Postgres Conference

Silicon Valley 2022

San Jose / United Statues

April 07-08, 2022

Tune PostgreSQL for Read/Write Scalability.

Technical Breakout

Ibrar Ahmed
Senior Database Architect
Percona LLC

PERCONA

Who am I?

IBRAR AHMED

Senior Software Architect

Percona LLC

@ibrar_ahmad

https://www.facebook.com/ibrar.ahmed

https://www.linkedin.com/in/ibrarahmed74/

Software Career

- Software industries since 1998.

PostgreSQL Career

- Working on PostgreSQL Since 2006.
- EnterpriseDB (Associate Software Architect core Database Engine) 2006-2009
- EnterpriseDB (Software Architect core Database Engine) 2011 - 2016
- EnterpriseDB (Senior Software Architect core Database Engine) 2016 – 2018
- Percona (Senior Software Architect core Database Engine) 2018 – Present

PostgreSQL Books

- PostgreSQL Developer's Guide
- PostgreSQL 9.6 High Performance

Database Performance

Database Performance

Key Factors about Database Performance

PostgreSQL Architecture

PostgreSQL Modules

PostgreSQL Performance Tuning

Tune your PostgreSQL Box

Database Benchmarking

Benchmark your database

Linux Kernel Tuning

Tune you Linux Box

02

Database Performance

Database Performance

Hardware

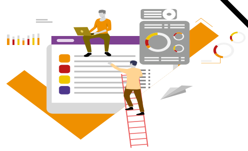
Choose your hardware wisely

Operating System

Operating System Selection and **Tuning**

Software

Software Impact on database performance



Workload

Choose Database according to your workload

Database

Choose your database according your data need

Software

Software Type

Impact of badly written and well-defined software

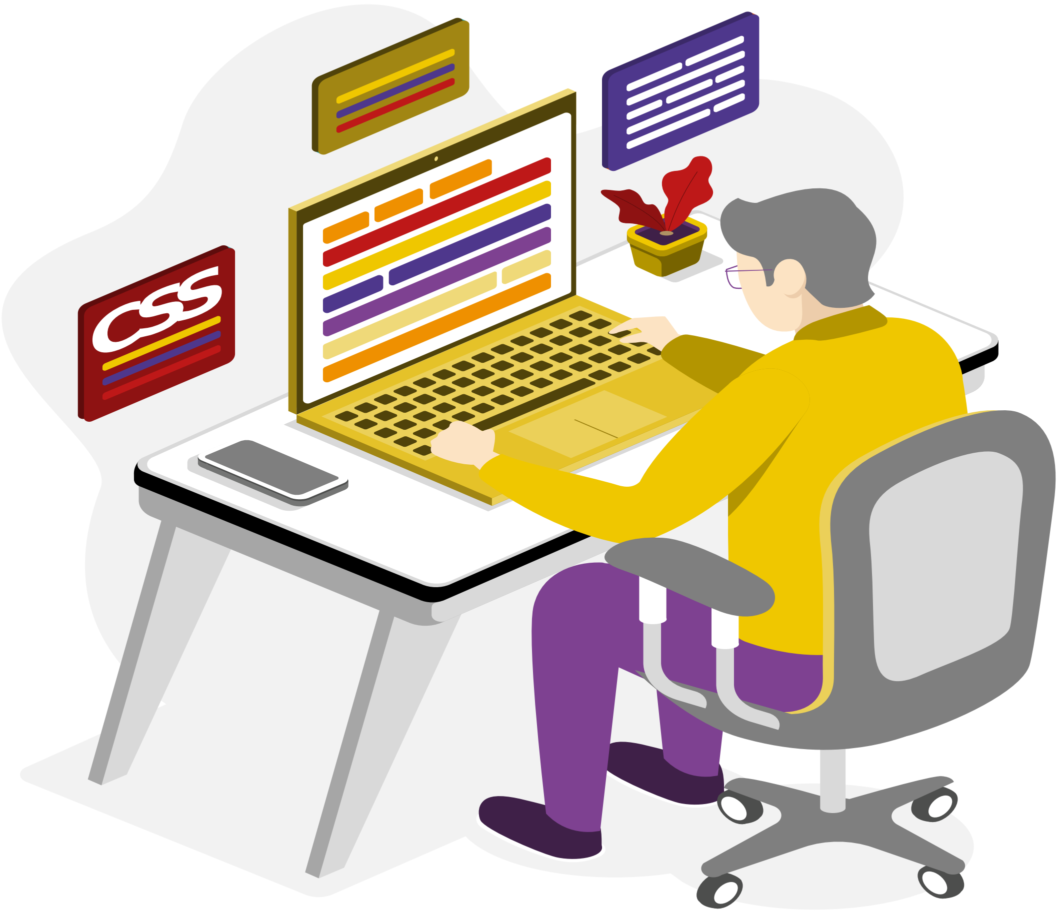
Queries

Impact of Database Queries on the database

Connectors

Choose your connector wisely

Hardware



Processor

CPU performance and number of processor cores

Memory

Memory requirement for your workload

Disk

Disk speed and size

Network

Network latency

Operating System

Environment

Operating system environment according to your database and application

