

# PostgreSQL Performance Tuning and Optimization

Devrim Gündüz  
Dave Page

October 2021



# Agenda

- Who is EDB?
- Designing the hardware
- Tuning the operating system
- Tuning PostgreSQL parameters
- Query tuning
- Partitioning

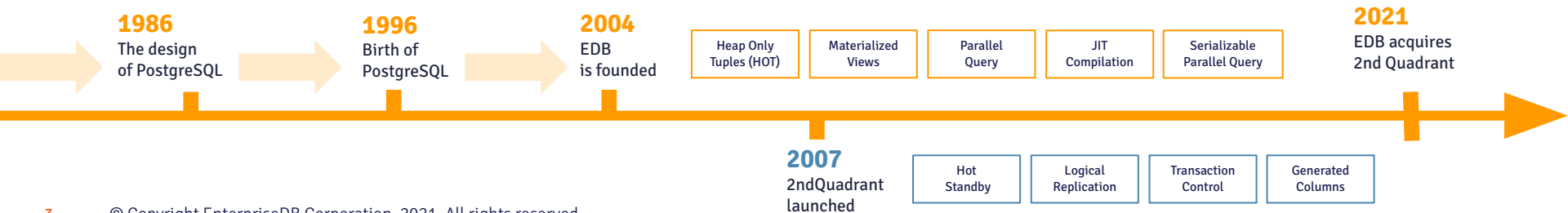


# Who is EDB?

We're database fanatics who care deeply about PostgreSQL

- Largest dedicated PostgreSQL company
- Enterprise PostgreSQL innovations
- Major PostgreSQL community leadership

## EDB supercharges PostgreSQL





# Designing the hardware: Bare metal

# Hardware design (Bare metal): CPU

- Which CPU's suit PostgreSQL more?
- CPU caches
- L1 and L2 cache
- L3 cache

# Hardware design (Bare metal): Disk

- **Depends on the application**
  - Read intensive / write intensive / mixed load
- **RAID and PostgreSQL**
  - RAID 1 (or 10) for WAL
  - RAID 10 for data
- **Tablespaces**
  - Slower/cheaper drives for archive data

# Hardware design (Bare metal): RAM

- Cheapest component
  - Go as big as you can
- Cache
  - More cache, less I/O
- Hotswap RAM
  - Avoid downtime for upgrades/replacements
  - Expensive!

# Hardware design (Bare metal): Network

- May sound irrelevant
- Faster network -> faster data transfer
  - Also, faster replication





# Designing the hardware: Virtual machines

# Hardware design (Virtual): Virtual machine

- Para-virtualisation vs. full
- Dedicated hardware
  - Noisy neighbours!
- Choose instance types carefully:
  - Number of cores
  - RAM
  - Network throughput
- NUMA pinning
  - Pin VMs to specific CPUs where possible

# Hardware design (Virtual): Disk

- Pre-allocation of disks
- RAID
  - No performance benefit using Linux MDRAID over multiple AWS EBS devices in our testing
  - There may be benefits in other environments; it depends on the network/storage architecture
- **Dedicated IOPs**
  - Provision storage with guaranteed IOP performance



# Tuning the operating system

# tuned (Adaptive system tuning daemon)

- Dynamic adaptive system tuning daemon
- RHEL's default tuning mechanism
- Optional for Debian/Ubuntu
- Anaconda (the RHEL installer) picks up a good default
- Needs some manual configuration
- Demo!

# Huge pages

- Huge pages allow allocation of much larger blocks of memory
- As the data grows more, PostgreSQL will cache more GBs of data in RAM
- Default page size: 4kB
- Disabled by default
- Requires a restart (of PostgreSQL)
- Demo!

