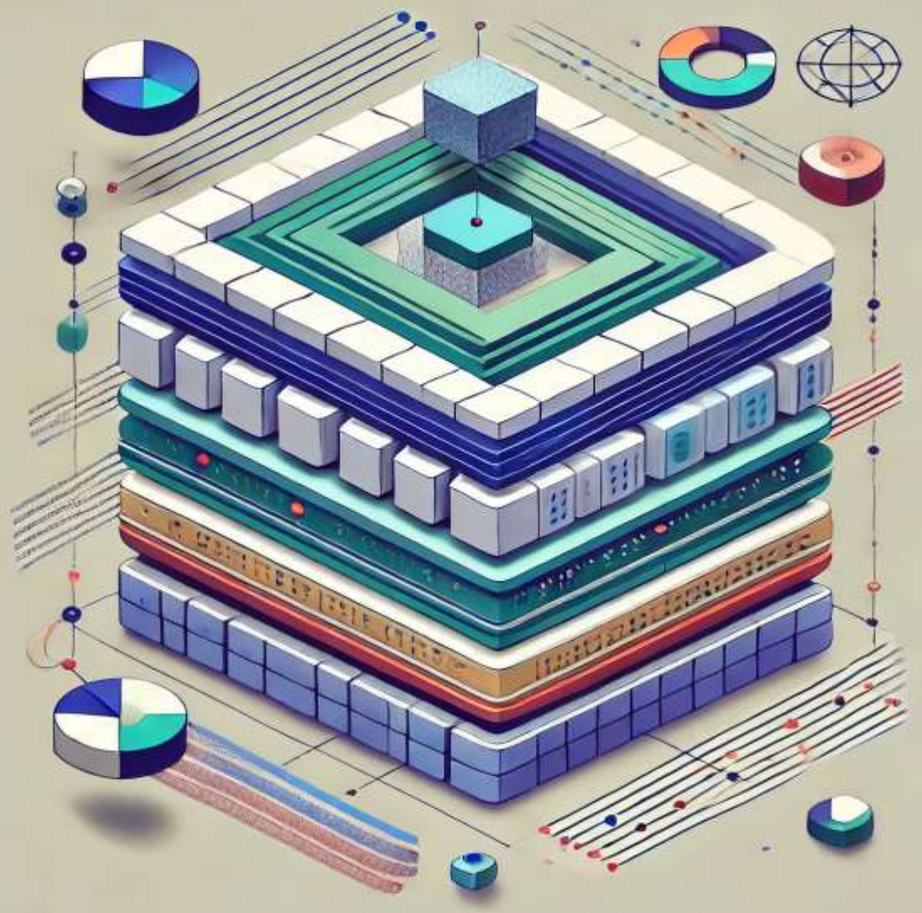


PostgreSQL: Page Buffer Overview

This presentation provides an in-depth look into PostgreSQL pages. It explores the buffer cache and the clock-sweep buffer eviction algorithm. Understand how PostgreSQL manages data efficiently.

Agenda: Understanding PostgreSQL Pages

1 Buffer Cache

Learn about the buffer cache. Understand its role in reducing I/O delays.

2 Page Structure

Delve into the structure of PostgreSQL pages. Explore how data is organized for optimal access.

3 Buffer Usage

View buffer usage during query execution

4 Buffer Management

Examine the clock-sweep algorithm. Discover how PostgreSQL manages memory.

Page Buffer

Shared Memory Area

It stores data from physical files into RAM. This avoids I/O delays.

Part of Shared Buffer Pool

Accessible to all PostgreSQL processes. Ensures efficient data sharing.

Cache Lookup

PostgreSQL checks buffer cache for data. Returns data from cache if available.