Compatibility

Operating system compatibility with your database and application

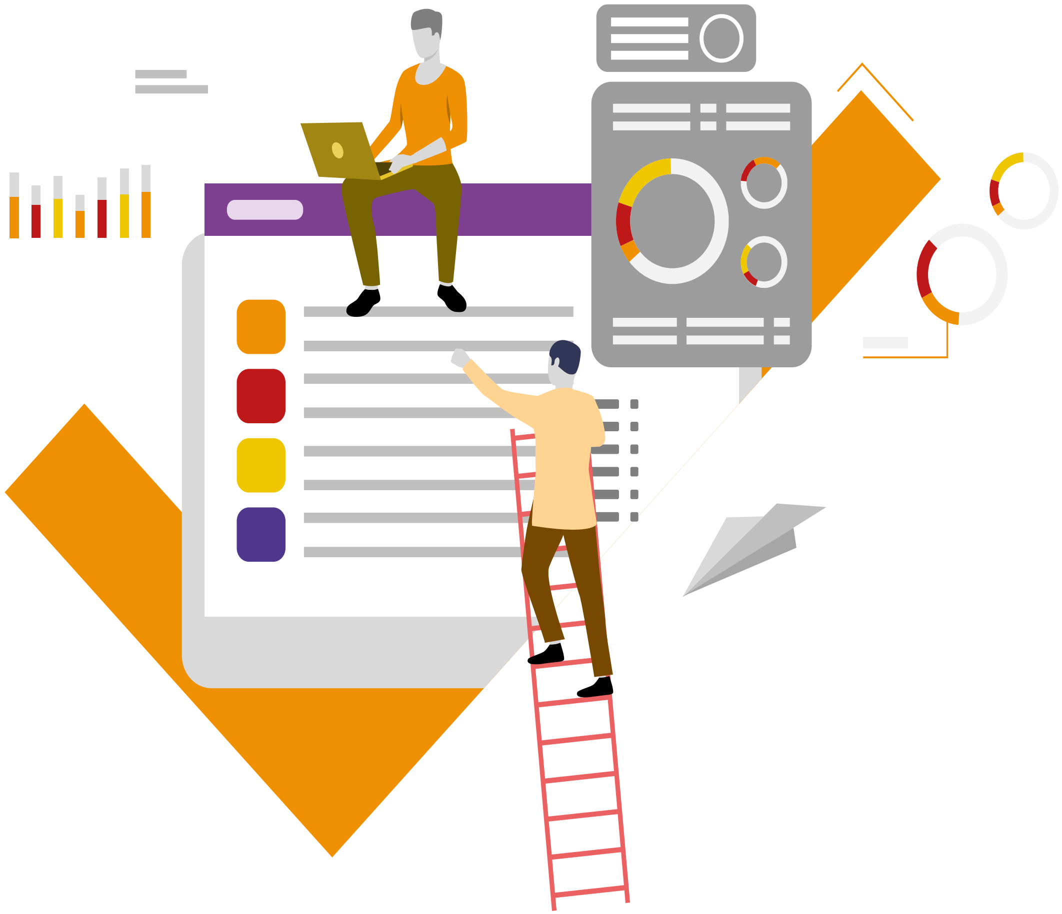
Performance

Operating system performance which suites your database

Support

How good is your support for operating system

Workload



SQL / NOSQL

Is your workload best suited for SQL or NOSQL?

Size of Workload

Size of workload is important for tunning

OLTP / OLAP

Type of workload, is it OLAP or OLTP

Read / Write Intensive

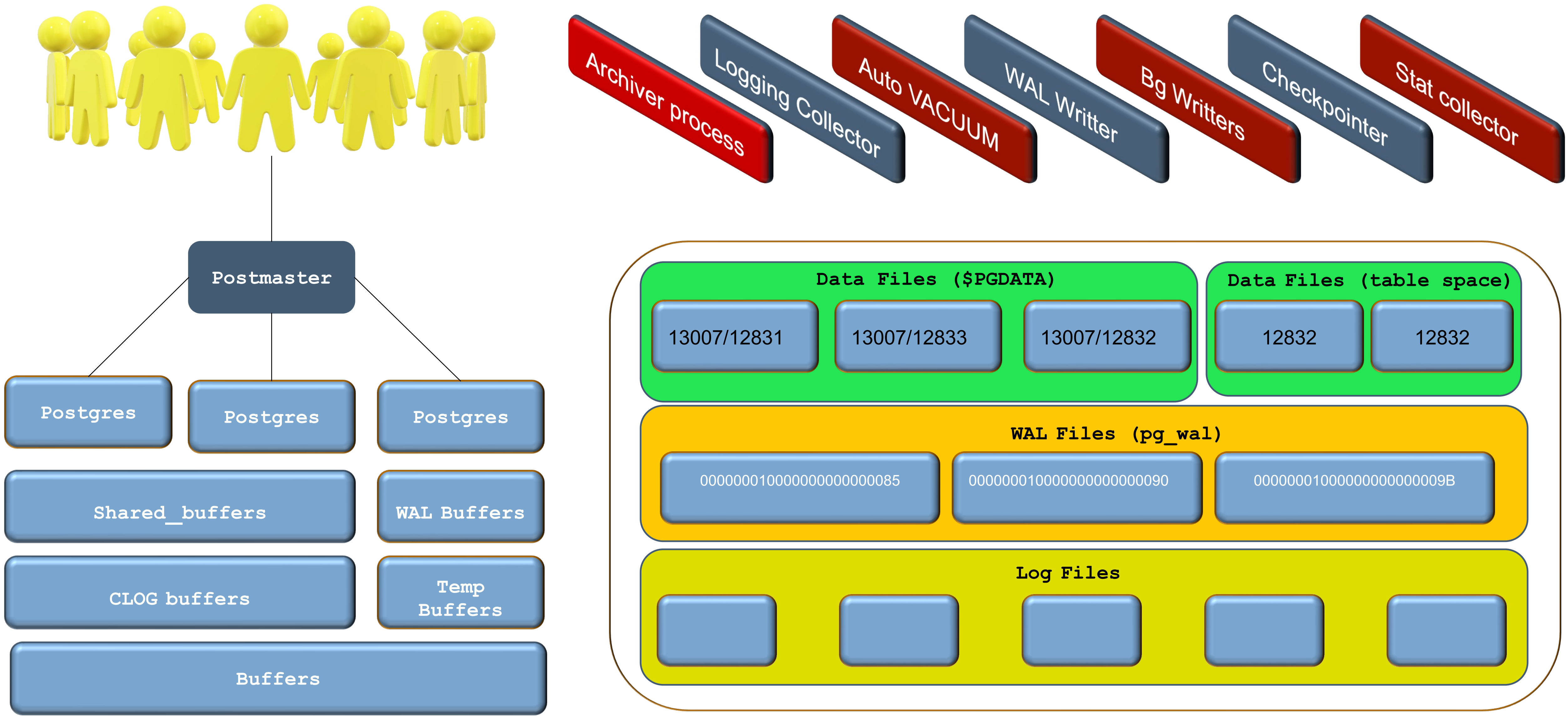
Is your workload Read or Write intensive?

Database

02

PostgreSQL Architecture

PostgreSQL Architecture



PERCONA

02

PostgreSQL Performance Tuning

PERCONA

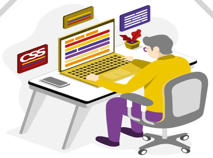
Database Performance

Tunning Parameter

PostgreSQL Tunning Parameters

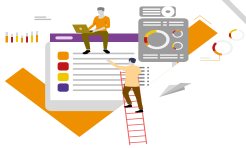
PostgreSQL Indexes

Impact of index on Database Performance



Query Analysis

Analyze your queries for optimal database performance



Partitioning

Partition your database when require

Monitoring

Monitor your database to identify the bottleneck

Tunning Parameters

Memory based configuration parameters

1. **`shared_buffers`**
   PostgreSQL Memory buffer

2. **`wal_buffers`**
   Buffer for WAL

3. **`effective_cache_size`**
   PostgreSQL Cache

4. **`work_mem`**
   Buffer for sorting

5. **`maintenance_work_mem`**
   Buffer for Maintenance activity

6. **`Other`**
   postgresql.conf file contains all the parameters

shared_buffers

- PostgreSQL uses its own buffer along with kernel buffered I/O.
- PostgreSQL does not change the information on disk directly then how?
- Writes the data to shared buffer cache.
- The backend process write that these blocks kernel buffer.

```
postgresql=# SHOW shared_buffers;
shared_buffers
----------------
128MB
(1 row)
```

The proper size for the POSTGRESQL shared buffer cache is the largest useful size that does not adversely affect other activity.

