6. Monitoring

Introduction to PostgreSQL

AGENDA

- Configuration Monitoring
- Base Performance Monitoring
- Advanced Performance Monitoring
- Tools

MAIN CONFIG FILES

- There are three main configuration files that present the starting point for any configuration:
  - postgresql.conf is the main cluster configuration file
    - *contains all the data required to start the cluster, set up processes (as WAL senders) and logging, and configure how the cluster will accept connections*
  - postgresql.auto.conf automatically generated and edited by the cluster itself
    - *contains parameters changed by the superuser from within the cluster.*
  - pg_hba.conf used to allow or deny the client connections to the cluster.
- A parameter can be defined multiple times but only the last definition found is used
  - Every parameter has a default value used if it is not defined in the config specs

RUNNING CONFIG

- You can inspect the running configuration by running a query on *pg_settings*
  - You will do this in more detail in the lab.

```
rod=# SELECT name, setting || ' ' || unit as current_value,
min_val, max_val, boot_val, reset_val
from pg_settings;
                 name                 | current_value |   min_val   |   max_val   |           boot_val            |             reset_val
--------------------------------------+---------------+-------------+-------------+-------------------------------+-----------------------
 allow_in_place_tablespaces           |               |             |             | off                           | off
 allow_system_table_mods              |               |             |             | off                           | off
 application_name                     |               |             |             |                               | psql
 archive_cleanup_command              |               |             |             |                               |
 archive_command                      |               |             |             |                               |
 archive_library                      |               |             |             |                               |
 archive_mode                         |               |             |             | off                           | off
 archive_timeout                      | 0 s           | 0           | 1073741823  | 0                             | 0
 array_nulls                          |               |             |             | on                            | on
 authentication_timeout               | 60 s          | 1           | 600         | 60                            | 60
 autovacuum                           |               |             |             | on                            | on
 autovacuum_analyze_scale_factor      |               | 0           | 100         | 0.1                           | 0.1
 autovacuum_analyze_threshold         |               | 0           | 2147483647  | 50                            | 50
 autovacuum_freeze_max_age            |               | 100000      | 2000000000  | 200000000                     | 200000000
 autovacuum_max_workers               |               | 1           | 262143      | 3                             | 3
 autovacuum_multixact_freeze_max_age  |               | 10000       | 2000000000  | 400000000                     | 400000000
 autovacuum_naptime                   | 60 s          | 1           | 2147483     | 60                            | 60
 autovacuum_vacuum_cost_delay         | 2 ms          | -1          | 100         | 2                             | 2
 autovacuum_vacuum_cost_limit         |               | -1          | 10000       | -1                            | -1
 autovacuum_vacuum_insert_scale_factor|               | 0           | 100         | 0.2                           | 0.2
 autovacuum_vacuum_insert_threshold   |               | -1          | 2147483647  | 1000                          | 1000
 autovacuum_vacuum_scale_factor       |               | 0           | 100         | 0.2                           | 0.2
 autovacuum_vacuum_threshold          |               | 0           | 2147483647  | 50                            | 50
 autovacuum_work_mem                  | -1 kB         | -1          | 2147483647  | -1                            | -1
 backend_flush_after                  | 0 8kB         | 0           | 256         | 0                             | 0
 backslash_quote                      |               |             |             | safe_encoding                 | safe_encoding
 backtrace_functions                  |               |             |             |                               |
 bgwriter_delay                       | 200 ms        | 10          | 10000       | 200                           | 200
 bgwriter_flush_after                 | 64 8kB        | 0           | 256         | 64                            | 64
 bgwriter_lru_maxpages                |               | 0           | 1073741823  | 100                           | 100
 bgwriter_lru_multiplier              |               | 0           | 10          | 2                             | 2
```

