? PostgreSQL Internal Data Flow (Read & Write Queries)

✅Overview of Key Components

Component	Role
Shared Buffers	In-memory cache for table/index data (like buffer pool)
WAL (Write Ahead Log)	Ensures durability – log first, write later (pg_wal directory)
Dirty Page	A page in shared buffer modified but **not** written to disk yet
Checkpointer	Background process that flushes dirty pages to disk
WAL Writer	Background process that writes WAL buffers to pg_wal directory
Archiver	Archives WAL files (if archive_mode = on)

? Step-by-Step Flow (INSERT/UPDATE/DELETE example):

Let's say you run:

UPDATE employees SET department = 'HR' WHERE emp_id = 1001;

? 1. Query Reaches PostgreSQL Backend

Sent by client → received by PostgreSQL → backend process handles it.

? 2. Buffer Lookup (Shared Buffers)

PostgreSQL calculates the **block (page)** number from emp_id = 1001.

Checks if that block is already in **shared_buffers**.

➤**If found: good! Work on the page in memory.**

➤**If not found: PostgreSQL reads from disk and puts that block into shared buffers.**

---

3. Modify the Page in Shared Buffer

The backend process updates the tuple inside the in-memory buffer page.

PostgreSQL uses **MVCC**: the old row is marked with an xmax and new row version is created with xmin.

The page is now **marked as "dirty"** (it has been modified but not yet flushed to disk).

---

4. WAL Generation

Before writing dirty pages to disk, **PostgreSQL generates a WAL record**:

Contains enough info to **redo** the change if the system crashes.

Stored in memory (WAL buffer – wal_buffers)

---

5. WAL Flushed to pg_wal Directory

The WAL writer process writes the WAL buffer to disk inside $PGDATA/pg_wal/ directory.

WAL File Format:

Default: 16 MB segments (000000010000000000000000A)

Files are sequentially numbered.

---

6. WAL Sync (Durability)

On COMMIT:

PostgreSQL **flushes WAL to disk (fsync)** to make it **durable** before acknowledging commit to the client.

Only **WAL** needs to be flushed – not the dirty page.

This is the core of **Write-Ahead Logging** – “log the change before applying it to disk.”

---

7. Dirty Page Stays in Shared Buffers

The updated page may stay in shared_buffers for seconds or minutes.

It is eventually flushed to disk by the **checkpointer**.

Checkpoint Process

PostgreSQL runs a **checkpoint** at intervals (controlled by checkpoint_timeout, max_wal_size, etc.).

During a checkpoint:

All dirty pages are **written to disk (heap/index)**.

A special **checkpoint WAL record** is written.

After checkpoint, you can **truncate or archive** WAL files safely.

Archiving WAL

If you have:

```
archive_mode = on
archive_command = 'cp %p /archive/%f'
```

Then:

1.After a WAL file is **filled & no longer needed for crash recovery**, PostgreSQL:

Calls archive_command.

Copies WAL file from pg_wal/ to your archive directory.

Archived WALs are used for:

Point-In-Time Recovery (PITR)

Streaming replication

Summary – What Happens Internally?

Step	Where it happens
Query parsed, planned	PostgreSQL backend
Data page fetched	Shared Buffers (or loaded from disk)
Page updated	Shared Buffers (now dirty)
WAL generated	WAL buffer (in memory)
WAL written to disk	pg_wal/ directory
On commit, WAL flushed	fsync (guaranteed durability)
Checkpoint flushes pages	Dirty pages written to heap files
WAL archived (if enabled)	archive_command copies to archive dir

---

? Extra Tips

pg_stat_bgwriter helps monitor checkpoints and dirty page activity.

WAL files are crucial for **recovery** and **replication**.

PostgreSQL never overwrites existing rows – always creates a new version.

Pages in shared_buffers can be reused (LRU policy) if memory is full.

WAL Archiving is **separate** from WAL streaming (used in streaming replication).