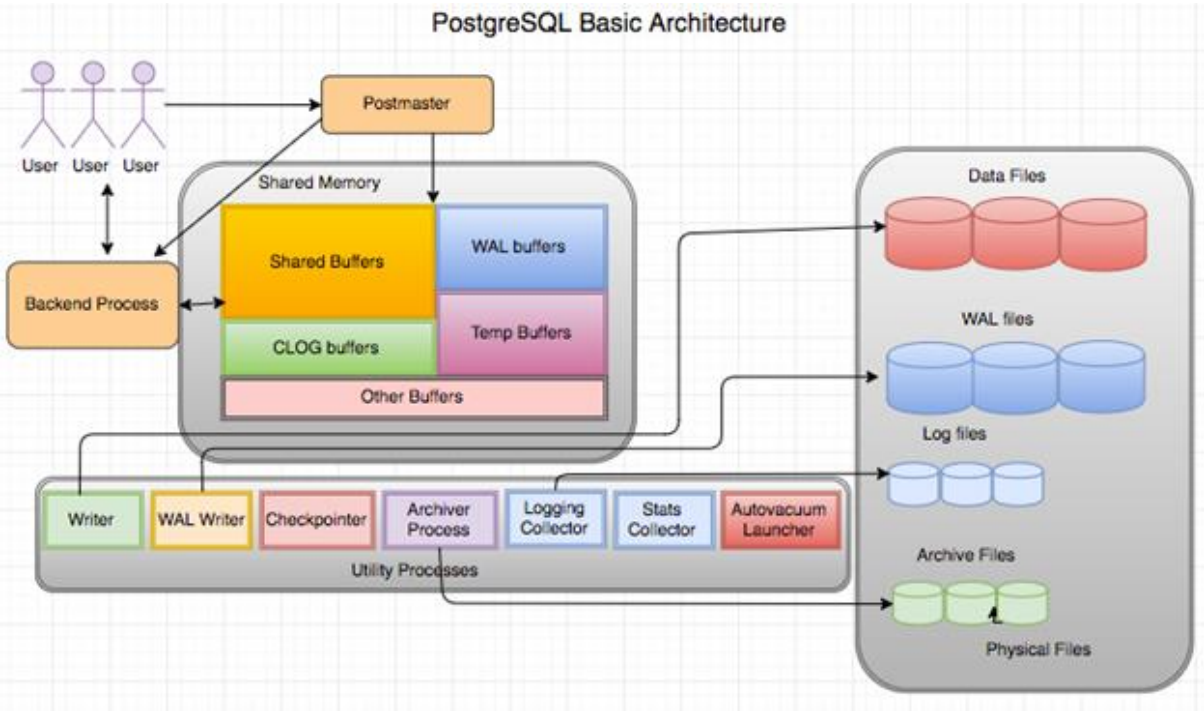
PostgreSQL Architecture

PostgreSQL is a powerful, open-source relational database management system (RDBMS) known for its robustness, extensibility, and standards compliance. Understanding its architecture is crucial for effective database management and optimization.



Below is an in-depth explanation of the key components and concepts of PostgreSQL architecture.

PostgreSQL Architecture Components

1. **PostgreSQL Instance (Server)**:
   - **Postmaster Process**: This is the main PostgreSQL server process responsible for initializing the database, handling connections, and starting other background processes. It listens for incoming connections and spawns new server processes for each connection.
2. **Database Cluster**:
   - A collection of databases managed by a single PostgreSQL server instance. A cluster contains several databases, and each database contains multiple schemas.
3. **Shared Memory**:

- **Shared Buffers**: This is the main memory area where PostgreSQL caches data blocks read from the disk. It improves performance by reducing disk I/O.
- **WAL Buffers**: Stores write-ahead log (WAL) entries before they are written to disk. WAL ensures data durability and crash recovery.
- **Work_mem**: Memory used for internal operations like sorting and hash tables during query execution.
- **Maintenance_work_mem**: Memory allocated for maintenance tasks like VACUUM, CREATE INDEX, and ANALYZE operations.

