A **checkpoint** in PostgreSQL is a process that writes all recent changes (or "dirty" pages) from memory to disk to make sure your data is safe and can be quickly recovered after a crash. Below is both a simple explanation and a more detailed breakdown:

Simple Explanation

-

What It Is:
Think of a checkpoint as a "save game" in your database. It saves all the changes made so far, ensuring that the database has a recent, consistent state on disk.

-
-

Why It Matters:
In case of a crash or power failure, the database doesn't have to replay every single transaction from the beginning. Instead, it starts recovery from the last checkpoint, saving time and reducing the risk of data loss.

-

Detailed Explanation

1.

Purpose of Checkpoints:

2.
    -

Durability:
Checkpoints ensure that all committed transactions are safely stored on disk. This guarantees that even if the system crashes, the changes made are not lost.

    -
    -

Faster Recovery:
When PostgreSQL restarts after a crash, it only needs to replay the

WAL (Write-Ahead Log) entries that occurred after the last checkpoint, which speeds up recovery.

- ○

3.

How It Works:

4.

- ○

Dirty Pages:
As you work with PostgreSQL, changes (insertions, updates, deletions) are made in memory (buffer cache). These changes are called "dirty" pages because they have been modified but not yet written to disk.

- ○
- ○

Flushing Data:
At a checkpoint, PostgreSQL writes these dirty pages to disk. This process flushes the memory changes to ensure that the disk reflects a consistent state of the database.

- ○
- ○

Checkpoint Record in WAL:
PostgreSQL writes a special checkpoint record in the WAL. This record marks the point up to which all changes have been written to disk. During recovery, PostgreSQL can start from this record.

- ○

5.

Control Parameters:

6.

- ○

checkpoint_timeout:
Sets the maximum time between checkpoints.

- ○
- ○

checkpoint_completion_target:
Spreads out the disk I/O workload by trying to complete the checkpoint process over a percentage of the checkpoint interval.

-
-

max_wal_size / checkpoint_segments:
Controls how much WAL data can be generated before a checkpoint is forced, ensuring that the WAL doesn't grow too large.

-

7.

Benefits:

8.

-

Crash Recovery:
Reduces the recovery time by limiting the amount of WAL that must be replayed.

-
-

WAL File Management:
Helps PostgreSQL manage and eventually remove old WAL files that are no longer needed for recovery.

-

Conclusion

-

In Simple Terms:
A checkpoint is like hitting "save" in your database to ensure that recent changes are permanently stored on disk.

-
-

In Detailed Terms:
It flushes all modified memory pages to disk, writes a checkpoint record in the WAL, and reduces the amount of work needed during crash recovery.

Parameters like checkpoint_timeout and checkpoint_completion_target control how often checkpoints occur and how they impact system performance.

-

How Checkpoints Trigger WAL Writer & Disk Writer in PostgreSQL

A **checkpoint** acts as a key event in PostgreSQL that ensures data consistency by flushing in-memory changes to disk. It directly impacts both the **WAL writer** and **disk writer** processes. Here's how:

---

1. Checkpoint & WAL Writer Interaction

□ Checkpoint triggers WAL writer because:

- A checkpoint records the current state of the database by ensuring that all transaction logs (WAL) are persisted to disk.
- When a checkpoint starts, PostgreSQL forces WAL writer to **flush any unwritten WAL records** from memory (WAL buffers) to the WAL files on disk.
- The WAL writer ensures that the WAL contains a checkpoint record, marking the last safe state for recovery.

Flow of actions:

1. Checkpoint process starts.
2. WAL writer is triggered to flush all WAL buffers to disk.
3. A special **checkpoint WAL record** is written in the WAL file.
4. This ensures that if a crash happens, PostgreSQL knows where to start recovery.

Example: If a checkpoint is scheduled every 5 minutes (checkpoint_timeout = 300s), the WAL writer will be triggered at least once every 5 minutes to write WAL data to disk.

---

2. Checkpoint & Disk Writer Interaction

□ Checkpoint triggers disk writer because:

- The main job of a checkpoint is to **flush all dirty pages from memory (shared buffers) to disk** to ensure durability.
- The disk writer (Background Writer & Checkpoint Writer) writes these pages gradually to reduce I/O spikes.
- This prevents excessive WAL accumulation, as PostgreSQL can remove older WAL files after they are safely written to disk.

Flow of actions:

1. Checkpoint process starts.

2. Disk writer (Background Writer) is triggered to flush dirty pages to disk.
3. Once all dirty pages are written, PostgreSQL records a checkpoint in the WAL.
4. Old WAL files can now be removed safely if they are no longer needed.

Example: If checkpoint_completion_target = 0.7, PostgreSQL tries to **spread out disk writes over 70% of the time** between checkpoints to avoid sudden I/O spikes.

Overall Checkpoint Workflow

Checkpoint Starts

- Ensures data durability and marks a consistent state.

WAL Writer Flushes WAL Buffers

- Writes WAL records to disk.
- Ensures the checkpoint record is safely stored in WAL.

Disk Writer Flushes Dirty Pages

- Writes modified database pages from memory to disk.
- Ensures data is permanently saved.

Checkpoint Record is Written

- Signals that all changes are safely on disk.
- Allows PostgreSQL to clean up older WAL files.

Key Takeaways

✅**Checkpoint triggers WAL Writer** → Ensures WAL files are written before the checkpoint is completed.
✅**Checkpoint triggers Disk Writer** → Ensures all dirty pages are written to disk for data consistency.
✅**Optimizing Checkpoints** → Helps balance write performance and avoid high I/O loads.