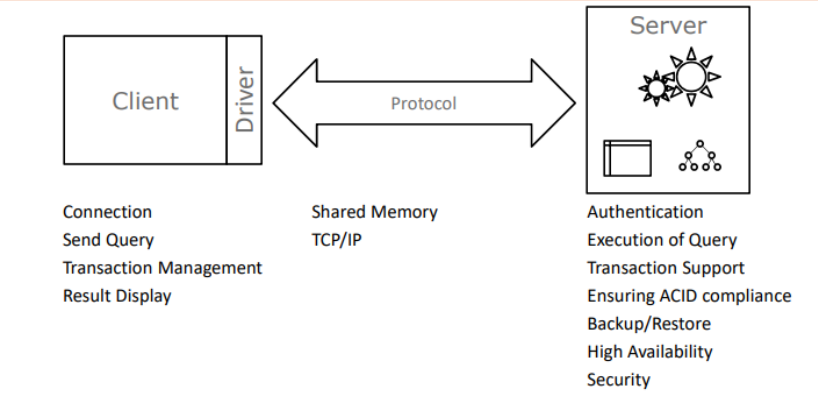
# NOTES
# FOR
# POSTGRESQL
# ARCHITECTURE



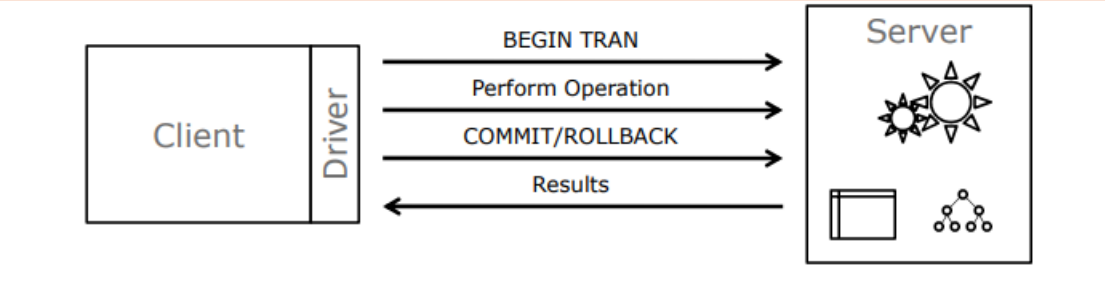
PREPARED BY: SUBHAM DASH
DBA CONSULTANT

# PostgreSQL Architecture

**Model**: PostgreSQL like other RDBMs follows Client/Server model.




**Transaction Flow:** Transactions are supported by PostgreSQL.




## Fully ACID compliant

A: Atomicity: All or nothing

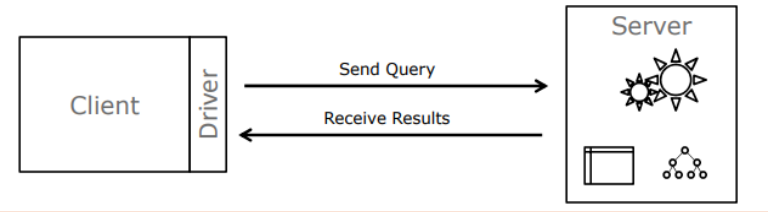
C: Consistency: Data correctness

I: Isolation: No influence of concurrent transactions

D: Durability: Committed data will survive failures

## Query Execution

PostgreSQL accepts Queries from clients, process them and returns Results back to clients.

