How to Prepare PostgreSQL Databases for Black Friday and Cyber Monday

Recommendations for PostgreSQL engineers working in e-commerce companies.
INCLUDED: Timeline and Checklist

Postgres.ai

About Postgres.ai

Postgres.ai helps companies maintain and optimize PostgreSQL databases in clouds and on-premise. Our customers benefit from our decades of experience in successfully scaling PostgreSQL in heavily loaded and rapidly growing projects.

We have a strong focus on performance and automation. If your engineers need to perform manual, repetitive, or even boring tasks, we can help. Waiting for long processes, like copying terabytes over the network, or not having access to the data when it's really needed; all can lead to development failures, slow down all essential processes, decrease time-to-market, and negatively affect your business.

To solve these problems, Postgres.ai invests in building open-source tools that boost the development processes for a company, lower budgetary costs for non-production infrastructure, and drastically improve time-to-market and product quality:

- Database Lab Engine: clone databases of any size *in just seconds* to boost development and testing
- Joe bot: a Postgres query optimization assistant, the best way for database developers to optimize SQL (it uses ultra-fast independent clones provided by Database Lab)
- postgres-checkup: automated PostgreSQL health checks

These tools are the building blocks of our SaaS offering, **Database Lab Platform**, that provides GUI, CLI, API, fine-grained user management, and advanced features for backend engineers, SQL developers, DBAs, and QA engineers.

Learn more about **Database Lab**: https://postgres.ai/docs/

Participate in our Private Beta program: https://postgres.ai/console/

Reach out to the Postgres.ai team:
team@postgres.ai
+1-650-441-6163

Table of Contents

Special thanks to Alexander Kukushkin, Database Engineer at Zalando SE and developer of Patroni (a popular template for Postgres HA), for reviewing this paper and invaluable comments.

Introduction

In e-commerce, some days and some weeks are very special each year. Black Friday and Cyber Monday (BFCM) in the USA, Boxing Day in Great Britain, Singles Day in China – these are just a few examples of major online shopping days in various countries. Such days are special for e-commerce companies: they try to squeeze as much as possible from the increased activity of customers, selling significantly more goods and services than usual.

For technical staff, and for database experts specifically, it implies additional responsibilities. In most cases, traffic and overall loads to the systems are increased. Any performance degradations or downtime risks are especially painful and need to be fully prevented to avoid direct and indirect losses for businesses.

PostgreSQL has been used to build mission-critical systems in large e-commerce companies for many years. Companies of any size trust PostgreSQL to work with essential data and operations, such as Customer Data, information about Orders, actions with Cart, or transactions during Checkout.

In this paper, we start by describing how a year-long cycle in a mid- or large-size e-commerce company would look like. Then we discuss the main tasks for DBAs/DBREs and other experts dealing with PostgreSQL in an e-commerce company. Finally, we provide a checklist that can be useful for database experts preparing for peak seasons at any point in time.

With certain corrections, the recommendations provided in this paper can be useful to any other types of companies where large customer activity spikes are common (example: marketing campaigns causing increased traffic).

DBA’s year in an e-commerce company

Some e-commerce companies have yearly development life cycles. Here are some basic examples for an abstract US e-commerce company that benefits from Black Friday (in 2020, it's November 26), and subsequent Cyber Monday (November 30 in 2020). You might want to adjust it for your location, calendar, and industry.

- **January and February:** Thoroughly plan big projects, prepare to build a good foundation for the new cycle.
- **March and April:** Time to work on scalability and improve infrastructure. It's time for big steps.
- **May and June:** Continue all the big moves started in previous months, improve observability (monitoring and troubleshooting tooling) and, reliability (backups and DR strategy, HA).

