PostgreSQL VACUUM vs ANALYZE

`**VACUUM**` and `**ANALYZE**` are two essential maintenance operations in PostgreSQLthat help manage data storage efficiency and query performance. They are crucialfor maintaining a healthy and performant database system.

VACUUM

VACUUM process:-

- This is the most common type of VACUUM and is performed

It reclaims space and optimizes the table by removing dead rows (rows that are no longer needed) and marking space as available for future data storage.

- It also updates table statistics,

which is important for query optimization.

```
command:-
select 'VACUUM VERBOSE ANALYZE ' ||relname|| ';' from pg_stat_all_tables where relname
not like 'pg_%' and n_dead_tup > 0;
```

1. Understanding VACUUM in PostgreSQL:-

- **Purpose**:

`**VACUUM**` reclaims storage occupied by dead tuples. In PostgreSQL, when data isupdated or deleted, the old data is not immediately removed; instead, it is marked asobsolete. This allows transactions that started before the data was updated to stillsee the old data. These obsolete rows are known as “dead tuples.”

- **How it Works:**-

❖ `**VACUUM**` scans the database tables and removes these dead tuples, making thespace available for future data inserts and updates.- It also updates the visibility map, which helps optimize future queries and vacuumoperations.

❖ **- There are two main types of `VACUUM`:**— Standard **VACUUM**: Removes dead tuples and updates statistics.— **VACUUM FULL**: Rewrites the entire table, compacting it by removing dead tuplesand reclaiming the space. This can be resource-intensive and requires a table lock,making it less suitable for use in production environments without carefulplanning.

- **Advantages:-**

❖ **Space Reclamation:**

Frees up disk space, making it available for new data.
- Prevents Transaction ID Wraparound: Regular vacuuming is critical to preventtransaction ID wraparound issues, which can lead to database corruption if notmanaged properly.
- Performance Improvement: Reduces bloat, which can significantly improveperformance by reducing the amount of data that needs to be scanned duringqueries.

Automatic VACUUM (Autovacuum)

PostgreSQL comes with an Autovacuum feature, which automatically triggersVACUUM operations based on the database activity and the configuration settings.The Autovacuum daemon continuously monitors all tables and runs VACUUM whennecessary.

Configuring Autovacuum: It is configured using parameters such as

autovacuum_vacuum_threshold,
autovacuum_vacuum_scale_factor,**autovacuum_max_workers**, and

autovacuum_vacuum_threshold:-

This is the **minimum number of dead rows** in a table before autovacuum starts.

autovacuum_vacuum_scale_factor:-

This is the **percentage of table size** (in terms of rows) that can be dead before autovacuum runs.

autovacuum_max_workers:-

This is the **maximum number of autovacuum processes** that can run at the same time across the whole database.

autovacuum_naptime:-

· It’s the **time delay (in seconds)** between two rounds of autovacuum checks.

· After one round finishes, PostgreSQL **waits this long** before checking again if any table needs cleaning.

others in the **postgresql.conf** file.

- **Advantages**:

Autovacuum helps automate the maintenance process withoutmanual intervention.

Challenges: Default settings might not be optimal for all workloads. For high-traffic tables, you may need to fine-tune these settings.

Example: Configuring Autovacuum
-- Adjust the autovacuum settings for a specific tableALTER TABLE your_table_name SET (autovacuum_vacuum_threshold = 50, autovacuum_vacuum_scale_factor = 0.2);

2.2 Manual VACUUM and Scheduling

While Autovacuum is useful, there are scenarios where manual VACUUMcommands are necessary:

- **Heavy Update/Delete Operations**:

After large data modifications, runningVACUUM ANALYZE manually can help in reclaiming storage quickly andupdating the statistics used by the query planner.

- **Scheduled Maintenance**:

For databases with predictable workloads, schedulingVACUUM during off-peak hours can prevent any potential performancedegradation during busy times.

Example: Running Manual VACUUM

-- Reclaim storage and update planner statistics for a tableVACUUM ANALYZE your_table_name;

VACUUM FULL

VACUUM FULL Process:-

- This is the most resource-intensive type of VACUUM. It reclaims space and compacts the table by creating a new copy of the table and its indexes, then swapping them with the old ones.
- It is used when significant space reclamation is needed but should be used sparingly due to its impact on database performance and downtime.

```
command:-
select 'VACUUM FULL VERBOSE ANALYZE ' ||relname|| ';' from pg_stat_all_tables where relname not like 'pg_%' and n_dead_tup > 0;
```

Note:-
In your case, with an 88 GB database,
you should ensure that you have at least 88 GB of free disk space available on the partition where your PostgreSQL data directory resides.

select relname, n_live_tup, n_dead_tup

2.3 Using VACUUM FULL for Database Optimization:-

- **VACUUM FULL** should be used cautiously as it locks tables. It is suitable for tablesthat have accumulated significant bloat due to frequent updates or deletes but arenot accessed regularly.