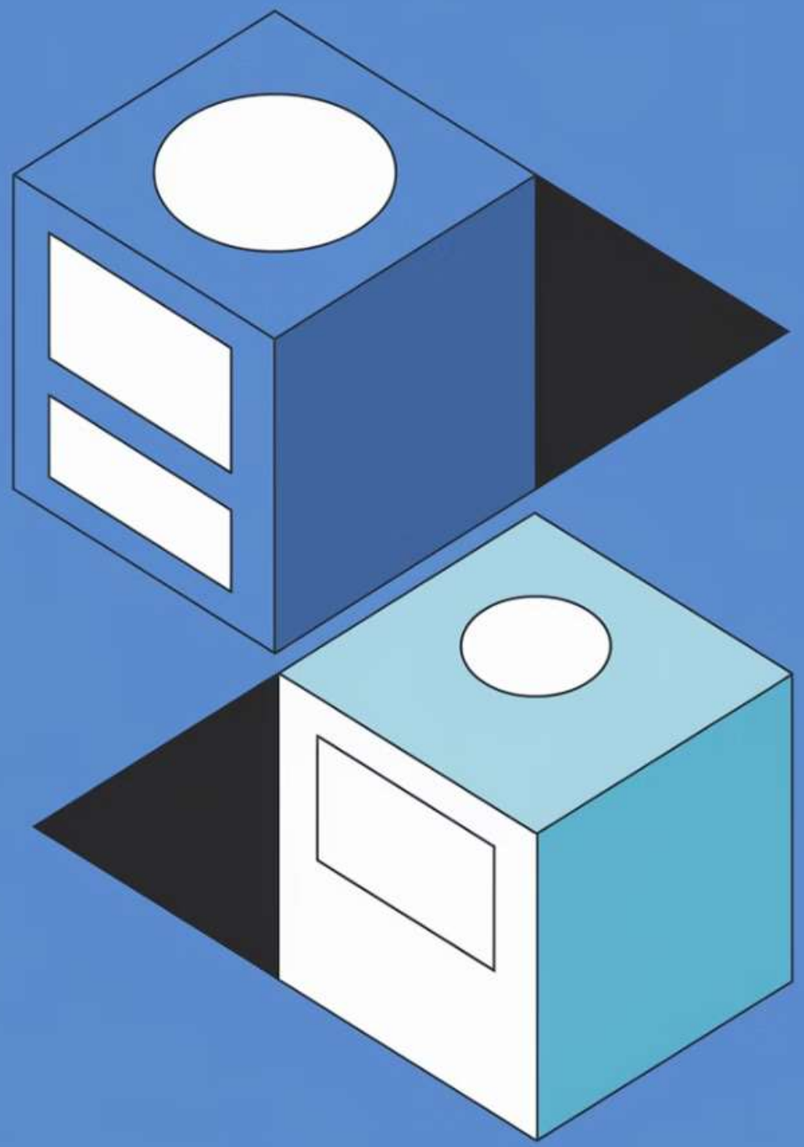
Configuration

Controlled by shared_buffers parameter (default: 128 MB).

Initialization

InitBufTable in backend/storage/buffer/freelist.c

The page buffer is crucial for performance. It reduces the need to read from disk.



Page Buffer & OS Cache

Double Buffering

Data is cached in both PostgreSQL and OS Cache. This is default behaviour.

Direct I/O

It eliminates double buffering. Requires architectural changes for support.

Portability

Current setup prioritizes portability. Ensures compatibility across systems.

Understanding the differences is key. It helps optimize performance.
Asynchronous I/O (AIO) was attempted but not implemented due to portability issues.

Understanding effective_cache_size

1

Query Cost

Used to determine query cost. Assumes total cache size for the query.

2

No Memory Allocation

No effect on shared memory size. Does not change allocated memory.

3

Index Scans

Determines index vs. sequential scans. Affects query execution strategy.

This parameter is essential for planning queries. It does not allocate memory

Page Size and Data Storage

1

Page Size

Compile-time parameter. Default size is 8 KB.