# Optimizing filesystem

- Get more from the filesystem
  - “Noatime”
  - PostgreSQL does not rely on file access time
  - Disabling it saves CPU cycles

# Filesystem type selection

- Several options available
- XFS is the most popular (and default on major OSes)
- Do not turn off journaling
- Btrfs is not quite there *\*yet\**





# PostgreSQL tuning starting points

# PostgreSQL tuning

- Many of the default parameters are not suitable for production usage
  - Default config is designed to "run anywhere", e.g R-Pi, POS machines.
- Some parameters should always be changed
- A great way to improve performance

# PostgreSQL tuning: Connections

- `max_connections`
  - Rule of thumb: Not more than needed, to reduce the size of pre-allocated data structures
  - In an ideal world matches the number of CPU cores, but often 2:1 or 4:1
    - Consider using a pooler if there's a need for hundreds of connections

# PostgreSQL tuning: Resource usage

- **shared\_buffers**
  - Main 'database cache'. Depends on RAM, no more than 50% of what's available
- **work\_mem**
  - 'Working' memory for queries. This is per sort/hash table operation, so be careful
- **maintenance\_work\_mem**
  - Memory used for maintenance operations such as VACUUM. Depends on the available RAM, but usually 1-4 GB
- **autovacuum\_work\_mem**
  - -1 uses maintenance\_work\_mem
- **effective\_io\_concurrency**
  - Number of IO operations that can be expected to execute in parallel
  - Depends on the drives, usually a few hundred for SSDs and NVMe drives

# PostgreSQL tuning: WAL

- **wal\_compression**
  - Set this to on in most cases, to reduce I/O at the cost of some CPU
- **wal\_log\_hints**
  - Log hint bits in WAL. Useful for pg\_rewind, so always “on”
- **wal\_buffers**
  - The amount of shared memory used for un-written WAL data. 64MB is recommended (4 WAL files)
- **checkpoint\_completion\_target**
  - The target checkpoint completion time, as a fraction of the time between checkpoints
  - 0.5 by default prior to v14
  - 0.9 as of v14, and use that value for all Postgres versions

# PostgreSQL tuning: WAL

- `checkpoint_timeout`
  - Maximum time between checkpoints
  - Depends on the database load. Longer timeout may end up with longer recovery times, lower values may end up with more I/O (and also full page writes)
- `max_wal_size`
  - Causes a checkpoint once X MB of WAL has been written
  - Set this to a value high enough so that Postgres will checkpoint because of `checkpoint_timeout`.
  - Soft limit

# PostgreSQL tuning: Query tuning

- **seq\_page\_cost**
  - Cost of reading a page sequentially from disk
- **random\_page\_cost**
  - Cost of reading a random page from disk
  - Faster drives -> lower costs
- **cpu\_tuple\_cost**
  - Cost of processing one row (tuple) in a query
  - Start with 0.03
- **effective\_cache\_size**
  - A “hint” to the query planner, not a “reserved” space unlike `shared_buffers`
  - Usually 50% - 75% of the available RAM

# PostgreSQL tuning: Client connection defaults

- `idle_in_transaction_session_timeout`
  - Used to terminate sessions that remain idle in a transaction for too long
  - Avoids locks and maintenance issues
- `shared_preload_libraries`
  - `pg_stat_statement`: very, very useful for monitoring/tuning queries



# PostgreSQL tuning: Autovacuum

- **log\_autovacuum\_min\_duration**
  - Logs autovacuum durations
  - 0 logs all of them
- **autovacuum\_max\_workers**
  - More workers -> more frequent vacuum/analyze
  - 5 as a starting point
- **autovacuum\_vacuum\_cost\_limit**
  - Useful for throttling autovacuum/autoanalyze
  - 3000 is a good starting point.

# PostgreSQL tuning: Reporting and logging

- **log\_temp\_files**
  - Useful for logging temp files, caused by lack of work\_mem parameter.
- **log\_checkpoints**
  - Useful for processing checkpoint performance. Set to on.
- **timed\_statistics (EPAS-only)**
  - DRITA: Dynamic Runtime Instrumentation Tools Architecture
  - Set this to on.