4. **Background Processes**:
- **Background Writer (bgwriter)**: Periodically writes dirty pages from the shared buffers to the disk to ensure the buffer cache has space for new pages.
- **WAL Writer (walwriter)**: Writes WAL data from WAL buffers to the disk.
- **Autovacuum Daemon**: Manages automatic vacuuming of the database to reclaim storage and maintain table statistics.
- **Stats Collector**: Collects and aggregates database activity and performance statistics.

5. **Process Architecture**:
- **Postmaster Process**: The parent process that initializes the database system and handles incoming connections.
- **Backend Processes**: Individual server processes spawned by the Postmaster to handle client connections. Each connection to the database has a corresponding backend process.
- **Background Worker Processes**: Custom or built-in background jobs that perform specific tasks, such as parallel queries.

6. **Storage**:
- **Data Directory**: The file system directory where PostgreSQL stores its database files, configuration files, and other metadata.
- **Data Files**: Files that store actual database tables and indexes.
- **WAL Files**: Logs of all changes made to the database. Used for crash recovery and replication.
- **Configuration Files**: Key configuration files include postgresql.conf (main configuration), pg_hba.conf (client authentication), and pg_ident.conf (user identity mapping).

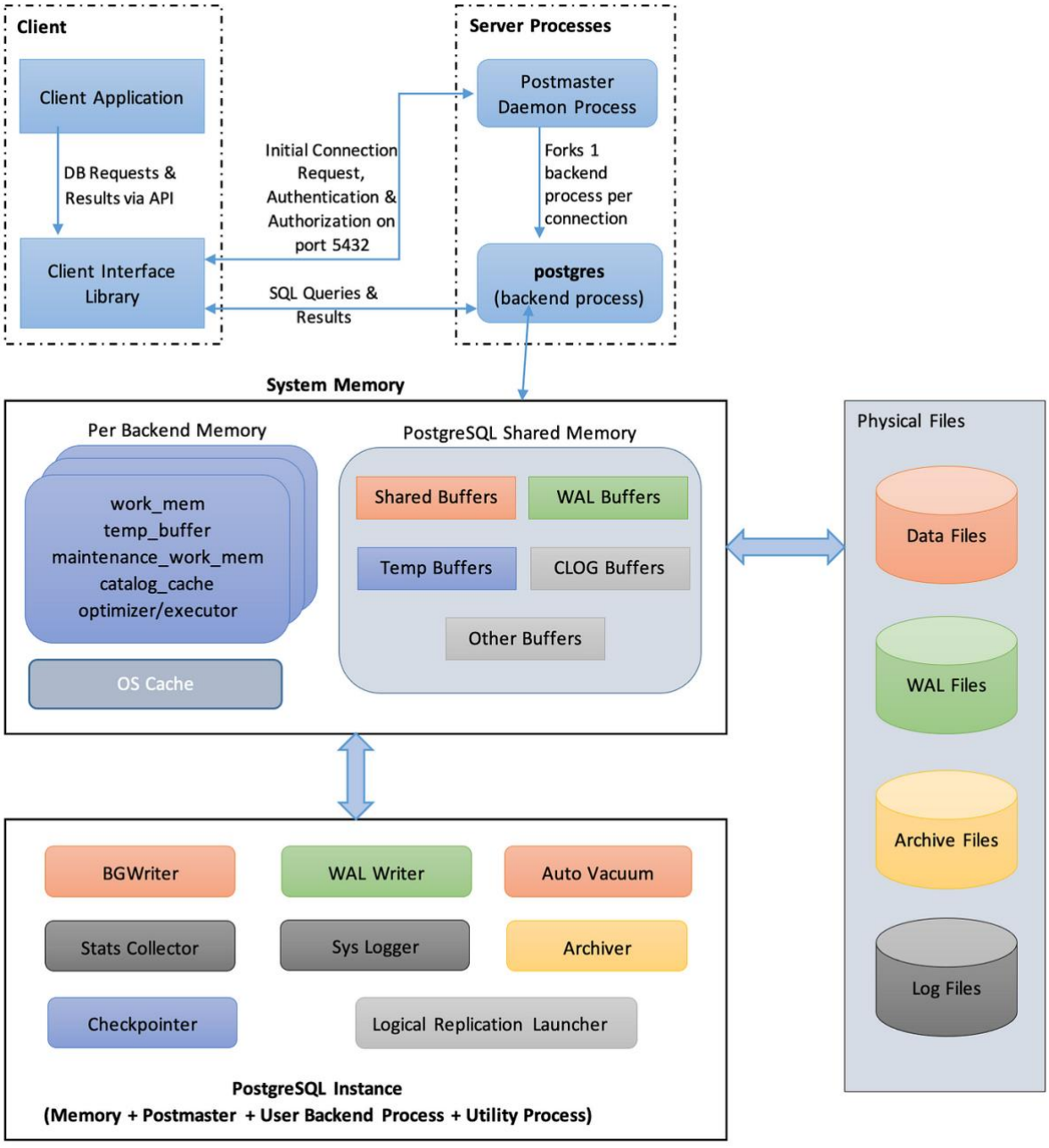
7. **Logical Storage Structures**:
- **Tablespaces**: Logical storage units that map to file system locations. They allow for the distribution of data across different storage locations.
- **Databases**: Logical collections of schemas, tables, indexes, functions, and other objects.
- **Schemas**: Namespace within a database containing tables, views, functions, and other objects. Schemas help organize objects logically.