2

Data Storage

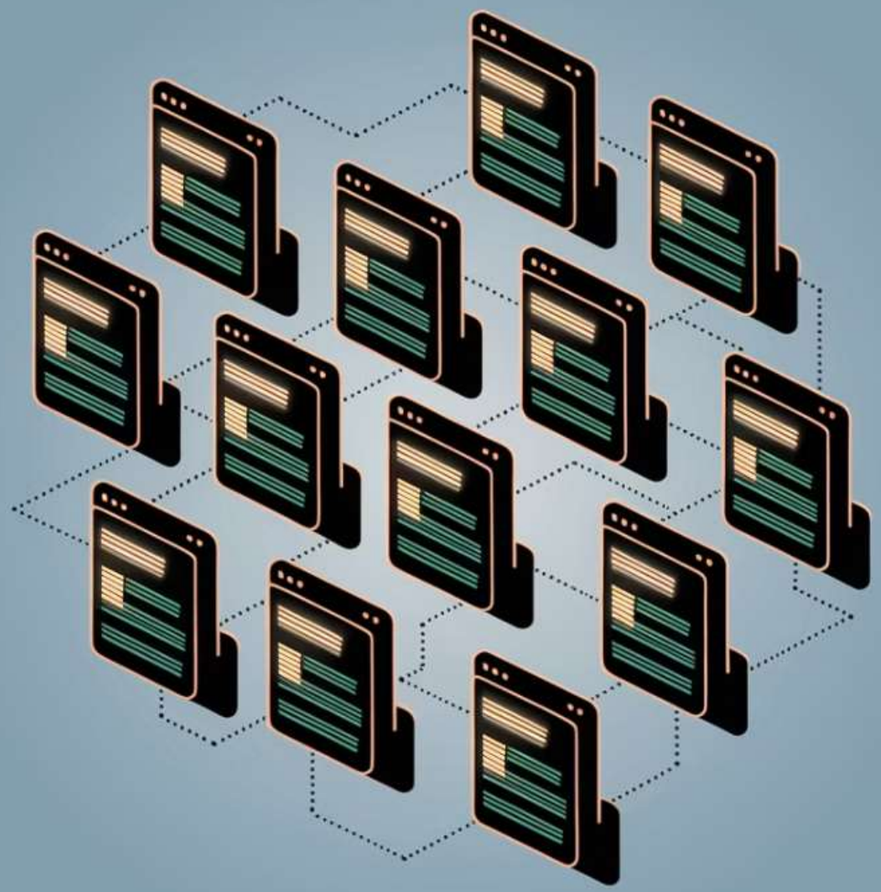
Table data and indexes are stored. Stored across multiple pages.

3

Data Retrieval

Indexes locate relevant pages. Avoids full table scans.

Efficient data management is critical. It ensures fast access.



Agenda: Understanding PostgreSQL Pages

1 **Buffer Cache**

Learn about the buffer cache. Understand its role in reducing I/O delays.

2 **Page Structure**

Delve into the structure of PostgreSQL pages. Explore how data is organized for optimal access.

3 **Buffer Usage**

View buffer usage during query execution

4 **Buffer Management**

Examine the clock-sweep algorithm. Discover how PostgreSQL manages memory.

