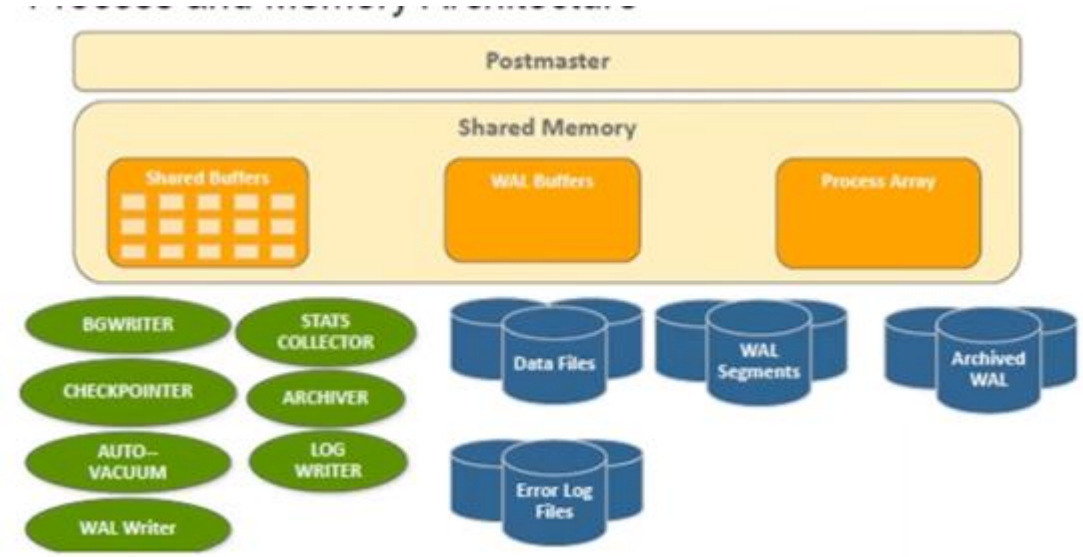
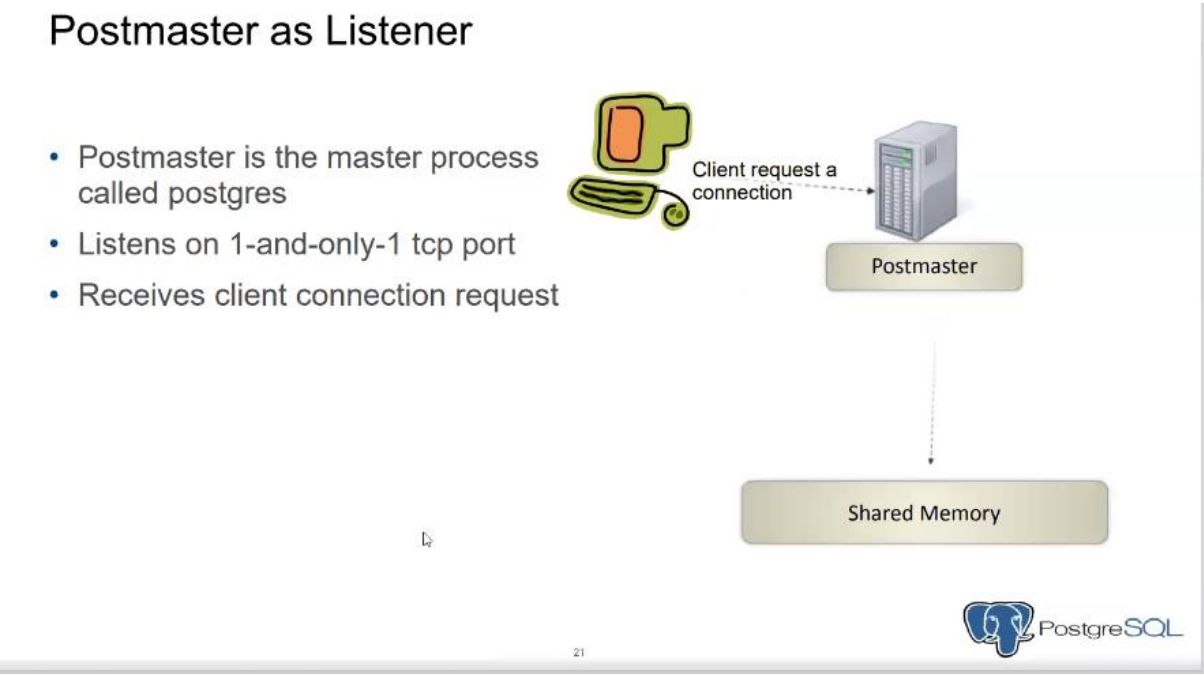
PostgreSQL Architecture



Postmaster

- The **Postmaster** is the main controller process that manages connections, initiates child processes, and oversees the overall database system.
- Postmaster process acts as the main server process and functions as the **listener** for client connections.



2. Shared Memory

PostgreSQL uses shared memory for caching and storing critical information. It includes:

- **Shared Buffers**: Cache frequently accessed data pages from disk.
- **WAL Buffers**: Cache for the Write-Ahead Log (WAL) entries, which ensures data integrity.
- **Process Array**: An array to track the various running processes.

