
  - pg_stat_statements
  - pg_stat_monitor

Percona stands for evolution

Percona stands for ease-of-use

Percona stands for freedom

Percona & PostgreSQL - Better Together


Are you passionate about Open Source?!
We're hiring!
Join us!
#RemoteWork
percona.com/careers

If you have 30 mins, I’d love to talk to you about PostgreSQL.

Thank you!

Questions?