Page Structure

Header Data

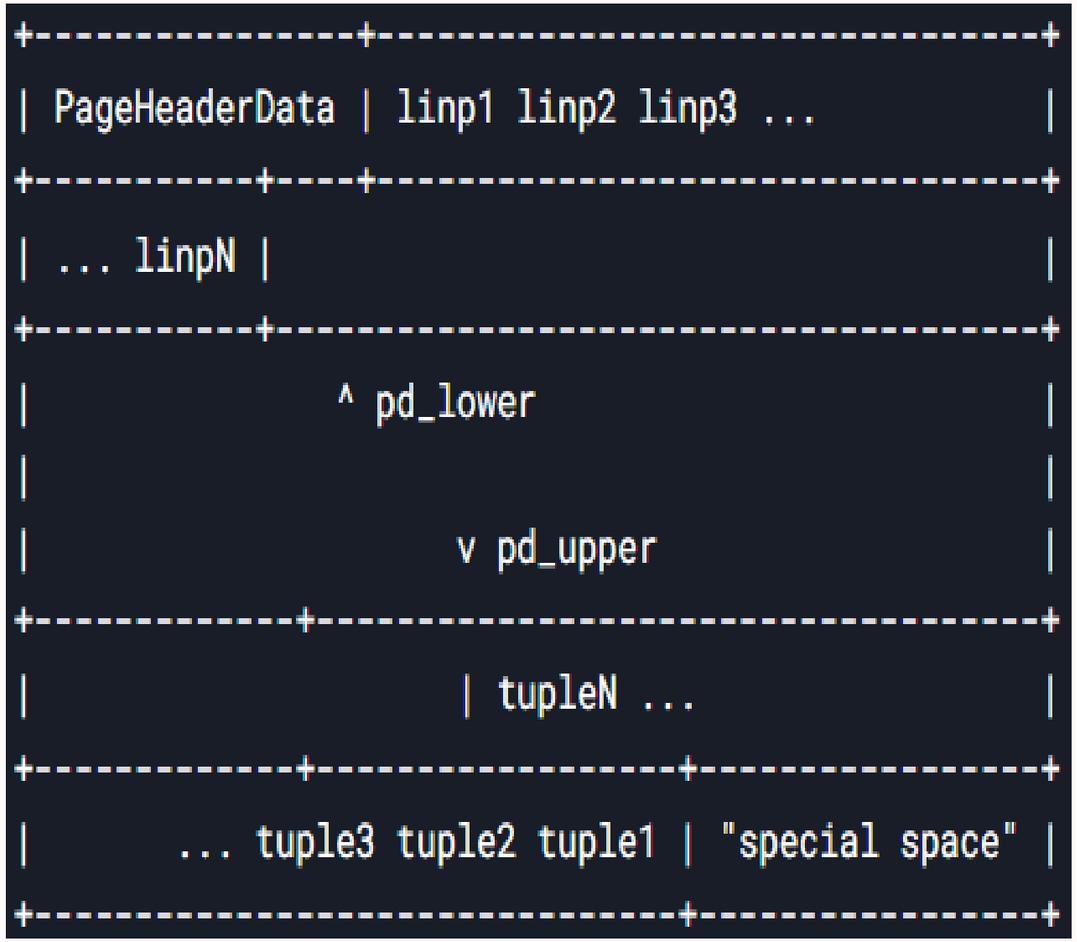
24 bytes long. Contains meta data about the page, including free space pointers.

Data Storage

ItemIds, followed by free space and Items

Special Space

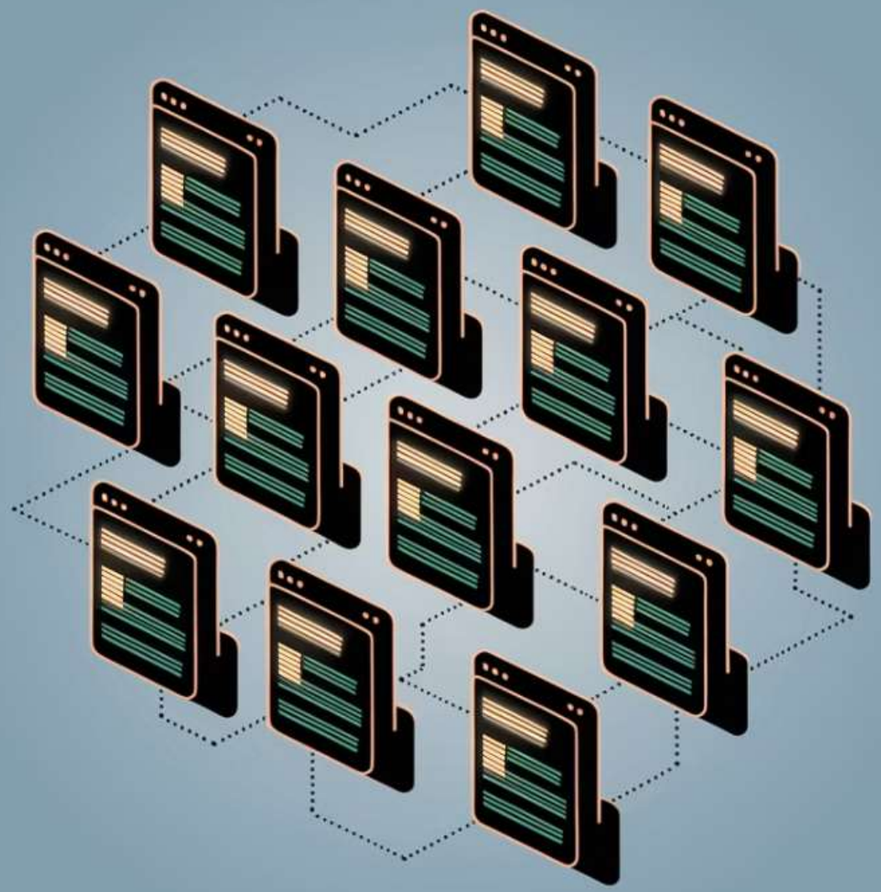
Index access method specific data. Empty in ordinary tables (heap pages).



