Your remote PostgreSQL DBA Team
data egret

PostgreSQL worst practices

at PGDay.Paris 2024

Ilya Kosmodemiansky

ik@dataegret.com

$ whoami

- CEO und Founder @ Data Egret GmbH
- Work with Postgres since 7.*
- PostgreSQL Contributor
- Co-organizer of PUG Frankfurt (am Main) and Postgres Europe conferences

SECURING YOUR DATABASE AVAILABILITY, SO THAT YOUR TEAM CAN FOCUS ON NEW FEATURE DEVELOPMENT.

- Migrations
- DB audit
- Performance optimisation
- Backup & restore
- Architectural review
- Advising Data Science teams
- Developer training

on premise & cloud

EXPERTISE

Senior DBA with **10+ years** of PostgreSQL administration **experience**

DEVELOPMENT

Involved in **new feature and extension development**

TAILORED APPROACH

Felxible approach and **dedicated team** focused on success of your project

COMMUNITY

Contributing Sponsor. Committed and deeply involved in the community

Best practices are just boring

- Never follow them, try worst practices
- Only those practices can really help you to screw the things up most effectively
- PostgreSQL consultants are nice people, so try to make them happy

How it works

- I have a list, a little bit more than 140 worst practices
- I do not make this stuff up, all of them are real-life examples
- I reshuffle my list every time before presenting and extract some amount of examples
- Well, there are some things, which I like more or less, so it is not a very honest shuffle

0. Do not use indexes (a test one!)

- Basically, there is no difference between full table scan and index scan
- You can check that. Just insert 10 rows into a test table on your test server and compare.
- Nobody deals with more than 10 row tables in production!

1. Try to create as many indexes as you can

- Indexes consume no disk space
- Indexes consume no shared_buffers
- There is no overhead on DML if one and every column in a table covered with bunch of indexes
- Optimizer will definitely choose your index once you created it
- Keep calm and create more indexes

2. Postgres likes long transactions

- Always call external services from stored procedures (like sending emails)

2. Postgres likes long transactions

- Always call external services from stored procedures (like sending emails)
- Oh, it is arguable... It can be, if 100% of developers were familiar with word timeout

2. Postgres likes long transactions

- Always call external services from stored procedures (like sending emails)
- Oh, it is arguable... It can be, if 100% of developers were familiar with word timeout
- Anyway, you can just start transaction and go away for weekend

3. Massive DDLs can never harm your database

- Put them all in a massive single transaction
- forget about **statement_timeout**
- Use "rush hour" and never discuss your plans with other teams
- Leave it running and go for lunch

4. Always use defaults

- Postgres can work on a big server as we as in a coffeemaker, we can keep the same configuration
- Never increase **shared_buffers** if your database is large
- **initdb --locale=C** is always the best choice
- Running Postgres without checksumms enabled would improve your skills and experience

5. as a DBA, never talk to your developers team

- You know things better
- Developers exist to abuse databases, not to create any value for the company
- Best answers for any question are "No!" closely followed by "Why?"
- Teamwork is an artificial construct created by managers to make DBAs busy

6. as a Developer, never talk to your DBAs team

- You know things better
- Nobody knows why DBAs are still exist, but you are here to create value for the company
- Never ask any question, when you ruin thing they might would explain things anyway
- Teamwork is an artificial construct created by managers to make developers busy

7. Always keep all your time series data

- Time series data like tables with logs or session history should be never deleted, aggregated or archived, you always need to keep it all
- You will always know where to check, if you run out of disk space
- You can always call that Big Data
- Solve the problem using partitioning… one partition for an hour or for a minute

8. Never upgrade your Postgres

- Upgrades are complex and painful
- They would be less complex and painful if you perform them as rare as you can
- Postgres is a dinosaur (as any RDBMS), you would never miss any cool new feature if you upgrade ones in 10 years

9. Never use graphical monitoring

- It allows you to tell what happened with Postgres for example yesterday at 2:00 am
  - which makes your work routine and boring
- It allows you to see the trend and prevent disasters
  - your boss would forget your phone number without those disasters

10. Never use Foreign Keys

- Consistency control at the application level always works as expected
- You will never get data inconsistency without constraints
- Even if you already have a bulletproof framework to maintain consistency, could it be a good enough reason to use it?

11. Turn autovacuum off

- It is quite an auxiliary process, you can easily stop it
- There is no problem at all to have 100Gb data in a database that is 1Tb in size
- 2-3Tb RAM servers are cheap, IO is the fastest thing in modern computing
- Besides that, everyone likes BigData

12. Be in trend, be schema-less

- You do not need to design the schema
- You need only one table, two columns: id bigserial and extra jsonb
- JSONB datatype is pretty effective in PostgreSQL, you can search in it just like in a well-structured table
- Even if you put a 500M of JSON in it
- Even if you have 1000+ tps

13. Be agile, use EAV

- You need only 3 tables: **entity**, **attribute**, **value**

13. Be agile, use EAV

- You need only 3 tables: **entity**, **attribute**, **value**
- At some point add the 4th: **attribute_type**

13. Be agile, use EAV

- You need only 3 tables: **entity**, **attribute**, **value**
- At some point add the 4th: **attribute_type**
- Whet it starts to work **slow**, just call those four tables The Core and add 1000+ tables with denormalized data

13. Be agile, use EAV

- You need only 3 tables: **entity**, **attribute**, **value**
- At some point add the 4th: **attribute_type**
- Whet it starts to work **slow**, just call those four tables **The Core** and add 1000+ tables with denormalized data
- If it is not enough, you can always add **value_version**

14. Move joins to your application

- Just *select ** a couple of tables into the application written in your favorite programming language
- Then join them at the application level

14. Move joins to your application

- Just *select ** a couple of tables into the application written in your favorite programming language
- Then join them at the application level
- Now you only need to implement nested loop join, hash join and merge join as well as query optimizer and page cache

14. Move joins to your application

- Just *select ** a couple of tables into the application written in your favorite programming language
- Then join them at the application level
- Now you only need to implement nested loop join, hash join and merge join as well as query optimizer and page cache
- **…and remember: SQL is specifically designed for the purpose, you should never use such tools!**

15. Need to run Postgres in a container?

- Never use persistent storage
- Always build your own container image
- Never use any existing operator

16. Do everything under superuser

- It keeps everything simple (up to certain moment)
- If your application runs out of connections it always can use **superuser_reserved_connections**

17. Are there better options as scram-sha-256?

- **md5** !
- **trust** is even better
- Make sure your passwords are in **.pgpass** laying everywhere

18. Always use timestamps without time zone

- You application would always work in a single timezone
- This timezone would never change

19. Even if you want to backup your database...

- Use virtual machine snapshot
- Use replication instead of backup
- Use pg_dump instead of backup
- Write your own backup script
- As complicated as possible, combine all external tools you know
- Never perform a test recovery
- **Do not use pgBackRest**

And don't forget

That was WORST practice talk!

You can always use these slides and ideas

Under **WPL aka Worst Practice Licence**:

- Take it!
- Use it in production!
- **#blamemagnus**

You also can share your results with me and I include them in my deck

But if you want to learn something more useful...

WEBINAR

18 APRIL, 13:00-14:00CET

INTRODUCTION TO POSTGRESQL BACKUPS

This webinar will provide you with the methodologies and key concepts to strengthen your PostgreSQL backup strategies.

- Write-Ahead Logs
- Point-In-Time Recovery
- Continuous archiving
- Live backup
- Data restore key points

Register
to elevate your expertise in PostgreSQL backup process!

Stefan Fercot
Senior PostgreSQL Expert
Data Egret GmbH