# Fine tuning based on workload analysis

# Finding slow queries

- How to find slow queries
  - `log_min_duration_statement`
  - `pg_stat_statements`
  - `pgbadger`

# Rewriting queries

- Expressions can prevent use of indexes. Don't use:

```
SELECT * FROM t
WHERE t.a_timestamp + interval '3 days' < CURRENT_TIMESTAMP
```

Instead, use naked columns:

```
SELECT * FROM t
WHERE t.a_timestamp < CURRENT_TIMESTAMP - interval '3 days'
```

# Rewriting queries

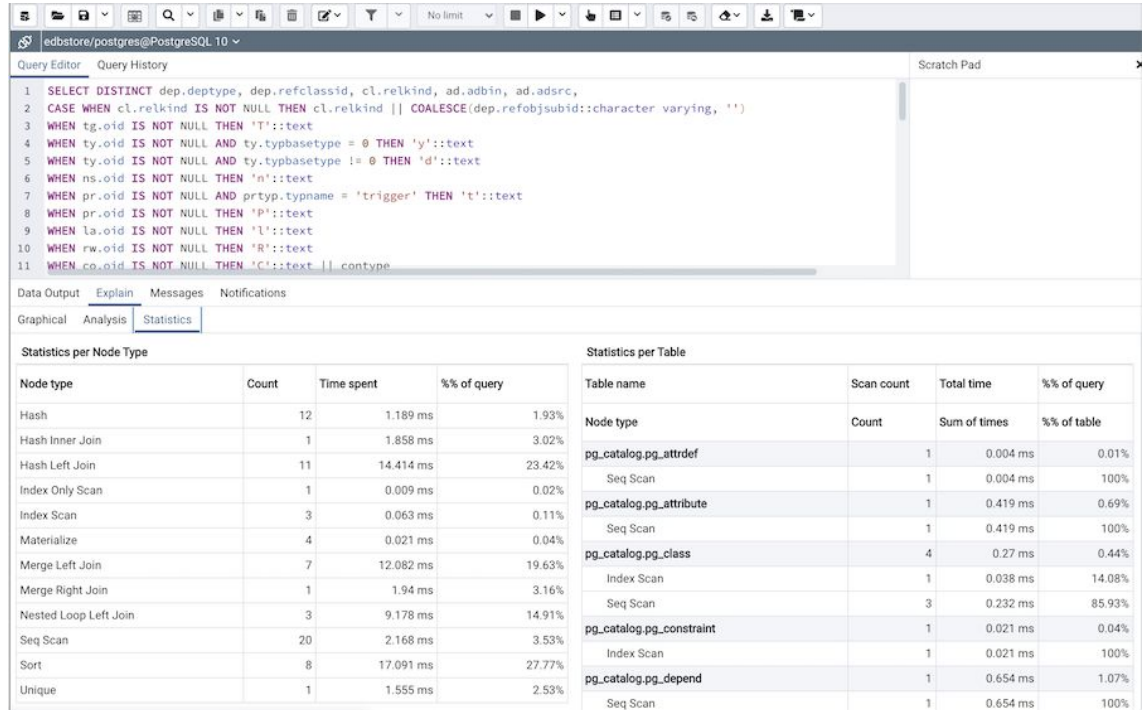
- Other patterns to review and fix:
  - `SELECT ... WHERE x NOT IN (SELECT ...)`
  - Imprecise joins in queries, "fixed" with `DISTINCT`
  - `GROUP BY` least complex types before more complex types for efficiency
  - Unnecessary use of CTEs prior to PostgreSQL 12

# EXPLAIN (ANALYZE)

- One of the best friends of a PostgreSQL DBA!
- Use it!
- Don't forget to use inside a BEGIN...ROLLBACK block :-)

# EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



```

1 SELECT DISTINCT dep.deptype, dep.refclassid, cl.relkind, ad.adbin, ad.adsrc,
2 CASE WHEN cl.relkind IS NOT NULL THEN cl.relkind || COALESCE(dep.refobjsubid::character varying, '')
3 WHEN tg.oid IS NOT NULL THEN 't'::text
4 WHEN ty.oid IS NOT NULL AND ty.typtype = 0 THEN 'y'::text
5 WHEN ty.oid IS NOT NULL AND ty.typtype != 0 THEN 'd'::text
6 WHEN ns.oid IS NOT NULL THEN 'n'::text
7 WHEN pr.oid IS NOT NULL AND pr.typtype = 'trigger' THEN 't'::text
8 WHEN pr.oid IS NOT NULL THEN 'P'::text
9 WHEN la.oid IS NOT NULL THEN 'l'::text
10 WHEN rw.oid IS NOT NULL THEN 'R'::text
11 WHEN co.oid IS NOT NULL THEN 'C'::text || contype

```

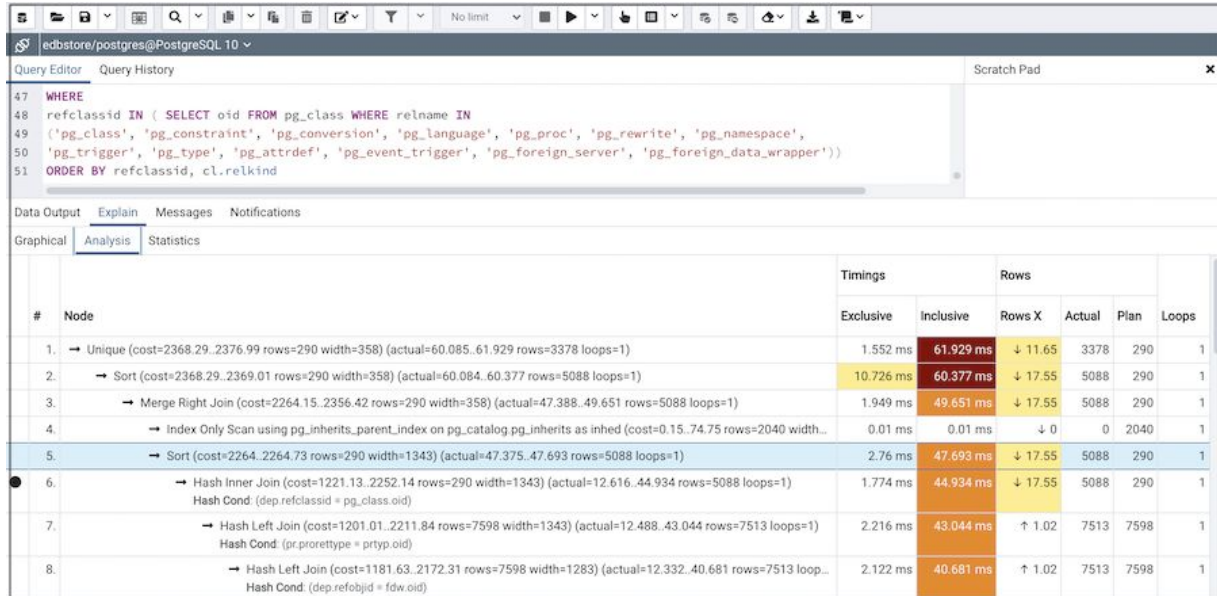
Statistics per Node Type			
Node type	Count	Time spent	% of query
Hash	12	1.189 ms	1.93%
Hash Inner Join	1	1.858 ms	3.02%
Hash Left Join	11	14.414 ms	23.42%
Index Only Scan	1	0.009 ms	0.02%
Index Scan	3	0.063 ms	0.11%
Materialize	4	0.021 ms	0.04%
Merge Left Join	7	12.082 ms	19.63%
Merge Right Join	1	1.94 ms	3.16%
Nested Loop Left Join	3	9.178 ms	14.91%
Seq Scan	20	2.168 ms	3.53%
Sort	8	17.091 ms	27.77%
Unique	1	1.555 ms	2.53%