- **July and August:** Finish all big projects, perform performance optimization, and start load testing.
- **September:** Continue with load testing, performance, and scalability optimization.
- **October:** More load testing, natural growth of traffic. Review plans for the peak season, including elevated attention to all the systems, readiness for any incidents: train staff and practice troubleshooting. This is also time to perform final health checks, double-checking the effects of optimization steps.
- **November:** Feature freeze, no changes can be deployed. All is ready for high traffic during Black Friday and Cyber Monday, which may be 3-10 times more than usual.Prior to, all technical staff should be well-rested and fully prepared during the most critical hours and days.
- **December:** Time for postmortem. What worked well? What went wrong? What can we learn for the future? It's important to document everything in detail, with numbers, screenshots of monitoring dashboards, and so on. A year is quite a long time for human memory, so it is always good to have historical data for better analysis and planning.

DBA's tasks when preparing for a peak season

If you are responsible for the health of the databases, how can you make sure you are better prepared? Here is the checklist of the most important aspects to cover. Ideally, you need to start using it in the very beginning – right after the previous peak season has finished. For the BFCM example above, it means you need to start in December. However, if you're approaching the end of the current season, do not worry: review the list right now and decide what and how you can improve during the days left. Even if you are short in time, there are always ways to improve the state of your databases.

1. Observability tools

Increased traffic and sales numbers mean potential performance and scalability issues. Sometimes, peak hours may show 10x more load on specific parts of your systems compared to regular days.

When it comes to optimization for growth, It is hard to overestimate the importance of the observability tools. First of all, monitoring.

Regardless of the monitoring system you are using or are going to use, consider the following as must-have items for your database monitoring (see also: community-built PostgreSQL Monitoring Checklist):

1. General availability, uptime

2. Basic metrics for system resources: CPU (overall and per each core), memory, disk IO, network
3. Disk space
4. Connection stats (number of connections by state, by users, by the database, by client address, by application)
5. QPS, TPS
6. Long-running transactions, idle transactions, long-running queries
7. Transaction ID wraparound risks
8. IO activities of DBMS subsystems (write-ahead log writes, background jobs, data writes, replication, regular logging)
9. Replication stats, including replication delays
10. The health of backups:
    1. periodical full or incremental backups
    2. archiving stream of WAL segments
    3. status of backup verification (automated checks that it is possible to restore from a backup to a healthy state)
11. Two special and very powerful capabilities:
    1. session analysis based on wait events (such as Performance Insights in Amazon Aurora, or pgsentinel for PostgreSQL)
    2. query analysis (aggregated stats of queries, including a number of calls, duration, IO stats for each query group; for Postgres, it is usually based on pg_stat_statements and sometimes slow query logging)

In addition to a regular monitoring system that provides some historical data with a focus on the current moment and the latest days and weeks, as well as some alerts, you may want to:

- Install observability extensions[1]:
    - pg_stat_statements is usually considered as a must-have
    - pg_stat_kcache and other PoWA extensions for additional data
    - pgsentinel for session history

[1] When installing a new extension, it is always worth remembering about the Observer Effect: any additional bits of observability come with extra costs. For example, be careful with enabling timing in auto_explain, enabling pg_stat_statements.track_utility in pg_stat_statements, and automatic, not well-tested upgrades to the new versions of extensions. Thoroughly test and check the overhead first!

    - auto_explain for advanced logging

- Set up additional dashboards that can be helpful in troubleshooting of specific problems (such as locking issues, increased disk IO, or spikes in the number of sessions approaching max_connections)
- Adopt ad-hoc tools for PostgreSQL troubleshooting such as:
    - pgCenter for the live top-like look at various pg_stat_** views and query analysis based on pg_stat_statements
    - postgres_dba to analyze to the state of the database from psql
    - PASH Viewer for wait-based session analysis
- Start using postgres-checkup (a part of Postgres.ai Database Lab Platform) for automated multi-node health checks of PostgreSQL clusters and deep SQL macro-analysis based on pg_stat_statements
- Update documentation, runbooks, organize internal training sessions for DBAs/DBREs and SREs to improve the troubleshooting skills of teams

2. Prepare for unexpected

It is ***highly*** recommended to train all database engineers to use monitoring dashboards for quick troubleshooting. If you can, think about possible risks, analyze past experiences, and describe possible incidents with solutions on how to resolve them. Some examples of such incidents:

- Increased disk IO affecting overall performance
- Increased number of sessions, approaching max_connections
- Quickly decreasing free disk space
- Locking issues
- Sudden degradation of the performance of specific SQL queries
- Increased replication lags
- Unexpected PostgreSQL restarts, crashes
- Backup creation failures or failures of backup verification

In most cases, the hypothetical troubleshooting solution steps must include instructions on how to localize the problem by performing a top-down analysis. For example, when dealing with an increased number of connections, segmentation of all connections by client_addr, application_name, state, and historical overview of the segments may be extremely helpful. Of

course, very well organized monitoring, as described in the previous chapter, can save a lot of headaches.