- **When to Use VACUUM FULL**:

Use it during maintenance windows or forarchiving tables.

Drawbacks: It can cause long downtime on large tables.

- **Example:**

Using VACUUM FULL
-- Perform a full vacuum to reclaim storage and compact the tableVACUUM FULL your_table_name;

3.Monitoring and Optimizing VACUUM Performance

To ensure VACUUM operations run efficiently, it’s crucial to monitor theirperformance and adjust configurations as necessary.

3.1 Monitoring VACUUM with PostgreSQL Logs

PostgreSQL provides logging options to monitor VACUUM activities. Enabling**log_autovacuum_min_duration** helps log all **autovacuum** runs that take longer thanthe specified duration.

-- Set logging for autovacuum durationSET log_autovacuum_min_duration = 1000; -- Log autovacuum processes that take more than 1 second

ANALYZE

- **Purpose**

1. ANALYZE helps PostgreSQL understand what kind of data is inside your tables.

2. With this knowledge, the database can choose the fastest way to run your queries.

� **Example:** Like a chef checking what ingredients are in the kitchen before deciding what dish to cook.

- How it Works

- It takes a small sample of rows from a table.

- It checks things like:

1. Which values are most common

2. How many unique values exist

3. How the data is spread out

Then it saves this info in PostgreSQL’s system catalog (a kind of internal memory).

� Example: Like Google Maps sampling live traffic data to see where the jams are.

- **Advantages (in simple words)**

- Faster queries – PostgreSQL knows the “shortcuts” to answer your query.

- Better decisions – The planner can pick the most efficient way to join tables or filter data.

- Saves resources – Reduces CPU, memory, and disk usage by avoiding bad query plans.

- Keeps performance consistent – Queries won’t suddenly get slow because of outdated statistics.

Advantages:-
Improved Query Performance: By providing the query planner with accurate datadistribution statistics, `**ANALYZE**` helps the planner choose the most efficientexecution plans, leading to faster query performance.

- Optimal Index Usage: Helps in the effective use of indexes, as the planner canbetter understand the selectivity of indexed columns.

When is VACUUM and ANALYZE Important?

1. **Regular Maintenance:**—

Regularly running `**VACUUM**` and `**ANALYZE**` is crucial for maintaining databaseperformance and preventing data bloat. Many PostgreSQL installations use`autovacuum`, an automated process that periodically runs these operations.**— Autovacuum:** Autovacuum automatically performs `**VACUUM**` and `**ANALYZE**` ontables that need maintenance, based on thresholds related to the number of tuplesupdated or deleted. It's essential to ensure that `**autovacuum**` settings areappropriately configured for your workload.

2. **After Large Data Changes:**—

After bulk inserts, updates, or deletes, it is often necessary to run `**VACUUM**` and`**ANALYZE**` to reclaim space and update statistics. This ensures that the database

remains efficient and queries continue to perform well.— For instance, after a batch update that affects many rows, running `**VACUUMANALYZE**` can help reclaim space and provide fresh statistics to the query planner.

3. **Before Query Optimization:**—

Before optimizing queries, it is advisable to run `ANALYZE` to ensure that theplanner has up-to-date statistics. This can significantly impact the planner's abilityto choose the most efficient query execution plan.

4. **Preventing Transaction ID Wraparound:**—

PostgreSQL uses a 32-bit counter for transaction IDs, and if this counter wrapsaround, it can lead to data corruption. Regular `**VACUUM**`ing of tables is necessaryto prevent this by advancing the "oldest transaction ID" in the system.

CLUSTER Operations

- The CLUSTER command physically reorganizes a table according to an index. Subsequent data entries may not follow this order, and the operation locks the table, preventing access until complete.