RUNNING CONFIG

- The pg_settings contains for each parameter
  - The file and the line number from where a parameter has been loaded.
  - Used to find where a configuration parameter has been set
  - In the screenshot, only the parameters set in a file are listed.

```
rod=# SELECT name, setting as current_value,
sourcefile, sourceline, pending_restart
from pg_settings
where sourcefile is Not null;
            name             |   current_value    |              sourcefile              | sourceline | pending_restart
-----------------------------+--------------------+--------------------------------------+------------+-----------------
 DateStyle                   | ISO, YMD           | /var/lib/pgsql/data/postgresql.conf |        715 | f
 default_text_search_config  | pg_catalog.english | /var/lib/pgsql/data/postgresql.conf |        741 | f
 dynamic_shared_memory_type  | posix              | /var/lib/pgsql/data/postgresql.conf |        153 | f
 lc_messages                 | en_CA.UTF-8        | /var/lib/pgsql/data/postgresql.conf |        731 | f
 lc_monetary                 | en_CA.UTF-8        | /var/lib/pgsql/data/postgresql.conf |        733 | f
 lc_numeric                  | en_CA.UTF-8        | /var/lib/pgsql/data/postgresql.conf |        734 | f
 lc_time                     | en_CA.UTF-8        | /var/lib/pgsql/data/postgresql.conf |        735 | f
 listen_addresses            | *                  | /var/lib/pgsql/data/postgresql.conf |         60 | f
 log_filename                | postgresql-%a.log  | /var/lib/pgsql/data/postgresql.conf |        465 | f
 log_rotation_age            | 1440               | /var/lib/pgsql/data/postgresql.conf |        469 | f
 log_timezone                | America/Winnipeg   | /var/lib/pgsql/data/postgresql.conf |        603 | f
 log_truncate_on_rotation    | on                 | /var/lib/pgsql/data/postgresql.conf |        474 | f
 logging_collector           | on                 | /var/lib/pgsql/data/postgresql.conf |        457 | f
 max_connections             | 100                | /var/lib/pgsql/data/postgresql.conf |         65 | f
 max_wal_size                | 1024               | /var/lib/pgsql/data/postgresql.conf |        247 | f
 min_wal_size                | 80                 | /var/lib/pgsql/data/postgresql.conf |        248 | f
 port                        | 5432               | /var/lib/pgsql/data/postgresql.conf |         64 | f
 shared_buffers              | 16384              | /var/lib/pgsql/data/postgresql.conf |        130 | f
 TimeZone                    | America/Winnipeg   | /var/lib/pgsql/data/postgresql.conf |        717 | f
(19 rows)
```

CONFIG SETTINGS

- For each of the parameters we looked at in the the previous slides, we can use the SHOW command.
  - We can also use the SET command to override for the current session
  - Or ALTER SYSTEM SET to permanently override the parameter

```
rod=# SHOW work_mem ;
 work_mem
----------
 4MB
(1 row)

rod=# SET work_mem = '8MB';
SET
rod=# SHOW work_mem ;
 work_mem
----------
 8MB
(1 row)

rod=# ALTER SYSTEM SET work_mem = '8MB';
ALTER SYSTEM
rod=# SHOW work_mem ;
 work_mem
----------
 8MB
(1 row)
```

LOCATING CONFIG

- Because the parameters can be set in different files, the table pg_file_settings can show exactly where the current parameter value was applied from.
  - Because we overrode the value of work_mem with the ALTER SYSTEM command, we see that here

```
rod=# SELECT name, setting, sourcefile, sourceline,
applied
from pg_file_settings
where name = 'work_mem';
   name   | setting |                sourcefile                | sourceline | applied
----------+---------+------------------------------------------+------------+---------
 work_mem | 8MB     | /var/lib/pgsql/data/postgresql.auto.conf |          3 | t
(1 row)
```

LOCATING CONFIG

- We can also find out which parameters were not successfully applied but checking the applied flag and any possible errors
  - In this case, no parameters were not applied
  - The error field may contain a message describing the error that prevented the parameter from not being set, like an invalid or out of range value;