The hardest part of these preparations is preparing for the unexpected. Learn from other's experiences. Numerous articles, blog posts, and conference talks found online can help to understand what bad things may happen under increased load and how to deal with them.

3. Capacity planning

To prepare for the peak time, you need to review the current utilization and saturation risks for all essential components:

- Computing Power
    - CPU load
    - Memory utilization, the state, and effectiveness of the caches (both the buffer pool configured via shared_buffers and OS file cache)
- Storage and Network
    - Disk space
    - Disk IO (disk IO utilization, and four metrics: read and write IOPS + read and write throughput)
    - Network utilization

Additionally, you may want to review the utilization of specific components to ensure that there is enough room for 2x, 3x, or 10x growth, depending on business expectations. One of the sound examples here is a popular connection pooler, pgBouncer. It is single-threaded, so with an increased load, it is easy to reach saturation of a single CPU core. It depends on your CPU model and types of queries. As an example, for modern CPUs and moderately complex queries generated by ORMs, such as Ruby on Rails or Hibernate, one may expect hitting this wall at levels such as 10-20 thousand QPS (queries per second). To resolve this, consider using multiple processes and the latest versions of pgBouncer supporting SO_REUSEPORT, or a modern replacement for pgBouncer, Odyssey, which is multi-threaded.

Another very basic example is a saturation of computing or disk IO capacities on the primary server. Besides SQL query optimization and index maintenance/cleanup that will be discussed in the next two chapters, the main solutions here are:

- Vertical scaling:
    - Add more computing power:
        - More CPU cores or vCPUs – modern PostgreSQL versions scale very well on dozens and even hundreds of vCPUs

        - More RAM for the buffer pool (correspondingly increasing shared_buffers; again, modern PostgreSQL versions work very well with very large shared_buffers values), and OS file cache (if database size exceeds RAM, normally all available “free” memory will be eventually filled with this type of cache, implicitly improving PostgreSQL performance)

    - Use of more performant disks (that support higher IOPS and throughput values) – switching to SSD or, even better, to NVMe SSD[2] is a very good idea in the most cases if at least of one of the following is applicable to your databases:

        - Doesn’t fit RAM

        - You need an acceptable performance of cold-cache queries

        - The workload has a lot of writes

- Split writes and read queries. There are automated solutions for this, such as pgpool, but we do not recommend using them: there are a lot of restrictions and caveats, not speaking of the performance penalty when you use intermediate software with parsing (and parsing is needed to perform the split). All modem frameworks, ORMs support work with two connections well, and design patterns CQRS (Command Query Responsibility Segregation), CQR (Command Query Separation) became very popular. Of course, it requires some work to teach the application code to use two database connections, but this approach gives better control and performance.

What numbers should you expect from the busiest hours of peak days? First, use the historical data from the past years (and always ensure that you save observations for the current year, to use in the future). Next, look at the behavior at the busiest “normal” days and apply some multiplier. Ask leadership to provide the expected numbers for traffic growth and customer activity. This will help determine the multiplier – depending on the business expectations, it can be 2x, 3x, or even 10x and more.

If you are using postgres-checkup, and combine the information from it with any basic monitoring, here is the recipe to understand how much room for the growth your databases have:

- Consider postgres-checkup reports for typical busiest hours for a regular week.
- **CPU:**
    - Use reports K001, K002, and K003 that contain deep workload analysis. The “total_time” metric can suggest the overall database utilization during these hours. Particularly, this metric derivative is measured in “seconds per seconds” and serve as an estimate of how much CPU power is used: say, if you see 4 sec/sec in the “total_time” column of K001, it means that at least 4 vCPUs or

[2] if you are on NVMe, a couple of pieces of advice: 1) consider enabling the "discard" filesystem options for automatic online TRIM; 2) old Linux kernels might not have a good support for NVMe

cores are needed to process the workload. Of course, this should be considered as just a rough estimate since in databases, the observed load does not depend linearly on the incoming traffic.