- **CLUSTER Examples:-**

```
create table testcluster (id int, name text);
CREATE TABLE

insert into testcluster values(2,'B'),(1,'A'),(3,'A'),(5,'C'),(4,'B');
INSERT 0 5

select * from testcluster;
 id | name
----+------
  2 | B
  1 | A
  3 | A
  5 | C
  4 | B
(5 rows)

create index idx_id on testcluster(id);
CREATE INDEX

postgres=# cluster testcluster using idx_id;
CLUSTER

postgres=# select * from testcluster;
 id | name
----+------
  1 | A
  2 | B
  3 | A
  4 | B
  5 | C
(5 rows)
```

- **Monitoring CLUSTER Status**

```
SELECT * FROM pg_stat_progress_cluster;
```

REINDEX Operations

- REINDEX rebuilds and updates old indexes. It is useful in the following situations:

1. If index data is corrupted or damaged.
2. To clean up indexes with too many invalid and empty entries.
3. To apply parameter changes related to the index.
4. To correct a previously failed index operation.

- **REINDEX Examples**

```
REINDEX INDEX index_name;
REINDEX TABLE table_name;
REINDEX SCHEMA schema_name;
REINDEX DATABASE db_name;
REINDEX TABLE CONCURRENTLY cs_agreement;
REINDEX (verbose, CONCURRENTLY, TABLESPACE pg_default) DATABASE db_name;
REINDEX (verbose) SYSTEM;
```

Statistical Data

- Gathers information about database objects, such as access frequency, how often the data block is found in the cache, and the frequency of disk reads. This data is stored in relevant statistics tables.

```
pg_stat_activity
pg_stat_replication
pg_stat_user_tables
pg_stat_user_indexes
pg_statio_user_tables
pg_statio_user_indexes
```

Table Statistics

Table statistics are used by the query planner to generate the most efficient plan. Database content statistics are stored in the pg_statistic catalog. The ANALYZE command updates these statistics.

Lock Mechanism

- The lock mechanism ensures data integrity during concurrent read/write operations in the database. PostgreSQL uses page-level locking to control access to pages in the shared_buffer.

Starting a Transaction

```
postgres=# create table emp (salary int , number int);
CREATE TABLE

postgres=# insert into emp (salary, number) values (1000, 12);
INSERT 0 1

postgres=# Begin;
BEGIN

postgres=*# update emp set salary=10 where number=12;
UPDATE 1

postgres=*# commit;
COMMIT
```

Monitoring Locks

```
SELECT * FROM pg_locks pl LEFT JOIN pg_stat_activity psa ON pl.pid = psa.pid;
SELECT a.datname, c.relname, l.transactionid, l.mode, l.GRANTED, a.usename, a.query,
a.query_start, age(now(), a.query_start) AS "age"
FROM pg_stat_activity a
JOIN pg_locks l ON l.pid = a.pid
JOIN pg_class c ON c.oid = l.relation
ORDER BY a.query_start;
```

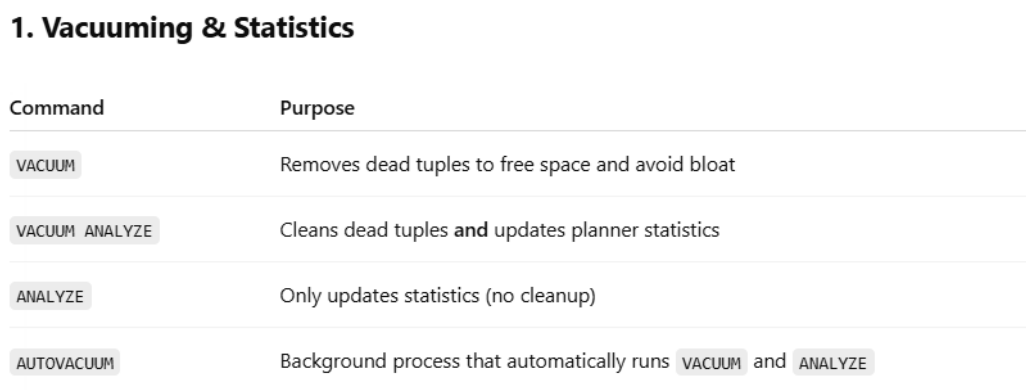
-- Also, you can see idle in transaction

```
watch -n 2 "psql -c \"SELECT pid, datname, usename, state, query, state_change FROM
pg_stat_activity WHERE state = 'idle in transaction';\""
```