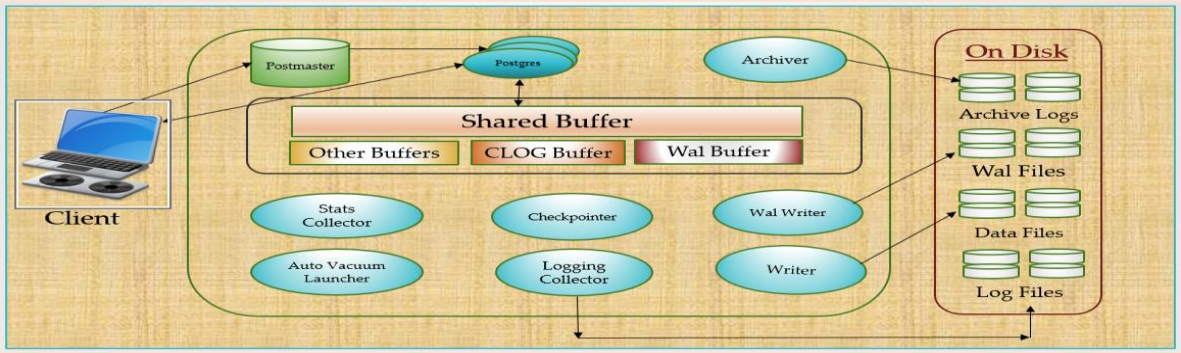



PREPARED BY: SUBHAM DASH
DBA CONSULTANT

## Query Processing

| | | |
|---|---|---|
| **Parsing** | : Check Syntax, Catalogs | : Parse Tree |
| **Transformation** | : Rewrites query as per the Rule System | : Query Tree |
| **Planning** | : Binding, Check Statistics, do Cost based analysis | : Execution Plan |
| **Execution** | : Execute Plan | : Generate Result set |

## PostgreSQL Architecture Fundamentals & PostgreSQL Process & Memory Architecture




- PostgreSQL is a relational database management system with a client-server architecture.
  - Client-server architecture means which request and receive services from a centralized server or a host computer.
  - Client is those which run application on local work station, our PC send request to the server and the server response to our request.
- PostgreSQL use “process per-user” client/server model.
  - Each user is granted or given a particular process in PostgreSQL.
  - So, when try to connect to PostgreSQL i will be given a separate process which will stay there till the connection or till the computer is logoff or shutdown.
- PostgreSQL's has a set of processes and memory structures which constitutes an instance.
  - An instance is a combination of processes and memory structures.
  - On a server there are many instances, and each instance will have its own processes and memory structure.
  - We cannot share the processes and memory of one instance with another.
- Programs run by clients connects to the server instance and request read and write operation.
  - The client connects to the server, it asks for certain operation like read & write, and Postgres architecture supports that kind.

### Architecture Classification

PostgreSQL architecture is classified into following sections.

- Memory Structure
- Background processes
- Physical Files
- Logical Structure

PREPARED BY: SUBHAM DASH
DBA CONSULTANT

**Memory Structure**

- Shared Buffer
- WAL Buffer
- CLOG Buffer
- Work Memory
- Maintenance Work Memory

**Background Processe**

- WAL Writer
- BG Writer
- Checkpointer
- Stat Collector
- Logging Collector
- Auto-vacuum Launcher