- Also, check CPU load and LA (load average) metrics in monitoring, and do the math based on the findings and the forecasted values. A basic rule: CPU load should not be higher than 50%, even during peak hours (forecasted).
- If needed, consider vertical scaling: more vCPUs (cores), switching to newer types of CPUs.

- **MEMORY AND DISK IO:**
  - Check “Cache Effectiveness” the report A004 “Cluster Information” showing hit/(read+hit) ratio of the buffer pool, good numbers for OLTP workloads are 99% and higher.
  - Check RAM metrics in your monitoring:
    - How many GiB are allocated for the buffer pool?
    - How many GiB are used by Postgres backends?
    - By non-Postgres processes?
    - How many GiB are normally left “unused” directly, and therefore are used by the OS file cache (here we suppose that DB size exceeds RAM)?
    - What is the percentage of dirty buffers in the page cache?
  - Check disk IO metrics (IOPS and throughput both for reads and writes) and compare it to what you know about your storage system. Segmentation can be very helpful again:
    - How many IOPS and MiB/s are associated with checkpoints?
    - How many IOPS and MiB/s are caused by checkpoints, bgwriter?
    - ...by Postgres backends reading the data or writing the data pages themselves?
    - How many IOPS and MiB/s are caused by WAL writer?
    - How many IOPS and MiB/s are caused by autovacuum?
    - How many IOPS and MiB/s are caused by writing temporary files? (Related: “Temp Files” in A004 “Cluster Information”.)
    - How many IOPS and MiB/s are caused by writing PostgreSQL logs and other types of logs on the server?

- For advanced analysis of how the PostgreSQL buffer pool works for a particular database and workload, pg_buffercache extension may be very useful.
- In postgres-checkup, reports K001, K002, K003 show hits and reads for the buffer pool (shared_blks_hit, shared_blks_read) and can help understand what parts of the workload cause most disk IO (keep in mind that the shared_blks_read metric is counting only reads to the Postgres buffer pool, some of these reads may be "hits" in the OS cache, not involving disk IO – for a more detailed analysis, you need pg_stat_kcache in addition to pg_stat_statements).
- As a result, you might decide to increase the buffer pool size and/or add more RAM to prepare for the growth.

The next two chapters describe steps that may (and do, in most cases) improve the overall picture for the capacity planning. For example, index maintenance almost always reduces used disk space, improves performance in various aspects (it may be CPU load, memory usage, disk IO, and network IO). Therefore, it may be tempting to do health checks, housekeeping, and query optimization first, and only then capacity planning. However, the real-world experience tells us that it's better not. It is worth starting with capacity planning first, prepare the plan not relying on possible improvements, and then adjust if necessary. This is better because:

- Capacity planning and scaling in may take time and budgets, so the earlier you start, the better
- Optimization activities can also take a significant amount of time, and if you do not fully rely on them, you are in a safer position
- Last but not least, in some cases, positive effects from an optimization may be temporary, quickly vanishing – in such a case, if you relied on these effects, you might be short in time to react and mitigate the consequences

4. Health Checks. Index Maintenance and Bloat Control

It is highly recommended to do comprehensive health checks for mission-critical databases at least once per quarter. When planning for the peak season, you need to do it at least once.