—Bruce Momjian

wal_buffer

- Do you have Transaction? Obviously
- WAL – (Write Ahead LOG) Log your transactions
- Size of WAL files 16MB with 8K Block size (can be changed at compile time)
- PostgreSQL writes WAL into the buffers(*wal_buffer*) and then these buffers are flushed to disk.

Bigger value for wal_buffer in case of lot of concurrent connection gives better performance.

effective_cache_size

- This used by the optimizer to estimate the size of the kernel's disk buffer cache.
- The effective_cache_size provides an estimate of the memory available for disk caching.
- It is just a guideline, not the exact allocated memory or cache size.

work_mem

- This configuration is used for complex sorting.
- It allows PostgreSQL to do larger in-memory sorts.
- Each value is per session based, that means if you set that value to 10MB and 10 users issue sort queries then 100MB will be allocated.
- In case of merge sort, if x number of tables are involved in the sort then **`x * work_mem`** will be used.
- It will allocate when required.
- Line in EXPLAIN ANALYZE "Sort Method: external merge Disk: 70208kB"

work_mem

```
postgres=# SET work_mem ='2MB';
postgres=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY id;
                                                            QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=848382.53..1917901.57 rows=9166666 width=9) (actual time=5646.575..12567.495
rows=11000000 loops=1)
   ->  Sort  (cost=847382.51..858840.84 rows=4583333 width=9) (actual time=5568.049..7110.789 rows=3666667
loops=3)
Planning Time: 0.055 ms
Execution Time: 13724.353 ms
```

```
postgres=# SET work_mem ='1GB';
postgres=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY id;
                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Sort  (cost=1455965.01..1483465.01 rows=11000000 width=9) (actual time=5346.423..6554.609 rows=11000000
loops=1)
   Sort Key: id
   Sort Method: quicksort  Memory: 916136kB
   ->  Seq Scan on foo  (cost=0.00..169460.00 rows=11000000 width=9) (actual time=0.011..1794.912
rows=11000000 loops=1)
 Planning Time: 0.049 ms
 Execution Time: 7756.950 ms
```

maintenance_work_mem

- maintenance_work_mem is a memory setting used for maintenance tasks.
- The default value is 64MB.
- Setting a large value helps in tasks like
  - VACUUM
  - RESTORE
  - CREATE INDEX
  - ADD FOREIGN KEY
  - ALTER TABLE.

maintenance_work_mem

```
CHECKPOINT;
SET maintenance_work_mem='10MB';
SHOW maintenance_work_mem;
 maintenance_work_mem
----------------------
 10MB
(1 row)
postgres=# CREATE INDEX idx_foo ON foo(id);
Time: 12374.931 ms (00:12.375)
```

```
CHECKPOINT;
SET maintenance_work_mem='1GB';
SHOW maintenance_work_mem;
 maintenance_work_mem
----------------------
 1GB
(1 row)
postgres=# CREATE INDEX idx_foo ON foo(id);
Time: 9550.766 ms (00:09.551)
```

PERCONA

synchronous_commit

- This is used to enforce that commit will wait for WAL to be written on disk before returning a success status to the client.
- This is a trade-off between performance and reliability.
- Increasing reliability decreases performance and vice versa.

Synchronous commit doesn't introduce the risk of *corruption*, which is really bad, just some risk of data *loss*.

https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

Tunning Parameters

I/O based configuration parameters

01

checkpoint_timeout

Time for checkpoint

3

max_wal_size

Max WAL size

5

checkpoint_completion_target

Checkpoint completion target

checkpoint_timeout

- PostgreSQL writes changes into WAL. The checkpoint process flushes the data into the data files.
- More checkpoints have a negative impact on performance.
- Frequent checkpoint reduce the recovery time

checkpoint_completion_target

- Specifies the target of checkpoint completion, as a fraction of total time between checkpoints.
- This parameter can only be set in the postgresql.conf file or on the server command line