8. **Concurrency Control**:
- **MVCC (Multi-Version Concurrency Control)**: Ensures data consistency and isolation by maintaining multiple versions of data rows. It allows concurrent

transactions to read and write without blocking each other, using a mechanism called snapshots.

9. **Indexing**:
   - Various indexing methods are available, including B-Tree, Hash, GiST, SP-GiST, GIN, and BRIN, which help optimize query performance.
10. **Replication and High Availability**:
    - **Streaming Replication**: Provides real-time data replication to standby servers for high availability and disaster recovery.
    - **Logical Replication**: Allows selective replication of data changes based on user-defined configurations.

PostgreSQL Process Flow

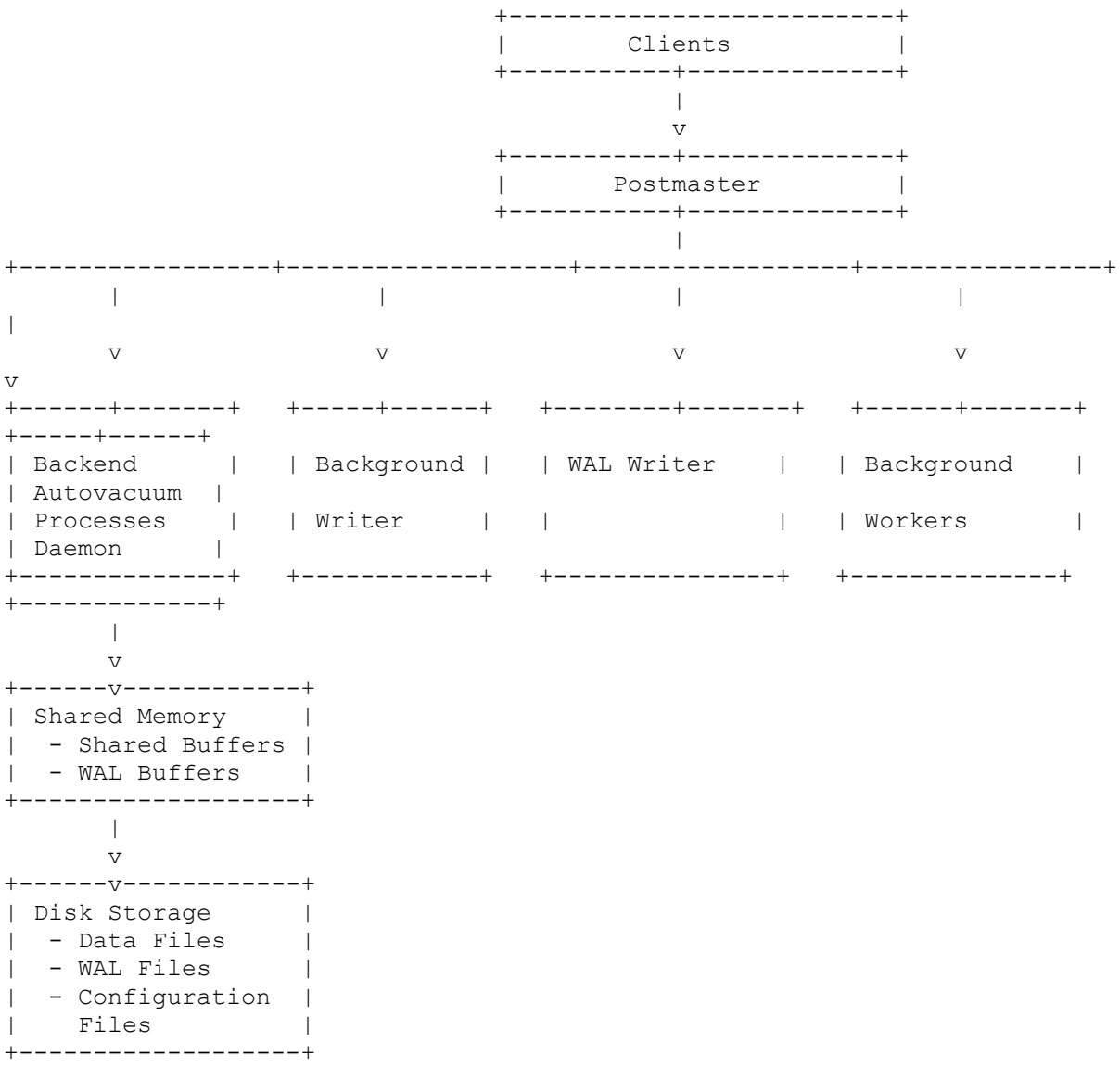


1. **Client Connection**: When a client connects to PostgreSQL, the Postmaster process accepts the connection and spawns a new backend process to handle it.
2. **Query Processing**:
   - **Parser**: Converts SQL queries into a parse tree.
   - **Planner/Optimizer**: Converts the parse tree into an execution plan, optimizing for performance.
   - **Executor**: Executes the plan by retrieving and manipulating data as needed.
3. **Buffer Management**: Data is fetched from the disk into the shared buffers. The buffer manager ensures that frequently accessed data is cached in memory.
4. **Transaction Management**: Ensures ACID properties (Atomicity, Consistency, Isolation, Durability) through MVCC and WAL.
5. **Background Tasks**: Processes like the background writer and autovacuum daemon ensure data integrity and optimal performance by managing memory and storage

Summary