Such health checks include various activities that may include:

- Reviewing software versions that are used on all database servers (Postgres itself, extensions, adjacent software)
- State of backups and PITR strategy (see below)
- HA characteristics of the system and failover readiness
- PostgreSQL configuration settings
- Query analysis (see below)

- Index health analysis
- Data corruption checks and review
- Bloat analysis

These activities can be done using various SQL snippets that every DBA has. Some examples of toolsets:

- pg-utils by Data Egret
- pgx_scripts by PostgreSQL Experts
- postgres_dba, an interactive toolset working right in psql, by Nikolay Samokhvalov (Postgres.ai Founder)

With these tools, the health check process is still rather manual and requires lots of time, especially if you have many databases to review. For holistic and automated analysis, consider using postgres_checkup by Postgres.ai, which is now a part of the Database Lab Platform, but can be used separately. It covers the majority of the mentioned topics, doesn't require installing anything on the observed servers, analyses primary with its secondaries, and provides the detailed reports in Markdown, HTML, and PDF.

One of the key activities that can follow the health checks are:

- Index maintenance:
    - Remove unused (on all nodes) indexes for significant (30+ days) period of time
    - Get rid of redundant indexes
    - Reconsider and optimize index sets and types of indexes
- Bloat control:
    - Usually, what hurts the most and what needs to be taken care of first is the bloat in indexes (tools: REINDEX CONCURRENTLY in Postgres versions 12 and newer; pg_repack)
    - Sometimes, table bloat is also an issue (tool: pg_repack)

In many cases, these actions can be very beneficial terms of performance. As already mentioned, once they are done, you might even re-consider your capacity planning results (although, again, it needs to be done with caution: sometimes, if you don't automate the processes and the negative states return rather quickly, you might prefer to ignore optimization effects when performing the capacity planning).

Finally, various experiments on database clones may be invaluable when you explore your systems during the analysis phase or prepare for the changes in production. Postgres.ai

philosophy is: always experiment first when you need something, on database clones that are as close to production ones as possible.

> *The Database Lab technology by Postgres.ai (there are Open Source / Community Edition and Enterprise Edition) allows you to clone databases of any size just in a few seconds. Many engineers can conduct various experiments on a single Database Lab Engine, saving a lot of money, time, and improving the overall efficiency of development and administration processes related to PostgreSQL databases.*

5. Query analysis and performance optimization

Query optimization is quite a big topic and requires continuous and dedicated efforts from both engineers and DBAs. This topic goes beyond the scope of this paper.

> *Recommended material: "Seamless SQL Optimization", a talk by Nikolay Samokhvalov (Postgres.ai Founder) at PGCon-2020.*

Open-source tools for macro-analysis of SQL queries (a type of analysis when the workload is analyzed as a whole, organizing queries into groups and observing various metrics for each query separately):

- pg_stat_statements extension, a de facto standard for query macro-analysis for PostgreSQL databases
- auto_explain
- pgBadger
- PoWA stat extensions (pg_stat_kcache and others)
- PASH Viewer and pgsentinel
- pgCenter
- postgres-checkup by Postgres.ai (reports K001, K002, K003)

For micro-analysis (analysis of a particular SQL query), the key tool is the standard SQL command EXPLAIN, and its version that allows you to see execution plans with all the details – EXPLAIN (ANALYZE, BUFFERS).

> *The key question for using the EXPLAIN command is where you use it: performing modifications, executing long-running queries, or changing the database schema on production databases just for analysis is not recommended. The recommended way is to do these actions on clones. Database Lab by Postgres.ai and Joe bot (that works on top of it) can be exceptionally helpful for these kinds of tasks. Learn more: https://Postgres.ai.*

6. Backups and DR strategy

Having reliable backups and a good DR (Disaster Recovery) strategy are critical goals. If you are using a managed database such as Amazon RDS, you already have tools and processes established, and all you need is just to learn how they work and what are the characteristics (see RPO and RTO). If you are on a self-managed PostgreSQL, you have to take care of backups yourself.You might want to choose one of the emerging backup solutions supporting incremental backups and PITR (point-in-time recovery) such as WAL-G, Barman, or pgBackRest.

Do the math and prepare properly – answer the following basic questions:

- How much WAL data per day is normally generated, and how much you expect for peak days?
- How long does it take to perform the full backup? If it's daily, will it overlap with the busiest hours during peak days?
- How long does it take to restore from a full backup and replay a full day of WALs?
- Do you have enough backup storage?

