Important PostgreSQL Parameters: Understanding Their Importance and Recommended Values

Have you ever experienced your database slowing down as the amount of data increases? If so, one important factor to consider is tuning PostgreSQL parameters to match your specific workload and requirements.

PostgreSQL has many parameters because it is designed to be highly flexible and customizable to meet a wide range of use cases and workloads. Each parameter allows you to fine-tune different aspects of the database, such as memory management, query optimization, connection handling, and more. This flexibility helps database administrators to optimize performance based on hardware resources, workload requirements, and specific business needs.

In this blog, I will cover some of the important PostgreSQL parameters, explain their role, and provide recommended values to help you fine-tune your database for better performance and scalability.

Disclaimer: The recommended PostgreSQL parameter values in this blog serve as a starting point for tuning. The optimal values depend on multiple factors, including workload type, hardware specifications, and database size. Always analyze your system's performance and adjust these settings based on real-world usage and monitoring.

Memory-Related Parameters

Memory-related parameters in PostgreSQL control how the database allocates and manages memory. Tuning these settings is important for improving query performance and preventing resource bottlenecks.

Name: work_mem
Description: Sets the maximum amount of memory used by internal operations like sorts and hashes before writing to disk. Increasing it can improve performance for complex queries
Default: 4MB
Recommended: Typically, setting work_mem to 1-2% of the total system's available memory is recommended, i.e., if the total system memory is 256 GB, assign 3 to 5 GB for work_mem.
Note: This may lead to higher memory usage for operations that involve sorting.

Name: shared_buffers
Description: Determines the amount of memory allocated for caching database data.
Default: 128MB
Recommendation: Typically, setting shared_buffers to 25-40% of the total system memory is recommended, i.e., if the total system memory is 256 GB, assign 64-102 GB for shared_buffers.

Name: maintenance_work_mem
Description: Specifies the amount of memory used for maintenance operations like VACUUM, CREATE INDEX, and ALTER TABLE. Increasing it can speed up these operations.
Default: 64MB
Recommendation: it's recommended to set 5-10% of the total system memory, i.e., if the total system memory is 256 GB, assign 13 to 26 GB for maintenance_work_mem.

`effective_cache_size`

Tells PostgreSQL how much memory the OS will use for caching.

Tip: Set to about 50%-75% of available memory.

Parallelism-Related Parameters

Parallelism-related parameters in PostgreSQL control how tasks are divided and processed across multiple CPU cores. These settings are important for improving query performance by enabling faster data processing and reducing execution time.

Name: max_parallel_maintenance_workers
Description: Sets the maximum number of parallel workers that can be used for maintenance operations like VACUUM and CREATE INDEX. Increasing it can speed up these operations.
Default: 2
Recommendation: If you have 48 cores, you can set this to 12% of the available cores, i.e., assign 6 cores for parallel maintenance operations such as VACUUM or CREATE INDEX.

Name: max_parallel_workers
Description: Defines the maximum number of parallel workers that can be used for any query execution. such as parallel scans, joins, and aggregates
Default: 8
Recommendation: If you have 48 cores, it's recommended to set max_parallel_workers to 75% of the available cores, i.e., assign 36 cores for parallel query execution.

Name: max_parallel_workers_per_gather
Description: Controls the maximum number of parallel workers used for a single query operation (like a JOIN or SCAN). It determines how many workers can be assigned to a single part of a query, improving performance for resource-heavy queries.
Default: 2
Recommendation: If you have 48 cores, it's recommended to set 1/6 of the total available CPU cores, i.e., assign 8 cores per parallel query operation.

Name: max_worker_processes
Description: Defines the maximum number of background worker processes that the PostgreSQL server can use for various tasks, such as parallel query execution, logical replication, and other maintenance tasks.
Default: 8
Recommendation: If you have 48 cores, you can set it to 100% of the available cores, i.e., assign all 48 cores for max_worker_processes.

JIT-Related Parameters

JIT-related parameters in PostgreSQL manage the Just-In-Time compilation of queries to improve execution speed. These settings are important for optimizing query performance by converting queries into machine code, making them run faster.

Name: jit
Description: Enables or disables Just-In-Time (JIT) compilation for queries, improving performance for complex queries by compiling parts of a query into machine code at runtime.
Default: on
Recommendation: Enable (on) if queries are CPU-bound.

Note with JIT operations: Setting JIT parameters too aggressively or too low can have drawbacks. Lower values enable more JIT compilation, while higher values keep it selective. So, if set too low, unnecessary parts of the query may be compiled, potentially increasing execution time instead of improving performance.

Name: jit_above_cost
Description: Sets the minimum query cost threshold for which JIT compilation will be considered. Queries

above this cost will be JIT-compiled.
Default: 100000
Recommendation: typically 100,000 is a good starting number increase this number if queries are becoming more CPU-bound and include complex operations.

Name: jit_inline_above_cost
Description: Sets the minimum cost threshold above which function calls in queries will be inlined when JIT is enabled, reducing function call overhead. Inlining means the function is merged directly into the query.
Default: 500000
Recommendation: 500,000 is a good starting point; increase it for more selective JIT use or decrease it to apply JIT to more general functions.

