How to take a dump from a Wal file

♨️pg_walinspect♨️
vs
📍pg_waldump📍

DATABASE BOX
ORACLE/POSTGRESQL/MYSQL

View contents of WALs using pg_waldump:

1. First find if you have `pg_waldump` in your system.
-> which pg_waldump -> pg_waldump not found
It is cool, just check if you have **postgresql-contrib** installed or not
If yes, then maybe like me you do not have it in path, so give absolute path

```
-> /usr/lib/postgresql/<PG-Version>/bin/pg_waldump
```

2. Now first go to this directory /var/lib/postgresql/14/main/pg_wal/, you'll need root access

3. You'll see files like this "00000001000000000000000A", just pick any two subsequent files and run it

4. This will run it and give output in wal_dump.txt file:
Example 1: dump all data between two wal file:

```
$ ./pg_waldump -p /var/lib/postgresql/14/main/pg_wal/   00000001000000000000000A
00000001000000000000000B > ~/<Your-output-path>/wal_dump.txt
```

Example 2:
We want to see contents of a wal file from lsn1 to lsn2:

```
pg_waldump -s 76/7E000060 -e 76/7E000108 00000001000000760000007E

rmgr: Heap        len (rec/tot):     57/    57, tx:   59555584, lsn: 76/7E000060, prev 76/7E000028,
desc: INSERT+INIT off 1 flags 0x08, blkref #0: rel 1663/5/53434 blk 0
rmgr: Transaction len (rec/tot):     46/    46, tx:   59555584, lsn: 76/7E0000A0, prev 76/7E000060,
desc: COMMIT 2023-02-13 16:25:19.441483 EST
rmgr: Standby     len (rec/tot):     50/    50, tx:          0, lsn: 76/7E0000D0, prev 76/7E0000A0, desc:
RUNNING_XACTS nextXid 59555585 latestCompletedXid 59555584 oldestRunningXid 59555585
```

Example 3:

```
postgres$ /usr/lib/postgresql/14/bin/pg_waldump -p /var/lib/postgresql/11/main/pg_wal -s
0/353AF21C -e 0/353BE51C
```

View details of WALs using pg_walinspect:

low-level WAL inspection

The pg_walinspect module available from postgres 15 above, provides SQL functions that allow you to inspect the contents of write-ahead log of a running PostgreSQL database cluster at a low level, which is useful for debugging, analytical, reporting or educational purposes. It is similar to **pg_waldump**, but accessible through **SQL** rather than a separate utility.
All the functions of this module will provide the WAL information using the server's current timeline ID.

Note

The pg_walinspect functions are often called using an LSN argument that specifies the location at which a known WAL record of interest *begins*. However, some functions, such as **pg_logical_emit_message**, return the LSN *after*the record that was just inserted.

Tip

All of the pg_walinspect functions that show information about records that fall within a certain LSN range are permissive about accepting ***end_lsn*** arguments that are after the server's current LSN.

Using an ***end_lsn*** "from the future" will not raise an error.

It may be convenient to provide the value FFFFFFFF/FFFFFFFF (the maximum valid pg_lsnvalue) as an ***end_lsn*** argument.
This is equivalent to providing an ***end_lsn*** argument matching the server's current LSN.
By default, use of these functions is restricted to superusers and members of the pg_read_server_files role. Access may be granted by superusers to others using GRANT.

General Functions

pg_get_wal_record_info(in_lsn pg_lsn) returns record.

Gets WAL record information about a record that is located at or after the ***in_lsn*** argument. For example:

```
postgres=# SELECT * FROM pg_get_wal_record_info('0/E419E28');
+-------------------------------------
start_lsn        | 0/E419E28
end_lsn          | 0/E419E68
prev_lsn         | 0/E419D78
xid              | 0
resource_manager   | Heap2
record_type            | VACUUM
record_length          | 58
main_data_length     | 2
fpi_length               | 0
description   | nunused: 5, unused: [1, 2, 3, 4, 5]
block_ref      | blkref #0: rel 1663/16385/1249-fork main blk 364
```

If ***in_lsn*** isn't at the start of a WAL record, information about the next valid WAL record is shown instead.
If there is no next valid WAL record, the function raises an error.
pg_get_wal_records_info(start_lsn pg_lsn, end_lsn pg_lsn) returns setof record

Gets information of all the valid WAL records between ***start_lsn*** and ***end_lsn***. Returns one row per WAL record.