Session Kill

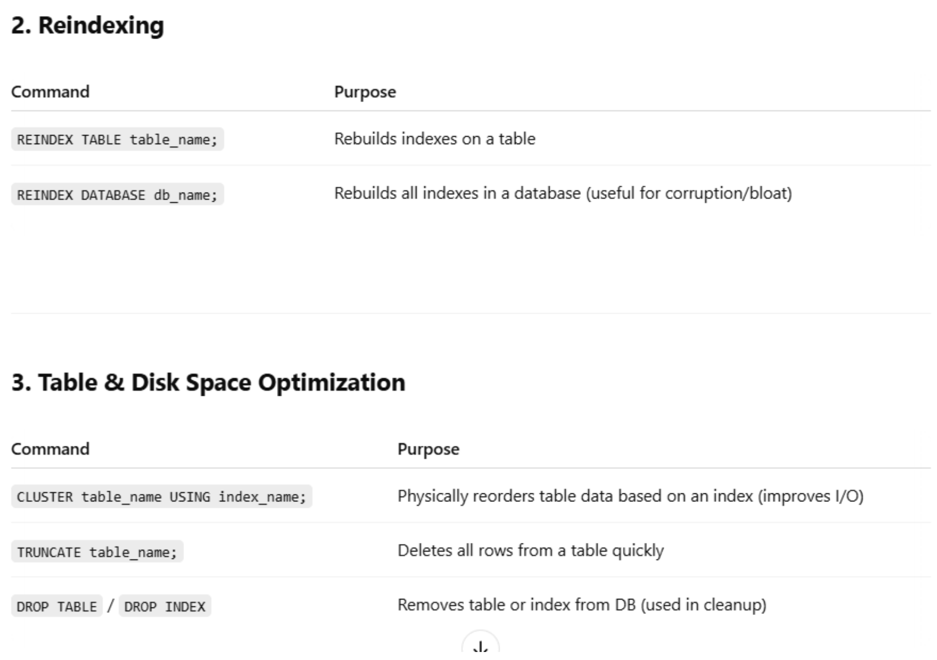
```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE pid <> pg_backend_pid() -- don't kill your connection!
AND datname = 'exampledb'; -- don't kill the connections to other databases
```

These maintenance operations will make your PostgreSQL database more efficient and secure. For more detailed and technical articles like this, keep following our blog on Medium. If you have any questions or need further assistance, feel free to reach out in the comments below and directly.



1. Vacuuming & Statistics

Command	Purpose
`VACUUM`	Removes dead tuples to free space and avoid bloat
`VACUUM ANALYZE`	Cleans dead tuples **and** updates planner statistics
`ANALYZE`	Only updates statistics (no cleanup)
`AUTOVACUUM`	Background process that automatically runs `VACUUM` and `ANALYZE`

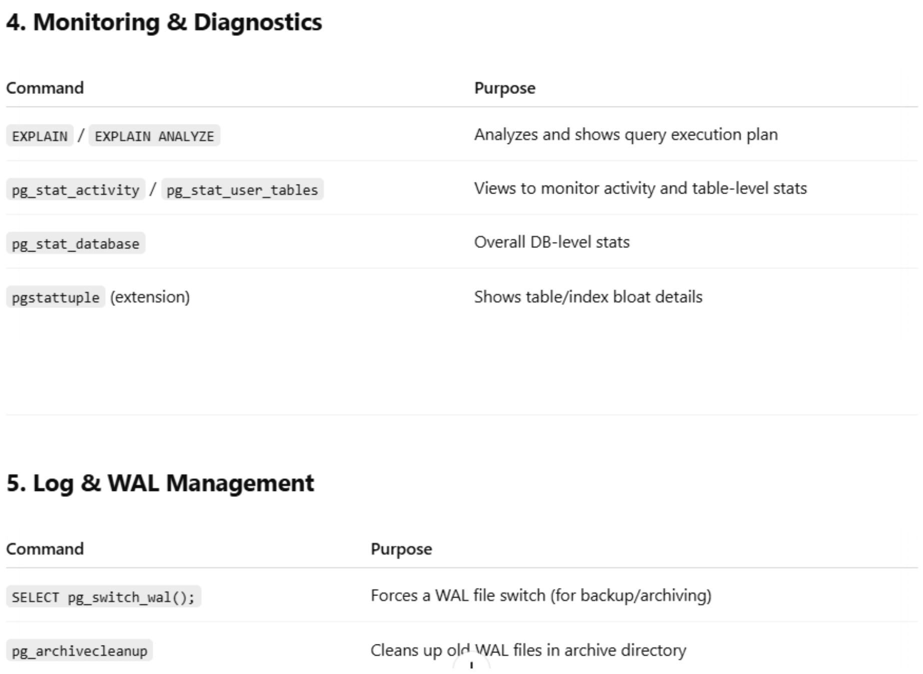


2. Reindexing

Command	Purpose
`REINDEX TABLE table_name;`	Rebuilds indexes on a table
`REINDEX DATABASE db_name;`	Rebuilds all indexes in a database (useful for corruption/bloat)