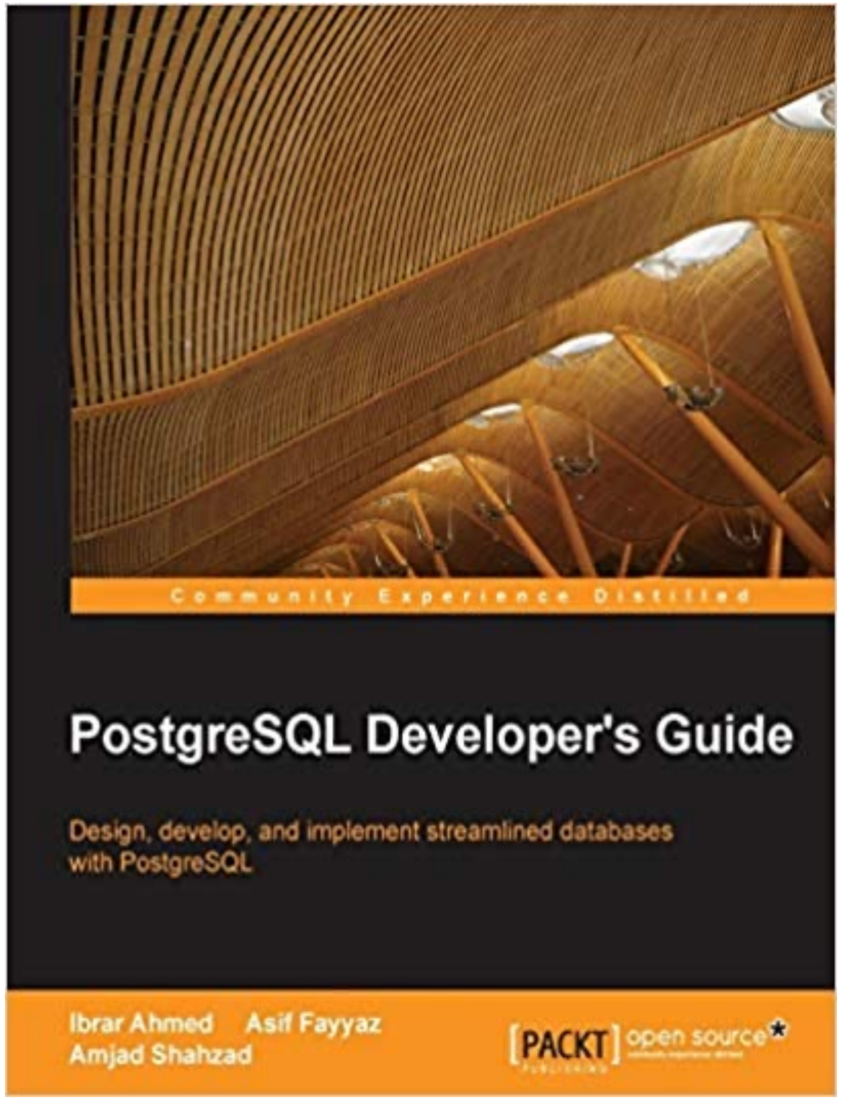
max_wal_size

- Maximum size to let the WAL grow during automatic checkpoints.
- This is a soft limit; WAL size can exceed max_wal_size under special circumstances.
- This parameter can only be set in the postgresql.conf file or on the server command line.

02

PostgreSQL Indexes

Heap / Index



PostgreSQL comes with two main types of triggers: **row-level trigger** and **statement-level trigger**. These are specified with `FOR EACH ROW` (row-level triggers) and `FOR EACH STATEMENT` (statement-level triggers). The two can be differentiated by how many times the trigger is invoked and at what time. This means that if an `UPDATE` statement is executed that affects 10 rows, the row-level trigger will be invoked 10 times, whereas the statement-level trigger defined for a similar operation will be invoked only once per SQL statement.

Triggers can be attached to both tables and views. Triggers can be fired for tables before or after any `INSERT`, `UPDATE`, or `DELETE` operation; they can be fired once per affected row, or once per SQL statement. Triggers can be executed for the `TRUNCATE` statements as well. When a trigger event occurs, the trigger function is invoked to make the appropriate changes as per the logic you have defined in the trigger function.

The triggers defined with `INSTEAD OF` are used for `INSERT`, `UPDATE`, or `DELETE` on the views. In the case of views, triggers fired before or after `INSERT`, `UPDATE`, or `DELETE` can only be defined at the statement level, whereas triggers that fire `INSTEAD OF` on `INSERT`, `UPDATE`, or `DELETE` will only be defined at the row level.

Triggers are quite helpful where your database is being accessed by multiple applications, and you want to maintain complex data integrity (this will be difficult with available means) and monitor or log changes whenever a table data is being modified.

The next topic is a concise explanation of tricky trigger concepts and behaviors that we discussed previously. They can be helpful in a database design that involves triggers.

Tricky triggers

In `FOR EACH ROW` triggers, function variables contain table rows as either a `NEW` or `OLD` record variable, for example, in the case of `INSERT`, the table rows will be `NEW`, for `DELETE`, it is `OLD`, and for `UPDATE`, it will be both. The `NEW` variable contains the row after `UPDATE` and `OLD` variable holds the row state before `UPDATE`.

Hence, you can manipulate this data in contrast to `FOR EACH STATEMENT` triggers. This explains one thing clearly, that if you have to manipulate data, use `FOR EACH ROW` triggers.

The next question that strikes the mind is how to choose between row-level `AFTER` and `BEFORE` triggers.