**Physical Files**

- Archive Logs
- WAL Files
- Data Files
- Log Files

**Logical Structure**

- Tablespace
- Users
- Databases

**HOW THE ARCHITECTURE WORK IN REALTIME?**

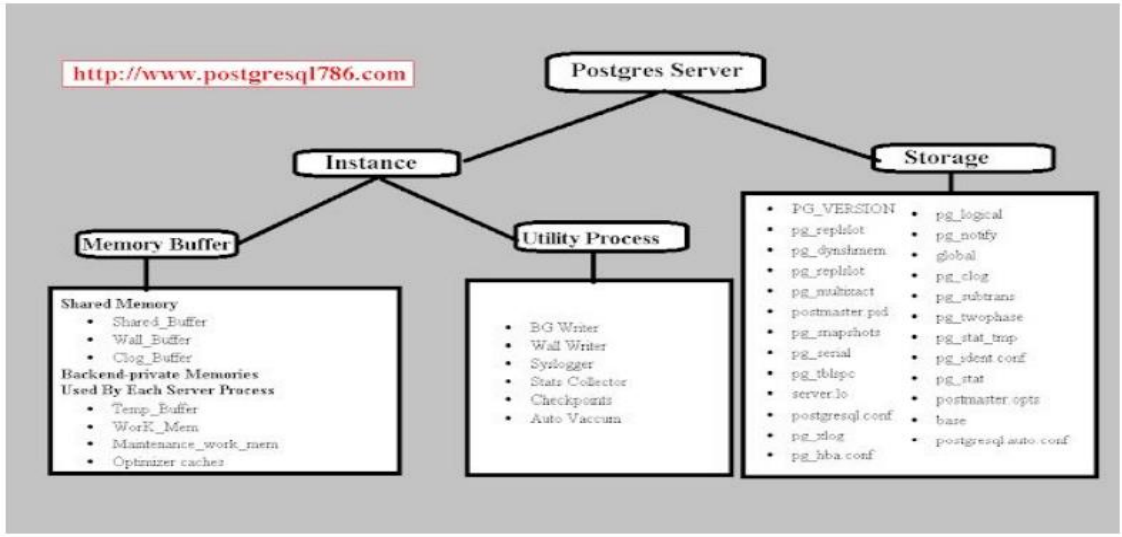
- A user/client sends a request from his workstation to connect to PostgreSQL.
- The request is picked up by postmaster which checks the user credential and IP address are valid.
- On success postmaster starts a new process called postgres and handover the connection to postgres.
- Postgres is connected to shared buffer.
- Now the client can communicate with the database, and the connection is kept alive till the user is logged in, the user can read and write the data.
- Assume the client/user sends a select request, the data is loaded into the shared buffer from the datafiles.
- Shared buffer sent back the data to the client.
- Insert, Update, Delete can make changes to data are called transaction.
- Assume the client/user wants to make any changes to a record, given an update request, these changes are made in shared buffer.
- A copy of the transaction are recorded in wal buffer.
- Whenever client/user commit to change, the wal buffer will invoke the wall-writer, which will write all changes of the transaction to the wal files.
- The changes which are made in shared buffer, the actual data is mark as committed and it change. It doesn't get written to the datafiles right away.

PREPARED BY: SUBHAM DASH
DBA CONSULTANT

- Checkpointer process which get invoke in every 5 mins (default) it signal the writer to write all the dirty buffer where the changes have been made but not written to the data files to write it to the data files.
- Checkpointer also ensure that all files are in sync.
- Generally in system, we enable an additional process called archiver, it keeps ticking the wal files, if the wal files get full or switches between them archiver process copy the wal files to archive files.
- This will help during the time of recovery and provide additional level of security by copying the wal files.
- Clog (commit log buffer) all the transaction are committed are marked in this buffer. So in case of crash or server reboot, these clog buffer is referred to check all the committed transaction.
- Other buffer like work_memory buffer, maintenance work memory buffer and temp buffer.
- The job of the stats collector is to collect and report information about the server activity, then update this information in the database directory.
- So, database directory is updated about the latest happening in the server.
- The Auto-vacuum launcher when enabled it is the responsible of vacuum is to carry vacuum operation on fragmented tables.

  EXAMPLE: Assume we have a big table of 500 GB where we perform multiple updates, inserts, and delete query in that case table become fragmented.
- Auto-vacuum launcher goes ahead and does maintenance on that particular fragmented table.
- Logging collector which primary job is to ensure all logging & tracing information in the error log.

  EXAMPLE: We started a PostgreSQL instance, and it doesn't start. The first thing to check is the log files to see the error due to which PostgreSQL is not starting. It is the job of log collector to write all the error logs in the log file.




PREPARED BY: SUBHAM DASH
DBA CONSULTANT

## Explanations of Memory Structure

### Shared Buffer

Sets the amount of memory the database server uses for shared memory buffers. The default is typically (128MB) but might be less if your kernel settings will not support. This setting must be at least 128 kilobytes. However, settings significantly higher than the minimum are usually needed for good performance. This parameter can only be set at server start.

A reasonable starting value for shared buffers is 25% of the memory in your system. There are some workloads where even large settings for shared buffers are effective, but because PostgreSQL also relies on the operating system cache, it is unlikely that an allocation of more than 40% of RAM to shared buffers will work better than a smaller amount.

- User cannot access the datafile directly to read or write any data.
- Any select, insert, update or delete to the data is done via shared buffer area.
- The data that is written or modified in this location is called “Dirty data“.
- Dirty data is written to the data files located in physical disk through background writer process.
- Shared Buffers are controlled by parameter named: shared_buffer located in postgresql.conf file.

The purpose of Shared Buffer is to minimize DISK IO. For this purpose, the following principles must be met.