Remember that unverified backups (a.k.a. Schrödinger's backups) are the last thing that you'd want to have in your backup and DR strategy. Set up an automated process of backup verification, with backup validation.

> *If you already have Database Lab Engines set up for your PostgreSQL production databases, provisioned in "physical" mode based on backups, with "continuously updated" mode, then you already have basic verification of the WAL stream. Reach out to the Postgres.ai team to learn more about how Database Lab can be used for backup verification and data corruption checks.*

7. Tests, load tests

There is no such thing as enough testing. When you prepare for a peak season, you need a holistic approach for this topic:

- Build test environments that are as close to real-life ones as possible
    - For micro-analysis (when only a single query is analyzed and rest of workload is ignored – for example, running EXPLAIN and optimizing SQL queries), thin clones are very useful (see the chapter on Query analysis above).
    - For macro-analysis, load testing, you need separate full-sized clones, with same-scale resources. You also need tooling for repeating, mirroring, or simulating the real-life traffic. This may be challenging.
    - Finally, one of the ultimate testing approaches that large companies use today is performing load testing right on production. This approach is not trivial to

implement, but very effective; helping to test all the parts of your system and the teams and processes working together.

- Consider performing full end-to-end HA testing for your databases and applications:
    - Are the systems really ready for a failover?
    - How long will it take to perform failover and return back to normal?
- Review DR strategy, recovery plans, and thoroughly test them.
- Ensure that all engineers that might be involved in the operations listed above are familiar with corresponding documentation, and participate in the testing, exercises, and rehearsals of the various disaster scenarios.

8. Freeze

Quite simple, but very important – just two rules here:

1. Avoid changes during the several days (maybe a few weeks) before peak days (freeze the development: postpone all non-urgent deployments for the post-peak time).
2. Have a good rest before the main days! Anyone who is involved in operations during peak days needs to be fresh, so the days off with quality time have to be planned. Also, remember, a single person can't stay fresh for several days in a row! It is important to get multiple database engineers involved and to build a schedule in which everyone gets enough time to recover between shifts.

Timeline

This table can help you to develop your own preparation plan. Use it as a recommendation showing when the focus for a specific topic should be made. Of course, it does not need to be taken literally, it is recommended to adjust it for your particular scenario.

Tools	Jan Feb	Mar Apr	May Jun	Jul Aug	Sep	Oct	Nov	Dec
	Planning	Initial health checks. Capacity planning.	Improve: monitoring, backups, DR, HA	Configuration tuning. Scale-in	Performance optimization	Load testing. Trainings	Deployment freeze, review runbooks, rest. Action (BFCM)!	Postmortem
Whiteboard	🔴 🔴	🟡 🟡	🟡 🟡	🟢 🟢	🟢	🟢	🟢	🔴
Documentation	🔴 🔴	🟡 🟡	🟡 🟡	🟡 🟡	🟡	🔴	🟡	🔴
Monitoring	🟡 🟡	🔴 🔴	🔴 🔴	🟡 🟡	🟡	🔴	🔴	🟡
Capacity planning	🟡 🟡	🔴 🔴	🟡 🟡	🟢 🟢	🟢	🟢	🟢	🟢
Tools for index maintenance and bloat control	🟢 🟢	🟡 🔴	🟡 🟢	🟢 🟢	🟡	🔴	🟡	🟢
Backups	🟡 🟡	🟡 🟡	🔴 🔴	🟡 🟡	🟡	🟡	🟡	🟡
HA solutions, auto-failover	🟡 🟡	🟡 🟡	🔴 🔴	🟡 🟡	🟡	🟡	🟡	🟡
Connection pooling	🟢 🟢	🟢 🟢	🟢 🟢	🔴 🔴	🟢	🟢	🟢	🟢
SQL performance analysis and optimization	🟡 🟡	🟡 🟡	🟡 🟡	🟡 🟡	🔴	🟡	🟡	🟡
Load testing tools	🟡 🟡	🟡 🟡	🔴 🔴	🔴 🔴	🔴	🔴	🟡	🟡
Non-production environments	🟡 🟡	🟡 🟡	🔴 🔴	🔴 🔴	🔴	🔴	🔴	🟡