PostgreSQL Indexes

B-Tree

PostgreSQL default index
Based on B-Tree

Hash

PostgreSQL Index Method
Based on Hasing

Brin

Block Range Index

GIST

Generalized Search Tree

GIN

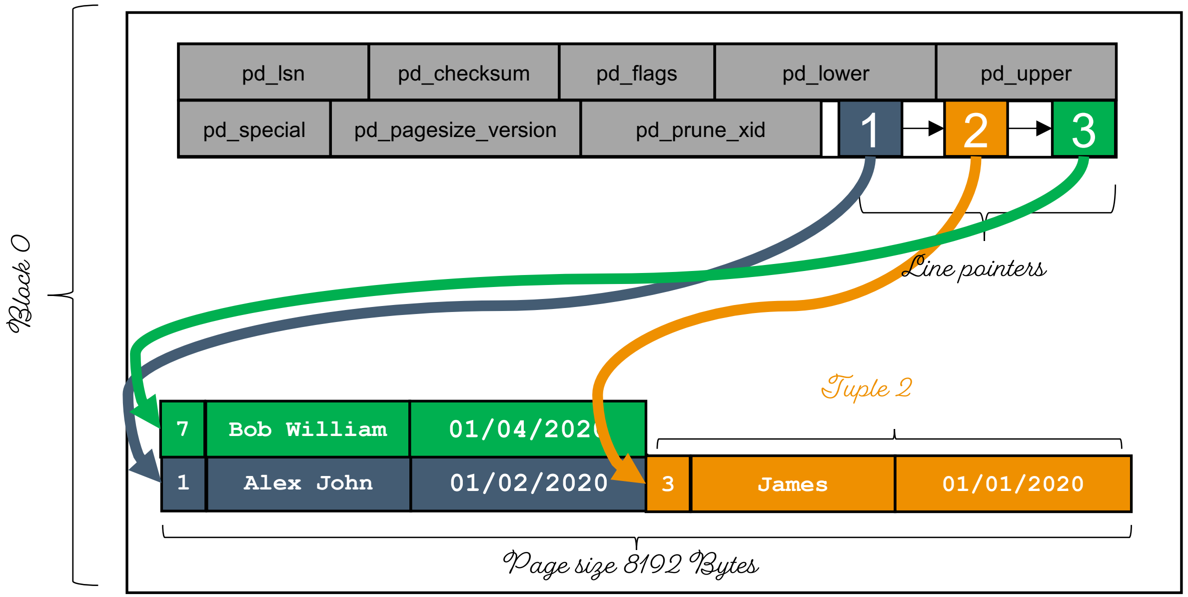
Generalized Inverted Index

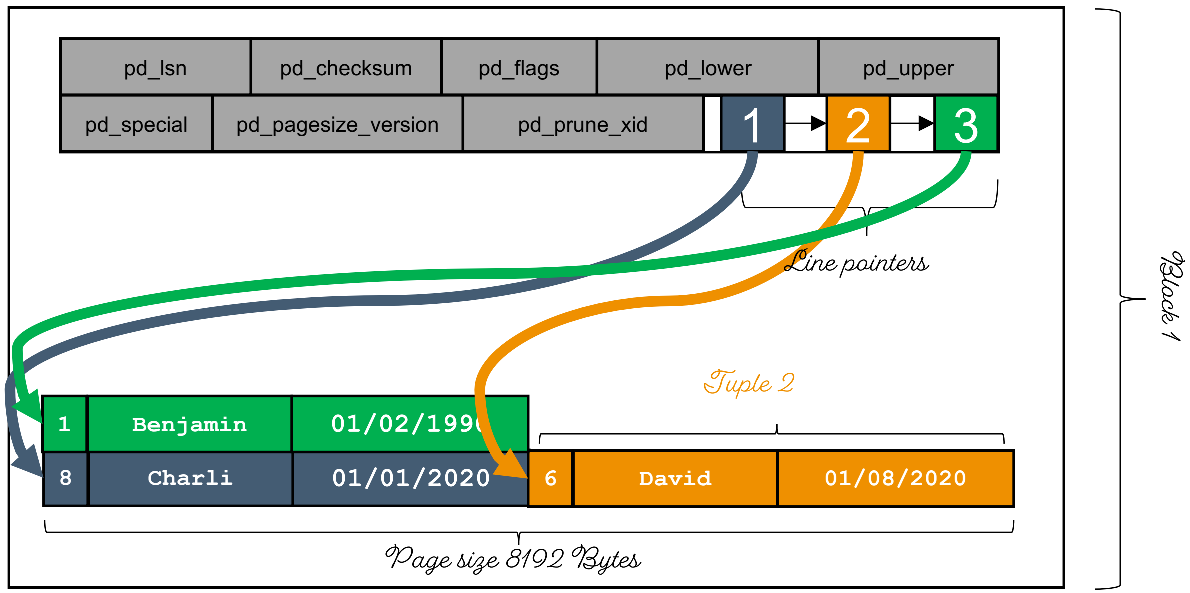
Sequential Scan

```
SELECT * FROM admin WHERE dt < '2021/04/01';
 id  |     name       |      dt
-----+--------------+------------
  3  | James          | 2020-01-01
  1  | Alex Johns     | 2020-01-02
  7  | Bob William    | 2020-01-04
  8  | Charli         | 2020-01-01
  6  | David          | 2020-08-02
  9  | Benjamin       | 1990-01-02
```

```
SELECT ctid, * FROM admin WHERE id = 8;

 ctid   | id | name
--------+----+------
 (1,0)  | 16 | Charli
(1 rows)
```