PostgreSQL architecture is designed for reliability, performance, and extensibility. It uses a combination of shared memory and process-based management to handle client connections, perform query execution, and manage data storage. Its robust concurrency control, indexing methods, and replication mechanisms make it suitable for various applications, including large-scale enterprise systems

Diagram of PostgreSQL Architecture



PostgreSQL Architecture Key Components

Let's see each component of the PostgreSQL architecture in the following section.

1. Postmaster Supervisor Process of PostgreSQL

Postmaster works as the **Supervisor process** in the PostgreSQL architecture and it is the first process that gets started after PostgreSQL start. It works as the Listener and is responsible for authenticating and authorizing the incoming request from the client and assigning a new process called **Postgres** for each connection. Postmaster also keeps monitoring the process and starts if it is dead.

2. Shared Memory Segments of PostgreSQL

The Shared Memory Segments are the buffer cache in memory that is reserved for transactions and Maintenance activity. There are different Shared Memory segments allocated to perform a different operation. The following are the major shared memory segments.

2.1 Shared Buffer

The **Shared Buffer** is the memory area in the PostgreSQL instance that is used for any **insert, update, delete, or select operation** because users can't access the data files directly. The data that are modified or updated is called the **Dirty Data** and written to the physical data files through the background process called the **writer process**. The property of **Shared Buffer** is controlled by the **shared_buffer** parameter that is present in the **postgresql.conf** file.

