**Shared Buffe and Shared Table**

**1. Shared Buffers**

**shared\_buffers** is a PostgreSQL **configuration parameter** that determines the amount of **memory allocated to cache data pages** (tables, indexes) in memory, reducing disk I/O.

**Key Points:**

* It is a portion of RAM reserved inside PostgreSQL.
* Frequently accessed data is stored here for faster access.
* It improves performance by avoiding frequent disk reads.

**Example:**

If shared\_buffers = 2GB, then 2 GB of RAM is used for caching table and index pages.

**Analogy:**

Think of shared\_buffers as PostgreSQL's internal memory "workspace" for holding data temporarily.

**2. Shared Table**

A **shared table** in PostgreSQL refers to a **table that is visible across all databases** in the same cluster. These are **rare and system-level** tables.

**Key Points:**

* Most tables belong to a single database.
* Shared tables are used for global PostgreSQL objects like roles, tablespaces.
* Only certain **catalog tables** are shared (e.g., pg\_authid, pg\_tablespace).

**Example:**

* pg\_database: visible only inside a database
* pg\_authid: shared table, same across all databases

You can check if a table is shared by querying:

SELECT relname FROM pg\_class WHERE relisshared = true;

**Summary Comparison**

| **Feature** | **Shared Buffers** | **Shared Table** |
| --- | --- | --- |
| Type | Memory (configuration) | Catalog table |
| Purpose | Caches data for faster access | Stores data used by all databases |
| Scope | Applies at the server level | Exists cluster-wide |
| Controlled by | shared\_buffers setting in postgresql.conf | Internal system catalog design |
| Example | Buffering rows from a user table | pg\_authid, pg\_tablespace |

**In Short:**

* **Shared Buffers**: A memory cache to improve performance.
* **Shared Table**: A system table shared across all databases in a PostgreSQL cluster.

**Use case:**

**1. Shared Buffers**

**Purpose:**

shared\_buffers is a chunk of memory used to cache table and index pages in RAM. It improves performance by reducing the number of disk reads.

**Use Cases:**

**a. Query Performance Optimization**

* When the same table or index is accessed repeatedly, shared buffers prevent repeated disk I/O by storing pages in memory.
* Frequently queried data (like lookup tables or hot rows) benefits the most.

**b. Bulk Reads and Writes**

* During large operations (like sequential scans, joins, or batch inserts), shared buffers can hold more data temporarily to minimize disk access.

**c. Write-Ahead Logging Support**

* Modified pages are first written to shared buffers, then flushed to disk based on checkpoint or background writer behavior.

**d. Concurrency Handling**

* Multiple sessions can read from or write to shared buffers, allowing concurrent access to cached data safely via locking mechanisms.

**e. Buffer Eviction Tuning**

* PostgreSQL maintains an LRU (least recently used) style buffer eviction policy. Tuning shared\_buffers helps control how long data stays in memory.

**2. Shared Tables**

**Purpose:**

Shared tables are system catalog tables that are visible and accessible across **all databases** in a PostgreSQL cluster.

**Use Cases:**

**a. Global Metadata Management**

* Store data that is common to all databases, such as:
  + User roles (pg\_authid)
  + Tablespaces (pg\_tablespace)
  + Replication slots and configurations

**b. Cluster-wide User Management**

* Roles created in PostgreSQL (with CREATE ROLE) are stored in shared tables like pg\_authid. These roles are accessible across all databases, enabling consistent permissions and authentication.

**c. Cluster Management Tools**

* Tools and admin queries that need to inspect system-wide state use shared tables (e.g., monitoring roles or listing all tablespaces).

**d. System Bootstrapping**

* During cluster initialization and startup, PostgreSQL needs to access minimal critical metadata (e.g., login roles) that must be available even before a specific database is selected.

**TUNIG:**

**1. Tuning Parameters for shared\_buffers**

shared\_buffers is one of the most important PostgreSQL performance parameters. It defines how much memory PostgreSQL uses for caching data.

**Key Parameter:**

shared\_buffers = 128MB # default;

should be increased in production

**Tuning Guidelines:**

| **System RAM** | **Recommended shared\_buffers** |
| --- | --- |
| < 2 GB | 20–25% of RAM |
| 2–16 GB | 25–40% of RAM |
| > 16 GB | 25% of RAM (start there, test further) |

Example: If your server has 16 GB of RAM, start with shared\_buffers = 4GB.

**Related Parameters:**

| **Parameter** | **Description** |
| --- | --- |
| work\_mem | Per-operation memory for sorts, joins (not directly related but affects performance alongside shared\_buffers) |
| effective\_cache\_size | Estimate of OS file system cache, helps planner make better decisions |
| maintenance\_work\_mem | Memory for maintenance operations (e.g., vacuum, create index) |
| bgwriter\_lru\_maxpages | Controls how many buffers the background writer flushes per cycle |
| checkpoint\_completion\_target | Affects how aggressively pages are written to disk before checkpoint |

**Check Memory Usage:**

Use this SQL to see how shared buffers are being used:

SELECT count(\*) FROM pg\_buffercache;

You may need to install the pg\_buffercache extension.

**2. Tuning for Shared Tables**

Shared tables are internal system catalog tables used by PostgreSQL. **They are not directly tunable** like user tables, but their performance and behavior are affected by broader system settings.

**Indirect Tuning Areas:**

| **Parameter** | **Effect on Shared Tables** |
| --- | --- |
| max\_connections | More connections may increase access to shared catalogs like pg\_authid |
| shared\_preload\_libraries | Extensions like pg\_stat\_statements can increase access to shared catalogs |
| wal\_level = replica | Required for features like replication, which involves shared catalog reads |
| autovacuum settings | Ensure system catalogs are vacuumed regularly |
| stats\_temp\_directory | Temp location for shared catalog stats collection |