Understanding the page layout helps in debugging. It optimizes data retrieval.

PageHeaderData

Field	Type	Length	Description
pd_lsn	PageXLogRecPtr	8 Bytes	Log Sequence Number(LSN): next byte after last byte of WAL record for last change to this page
pd_checksum	uint16	2 Bytes	Page checksum
pd_flags	uint16	2 Bytes	Flag bits
pd_lower	LocationIndex	2 Bytes	Offset to start of free space
pd_upper	LocationIndex	2 Bytes	Offset to end of free space
pd_special	LocationIndex	2 Bytes	Offset to start of special space
pd_pagesize_version	uint16	2 Bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 Bytes	Oldest unpruned XMAX on page, or zero if none



Agenda: Understanding PostgreSQL Pages

1 **Buffer Cache**

Learn about the buffer cache. Understand its role in reducing I/O delays.

2 **Page Structure**

Delve into the structure of PostgreSQL pages. Explore how data is organized for optimal access.

3 **Buffer Usage**

View buffer usage during query execution

4 **Buffer Management**

Examine the clock-sweep algorithm. Discover how PostgreSQL manages memory.

Buffers after… DB Startup

```
psql (17.4)
Type "help" for help.

omdb=# show shared_buffers;
 shared_buffers
----------------
 128MB
(1 row)

omdb=# SELECT relpages FROM pg_class WHERE relname = 'casts';
 relpages
----------
    10632
(1 row)

omdb=# select count(*) from pg_buffercache;
 count
-------
 16384
(1 row)

omdb=# SELECT n.nspname, c.relname, count(*) AS buffers
FROM pg_buffercache b JOIN pg_class c
ON b.relfilenode = pg_relation_filenode(c.oid) AND
b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname = current_database()))
JOIN pg_namespace n ON n.oid = c.relnamespace
where n.nspname = 'public' GROUP BY n.nspname, c.relname
ORDER BY n.nspname,3 DESC;
 nspname | relname | buffers
---------+---------+---------
(0 rows)
```

This shows the buffer state after database startup. The buffers are initialized but not allocated for the table ‘casts’.

Buffers after... Query Execution Index scan

```
omdb=# explain (analyze,buffers,verbose,memory) select * from casts where movie_id = 11;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using casts_pkey on public.casts  (cost=0.43..8.71 rows=16 width=36) (actual time=1.808..3.842 rows=35 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Index Cond: (casts.movie_id = 11)
   Heap Fetches: 35
   Buffers: shared hit=4 read=14
 Planning:
   Buffers: shared hit=81 read=7
   Memory: used=13kB  allocated=32kB
 Planning Time: 1.892 ms
 Execution Time: 3.859 ms
(10 rows)

omdb=# SELECT n.nspname, c.relname, count(*) AS buffers
FROM pg_buffercache b JOIN pg_class c
ON b.relfilenode = pg_relation_filenode(c.oid) AND
b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname = current_database()))
JOIN pg_namespace n ON n.oid = c.relnamespace
where n.nspname = 'public' GROUP BY n.nspname, c.relname
ORDER BY n.nspname,3 DESC;
 nspname |  relname   | buffers
---------+------------+---------
 public  | casts      |      10
 public  | casts_pkey |       5
(2 rows)
```

This image shows the buffers after an index scan. Relevant pages are loaded into the buffer cache.