1. You need to access very large (tens, hundreds of gigabytes) buffers quickly.
2. You should minimize contention when many users access it at the same time.
3. Frequently used blocks must be in the buffer for as long as possible.

### Wall buffer

The amount of shared memory used for WAL data that has not yet been written to disk. The default setting of -1 selects a size equal to 1/32nd (about 3%) of shared buffers, but not less than 64kB nor more than the size of one WAL segment, typically 16MB. This value can be set manually if the automatic choice is too large or too small, but any positive value less than 32kB will be treated as 32kB. This parameter can only be set at server start. The contents of the WAL buffers are written out to disk at every transaction commit, so extremely large values are unlikely to provide a significant benefit. However, setting this value to at least a few megabytes can improve write performance on a busy server where many clients are committing at once. The auto-tuning selected by the default setting of -1 should give reasonable results in most cases.

- Write ahead logs buffer is also called “Transaction log Buffers”.
- WAL data is the metadata information about changes to the actual data and is sufficient to reconstruct actual data during database recovery operations.

- WAL data is written to a set of physical files in persistent location called "WAL segments" or "checkpoint segments".
- Wal buffers are flushed from the buffer area to wal segments by wal writer.
- Wal buffers memory allocation is controlled by the **Wal buffers** parameter

The WAL buffer is a buffer that temporarily stores changes to the database. The contents stored in the WAL buffer are written to the WAL file at a predetermined point in time. From a backup and recovery point of view, WAL buffers and WAL files are very important.

**Clog and other Buffers**

- CLOG stands for "commit log", and the CLOG buffers is an area in operating system RAM dedicated to hold commit log pages.
    - CLOG BUFFERS are one of the SLRU-style buffers oriented toward circular "rings" of data, like which transaction numbers have been committed or rolled back.
    - The commit logs have committed status of all transactions and indicate whether or not a transaction has been completed (committed).
- Work Memory is a memory reserved for either a single sort or hash table (Parameter: Work_mem)
    - Specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files.
    - The value defaults to four megabytes (4MB). Note that for a complex query, several sort or hash operations might be running in parallel each operation will be allowed to use as much memory as this value specifies before it starts to write data into temporary files.
    - Several running sessions could be doing such operations concurrently. Therefore, the total memory used could be many times the value of work_mem.
    - It is necessary to keep this fact in mind when choosing the value. Sort operations are used for ORDER BY, DISTINCT, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of IN subqueries.
- Maintenance Work Memory is allocated for Maintenance work (Parameter: maintenance_work_mem).
    - Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. It defaults to 64 megabytes (64MB).
    - Since only one of these operations can be executed at a time by a database session, and an installation normally doesn't have many of them running concurrently, it's safe to set this value significantly larger than work_mem.
    - Larger settings might improve performance for vacuuming and for restoring database dumps. Note that when autovacuum runs, up to

autovacuum_max_workers times this memory may be allocated, so be careful not to set the default value too high.

- It may be useful to control for this by separately setting autovacuum_work_mem.

➢ Temp Buffers are used for access to temporary tables in a user session during large sort and hash table. (Parameter: temp_buffers).

- Sets the maximum number of temporary buffers used by each database session.
- These are session-local buffers used only for access to temporary tables.
- The default is eight megabytes (8MB).
- The setting can be changed within individual sessions, but only before the first use of temporary tables within the session; subsequent attempts to change the value will have no effect on that session.
- A session will allocate temporary buffers as needed up to the limit given by temp_buffers.
- The cost of setting a large value in sessions that do not actually need many temporary buffers is only a buffer descriptor, or about 64 bytes, per increment in temp_buffers.
- However, if a buffer is actually used an additional 8192 bytes will be consumed for it (or in general, BLCKSZ bytes).