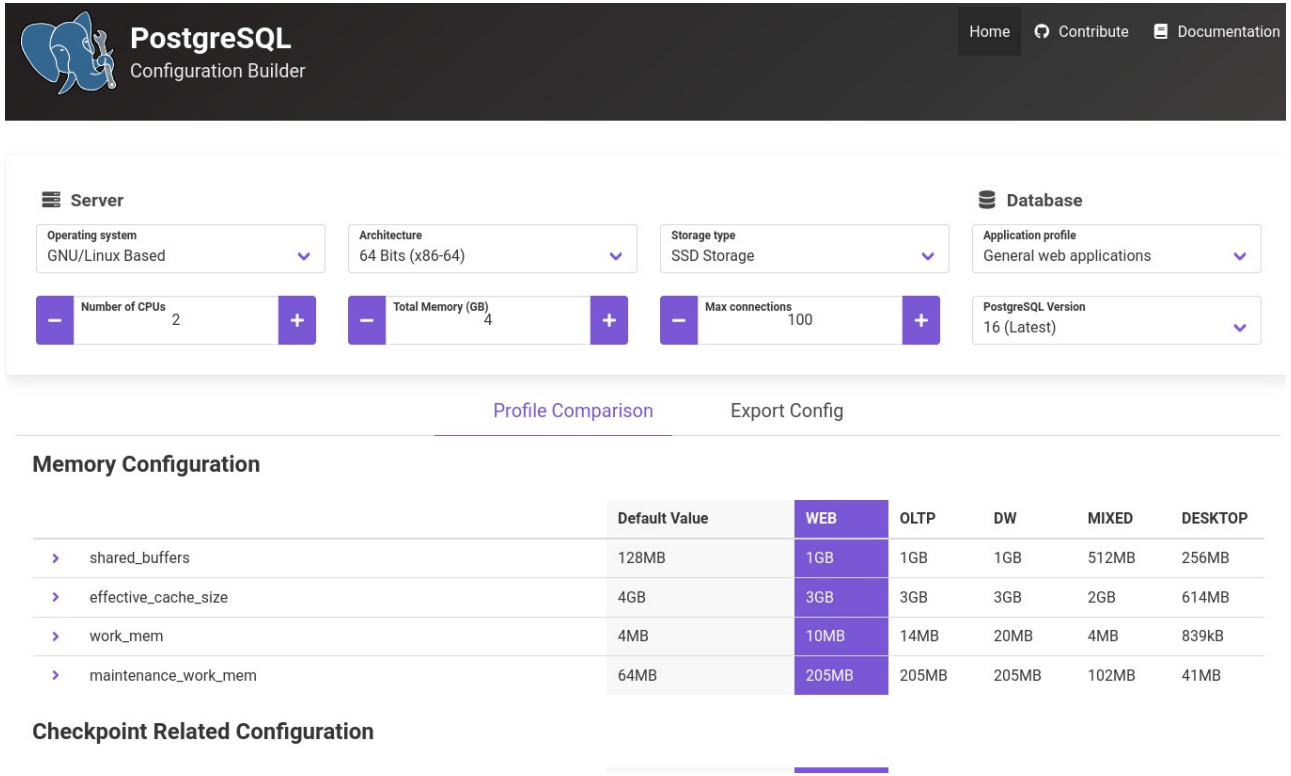
```
rod=# SELECT *
FROM pg_file_settings
WHERE applied = false;
 sourcefile | sourceline | seqno | name | setting | applied | error
------------+------------+-------+------+---------+---------+-------
(0 rows)
```

CONFIGURATION CONTEXTS

- Each configuration parameter belongs to a context
  - This is a group that defines when a change to the parameter can be applied.
  - Some parameters can be changed and take effect during the cluster's life cycle.
  - Others cannot and require the cluster to be restarted
- Contexts are:
  - internal: Value depends on the PostgreSQL source code and is established at compile time, and can only be changed by recompiling.
  - postmaster: The cluster must be restarted a change to be applied.
  - sighup: Changes when given a hang-up signal, usually a reload of the operating system service.
  - superuser-backend and backhand: Changes will be applied to both the client and administratorfrom the next connection of either type.
  - user and superuser: These changes will be applied immediately to the current connection,

CONFIGURATION GENERATORS

- Configuration generators are used to build a base configuration for a cluster that can be tuned.
  - We will not be exploring these in detail.
  - The lab explores a commonly used one ‘pgconfig.org’



LAB 6-1

- The lab description and documentation is in the Lab directory in the class repository

BASIC OS MONITORING

- Basic monitoring in PostgreSQL focuses on ensuring the general health and performance of the database.
- The key areas include checking system resources, database connectivity, instance availability, and basic performance metrics.
- Instance Availability and Connection Monitoring
  - pg_isready, simple command-line tool used to check the availability of a PostgreSQL instance. Equivalent to a "ping" for the database.
- System Resource Monitoring
  - CPU and Memory: Monitor CPU usage, memory usage, and swap space to ensure the database has sufficient resources.
  - Disk I/O: Monitor read/write latency, IOPS, and throughput using tools like iostat, vmstat, or system-specific monitoring solutions.
  - Disk Space: Regularly check disk space to prevent out-of-space errors, which can severely impact PostgreSQL, as it relies heavily on disk for data storage and temporary operations.