Buffers after... Query

```
omdb=# explain (analyze,buffers,verbose,memory) select * from casts where movie_id = 11;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using casts_pkey on public.casts  (cost=0.43..8.71 rows=16 width=36) (actual time=0.052..0.092 rows=35 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Index Cond: (casts.movie_id = 11)
   Heap Fetches: 35
   Buffers: shared hit=15
 Planning:
   Memory: used=13kB  allocated=16kB
 Planning Time: 0.186 ms
 Execution Time: 0.252 ms
(9 rows)

omdb=# explain (analyze,buffers,verbose,memory) select * from casts where movie_id = 11;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using casts_pkey on public.casts  (cost=0.43..8.71 rows=16 width=36) (actual time=0.027..0.046 rows=35 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Index Cond: (casts.movie_id = 11)
   Heap Fetches: 35
   Buffers: shared hit=15
 Planning:
   Memory: used=13kB  allocated=16kB
 Planning Time: 0.080 ms
 Execution Time: 0.064 ms
(9 rows)
```

Buffers are populated with data required by the query. This reduces disk I/O for subsequent access.

Buffers after… Full Table Scan

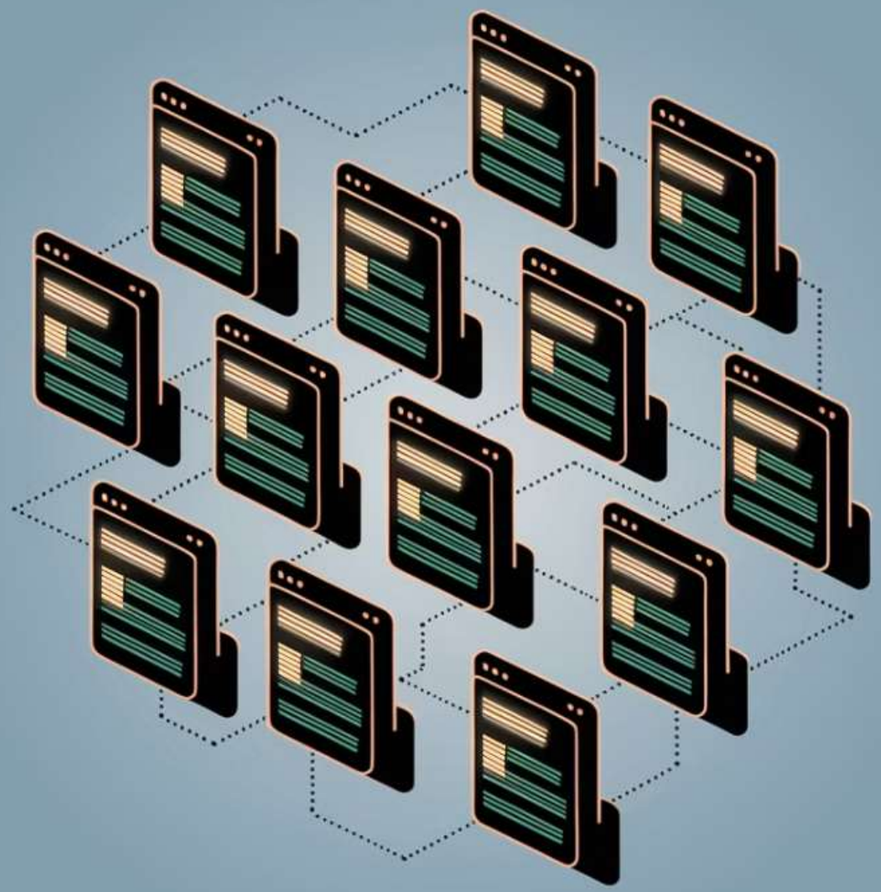
```
omdb=# explain (analyze,buffers,verbose,memory) select * from casts;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on public.casts  (cost=0.00..23258.44 rows=1262644 width=36) (actual time=0.940..122.156 rows=1241675 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Buffers: shared hit=9 read=10623
 Planning:
   Memory: used=9kB  allocated=16kB
 Planning Time: 0.089 ms
 Execution Time: 174.497 ms
(7 rows)

omdb=# SELECT n.nspname, c.relname, count(*) AS buffers
FROM pg_buffercache b JOIN pg_class c
ON b.relfilenode = pg_relation_filenode(c.oid) AND
b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname = current_database()))
JOIN pg_namespace n ON n.oid = c.relnamespace
where n.nspname = 'public' GROUP BY n.nspname, c.relname
ORDER BY n.nspname,3 DESC;
 nspname |   relname    | buffers
---------+--------------+---------
 public  | casts        |      42
 public  | casts_pkey   |       5
(2 rows)

omdb=# explain (analyze,buffers,verbose,memory) select * from casts;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on public.casts  (cost=0.00..23258.44 rows=1262644 width=36) (actual time=0.049..82.479 rows=1241675 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Buffers: shared hit=41 read=10591
 Planning:
   Memory: used=9kB  allocated=16kB
 Planning Time: 0.046 ms
 Execution Time: 127.320 ms
(7 rows)

omdb=# explain (analyze,buffers,verbose,memory) select * from casts;
                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on public.casts  (cost=0.00..23258.44 rows=1262644 width=36) (actual time=0.050..84.349 rows=1241675 loops=1)
   Output: movie_id, person_id, job_id, role, "position"
   Buffers: shared hit=73 read=10559
 Planning:
   Memory: used=9kB  allocated=16kB
 Planning Time: 0.047 ms
 Execution Time: 132.559 ms
(7 rows)
```