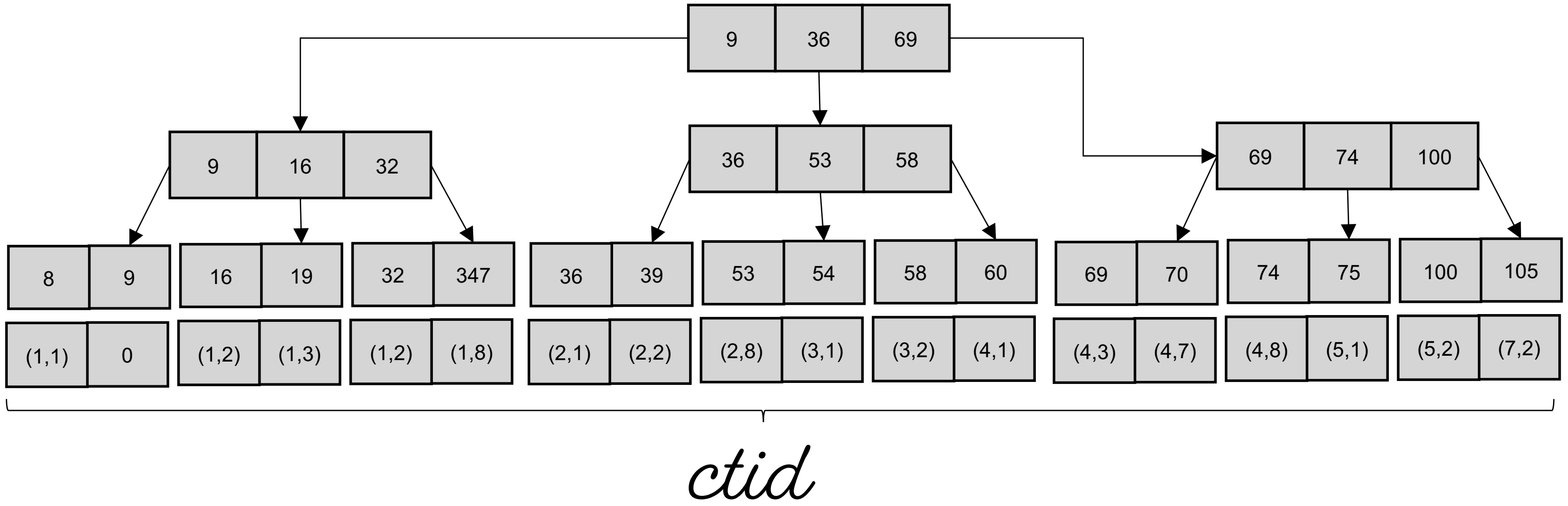


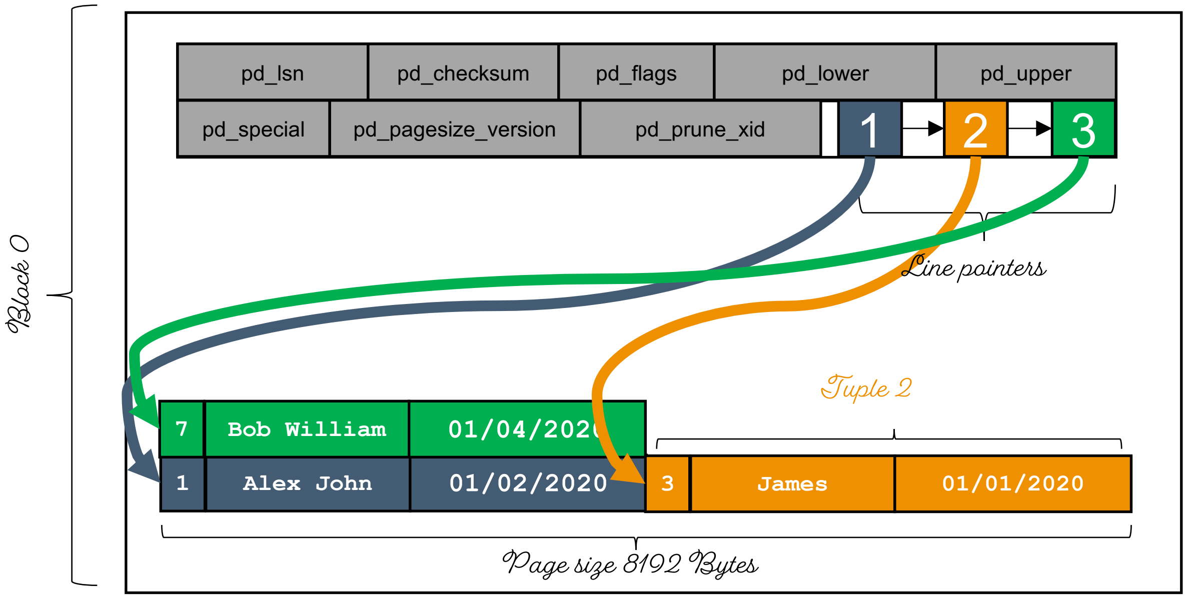
PERCONA

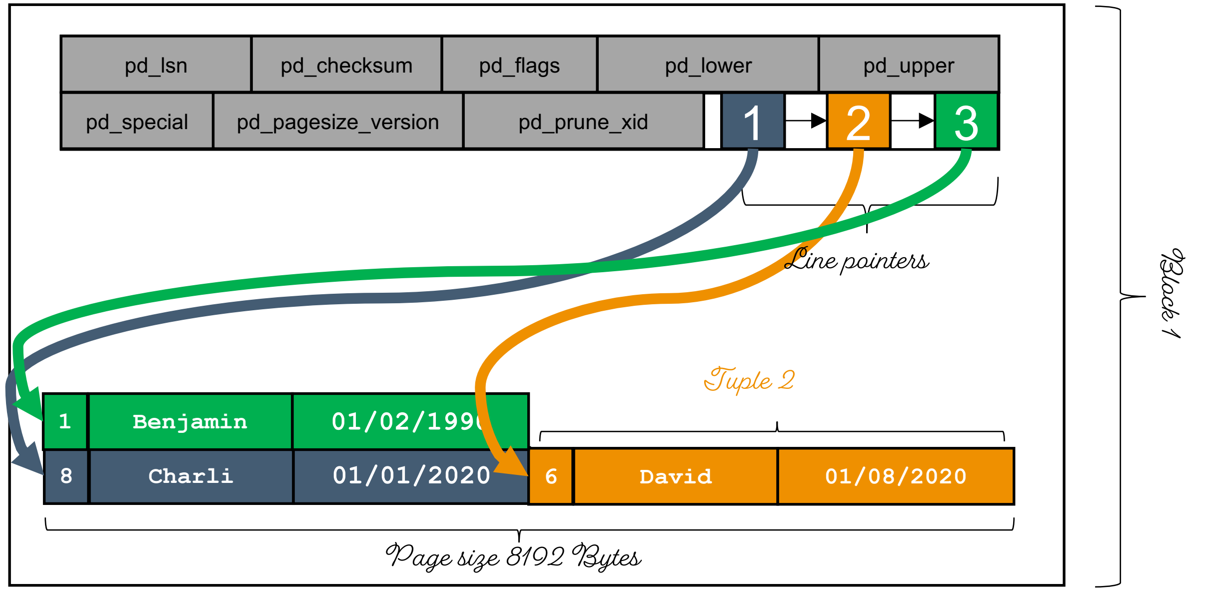
B-Tree Index

```
SELECT id, name FROM admin
WHERE id = 8;

 id    | name
-----+-------
  8    | Charli
(1 rows)
```