SYSTEM TOOLS MONITORING

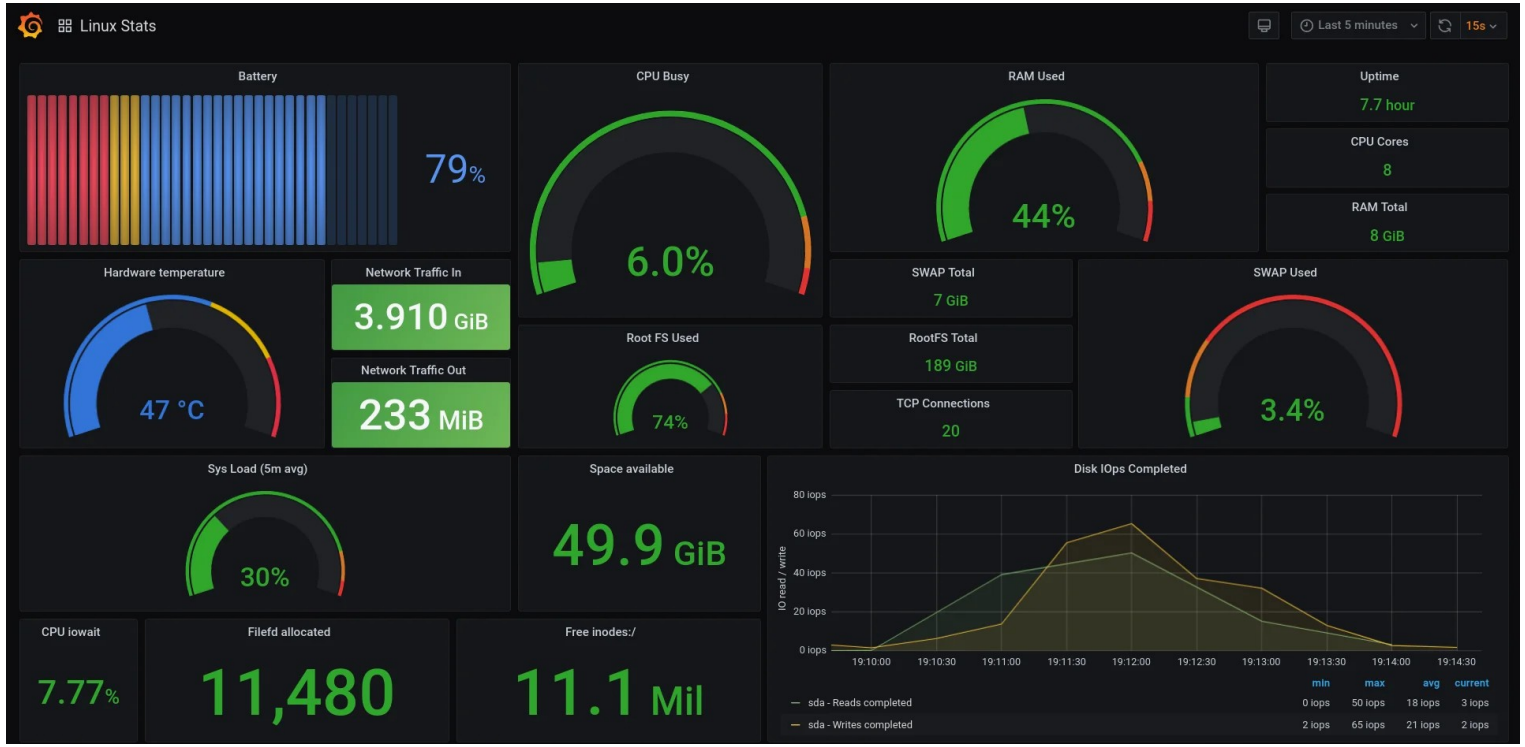
- Generally PostgreSQL instances run on Unix
  - There are a number of Unix system monitoring tools that can be used
  - These treat PostgreSQL like any other process
  - Useful for monitoring resource usage
- ‘ps’ - monitors system processes, checks the status of PostgreSQL processes
  - “ps aux | grep postgres”
- ‘top and htop’ - monitor real-time system performance, including CPU, memory usage being for each running processes.
- ‘vmstat’ - Monitor system performance metrics, including CPU, memory, and I/O statistics.
  - Check for I/O bottlenecks and memory swapping that could impact PostgreSQL performance.