Full table scan loads all pages into the buffer. More the usage more pages are retained in buffer for future use.



Agenda: Understanding PostgreSQL Pages

1 **Buffer Cache**

Learn about the buffer cache. Understand its role in reducing I/O delays.

2 **Page Structure**

Delve into the structure of PostgreSQL pages. Explore how data is organized for optimal access.

3 **Buffer Usage**

View buffer usage during query execution

4 **Buffer Management**

Examine the clock-sweep algorithm. Discover how PostgreSQL manages memory.

Buffer Management

Buffer Lookup

Done through a hash table. It ensures fast access.
backend/storage/buffer/buf_table.c

Buffer Replacement

Implements LRU. Uses a clock sweep algorithm.

Usage Counter

Tracks buffer access frequency.
Evicts buffers with zero usage.
Max value (5) defined by BM_MAX_USAGE_COUNT

Buffer Allocation

```
BufferAlloc() {
  Create buffer tag for requested block;
  Lookup block in shared buffer pool;
  if (found) { // Cache hit
    Increment pin_count;
    Increment usage count;
  }
  if (not found) { // Cache miss
    v = free_List;
    if (v) {
      Add buffer to ring;
      return;
    } else {
      v = find_victim(); // Run clock sweep algorithm
      if (v dirty) {
        Write to WAL;
        Read block into v;
        Increment ref count;
      }
    }
  }
}
```