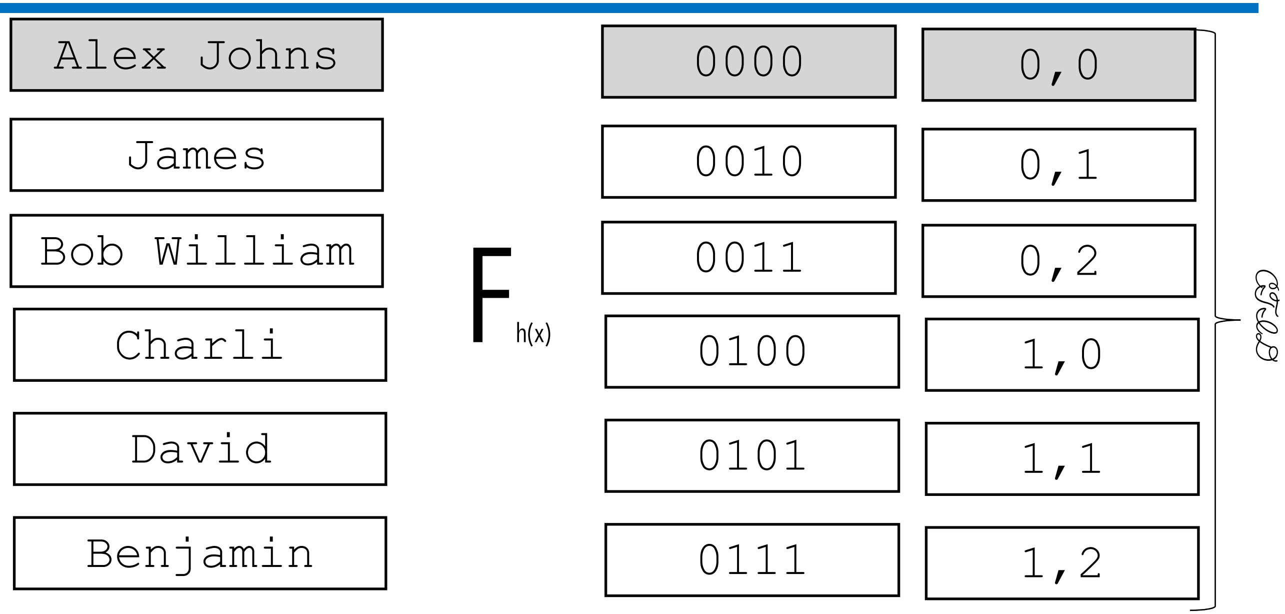


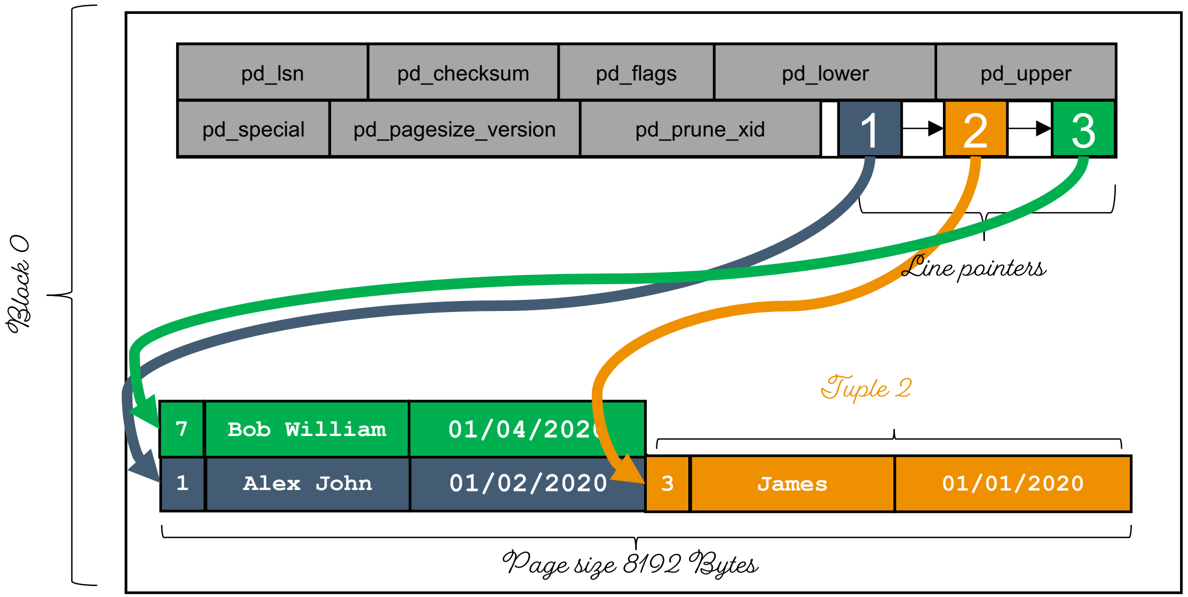


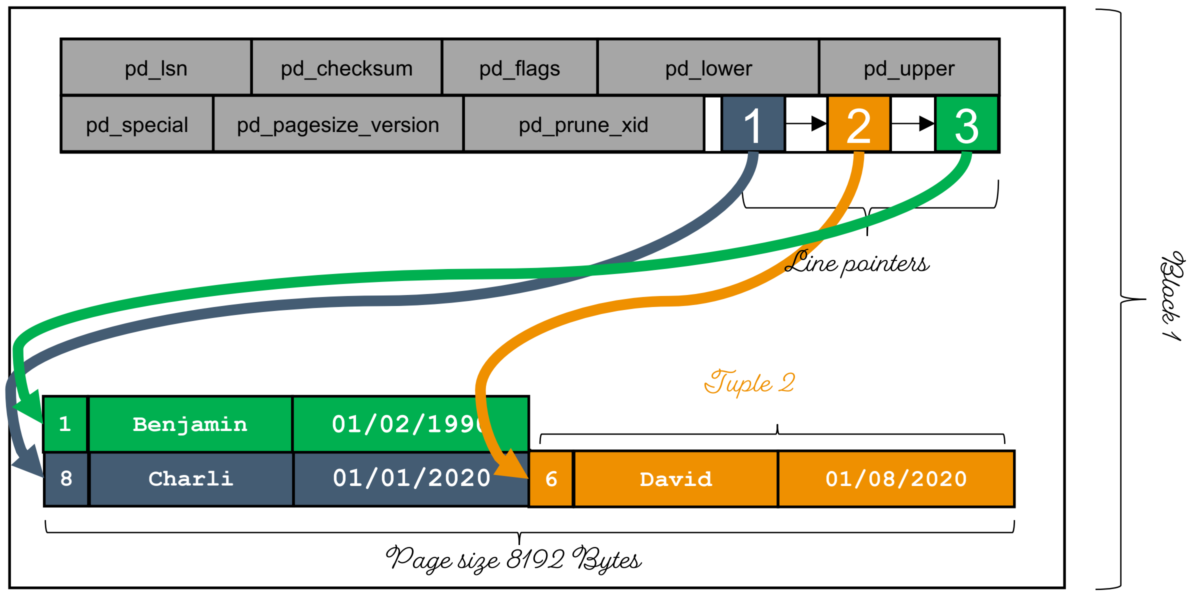
Hash Index

```
SELECT id, name FROM admin WHERE name
LIKE 'Alex Johns';

 id   | name
------+------
  16  | Alex Johns
(1 rows)
```







PERCONA

02

Linux Tuning for PostgreSQL

Linux Tunning

Memory Architecture

How to create indexes in PostgreSQL

Huge Pages

Linux Huge pages

Transparent Huge Pages

Linux Transparent Huge Pages

Tunning Parameters

Tune your Linux Box

Linux

Monitoring

Monitor your Linux OS

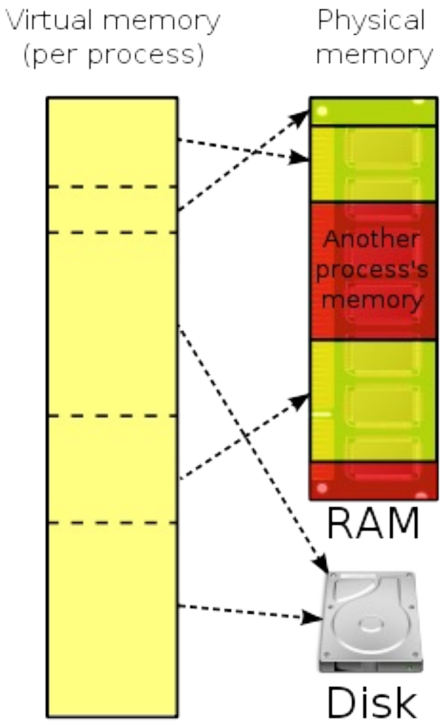
PERCONA

Input Output Handling

- Direct IO, Buffered IO and Double buffering
- PostgreSQL believes that the Operating system (Kernel) knows much better about storage and IO scheduling.
- PostgreSQL has its own buffering; and also needs the pages cache. Double Buffering
- It Increase the use of memory.
- And different kernel and setting behave differently.