SYSTEM TOOLS MONITORING

- ‘iostat’ - monitor disk I/O statistics
  - Identify disk read/write patterns and potential I/O bottlenecks that can affect database performance.
- ‘ netstat’ - Monitor network connections and statistics
- ‘pg_top’ - A PostgreSQL-specific tool based on top for monitoring active sessions, queries, and performance statistics.
  - View running queries, resource consumption by each session, and overall database activity.

SYSTEM MONITORING

- Graphical OS monitoring tools like Grafana and Prometheus are generally employed
  - Allow for various sort of alerts
  - Not covered in this course

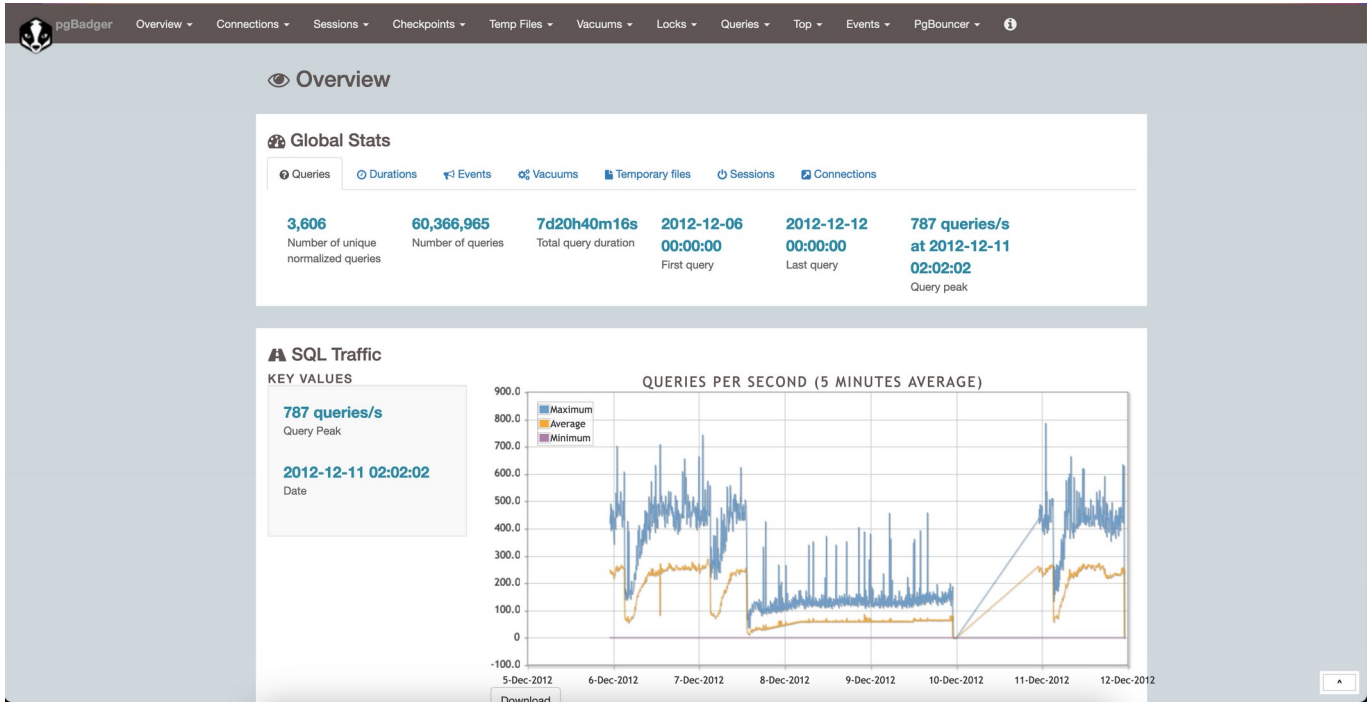


BASIC PERFORMANCE MONITORING

- Database Logs
  - Log Files: PostgreSQL logs important events such as connection attempts, errors, warnings, and slow queries.
  - Configuration: Set appropriate logging levels in postgresql.conf (log_min_duration_statement, log_error_verbosity, etc.) to capture relevant events without overwhelming the system with logs.
  - Log Analysis: Use tools like pgBadger or custom scripts to parse and analyze logs for insights into errors, performance issues, and suspicious activities.
- Basic Performance Metrics
  - Tools: pg_stat_activity, pg_stat_database, pg_stat_user_tables.
  - Key Metrics: Active sessions and connections.
  - Database-wide statistics such as transaction counts, number of deadlocks, and cache hit ratios.
  - Table-level statistics like sequential scans, index scans, and tuples read/returned.
  - Usage: These views provide an overall view of database activity and can be queried regularly or integrated into a monitoring dashboard.

PGBADGER

- Log analysis tool
  - Officially supported by PostgreSQL
  - Provides graphic and quantitative reports on activities.
  - Link is in the notes



COLLECTING STATS

- The cluster collects information about activities by means of the statistic collector
  - A dedicated process that collects cluster-wide information
  - Statistics are not in real time, they are are updated at least every 500 milliseconds by backend processes
  - Statistics within a transaction block are “frozen” and cannot be collected until the transaction terminates
  - Statistics are kept across shutdowns and restarts, but are lost after a crash and are reset.
  - Individual database stats can be reset with pg_stat_reset()

QUERIES AND SESSIONS

- *pg_stat_activity* catalog records every back end process active in the cluster
  - Only reports the last executed query from a session or connection.
  - Not updated until a new statement is executed.

```
state           |
backend_xid     |
rod=# \d pg_stat_activity
                    View "pg_catalog.pg_stat_activity"
      Column      |           Type           | Collation | Nullable | Default
------------------+--------------------------+-----------+----------+---------
 datid            | oid                      |           |          |
 datname          | name                     |           |          |
 pid              | integer                  |           |          |
 leader_pid       | integer                  |           |          |
