IN3020/4020 – Database Systems
Spring 2021, Week 7.1

Overview of Transaction Management and Introduction to Buffer Management

Dr. M. Naci Akkøk, Chief Architect, Oracle Nordics

Based upon slides by E. Thorstensen from Spring 2019

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

From INTRO:
The syllabus can be divided into three parts

- Main focus:
  - Queries, SQL and query optimization
  - ACID, DBMS characteristics and mechanisms
  - Other DBMS (slightly less than previous years)

- And an additional bit about emerging technologies and database research

We did this: Queries, SQL and query optimization

- SQL (repetition and a bit more)
- Query optimization (including relational algebra)
- Indexes, index usage and the underlying search
- Query plans and optimization

We are here: ACID, DBMS characteristics and Mechanisms

- Managing multiple concurrent uses/users and protecting the data
  - ACID – what it stands for
  - Transaction Management
  - Locking, logging, buffer & cache management
  - Relationships between these mechanisms
  - Synchronization/replication challenges

Overview

- Memory organization and data flow
- Files and blocks (pages)
- Working memory and shared cache

Memory organization

- (Almost) everything is measured in and stored in blocks.
- In addition to files on “disk”, we have
  - Shared memory
  - Working memory
- Each connection gets its own process.
- These have their own working memory for sorting / hash tables etc.
- Blocks from disk are put in common memory, everyone has access.

Shared memory

- Works both as a workspace on blocks and as a cache.
- All changes in tuples are made here.
- Tuples to be processed in a query are also read from here.
- Modified blocks must be written back to disk sooner or later.

Why own cache?

- The OS also has a cache, but this is not optimized for blocks.
- The DBMS knows more about its queries and can choose which individual blocks it caches.
- Can control writing back to disk better.

Some numbers

- IFI (typical) server could be something like:
  - 8GB shared memory
  - 32MB working memory
  - 8KB blocks

Blocks

- A block has a unique ID - which file it belongs to and its number.
- Makes blocks on disk and in memory "the same".

Blocks – Postgres examples

See: https://malisper.me/the-file-layout-of-postgres-tables/

Each table in Postgres is represented by one or more underlying files. Each 1GB chunk of the table is stored in a separate file. It is actually pretty easy to find the actual underlying files for a table. To do so, you first need to find the **Postgres data directory**, which is the directory in which Postgres keeps all of your data. You can find where the data directory is by running **SHOW DATA_DIRECTORY;**. When I run it locally, we see the following:

```
> SHOW DATA_DIRECTORY;
        data_directory
------------------------------
 /var/lib/postgresql/9.5/main
(1 row)
```

Now that you know where the Postgres data directory is, you will need to find where the files for the specific table we are looking for is located. To do so, you can use the *pg_relation_filepath* function with the name of the table you want to find the file for.

Blocks and tuples

- Each tuple is in a block.
- What if a value is too large for a block?
- Then a technique called TOAST (the Oversized-Attribute Storage Technique) is used.
- https://www.postgresql.org/docs/9.6/storage-toast.html: PostgreSQL uses a fixed page size (commonly 8 kB) and does not allow tuples to span multiple pages. Therefore, it is not possible to store very large field values directly. To overcome this limitation, large field values are compressed and/or broken up into multiple physical rows. This happens transparently to the user, with only small impact on most of the backend code. The technique is affectionately known as TOAST (or "the best thing since sliced bread"). The TOAST infrastructure is also used to improve handling of large data values in-memory.

Deletion of tuples

- Lazy deletion: Tuples are marked as deleted.
- Background process (vacuum) takes care of the later (deletes, reorganizes block, updates free space).
- May cause some fragmentation (half-full blocks).
- Can be fixed with vacuum full but is an expensive and locking operation.

Cache-control

- DBMS cache is the shared memory.
- Blocks read from disk are stored here and remain here as long as there is free space.
- Updates, reading for nested loop join etc. happen from here.
- Updates must be flushed to disk sooner or later.

Sooner or later, it will be full!

- If the shared memory does not have more free space, a block must be ejected. Remember “paging” logic?
- Some requirements for such a block:
  - Not in use by existing transaction
  - Not "dirty" (updated but not flushed)
- In addition, there are requirements related to multi-user problems.

Choosing a block to «sacrifice»

- Which block to evict?
- One that has not been used for a while?
- LRU: Least recently used. Requires keeping timestamps for each block and which one is the oldest.

- Better: One that has not been used for a while AND is not popular?
- Clock sweep algorithm!

Clock sweep

- We keep track of:
  - Pin bit (Boolean)
  - Usage count (Int)
- Usage count increases by 1 every time someone uses the block.
- All blocks are in a “cycle” - we go through them in order (round robin scheduling style).
- Blocks in use (pinned) are skipped, others get -1 (decremented) usage count until we find a block with 0 usage count.

Clock sweep, pseudocode

```
while true:
    b = next_page ()
    if pinned (b) == false:
        if use_count (b) == 0:
            return b
        use_count (b) = use_count (b) - 1
```

`next_page ()` goes through the blocks in a cycle (i.e., after the last block, the next block is # 1)

Why clock sweep?

- A block that received use count - 1 will not be checked again until we have gone all the way around the cycle of blocks.
- If it gets 0 in use count, it is a candidate for eviction if we can't find one that got it earlier.
- A block that was accessed 5 times for a while back lives longer than one that was accessed once.

Cache control, special cases

- If someone loads a large table, large portions of the cache will be thrown out.
- It's unlikely that the whole table is cached.

- Such cases are handled as special cases –

  free 32 (= $\frac{256}{8}$) blocks, reuse these.

Smarter cache control

- All blocks are equal, but some are more equal to others ☺
- Can divide blocks by type (data table, index, metadata ...)
- Least Recently Used (LRU) / Clock Sweep separately for each type.

Some notes (“hot” cases)

- Nested loop join: The smallest table should be used over and over again.
- Fine if it stays cached. Some algorithms will be able to use such information.