**Explanations of Utility (Background) Processes**

```
$ pstree -p 1125
postgres(1125) /usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
        ├─postgres(1249) postgres: logger process
        ├─postgres(1478) postgres: checkpointer process
        ├─postgres(1479) postgres: writer process
        ├─postgres(1480) postgres: wal writer process
        ├─postgres(1481) postgres: autovacuum launcher process
        ├─postgres(1482) postgres: archiver process
        ├─postgres(1483) postgres: stats collector process
```

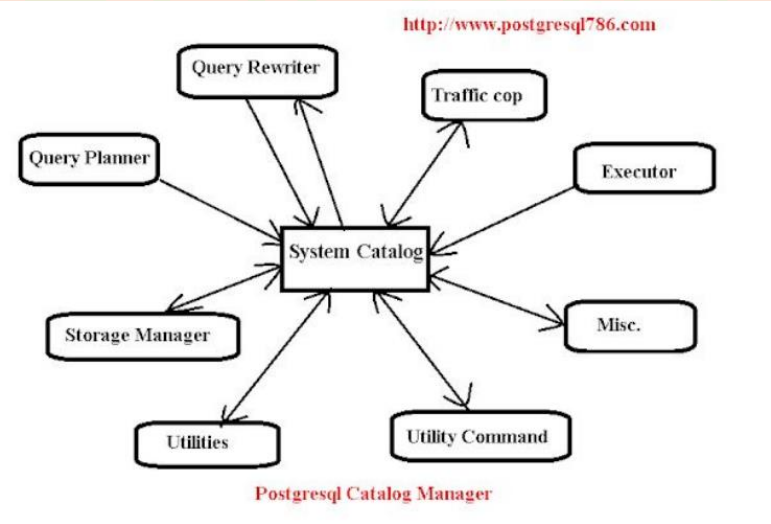
The list of background processes required for PostgreSQL operation are as follows.

| Process | Role |
|---|---|
| Logger | Write the error message to the log file. |
| checkpointer | When a checkpoint occurs, the dirty buffer is written to the file |
| Writer | Periodically writes the dirty buffer to a file. |
| wal writer | Write the WAL buffer to the WAL file |
| Autovacuum launcher | Fork autovacuum worker when autovacuum is enabled.It is the responsibility of the autovacuum daemon to carry vacuum operations on bloated tables on demand |
| archiver | When in Archive.log mode, copy the WAL file to the specified directory. |
| stats collector | DBMS usage statistics such as session execution information ( pg_stat_activity ) and table usage statistical information ( pg_stat_all_tables ) are collected. |

**BGWriter:** There is a separate server process called the background writer, whichse function is to issue writes of "dirty" (new or modified) shared buffers. It writes shared buffers so server processes handling user queries seldom or never need to wait for a write to occur. However, the background writer does cause a net overall increase in I/O load, because while a repeatedly-dirtied page might otherwise be written only once per checkpoint interval, the background writer might write it several times as it is dirtied in the same interval. The parameters discussed in this subsection can be used to tune the behavior for local needs.

**WallWriter:** WAL buffers are written out to disk at every transaction commit, so extremely large values are unlikely to provide a significant benefit. However, setting this value to at least a few megabytes can improve write performance on a busy server where many clients are committing at once. The auto-tuning selected by the default setting of -1 should give reasonable results in most cases. The delay between activity rounds for the WAL writer. In each round the writer will flush WAL to disk. It then sleeps for wal_writer_delay milliseconds, and repeats. The default value is 200 milliseconds (200ms). Note that on many systems, the effective resolution of sleep delays is 10 milliseconds; setting wal_writer_delay to a value that is not a multiple of 10 might have the same results as setting it to the next higher multiple of 10. This parameter can only be set in the postgresql.conf file or on the server command line.

**SysLogger:** Error Reporting and Logging




As per the figure, it is clearly understood that all – the utility processes + user backends + Postmaster Daemon are attached to syslogger process for logging the information about their activities. Every process information is logged under $PGDATA/pg_log with the file .log.

Debugging more on the process information will cause overhead on the server. Minimal tuning is always recommended. However, increasing the debug level when required. Click Here for further on logging parameters logging collector, which is a background process that captures log messages sent to stderr and redirects them into log files.

- log directory- data directory.
- log filename -The default is postgresql-%Y-%m-%d_%H%M%S.log.
- the default permissions are 0600

**Checkpointer:** When checkpoints occur, all the dirty pages must write to disk. If we increase the checkpoint_segments then checkpoint will occur less and so I/O will be less as it need to write less to disk. IF large amount of data is inserted there is more generation of checkpoints. Write-Ahead Logging (WAL) puts a checkpoint in the transaction log every so often.