2.2 Wal Buffer

The **Wal Buffer** also called the **Write ahead logs buffer** is the **Transaction log Buffers** which stores the metadata information of the changed data that is used to reconstruct the data during the database recovery operation. The **Transaction log Buffers** are written to the physical files **WAL Segments** or **Checkpoint Segments** by the background process called **Wal writer**. The property of **Wal Buffer** is managed by the **wal_buffers** parameter.

2.3 CLOG (Commit LOGs) Buffer

The PostgreSQL **CLOB** buffer is the **commit logs**, the area allocated in the main memory(RAM) to store the status of all transactions. It shows whether a transaction is completed or not. This buffer area is automatically managed by the database engine as there is no specific parameter for that. It is shared by all background servers and users in the PostgreSQL database.

2.4 Work Memory

The PostgreSQL **Work Memory** area is used when there is a **Sort** operation that includes Order By, Distinct, Merge join, and **hash table operation** that includes hash-join, hash-based aggregation, or the IN clause are involved in the database SQL query. The Work Memory is controlled by the work_mem parameter. In a complex SQL query, there could be multiple Sort and Hash operations and for each Sort and Hash operation, there is a memory allocated in RAM, hence it is suggested not to give a big value to this memory area otherwise it can exhaust all RAM space and will create an issue for other processes.

2.5 Maintenance Work Memory

The PostgreSQL **Maintenance Work Memory** area is used when the maintenance work is performed such as creating the Index, adding the index, adding the foreign key, and so on. It is controlled by the maintenance_work_mem parameter.

2.6 Temp Buffers

The PostgreSQL **Temp Buffers** area is used while accessing the temporary tables during the large sorting and hashing operations. These buffers are user session-specific.

3. Utility Background Processes of PostgreSQL

The **PostgreSQL Background processes** are important components of the PostgreSQL database. These processes are used to maintain the consistency between the memory and disk and due to this PostgreSQL database function properly. Each PostgreSQL background process has its role to play.

The following are the list of PostgreSQL background process.

- **Background Writer**
- **Checkpointer**
- **Autovacuum Launcher**
- **WAL Writer**
- **Statistics Collector**
- **Logging Collector**
- **Archiver**

Let's understand each PostgreSQL database background process in the following section.

3.1 Background Writer

The PostgreSQL **Background Writer** gets started by the **postmaster** when the PostgreSQL instance is started. The Background Writer is used to write the **Dirty Buffers** also called the **new or modified shared buffers** on the data files so that sufficient buffer space is available for use.

The PostgreSQL **Background Writer** follows the following three-parameter to write the dirty buffers from Shared Buffers to Data files.

- **bgwriter_delay (200ms by default, 10ms – 10s possible):-** This parameter is used to define the wait time of two successful execution.
- **bgwriter_lru_maxpages (100 pages by default, 0 – 1000 possible):-** This parameter is used to define the number of maximum buffers that can be written to data files in each iteration.
- **bgwriter_lru_multiplier (2.0 by default, 0-10.0 possible):-** This parameter is used to define the number of pages that will be cleaned for incoming dirty pages, and that is based on the count of the last delayed period. For example, if the value is set to 2 and the incoming pages are 10 so in this case, the dirty buffers will be cleaned unless there are 20 buffers that are not reaching.

3.2 Checkpointer

The **PostgreSQL Checkpoint** is an event that occurs at a specific time or manually by DBA to move Dirty Buffers(changed data or new data) from the memory(shared buffer) to disk(Data Files). The checkpoint is required in case of Crash recovery in which the latest checkpoint in the write-ahead log tells from which position the REDO recovery should start.

But how the changes are done in the database and what is the process? Basically for any DDL and DML statement, PostgreSQL requires the data to be present in the shared_buffers, If the data is not in shared buffers then PostgreSQL brings data from data files into shared buffers and then performs the DDL and DML operations. The modified blocks are called **Dirty Pages**. Once the commit command is submitted the detail about changes are written to the **Write ahead log** file on the disk and the **Dirty Pages** are written on the data files respectively.