Name: jit_optimize_above_cost
Description: Sets the cost threshold above which JIT-compiled functions will undergo additional optimization.
Default: 500000
Recommendation: 500,000 is a good starting point; increase it for more selective JIT use or decrease it to apply JIT to more general functions.

Connection-Related Parameters

Connection-related parameters in PostgreSQL manage how client applications connect to the database. Tuning these settings is crucial for handling large numbers of connections efficiently and ensuring stable performance under varying loads.

Name: max_connections
Description: Specifies the maximum number of concurrent connections allowed to the PostgreSQL database.
Default: 100
Recommendation: If your system has 48 cores, the recommended setting for max_connections would be (Number of cores * 3) * 2, i.e., (48 * 3) * 2 = 288 connections.
Note: If your requirements significantly exceed the recommended value, it's better to configure a connection pooler to meet your business needs effectively.

Name: idle_in_transaction_session_timeout
Description: Specifies the amount of time (in milliseconds) a session can remain idle in a transaction before it is automatically terminated.
Default: 0 (disabled)
Recommendation: Set a reasonable timeout based on workload requirements to avoid blocking issues caused by idle transactions. For most applications, a value between 30000 (30 seconds) to 60000 (1 minute) is a good starting point.

Name: idle_session_timeout
Description: Specifies the duration (in milliseconds) a session can remain idle (not executing queries) before it is automatically terminated. This helps free up system resources and prevents inactive connections from lingering indefinitely.
Default: 0 (disabled)
Recommendation: Set a timeout based on expected user activity to prevent unnecessary resource consumption. A value between 300000 (5 minutes) and 1800000 (30 minutes) is typically suitable for most environments.

Autovacuum parameters

We were working on a production environment, and we noticed that the customer had turned off the autovacuum process. So, naturally, we asked, "Why did you disable it?" And they replied, "Well, we saw

autovacuum was eating up so many resources, running forever, and making the system slow, so we decided to turn it off."

Now, here's the thing this approach isn't quite the solution. Autovacuum isn't the villain here. If it's taking too many resources, it's actually trying to clean up the mess. In our case, we figured out that the system was doing a write-heavy workload and generating lots of dead tuples. The default autovacuum settings just couldn't keep up with the load. Instead of disabling it, all you need to do is tune those autovacuum settings to match your system's needs and let it work its magic!

We have previously published an article on best practices for autovacuum parameters; you can read it here.

Before considering an upgrade to your system resources, it's worth focusing on tuning your database first. Fine-tuning your database settings can significantly enhance performance, often providing a more cost-effective solution without the need for extra hardware.

Important PostgreSQL Parameters: Understanding Their Importance and Recommended Values

Have you ever experienced your database slowing down as the amount of data increases? If so, one important factor to consider is tuning PostgreSQL parameters to match your specific workload and requirements.

PostgreSQL has many parameters because it is designed to be highly flexible and customizable to meet a wide range of use cases and workloads. Each parameter allows you to fine-tune different aspects of the database, such as memory management, query optimization, connection handling, and more. This flexibility helps database administrators to optimize performance based on hardware resources, workload requirements, and specific business needs.

In this blog, I will cover some of the important PostgreSQL parameters, explain their role, and provide recommended values to help you fine-tune your database for better performance and scalability.

Disclaimer: The recommended PostgreSQL parameter values in this blog serve as a starting point for tuning. The optimal values depend on multiple factors, including workload type, hardware specifications, and database size. Always analyze your system's performance and adjust these settings based on real-world usage and monitoring.

Memory-Related Parameters

Memory-related parameters in PostgreSQL control how the database allocates and manages memory. Tuning these settings is important for improving query performance and preventing resource bottlenecks.

Name: work_mem
Description: Sets the maximum amount of memory used by internal operations like sorts and hashes before writing to disk. Increasing it can improve performance for complex queries
Default: 4MB
Recommended: Typically, setting work_mem to 1-2% of the total system's available memory is recommended, i.e., if the total system memory is 256 GB, assign 3 to 5 GB for work_mem.
Note: This may lead to higher memory usage for operations that involve sorting.

Name: shared_buffers
Description: Determines the amount of memory allocated for caching database data.
Default: 128MB
Recommendation: Typically, setting shared_buffers to 25-40% of the total system memory is recommended, i.e., if the total system memory is 256 GB, assign 64-102 GB for shared_buffers.

Name: maintenance_work_mem
Description: Specifies the amount of memory used for maintenance operations like VACUUM, CREATE INDEX, and ALTER TABLE. Increasing it can speed up these operations.
Default: 64MB
Recommendation: it's recommended to set 5-10% of the total system memory, i.e., if the total system memory is 256 GB, assign 13 to 26 GB for maintenance_work_mem.

Parallelism-Related Parameters

Parallelism-related parameters in PostgreSQL control how tasks are divided and processed across multiple CPU cores. These settings are important for improving query performance by enabling faster data processing and reducing execution time.