For example:

```
postgres=# SELECT * FROM pg_get_wal_records_info('0/1E913618', '0/1E913740') LIMIT 1;
-----------------------------------
start_lsn        | 0/1E913618
end_lsn          | 0/1E913650
```

```
prev_lsn         | 0/1E9135A0
xid              | 0
resource_manager | Standby
record_type      | RUNNING_XACTS
record_length    | 50
main_data_length | 24
fpi_length       | 0
description      | nextXid 33775-latestCompletedXid 33774 oldestRunningXid 33775
block_ref        |
```

The function raises an error if ***start_lsn*** is not available.

pg_get_wal_block_info(start_lsn pg_lsn, end_lsn pg_lsn, show_data boolean DEFAULT true) returns setof record

Gets information about each block reference from all the valid WAL records between ***start_lsn*** and ***end_lsn*** with one or more block references. Returns one row per block reference per WAL record.

The first step, is to install the extension, which does not differ from any other extension:

```
postgres=# CREATE EXTENSION pg_walinspect;
```

This extension allows us to examine the records between two valid WAL **Log Sequence Number** (LSN in short): ***start_lsn*** and ***end_lsn***.
Those LSN can be obtained through the ***pg_current_wal_lsn*** function

```
postgres=# SELECT pg_current_wal_lsn(),now();
 pg_current_wal_lsn |             now
+-------------------------------
0/1230278                    | 2024-01-12 15:51:21.532482-03
(1 row)
```

Now we doing some dml on database and get new LSN:'0/12302B8'

Now we can using this functions to investigate between 2 mentiond Lsn:

```
postgres=# SELECT * FROM pg_get_wal_block_info('0/1230278', '0/12302B8');
 ----------------------------------
start_lsn            | 0/1230278
end_lsn              | 0/12302B8
prev_lsn             | 0/122FD40
block_id             | 0
reltablespace        | 1663
reldatabase          | 1
relfilenode          | 2658
relforknumber        | 0
relblocknumber       | 11
xid                  | 341
resource_manager     | Btree
record_type          | INSERT_LEAF
```

```
record_length          | 64
main_data_length    | 2
block_data_length   | 16
block_fpi_length      | 0
block_fpi_info          |
description              | off: 46
block_data               | -\x00002a00070010402630000070696400
block_fpi_data    |
```

This example involves a WAL record that only contains one block reference, but many WAL records contain several block references. Rows output by pg_get_wal_block_info are guaranteed to have a unique combination of ***start_lsn*** and ***block_id*** values.

Much of the information shown here matches the output that pg_get_wal_records_infowould show, given the same arguments. However, pg_get_wal_block_info unnests the information from each WAL record into an expanded form by outputting one row per block reference, so certain details are tracked at the block reference level rather than at the whole-record level. This structure is useful with queries that track how individual blocks changed over time. Note that records with no block references (e.g., COMMIT WAL records) will have no rows returned, so pg_get_wal_block_info may actually return *fewer* rows than pg_get_wal_records_info.

The reltablespace, reldatabase, and relfilenode parameters reference pg_tablespace.oid, pg_database.oid, and pg_class.relfilenode respectively. The relforknumber field is the fork number within the relation for the block reference.

Tip

The **pg_filenode_relation** function can help you to determine which relation was modified during original execution.

It is possible for clients to avoid the overhead of materializing block data. This may make function execution significantly faster.
When ***show_data*** is set to false, block_data and block_fpi_data values are omitted (that is, the block_data and block_fpi_data OUT arguments are NULL for all rows returned).
Obviously, this optimization is only feasible with queries where block data isn't truly required.
The function raises an error if ***start_lsn*** is not available.

pg_get_wal_stats(start_lsn pg_lsn, end_lsn pg_lsn, per_record boolean DEFAULT false) returns setof record

Gets statistics of all the valid WAL records between ***start_lsn*** and ***end_lsn***. By default, it returns one row per ***resource_manager*** type. When ***per_record*** is set to true, it returns one row per ***record_type***.

For example:

```
postgres=# SELECT * FROM pg_get_wal_stats('0/1E847D00', '0/1E84F500')
 WHERE count > 0 AND
"resource_manager/record_type" ='Transaction'  LIMIT 1;
+-------------------
resource_manager/record_type | Transaction
count                                    | 2
```

```
count_percentage           | 8
record_size                | 875
record_size_percentage     | 41.23468426013195
fpi_size                   | 0
fpi_size_percentage        | 0
combined_size              | 875
combined_size_percent | 2.8634072910530795
```

The function raises an error if ***start_lsn*** is not available.

♨♨♨♨♨♨♨♨♨♨
Regards,

Alireza Kamrani,
Senior RDBMS Consultant.