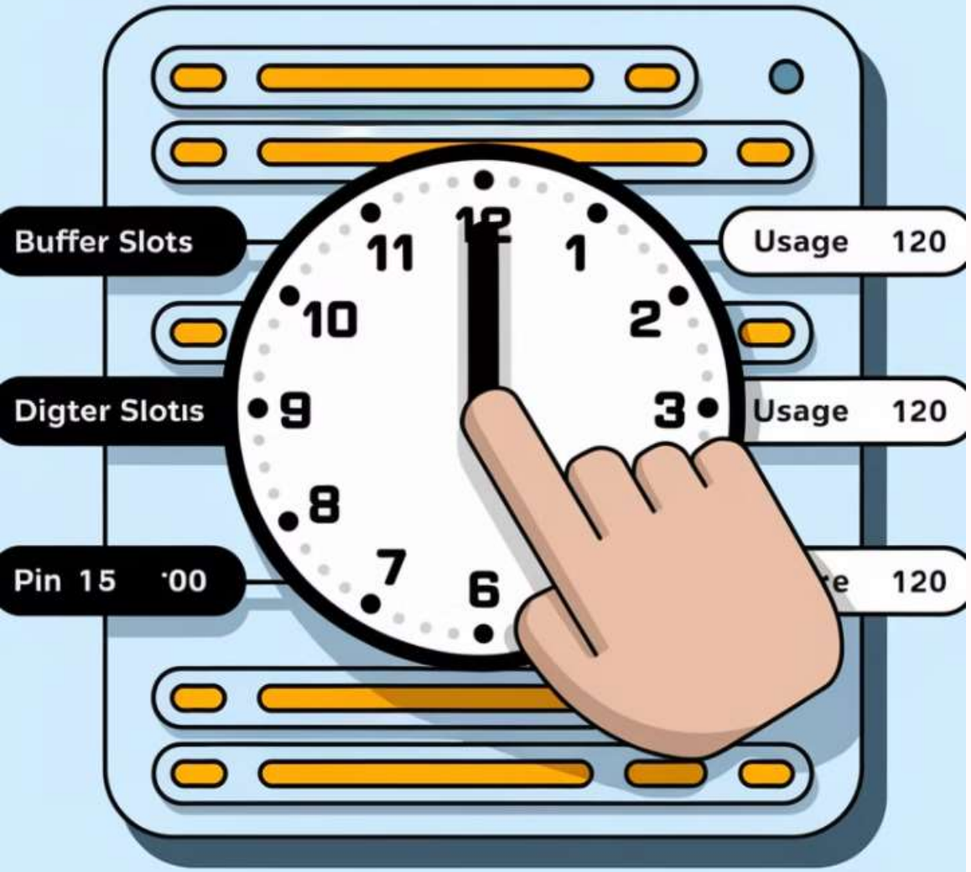
Buffer Allocation LRU - Clock Sweep Strategy

```
find_victim()
{
    int N = Max Pages in Buffer;
    int try = 0;

    while (try < N)
    {
        v = buffer[next_victim];
        if (v->pin_count > 0)
        {
            // Cannot evict this page as it is already being used
            try++;
            next_victim = (next_victim + 1) % N;
        }
        else
        {
            v->usage_count = v->usage_count - 1;
            if (v->usage_count == 0)
            {
                return v; // Found a victim
            }
            try++;
            next_victim = (next_victim + 1) % N;
        }
    }

    // If no free buffer is found after N attempts,
    // handle the situation (e.g., increase buffer space or reduce DB load)
    return out_of_buffer;
}
```

Clock Sweep Algorithm



next_victim ->

pin=1 usage=2
pin=0 usage=2
pin=0 usage=1
pin=0 usage=3
pin=0 usage=5

decrement usage by 1, check if usage is 0 (non-zero, move to next item in buffer)

next_victim ->

pin=1 usage=2
pin=0 usage=1
pin=0 usage=1
pin=0 usage=3
pin=0 usage=5

decrement usage by 1, check if usage is 0 (zero, victim found, return)

Monitoring Buffer Cache

1 pg_stat_io (PG 16)

Provides insights into I/O behavior. Fields : hits and evictions can be used to determine the cache hits and page swaps.

2 pg_buffercache

Displays real-time contents.
Each row represents a buffer in the shared cache.
High misses suggest low shared_buffers.

Thank you! Questions?

Appreciate your time and attention. We are now open for questions.