Statistics per Table			
Table name	Scan count	Total time	% of query
<b>pg_catalog.pg_attrdef</b>			
Seq Scan	1	0.004 ms	0.01%
<b>pg_catalog.pg_attribute</b>			
Seq Scan	1	0.419 ms	0.69%
<b>pg_catalog.pg_class</b>			
Index Scan	4	0.27 ms	0.44%
Seq Scan	1	0.038 ms	14.08%
<b>pg_catalog.pg_constraint</b>			
Seq Scan	3	0.232 ms	85.93%
<b>pg_catalog.pg_depend</b>			
Index Scan	1	0.021 ms	0.04%
Seq Scan	1	0.654 ms	1.07%



# EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



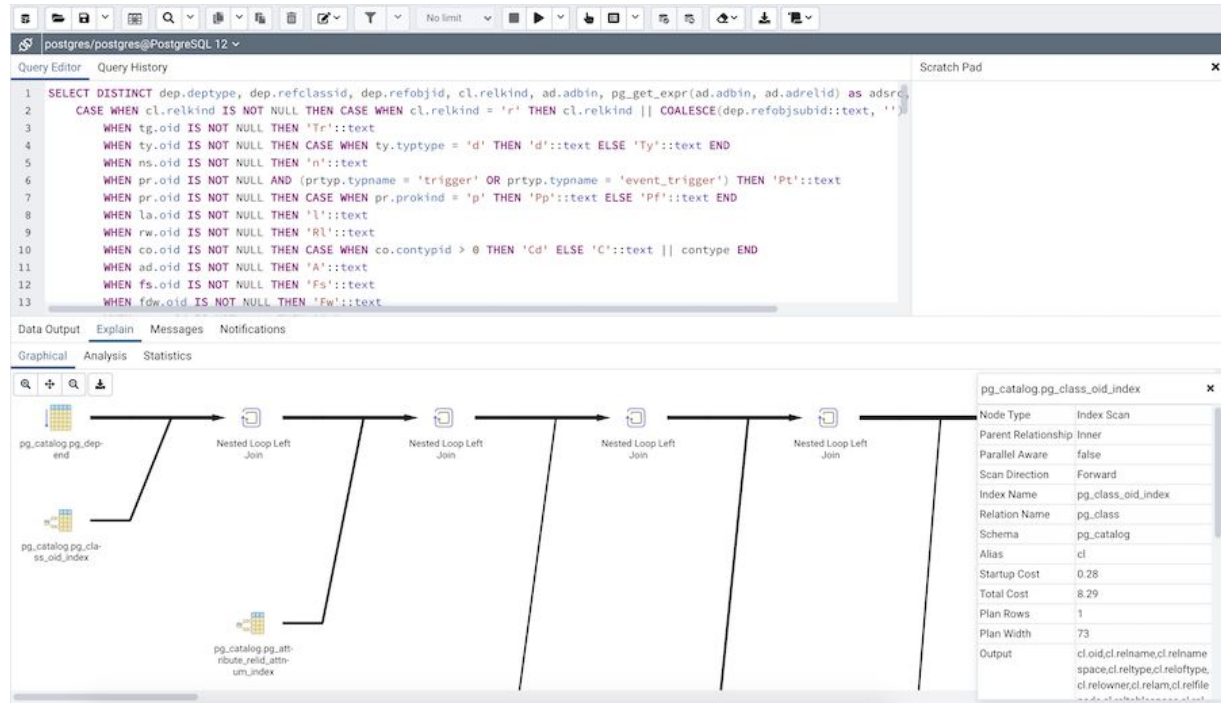
```

47 WHERE
48 reclassid IN ( SELECT oid FROM pg_class WHERE relname IN
49 ('pg_class', 'pg_constraint', 'pg_conversion', 'pg_language', 'pg_proc', 'pg_rewrite', 'pg_namespace',
50 'pg_trigger', 'pg_type', 'pg_attrdef', 'pg_event_trigger', 'pg_foreign_server', 'pg_foreign_data_wrapper'))
51 ORDER BY reclassid, cl.relkind
  