 usesysid         | oid                      |           |          |
 usename          | name                     |           |          |
 application_name | text                     |           |          |
 client_addr      | inet                     |           |          |
 client_hostname  | text                     |           |          |
 client_port      | integer                  |           |          |
 backend_start    | timestamp with time zone |           |          |
 xact_start       | timestamp with time zone |           |          |
 query_start      | timestamp with time zone |           |          |
 state_change     | timestamp with time zone |           |          |
 wait_event_type  | text                     |           |          |
 wait_event       | text                     |           |          |
 state            | text                     |           |          |
 backend_xid      | xid                      |           |          |
 backend_xmin     | xid                      |           |          |
 query_id         | bigint                   |           |          |
 query            | text                     |           |          |
 backend_type     | text                     |           |          |
```

QUERIES AND SESSIONS

```
rod=# SELECT usename, datname, client_addr, application_name,
backend_start, query_start,
state, backend_xid, query
FROM pg_stat_activity;
```

```
query           |
-[ RECORD 2 ]----+-----------------------------------------------------
usename          | postgres
datname          |
client_addr      |
application_name |
backend_start    | 2024-11-20 08:16:17.546575-06
query_start      |
state            |
backend_xid      |
query            |
-[ RECORD 3 ]----+-----------------------------------------------------
usename          | rod
datname          | rod
client_addr      |
application_name | psql
backend_start    | 2024-12-02 19:31:07.463686-06
query_start      | 2024-12-02 19:37:45.067357-06
state            | active
backend_xid      |
query            | SELECT usename, datname, client_addr, application_name,+
                 | backend_start, query_start,                            +
                 | state, backend_xid, query                              +
                 | FROM pg_stat_activity;
-[ RECORD 4 ]----+-----------------------------------------------------
usename          |
```

LOCKING

- MVCC ensures that only one transaction can modify a particular resource at a time (e.g., a row or table)
  - MVCC allows readers to access older versions of data
  - Does not inherently prevent multiple writers from conflicting.
  - Locks (such as row-level locks like SELECT ... FOR UPDATE) prevent such write conflicts.
  - Locks are required to control schema modifications, such as altering a table, creating an index, or dropping a table.
  - Locks manage access to non-database resources or custom application logic using advisory locks, which MVCC does not support.
  - Locks help enforce certain constraints, such as foreign key constraints or unique constraints, by preventing conflicting operations.
  - And other use cases.

