Understanding Checkpoint, LSN, and WAL in PostgreSQL

1. Checkpoint in PostgreSQL

A checkpoint is a crucial event in PostgreSQL that ensures data consistency and reduces recovery time in case of a crash.
It writes all dirty (modified) pages from shared memory (buffer cache) to disk and marks a point in the WAL (Write-Ahead Log) where recovery can begin.

Key Features:
- Ensures Durability: Flushes dirty pages to disk so committed transactions are permanent.
- Speeds Up Crash Recovery: Recovery can start from the last checkpoint rather than replaying the entire WAL.
- Controlled via Parameters:
* checkpoint_timeout: Maximum time interval between checkpoints.
* checkpoint_completion_target: Spreads out writes over the interval.
* checkpoint_segments (pre-9.5) / max_wal_size (newer versions): Controls WAL size before triggering a checkpoint.

Checkpoint Process:
1. Flush modified pages to disk.
2. Write a checkpoint record in the WAL.
3. Remove old WAL files based on retention settings.

2. LSN (Log Sequence Number)

A Log Sequence Number (LSN) uniquely identifies each WAL record in PostgreSQL.
It points to an exact location in the WAL, aiding in tracking database changes.

Key Features:
- Used in Replication: Determines how far a replica is synchronized with the primary.
- Aids in WAL Archiving & PITR: Recovery begins from a known LSN.
- Improves Crash Recovery: Only replays WAL records starting from

a specific LSN.

LSN Format & Example:
- Represented in HEX, e.g., 00000002/000000D0
*** First part (00000002): WAL segment number.**
*** Second part (000000D0): Offset within the segment.**

Common Commands:
- Current WAL LSN: SELECT pg_current_wal_lsn();
- Last Checkpoint LSN: SELECT checkpoint_lsn FROM pg_control_checkpoint();

3. Relation Between LSN & WAL File

LSN is a pointer to a specific location within the WAL file. PostgreSQL stores WAL files in the pg_wal directory (or pg_xlog in older versions).
Each WAL file has a fixed size (usually 16MB) and a naming pattern like:
00000001000000020000003C
which consists of:
- Timeline ID
- WAL Segment Number (matching the first part of the LSN)
- WAL Offset

Each WAL file covers a range of LSNs. For example, if a file covers 16MB:
- First LSN: 00000002/00000000
- Last LSN: 00000002/01000000

To determine which WAL file contains a specific LSN:
SELECT pg_walfile_name('00000002/000000D0');
This maps the LSN to the corresponding WAL file.

4. Step-by-Step Example: Mapping LSN to WAL File

Step 1: Check the Current LSN
Command:
SELECT pg_current_wal_lsn();
Example Output:

```
0/16B4F78
```

Step 2: Find the WAL File Containing This LSN

Command:

```
SELECT pg_walfile_name('0/16B4F78');
```

Example Output:

```
000000010000000000000016
```

Step 3: Verify the WAL File Path

Command:

```
ls -lh $PGDATA/pg_wal/ | grep 000000010000000000000016
```

Example Output:

```
-rw------- 1 postgres postgres 16M Mar 2 10:30
000000010000000000000016
```

Step 4: Generate a New LSN by Inserting Data

Commands:

```
CREATE TABLE test_lsn (id SERIAL, name TEXT);
INSERT INTO test_lsn (name) VALUES ('LSN Example');
```

Then check:

```
SELECT pg_current_wal_lsn();
```

Example Output:

```
0/16B5A40
```

Step 5: Determine the WAL File for the New LSN

Command:

```
SELECT pg_walfile_name('0/16B5A40');
```

Example Output:

```
000000010000000000000016
```

(Indicating the same WAL file is still in use)

Step 6: Check the Last Checkpoint LSN

Command:

```
SELECT checkpoint_lsn FROM pg_control_checkpoint();
```

Example Output:

```
0/16B4A20
```

Step 7: LSN in Replication (On a Standby Server)

Command:

```
SELECT sent_lsn, write_lsn, flush_lsn, replay_lsn FROM
```

```
pg_stat_replication;
```

Example Output:

```
sent_lsn   | write_lsn  | flush_lsn  | replay_lsn
0/16B4F78  | 0/16B4F78 | 0/16B4F78 | 0/16B4F00
```

Conclusion:

- LSN points to a specific position within the WAL.
- pg_walfile_name() maps an LSN to its corresponding WAL file.
- WAL files store database changes sequentially.
- LSNs are essential for replication and recovery.