The **PostgreSQL Checkpoint** triggers in the following condition.

- Manually by using the **CHECKPOINT** command.
- By setting the interval parameter **checkpoint_timeout**. The default value is 300 seconds.
- When the **Online Backup** starts the PostgreSQL Checkpoint triggers.
- When the function **pg_start_backup** executes a post that the PostgreSQL Checkpoint triggers.
- When the function **pg_basebackup** executes a post that the PostgreSQL Checkpoint triggers.
- When the WAL parameter **max_wal_size** reaches its maximum limit. The default value is 1 GB.
- When the **CREATE DATABASE / DROP DATABASE** commands are used to configure the database.

Once the **PostgreSQL Checkpoint** command triggers the following actions are performed.

- Check all **Dirty Pages** in the shared buffers.
- Write those **Dirty Pages** in the respective data files.
- Execute the **fsync()** function to record all up-to-date data on the disk.

3.3 Autovacuum Launcher

The **PostgreSQL Autovacuum Launcher** is the background process that is enabled by default and used for automating the execution of ANALYZE and the VACUUM commands. This process is enabled by default in PostgreSQL and runs on every autovacuum_naptime seconds if the autovacuum is set.

3.4 WAL Writer

The **PostgreSQL WAL Writer** background process is used to write the changed records from the **WAL Buffer** to the **WAL files** once the commit is issued.

3.5 Statistics Collector

The **PostgreSQL Statscollector** background is used to collect the stats about the server activity such as the number of records in the table, database details, index, and table access details, and report it to the optimizer dictionary(pg_catalog). This process is optional and by default, it is on. The stats information can be viewed by many views provided by PostgreSQL.

The below command shows all stats-related views in PostgreSQL.

```
postgres=# \d pg_stat_statements
```

The following parameters in the PostgreSQL **postgresql.conf** file defines the details that will be collected by the Stats collector.

- **track_activities:** This parameter monitors the ongoing command executed by any of the Server processes.
- **track_functions:** The UDF(User Defined Function) usage is tracked by this parameter.
- **track_counts:** The number of stats collected on tables and indexes is tracked by this parameter.
- **track_io_timing:** This parameter will track the number of reads and write blocks.

3.6 Logging Collector

The **PostgreSQL Logging Collector** background process is used to log the messages in the log file. It works when the following parameter value is set in the **postgresql.conf** configuration file.

```
log_destination = 'stderr'
logging_collector = ON
log_directory = 'pg_log'
```

3.7 Archiver

The **PostgreSQL Archiver** background process writes the WAL buffers to the WAL files when the database is in Archive.log mode.

4. Physical Files of PostgreSQL

The **PostgreSQL Physical Files** are used to store the actual data in the form of data files, the changed blocks in the WAL files, the server log details in the log files, the

Archive log information, and so on. The data in these files are stored permanently and used for their respective operation.

The following listed are the physical files in the PostgreSQL database.

- **Data Files**
- **Wal Files**
- **Log Files**
- **Archive Logs**

In the following section, we will see more detail about each PostgreSQL Physical file.

4.1 Data Files

The **PostgreSQL Data Files** are used to store the actual data. It stores the actual data and no such instruction or any kind of code information. When the user requests the data, PostgreSQL looks for the data in the shared buffer if it is not there then it loads data from the data files in the shared buffer and then processes further.

4.2 Wal Files

The **PostgreSQL WAL files** are used to store all changes from the WAL Buffer before the commit happens. WAL files are primarily used to maintain durability and consistency during a write operation on database storage.

4.3 Log Files

The **PostgreSQL Log files** store all logs related to the server, stderr, csvlog, Syslog, error message, warning message, informative message, and so on. It helps the database administrator to debug any issue in detail.

4.4 Archive Logs

The **PostgreSQL Archive Log** files are used to store the WAL segment on the disk. The Archive logs are used in case there is an unexpected crash that happens due to which the data is lost, in that scenario the Archive logs are used to repair/recover the database.