MOINTORING LOCKS

- The pg_locks special catalog records any locks that are acquired by different transactions and statements.
  - This catalog allows the system administrator to identify possible bottlenecks
  - Often queried by joining with pg_stat_activity to get more detailed information

```
backend_start    | 2024-11-26 09:15:23.091624-05
student=#  SELECT a.usename, a.application_name, a.datname, a.query,
student-# l.granted, l.mode
student-# FROM pg_locks l
student-# JOIN pg_stat_activity a ON a.pid = l.pid;
-[ RECORD 1 ]----+-------------------------------------------------
usename          | student
application_name | psql
datname          | student
query            | SELECT a.usename, a.application_name, a.datname, a.query,+
                 | l.granted, l.mode                                       +
                 | FROM pg_locks l                                         +
                 | JOIN pg_stat_activity a ON a.pid = l.pid;
granted          | t
mode             | AccessShareLock
-[ RECORD 2 ]----+-------------------------------------------------
```

MONITORING DATABASES

- Detailed information about databases are in the pg_stat_database catalog.
  - Provides information about COMMIT and ROLLBACK transactions, deadlocks, and conflicts.
  - If you see the numbers of COMMITs and ROLLBACKs grow quickly, this may be due to an application error or clients making errors in a database forcing rollbacks

```
student=# \x
Expanded display is off.
student=#  SELECT datname, xact_commit, xact_rollback, blks_read,
student-# conflicts, deadlocks,
student-# tup_fetched, tup_inserted, tup_updated, tup_deleted, stats_reset
student-# FROM pg_stat_database;
  datname  | xact_commit | xact_rollback | blks_read | conflicts | deadlocks | tup_fetched | tup_inserted | tup_updated | tup_deleted | stats_res
et
-----------+-------------+---------------+-----------+-----------+-----------+-------------+--------------+-------------+-------------+----------
---
           |           0 |             0 |       158 |         0 |         0 |       91407 |           38 |           8 |           5 |
 postgres  |      150825 |             2 |      1962 |         0 |         0 |      178499 |          821 |         111 |          91 |
 template1 |       22421 |             0 |      1210 |         0 |         0 |      182690 |        19080 |         754 |          34 |
 template0 |           0 |             0 |         0 |         0 |         0 |           0 |            0 |           0 |           0 |
 student   |        6895 |            19 |         0 |         0 |         0 |       55138 |           67 |           1 |          31 |
 zorgo     |        1980 |             3 |         0 |         0 |         0 |       21222 |           19 |           0 |           0 |
(6 rows)
```

ADVANCED MONITORING

- Advanced Performance Monitoring
  - pg_stat_bgwriter: Provides insights into background writer activity, checkpoints, and buffer management, which are critical for understanding I/O performance.
  - Query Optimization: Utilize EXPLAIN and EXPLAIN ANALYZE to review query plans and execution times, identifying areas where indexes, rewrites, or configuration changes could improve performance.
- Resource Contention and Wait Event Analysis
  - Wait Events: PostgreSQL has a wait event monitoring system that tracks what processes are waiting on, similar to Oracle's wait events.
  - Blocking and Deadlock Detection: Identify and resolve lock contention and deadlocks using pg_locks and pg_blocking_pids() functions.
  - Buffer and Cache Analysis: Monitor buffer usage and cache hit ratios through pg_buffercache and pg_stat_database. Low cache hit ratios can indicate inefficient queries or inadequate memory allocation.