- The CHECKPOINT command forces an immediate checkpoint when the command is issued, without waiting for a scheduled checkpoint. A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to disk. If executed during recovery, the CHECKPOINT command will force a restartpoint rather than writing a new checkpoint. Only superusers can call CHECKPOINT. The command is not intended for use during normal operation.

**Stats Collector:** PostgreSQL's statistics collector is a subsystem that supports collection and reporting of information about server activity. Presently, the collector can count accesses to tables and indexes in both disk-block and individual-row terms. It also tracks the total number of rows in each table, and information about vacuum and analyze actions for each table. It can also count calls to user-defined functions and the total time spent in each one.

- PostgreSQL also supports reporting of the exact command currently being executed by other server processes. This facility is independent of the collector process.
- The statistics collector transmits the collected information to other PostgreSQL processes through temporary files. These files are stored in the directory named by the stats_temp_directory parameter, pg_stat_tmp by default. For better performance, stats_temp_directory can be pointed at a RAM-based file system, decreasing physical I/O requirements. When the server shuts down cleanly, a permanent copy of the statistics data is stored in the pg_stat subdirectory, so that statistics can be retained across server restarts. When recovery is performed at server start (e.g. after immediate shutdown, server crash, and point-in-time recovery), all statistics counters are reset.

**Archiver:** Achiver process is optional process, default is OFF. Setting up the database in Archive mode means to capture the WAL data of each segment file once it is filled and save that data somewhere before the segment file is recycled for reuse.

- On Database Archivelog mode, once the WAL data is filled in the WAL Segment, that filled segment named file is created under PGDATA/pg_xlog/archive_status by the WAL Writer naming the file as ".ready". File naming will be "segment-filename.ready".
- Archiver Process triggers on finding the files which are in ".ready" state created by the WAL Writer process. Archiver process picks the 'segment-file_number' of .ready file and copies the file from $PGDATA/pg_xlog location to its concerned Archive destination given in 'archive_command' parameter(postgresql.conf).

- On successful completion of copy from source to destination, archiver process renames the "segment-filename.ready" to "segment-filename.done". This completes the archiving process.
- It is understood that, if any files named "segement-filename.ready" found in $PGDATA/pg_xlog/archive_status. They are the pending files still to be copied to Archive destination.

## Storage

- **data_directory:** Specifies the directory to use for data storage. This parameter can only be set at server start.
- **config_file:** Specifies the main server configuration file (customarily called postgresql.conf). This parameter can only be set on the postgres command line.
- **hba_file:** Specifies the configuration file for host-based authentication (customarily called pg_hba.conf). This parameter can only be set at server start.
- **ident_file:** Specifies the configuration file for Section 19.2 username mapping (customarily called pg_ident.conf). This parameter can only be set at server start.
- **external_pid_file:** Specifies the name of an additional process-ID (PID) file that the server should create for use by server administration programs. This parameter can only be set at server start.
- **PG_LOG:** It is not an actual postgres directory, it is the directory where RHEL stores the actual textual LOG.
- **PG_XLOG:** Here the write ahead logs are stored. It is the log file, where all the logs are stored of committed and un committed transaction. It contains max 6 logs, and last one overwrites. If archiver is on, it moves there.
- **PG_CLOG:** It contains the commit log files, used for recovery for instant crash.
- **PG_VERSION:** A file containing the major version number of PostgreSQL.
- **Base:** Subdirectory containing per-database subdirectories
- **Global:** Subdirectory containing cluster-wide tables, such as pg_database.
- **PG_MULTIXACT:** Subdirectory containing multitransaction status data (used for shared row locks)
- **PG_SUBTRANS:** Subdirectory containing subtransaction status data PG_TBLSPC: Subdirectory containing symbolic links to tablespaces.
- **PG_TWOPHASE:** Subdirectory containing state files for prepared transactions.
- **POSTMASTER.OPTS:** A file recording the command-line options the postmaster was last started with.
- **POSTMASTER.PID:** A lock file recording the current postmaster PID and shared memory segment ID (not present after postmaster shutdown)