Name: max_parallel_maintenance_workers
Description: Sets the maximum number of parallel workers that can be used for maintenance operations like VACUUM and CREATE INDEX. Increasing it can speed up these operations.
Default: 2
Recommendation: If you have 48 cores, you can set this to 12% of the available cores, i.e., assign 6 cores for parallel maintenance operations such as VACUUM or CREATE INDEX.

Name: max_parallel_workers
Description: Defines the maximum number of parallel workers that can be used for any query execution. such as parallel scans, joins, and aggregates
Default: 8
Recommendation: If you have 48 cores, it's recommended to set max_parallel_workers to 75% of the available cores, i.e., assign 36 cores for parallel query execution.

Name: max_parallel_workers_per_gather
Description: Controls the maximum number of parallel workers used for a single query operation (like a JOIN or SCAN). It determines how many workers can be assigned to a single part of a query, improving performance for resource-heavy queries.
Default: 2
Recommendation: If you have 48 cores, it's recommended to set 1/6 of the total available CPU cores, i.e., assign 8 cores per parallel query operation.

Name: max_worker_processes
Description: Defines the maximum number of background worker processes that the PostgreSQL server can use for various tasks, such as parallel query execution, logical replication, and other maintenance tasks.
Default: 8
Recommendation: If you have 48 cores, you can set it to 100% of the available cores, i.e., assign all 48 cores for max_worker_processes.

JIT-Related Parameters

JIT-related parameters in PostgreSQL manage the Just-In-Time compilation of queries to improve execution speed. These settings are important for optimizing query performance by converting queries into machine code, making them run faster.

Name: jit
Description: Enables or disables Just-In-Time (JIT) compilation for queries, improving performance for complex queries by compiling parts of a query into machine code at runtime.
Default: on
Recommendation: Enable (on) if queries are CPU-bound.

Note with JIT operations: Setting JIT parameters too aggressively or too low can have drawbacks. Lower values enable more JIT compilation, while higher values keep it selective. So, if set too low, unnecessary

parts of the query may be compiled, potentially increasing execution time instead of improving performance.

Name: jit_above_cost
Description: Sets the minimum query cost threshold for which JIT compilation will be considered. Queries above this cost will be JIT-compiled.
Default: 100000
Recommendation: typically 100,000 is a good starting number increase this number if queries are becoming more CPU-bound and include complex operations.

Name: jit_inline_above_cost
Description: Sets the minimum cost threshold above which function calls in queries will be inlined when JIT is enabled, reducing function call overhead. Inlining means the function is merged directly into the query.
Default: 500000
Recommendation: 500,000 is a good starting point; increase it for more selective JIT use or decrease it to apply JIT to more general functions.

Name: jit_optimize_above_cost
Description: Sets the cost threshold above which JIT-compiled functions will undergo additional optimization.
Default: 500000
Recommendation: 500,000 is a good starting point; increase it for more selective JIT use or decrease it to apply JIT to more general functions.

Connection-Related Parameters

Connection-related parameters in PostgreSQL manage how client applications connect to the database. Tuning these settings is crucial for handling large numbers of connections efficiently and ensuring stable performance under varying loads.

Name: max_connections
Description: Specifies the maximum number of concurrent connections allowed to the PostgreSQL database.
Default: 100
Recommendation: If your system has 48 cores, the recommended setting for max_connections would be (Number of cores * 3) * 2, i.e., (48 * 3) * 2 = 288 connections.
Note: If your requirements significantly exceed the recommended value, it's better to configure a connection pooler to meet your business needs effectively.

Name: idle_in_transaction_session_timeout
Description: Specifies the amount of time (in milliseconds) a session can remain idle in a transaction before it is automatically terminated.
Default: 0 (disabled)
Recommendation: Set a reasonable timeout based on workload requirements to avoid blocking issues caused by idle transactions. For most applications, a value between 30000 (30 seconds) to 60000 (1 minute) is a good starting point.

Name: idle_session_timeout
Description: Specifies the duration (in milliseconds) a session can remain idle (not executing queries) before it is automatically terminated. This helps free up system resources and prevents inactive connections from lingering indefinitely.
Default: 0 (disabled)
Recommendation: Set a timeout based on expected user activity to prevent unnecessary resource consumption. A value between 300000 (5 minutes) and 1800000 (30 minutes) is typically suitable for most environments.

Autovacuum parameters

We were working on a production environment, and we noticed that the customer had turned off the autovacuum process. So, naturally, we asked, “Why did you disable it?” And they replied, “Well, we saw autovacuum was eating up so many resources, running forever, and making the system slow, so we decided to turn it off.”

Now, here’s the thing this approach isn’t quite the solution. Autovacuum isn’t the villain here. If it’s taking too many resources, it’s actually trying to clean up the mess. In our case, we figured out that the system was doing a write-heavy workload and generating lots of dead tuples. The default autovacuum settings just couldn’t keep up with the load. Instead of disabling it, all you need to do is tune those autovacuum settings to match your system’s needs and let it work its magic!

We have previously published an article on best practices for autovacuum parameters; you can read it here.

Before considering an upgrade to your system resources, it’s worth focusing on tuning your database first. Fine-tuning your database settings can significantly enhance performance, often providing a more cost-effective solution without the need for extra hardware.