3. Table & Disk Space Optimization

Command	Purpose
`CLUSTER table_name USING index_name;`	Physically reorders table data based on an index (improves I/O)
`TRUNCATE table_name;`	Deletes all rows from a table quickly
`DROP TABLE` / `DROP INDEX`	Removes table or index from DB (used in cleanup)



4. Monitoring & Diagnostics

Command	Purpose
`EXPLAIN` / `EXPLAIN ANALYZE`	Analyzes and shows query execution plan
`pg_stat_activity` / `pg_stat_user_tables`	Views to monitor activity and table-level stats
`pg_stat_database`	Overall DB-level stats
`pgstattuple` (extension)	Shows table/index bloat details

5. Log & WAL Management

Command	Purpose
`SELECT pg_switch_wal();`	Forces a WAL file switch (for backup/archiving)
`pg_archivecleanup`	Cleans up old WAL files in archive directory

Conclusion

`**VACUUM**` and `**ANALYZE**` are critical for maintaining PostgreSQL performanceand stability. `**VACUUM**` reclaims space and prevents transaction ID wraparound,while `**ANALYZE**` updates statistics crucial for query planning. Regularly runningthese operations, either manually or through `**autovacuum**`, helps ensure that thedatabase remains efficient and performant, especially after significant datamodifications. Proper maintenance using `**VACUUM**` and `**ANALYZE**` is essentialfor any production PostgreSQL environment.

How Dead Tuples Affect PostgreSQL Performance & How to Fix Them

In PostgreSQL, dead tuples are old row versions left behind after UPDATE and DELETE operations. Because of MVCC (Multi-Version Concurrency Control), PostgreSQL doesn't immediately remove old data—it keeps them until VACUUM reclaims the space.

▯ How Dead Tuples Hurt Performance

1▯ Increased Disk Usage → More storage is used as tables grow unnecessarily.
2▯ Slow Queries → Sequential scans & index scans take longer due to bloated tables.
3▯ Inefficient Indexes → Indexes also store dead tuples, slowing down lookups.
4▯ Autovacuum Delays → If autovacuum isn't tuned well, dead tuples accumulate, making cleanup slower and more resource-intensive.

▯ How to Identify Dead Tuples

Run this query to check dead tuples in your tables:

```
SELECT relname, n_live_tup, n_dead_tup, last_autovacuum
FROM pg_stat_user_tables
ORDER BY n_dead_tup DESC;
```

□ If n_dead_tup is high, vacuuming is needed!

For a deeper check, use the pgstattuple extension:

```
CREATE EXTENSION IF NOT EXISTS pgstattuple;
SELECT relname,
pg_size_pretty(pg_total_relation_size(relid)) AS table_size,
(100 * (pgstattuple(relid)).dead_tuple_percent) AS dead_tuple_percentage
FROM pg_stat_user_tables
ORDER BY dead_tuple_percentage DESC;
```

□ If dead tuple percentage > 10%, table bloat is affecting performance.

□ How to Fix Dead Tuple Issues

1□ Run Manual Vacuum & Index Cleanup

□ Standard cleanup (safe to run anytime):

```
VACUUM ANALYZE;
```

□ Aggressive cleanup (locks the table, use in maintenance windows):

```
VACUUM FULL my_table;
```

□ Rebuild indexes to remove bloat from indexes:

```
REINDEX TABLE my_table;
```

2□ Optimize Autovacuum for Better Performance

Modify postgresql.conf to prevent dead tuple buildup:

```
autovacuum_vacuum_scale_factor = 0.05
autovacuum_max_workers = 5
autovacuum_naptime = 30s
autovacuum_vacuum_cost_limit = 2000
autovacuum_vacuum_cost_delay = 5ms
```

□ This ensures vacuum runs more frequently and efficiently!

□ Monitor & Automate Dead Tuple Cleanup

Enable autovacuum logging for better monitoring:

```
log_autovacuum_min_duration = 0
```

Track running autovacuum processes:

```
SELECT pid, age(datfrozenxid), relname, state, query
FROM pg_stat_activity
WHERE query LIKE 'autovacuum%';
```

▯ Final Takeaways

▯ Dead tuples slow down queries, increase disk usage, and bloat indexes.
▯ Use VACUUM ANALYZE regularly to keep tables optimized.
▯ Tune autovacuum settings to clean up dead tuples before they become a problem.
▯ Monitor & automate vacuuming to maintain a high-performance PostgreSQL database.