Background Processes

1. **WAL Writer**: Writes changes from the Write-Ahead Logging (WAL) buffer to disk.
2. **Checkpointer**: Periodically saves dirty buffers to disk to ensure data persistence.
3. **Background Writer**: Flushes dirty pages to disk, reducing the load on checkpoints.
4. **Archiver**: Handles WAL file archiving for backups.
5. **Logger process:** is responsible for handling all logging operations. It collects and writes error messages, activity logs, and other notices generated by the database server to log files.
6. **Autovacuum**: Cleans up dead tuples and updates table statistics.

Control logging behaviour through various settings in the **`postgresql.conf`** file, such as:

- **`log_min_messages`**: Defines the severity level of messages logged.
- **`log_directory`** and `log_filename`: Specify where the logs are stored.

`wal level` parameter controls the amount of Write-Ahead Logging (WAL) data that is generated by the database server. Especially concerning replication and point-in-time recovery.

wal_level Settings

1. **minimal**:
   - Only enough WAL is generated to ensure the durability of transactions.
   - This level does not support replication or point-in-time recovery.
2. **replica** (default for most installations):
   - Generates additional WAL for replication.
   - This level allows for streaming replication and point-in-time recovery.
3. **logical**:
   - Includes all the information needed for logical replication.
   - This level allows changes to be captured in a format that can be sent to other systems (like logical decoding).

`max_wal_size`

- **Definition**: Maximum size that WAL files can grow before forcing a checkpoint.
- **Default**: Typically **1 GB**.
- **Purpose**: Controls disk space used by WAL; limits accumulation to improve performance and reduce I/O.

`min_wal_size`

- **Definition**: Minimum size for WAL files that PostgreSQL tries to maintain.
- **Default**: Typically **80 MB**.
- **Purpose**: Ensures sufficient WAL space is available to minimize overhead from creating/removing WAL files.

```
postgres=# show max_wal_size;
 max_wal_size
--------------
 1GB
(1 row)

postgres=# show min_wal_size;
 min_wal_size
--------------
 80MB
(1 row)
```

Checkpoint Trigger:

- When the size of the WAL files exceeds the `max_wal_size` limit, PostgreSQL will automatically trigger a checkpoint.
- The `checkpoint_timeout` being reached.

Logging configuration is managed through the `postgresql.conf` file,

Key Logging Parameters

1. **log_destination**: Defines where to send log messages.
   - **Options**: `stderr`, `csvlog`, `syslog`, or `eventlog` (on Windows).
     `log_destination = 'stderr'` → `Standard Errors`

2. **logging_collector**: If enabled, it captures log messages and directs them to log files.
   - **Default**: `off`
     `logging_collector = on`

3. **log_directory**: Directory where log files are stored.

- **Default**: Typically set to `pg_log` or similar.
  ```
  log_directory = 'pg_log'

  under PGDATA/pg_log
  ```

4. **log_filename**: The name format for log files.
   ```
   log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'

   postgres=# show log_filename;
      log_filename
   ------------------
    postgresql-%a.log
   (1 row)
   ```
   %a → day of the week Mon,tue,wed ….

```
Primary postgres@Primary log$ ls -lrt
total 68
-rw-------. 1 postgres postgres 58864 Oct 21 06:24 postgresql-Sun.log
-rw-------. 1 postgres postgres   869 Oct 21 22:32 postgresql-Mon.log
-rw-------. 1 postgres postgres  2775 Oct 22 21:59 postgresql-Tue.log
```

5. **log_statement**: Controls which SQL statements are logged.
   - **Options**: `none`, `ddl`, `mod`, `all`.
     ```
     log_statement = 'all'
     ```

6. **log_min_error_statement**:Minimum error level for statements to be logged.
   `log_min_error_statement = 'error' | 'Warning'` → it will write errors or warning to the logs

7. **log_line_prefix**: Prefix to add to each log line, useful for adding timestamps, user info, etc.
   ```
   log_line_prefix = '%m [%p]: [%l-1] user=%u,db=%d '
   ```

8. **log_rotation_age**: Determines how often log files are rotated based on age.
   ```
   log_rotation_age = 1d
   ```

9. **log_rotation_size**:Maximum size of log files before rotation occurs.
   ```
   log_rotation_size = 10MB
   ```

10. **log_checkpoints** provides valuable insight into the database's checkpointing behavior,
    Default Value: `off`

Connection options:

```
Connection options:
  -h, --host=HOSTNAME      database server host or socket directory (default: "local socket")
  -p, --port=PORT          database server port (default: "5432")
  -U, --username=USERNAME  database user name (default: "postgres")
  -w, --no-password        never prompt for password
  -W, --password           force password prompt (should happen automatically)
```

Autovacuum is a background process that manages dead tuples and reclaims storage space automatically. This process ensures that table statistics are up-to-date and prevents table bloat. Key configuration options include:

- **`autovacuum_vacuum_threshold`:** The minimum number of dead tuples before a vacuum runs.
- **`autovacuum_analyze_threshold`:** Sets the threshold for triggering an analyze operation.
- **`autovacuum_naptime`:** Controls the interval between autovacuum runs.

Autovacuum works with the **Free Space Map (FSM)** to track and reuse free space efficiently, reducing storage bloat and maintaining performance.