```

#	Node	Timings		Rows			
		Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Unique (cost=2368.29..2376.99 rows=290 width=358) (actual=60.085..61.929 rows=3378 loops=1)	1.552 ms	61.929 ms	↓ 11.65	3378	290	1
2.	→ Sort (cost=2368.29..2369.01 rows=290 width=358) (actual=60.084..60.377 rows=5088 loops=1)	10.726 ms	60.377 ms	↓ 17.55	5088	290	1
3.	→ Merge Right Join (cost=2264.15..2356.42 rows=290 width=358) (actual=47.388..49.651 rows=5088 loops=1)	1.949 ms	49.651 ms	↓ 17.55	5088	290	1
4.	→ Index Only Scan using pg_inherits_parent_index on pg_catalog.pg_inherits as inhed (cost=0.15..74.75 rows=2040 width=...	0.01 ms	0.01 ms	↓ 0	0	2040	1
5.	→ Sort (cost=2264.2264.73 rows=290 width=1343) (actual=47.375..47.693 rows=5088 loops=1)	2.76 ms	47.693 ms	↓ 17.55	5088	290	1
6.	→ Hash Inner Join (cost=1221.13..2252.14 rows=290 width=1343) (actual=12.616..44.934 rows=5088 loops=1) Hash Cond: (dep.reclassid = pg_class.oid)	1.774 ms	44.934 ms	↓ 17.55	5088	290	1
7.	→ Hash Left Join (cost=1201.01..2211.84 rows=7598 width=1343) (actual=12.488..43.044 rows=7513 loops=1) Hash Cond: (pr.proreftype = prtyp.oid)	2.216 ms	43.044 ms	↑ 1.02	7513	7598	1
8.	→ Hash Left Join (cost=1181.63..2172.31 rows=7598 width=1283) (actual=12.332..40.681 rows=7513 loop...) Hash Cond: (dep.refobjid = fdw.oid)	2.122 ms	40.681 ms	↑ 1.02	7513	7598	1

# EXPLAIN (ANALYZE)

- pgAdmin 4 and Postgres Enterprise Manager have a nice GUI for EXPLAIN (ANALYZE)



The screenshot displays the pgAdmin 4 interface. The top pane shows a SQL query in the Query Editor. The bottom pane shows the Explain tab, which contains a graphical execution plan. The plan consists of a series of Nested Loop Left Joins. A detail window for the pg\_catalog.pg\_class\_oid\_index is open on the right, showing its properties.

Node Type	Index Scan
Parent Relationship	Inner
Parallel Aware	false
Scan Direction	Forward
Index Name	pg_class_oid_index
Relation Name	pg_class
Schema	pg_catalog
Alias	cl
Startup Cost	0.28
Total Cost	8.29
Plan Rows	1
Plan Width	73
Output	cl.oid,cl.relname,cl.relname space,cl.reltype,cl.relotype, cl.relnum,cl.relam,cl.refille

# EXPLAIN (ANALYZE)

- Avoid, or at least try to eliminate:
  - Bad estimates
  - External sorts
  - Hash batches
  - Heap fetches
  - Lossy bitmap scans
  - Wrong plan shapes

# Partitioning

# Partitioning

- Why/when do we need partitioning?
  - Maintenance
  - Parallelization
- Use cases
- Types of partitioning in PostgreSQL
- Automatic partitioning in EPAS
  - Demo!

# Partitioning

- Why/when do we need partitioning?
  - Maintenance
  - Parallelization
- Use cases
- Types of partitioning in PostgreSQL
- Automatic partitioning in EPAS
  - Demo!

# Conclusion

# Conclusion

- Hardware, operating system and PostgreSQL are the 3 main legs of tuning
- Getting more from the database is an ongoing process
- Make use of tools such as pgAdmin, PEM, pgBadger etc.
- Each new major version adds new parameters and features
- Keep up2date with minor versions



# Questions?



Additional Reading:

[PostgreSQL Performance Tuning and Optimization](https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization)

by Vik Fearing with Devrim Gündüz and Dave Page

<https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization>

For companies committed to open source PostgreSQL and tools: new EDB Community 360 Plan (includes break/fix).

Email [community360@enterprisedb.com](mailto:community360@enterprisedb.com) for details.