ADVANCED MONITORING

- Replication and High Availability Monitoring
  - Streaming Replication: Monitor replication status using views like pg_stat_replication, which shows the state of replication, lag, and connection status of replicas.
  - Failover and Recovery: Tools like repmgr or Patroni can be used to manage and monitor high availability setups, providing alerts and automation for failover scenarios.
- System-Level Metrics Integration
  - Prometheus and Grafana: Use Prometheus with exporters like node_exporter and postgres_exporter for detailed time-series data on system and database performance.
  - Detailed Metrics: Monitor CPU, memory, disk I/O, and network metrics alongside database-specific metrics for a holistic view of system health.
- Alerting and Automated Responses
  - Threshold-Based Alerts: Set up alerts for critical metrics such as high CPU usage, long-running queries, replication lag, and disk space.
  - Automated Responses: Implement automated actions in response to certain alerts, such as killing runaway queries, initiating failovers, or adjusting configuration parameters.

ADVANCED MONITORING

- Replication and High Availability Monitoring
  - Streaming Replication: Monitor replication status using views like pg_stat_replication, which shows the state of replication, lag, and connection status of replicas.
  - Failover and Recovery: Tools like repmgr or Patroni can be used to manage and monitor high availability setups, providing alerts and automation for failover scenarios.
- System-Level Metrics Integration
  - Prometheus and Grafana: Use Prometheus with exporters like node_exporter and postgres_exporter for detailed time-series data on system and database performance.
  - Detailed Metrics: Monitor CPU, memory, disk I/O, and network metrics alongside database-specific metrics for a holistic view of system health.
- Alerting and Automated Responses
  - Threshold-Based Alerts: Set up alerts for critical metrics such as high CPU usage, long-running queries, replication lag, and disk space.
  - Automated Responses: Implement automated actions in response to certain alerts, such as killing runaway queries, initiating failovers, or adjusting configuration parameters.

ADVANCED MONITORING

- Advanced Log Analysis
  - Log Aggregation and Analysis: Use tools like ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk to aggregate and analyze PostgreSQL logs for patterns, anomalies, and trends.
  - Custom Alerts from Logs: Set up alerts based on specific log entries, such as repeated connection failures, frequent checkpoint warnings, or slow queries exceeding a defined threshold.
- Key Differences from Oracle Monitoring
  - Schema and Object Ownership: PostgreSQL's schema design and object ownership can influence monitoring strategies, as it has a more granular permissions model compared to Oracle.
  - Autovacuum and Vacuum Monitoring: Unlike Oracle's automated segment management, PostgreSQL relies on autovacuum processes to reclaim space, which needs to be closely monitored and tuned to avoid performance degradation.
  - Configuration Flexibility: PostgreSQL allows extensive tuning via configuration files (postgresql.conf), requiring regular review and adjustment based on monitored performance data.

MONITORING TOOLS

- Sampling of the most common tools
  - Links to the tools are in the Notes file in the repository
- *PdAdmin*: A popular open-source tool that provides a graphical interface for managing PostgreSQL databases. It includes built-in monitoring and graphing tools.
- *pg_stat_statements*: A PostgreSQL extension that tracks execution statistics of all SQL statements executed by a server.
  - Available within PostgreSQL; it needs to be enabled by adding it to the shared_preload_libraries in the postgresql.conf file.
- *PgBadger*: A fast PostgreSQL log analyzer that generates detailed reports on performance based on log files.

MONITORING TOOLS

- *pg_top*: Similar to the Unix top command but for PostgreSQL, providing a real-time view of database processes, queries, and statistics.
- *PostgreSQL Exporter for Prometheus*: Collects PostgreSQL metrics and exports them to Prometheus for monitoring and alerting.
- *Pgmetrics*: A command-line tool that collects various statistics and configurations from a running PostgreSQL server and displays them in a detailed report format.
- *Percona*: A comprehensive monitoring tool that supports PostgreSQL and offers insights into database performance with advanced dashboards and alerts.
- *pg_stat_kcache:* A PostgreSQL extension that provides statistics on CPU and I/O usage for all SQL statements.

MONITORING TOOLS

- *TimescaleDB*: A time-series database based on PostgreSQL that includes additional features for monitoring and managing time-series data.
- *pgCluu*: A PostgreSQL clusters performance monitoring tool that collects, stores, and visualizes statistics from PostgreSQL and the operating system.

LAB 6-2

- The lab materials and instructions are in the repository

End Module