🔴 must have
🟡 recommended
🟢 nice to have

Want to learn more? Reach out to the Postgres.ai team:

team@postgres.ai

https://Postgres.ai

Checklist

- ❏ Observability tools
  - ❏ System metrics covered (CPU, RAM, disk IO, network IO, disk space)
  - ❏ PostgreSQL metrics covered (see PostgreSQL Monitoring Checklist)
  - ❏ Alerts are configured and tested. No extra noise, low level of false positives
  - ❏ Ad hoc troubleshooting tools prepared and practiced by personnel
- ❏ Prepare for unexpected
  - ❏ The existing past experience is analyzed and documented in the form of runbooks describing a problem, steps to troubleshoot, and steps to fix
  - ❏ The experience of others (learned from articles, blog posts, conference talks) is analyzed and described in the form of runbooks (less detailed)
- ❏ Capacity planning
  - ❏ Current usage and saturation risks analyzed, and the peak numbers forecasted:
    - ❏ CPU
    - ❏ Memory
    - ❏ Disk IO
    - ❏ Network IO
    - ❏ Disk space
    - ❏ Specific systems such as connection pooler or backup system
  - ❏ If needed, split reads and writes
  - ❏ If needed, add more resources to the existing servers
  - ❏ If needed, add more replicas
  - ❏ Review Capacity Planning after configuration tuning, SQL query optimization, index maintenance, and bloat reduction
- ❏ Health Checks
  - ❏ Check minor versions of PostgreSQL on all servers. If needed:
    - ❏ upgrade
    - ❏ eliminate version differences
  - ❏ Check versions of PostgreSQL extensions
    - ❏ Upgrade if needed (with proper testing in advance)
  - ❏ Review configuration of all PostgreSQL servers
    - ❏ If applicable, eliminate differences
    - ❏ If needed, perform basic tuning (such as tuning of the memory-related settings)
    - ❏ Consider checkpoint tuning, enabling WAL compression
    - ❏ Consider autovacuum tuning
    - ❏ Pay attention to max_connections. Analyze risks:
      - ❏ reaching max_connections
      - ❏ OOO (out-of-memory) or swapping risks

- ❏ Review the state of backups and PITR strategy
- ❏ Check HA characteristics of the system and failover readiness
- ❏ Analyze index health:
  - ❏ Identify unused indexes
  - ❏ Identify redundant indexes
- ❏ Analyze bloat:
  - ❏ Estimated index bloat
  - ❏ Estimated table bloat
  - ❏ (Optional) exact bloat levels
  - ❏ Plan reindexing/repacking for objects with a high level of bloat (say, exceeding 40%)

❏ Index Maintenance
- ❏ Eliminate unused (unused on all nodes during 30+ days) indexes
- ❏ Eliminate redundant indexes (test carefully first)

❏ Bloat Control
- ❏ Practice reindexing/repacking in a “Lab” environment (for example, using Database Lab)
- ❏ Perform the repacking of bloated tables. Re-analyze bloat levels
- ❏ Perform reindexing or repacking of bloated indexes. Re-analyze bloat levels

❏ Query analysis and performance optimization
- ❏ Identify query groups that have more than 25% of aggregated total_time, calls, buffer hits, and reads
- ❏ Collect specific examples of slow queries with execution plans
- ❏ Optimize (thoroughly test on clones first)

❏ Backups and DR strategy
- ❏ Carefully review the backup system. Learn the current RPO and RTO
- ❏ If needed, optimize, reduce the amount of WAL data generated, improve delays, and speed of the “create” and “restore” processes
- ❏ Carefully review the automated backup verification system. If you don’t have one, build it

❏ Tests, load tests
- ❏ Testing on environments identical to production
- ❏ Plan and perform periodical load tests on production (several weeks, a few times per week) involving all systems and various teams
- ❏ Test various failure scenarios, practice troubleshooting skills

❏ Freeze
- ❏ Freeze: Forbid and postpone any non-critical, non-urgent deployments at least a few days before peak days
- ❏ Have a good rest a few days before peak days