Virtual Memory

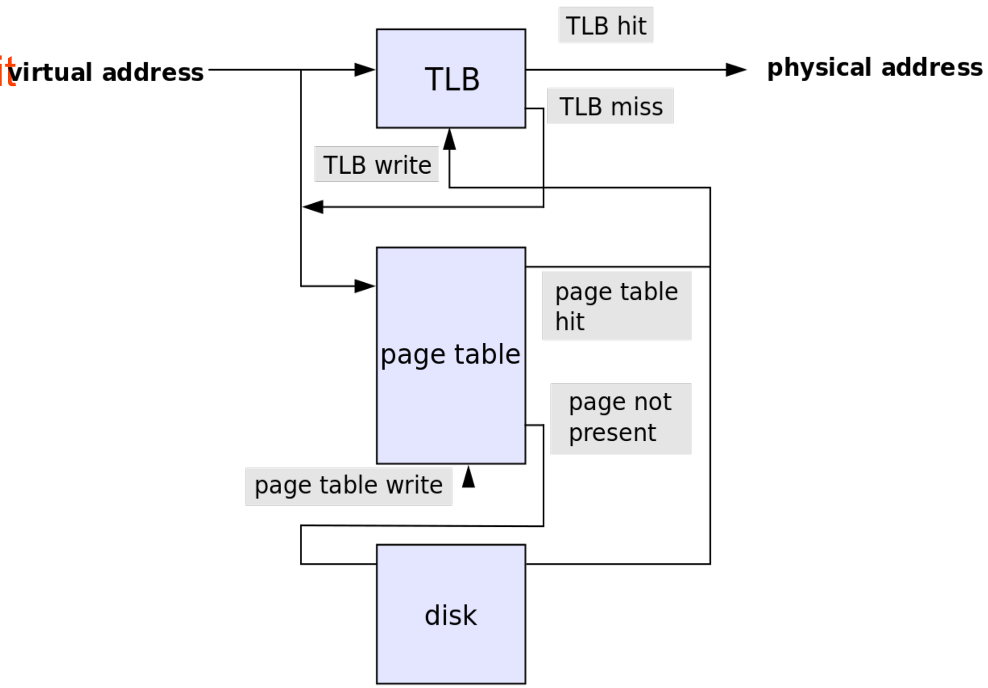
- Every process is given the impression that it is working with large, contiguous sections of memory
- Each process runs in its own dedicated address space
- Pages Table are used to translate the virtual addresses seen by the application into Physical Address



- https://en.wikipedia.org/wiki/Virtual_memory
- https://en.wikipedia.org/wiki/Page_table

Translation Lookaside Buffer (TLB)

- Translation Lookaside Buffer is a memory cache
- It reduce the time to access a user memory location
- If a match is found the physical address of the page is returned → TLB hit
- If match not found scan the page table (*walk*)
-    looking for the address mapping (entry) → TLB miss



Small page size bigger TLB size → expensive

Memory Pages

- PostgreSQL uses its own buffer along with kernel buffered I/O.
- PostgreSQL does not change the information on disk directly then how?
- Writes the data to shared buffer cache.
- The backend process write that these blocks kernel buffer.

Linux Page sizes

- Linux Page size is 4K
- Many modern processors support other page sizes

• If we consider a server with 256G of RAM:

large/huge pages

4K	67108864
2M	131072
1G	256

Classic Huge Pages

```
# cat /proc/meminfo
MemTotal:        264041660 kB
...
Hugepagesize:         2048 kB
DirectMap4k:        128116 kB
DirectMap2M:       3956736 kB
DirectMap1G:     266338304 kB

sysctl -w vm.nr_hugepages=256
```

PERCONA

Tunning Parameters

1 Classic Huge Pages

2 overcommit_memory
&
overcommit_ratio

3 Tranparent Huge Pages

4 dirty_background_ratio
&
dirty_background_bytes

5 swappines

6 vm.dirty_ratio
&
vm.dirty_bytes

Classic Huge Pages

- `# vi /etc/default/grub`
  - `GRUB_CMDLINE_LINUX_DEFAULT="hugepagesz=1GB default_hugepagesz=1G”`
- `# update-grub`
  - `Generating grub configuration file ...`
  - `Found linux image: /boot/vmlinuz-4.4.0-75-generic`
  - `Found initrd image: /boot/initrd.img-4.4.0-75-generic`
  - `Found memtest86+ image: /memtest86+.elf`
  - `Found memtest86+ image: /memtest86+.bin`
  - `Done`
- `# shutdown -r now`

Classic Huge Pages

- `# vim /etc/postgresql/10/main/postgresql.conf`
- `huge_pages=ON # default is try`
- `# service postgresql restart`

Transparent Huge pages

- The kernel works in the background (khugepaged) trying to:
  - "create" huge pages.
  - Find enough contiguous blocks of memory
  - Convert them into a huge page
- Transparently allocate them to processes when there is a "fit"

Disabling Transparent Huge pages

```
# cat /proc/meminfo | grep AnonHuge
AnonHugePages:        2048 kB
```

```
# ps aux | grep huge
```

```
root         42  0.0  0.0      0     0 ?        SN   Jan17   0:00 [khugepaged]
```

To disable it:

at runtime:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

at boot time:

```
GRUB_CMDLINE_LINUX_DEFAULT="(...) transparent_hugepage=never"
```

PERCONA

vm.swappines

- This is another kernel parameter that can affect the performance of the database.
- Used to control the swappiness (swapping pages to swap memory into RAM) behavior on a Linux system.
- The parameter can take anything from “0” to “100”.
- The default value is 60.
- Higher value means more aggressively swap.

vm.overcommit_memory and vm.overcommit_ratio

- Applications acquire memory and free that memory when it is no longer needed.
- But in some cases, an application acquires too much memory and does not release it. This can invoke the OOM killer.
- This is used to control the memory over-commit.
- It has three options
  - Heuristic overcommit, Do it intelligently (default); based kernel heuristics
  - Allow overcommit anyway
  - Don’t over commit beyond the overcommit ratio.

vm.dirty_background_ratio and vm.dirty_background_bytes

- The vm.dirty_background_ratio is the percentage of memory filled with dirty pages that need to be flushed to disk.
- Flushing is done in the background.
- The value of this parameter ranges from 0 to 100;

vm.dirty_ratio / vm.dirty_bytes

- The vm.dirty_background_ratio is the percentage of memory filled with dirty pages that need to be flushed to disk.
- Flushing is done in the foreground.
- The value of this parameter ranges from 0 to 100;

Blogs

https://www.percona.com/blog/2018/08/31/tuning-postgresql-database-parameters-to-optimize-performance/

https://www.percona.com/blog/2018/08/29/tune-linux-kernel-parameters-for-postgresql-optimization/

THANK YOU

GET IN TOUCH

@ibrar_ahmad

https://www.facebook.com/ibrar.ahmed

https://www.linkedin.com/in/ibrarahmed74/

www.pgelephant.com