

Intro to PostgreSQL: What To Know When You're Called In To Help

CodeMash 2023



Ryan Booz

PostgreSQL & DevOps
Advocate



[@ryanbooz](https://twitter.com/ryanbooz)



[/in/ryanbooz](https://www.linkedin.com/in/ryanbooz)



www.softwareandbooz.com



github.com/ryanbooz/presentations



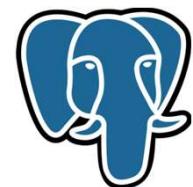
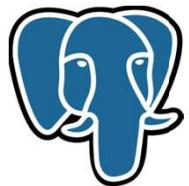
$$\text{👤} + [\text{👤}, \text{👤}, \text👤, \text{👤}, \text👤, \text👤] = \text{❤}$$

$$\text{👥}, \text{🎵}, \text{🐝}, \text{☕} = \text{💯}$$

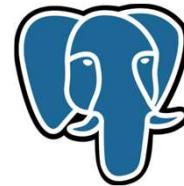


My Journey

Work



Hobby



1999

2004

2018

2020

2022



-T2371



Agenda

- 01** Back to the future
- 02** Feature Comparison
- 03** Terminology
- 04** Architecture
- 05** Getting Connected
- 06** Configuration Highlights
- 07** Create/Backup/Restore
- 08** SQL Differences
- 09** Query Tuning
- 10** Community & Help

01/10

Back to the future with PostgreSQL

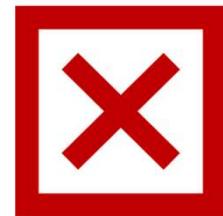


Post-gres-QL

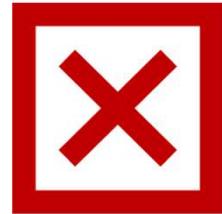


Post-gres

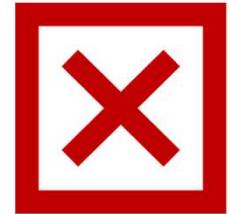




Post-gre



Post-gre-SQL



Post-gres-SQL

It's really PostgreSQL

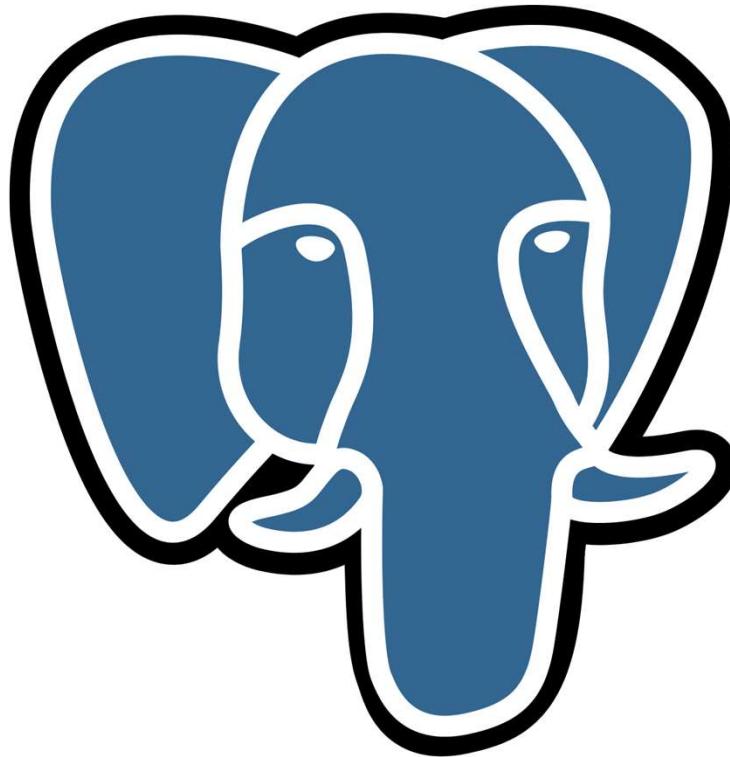
@postgre_s

[postgresql.org](#) Joined August 2018

1 Following 86 Followers

Followed by Pavlo Golub, Jessica Sharp, and 7 others you follow





Slonik
(Slony)

1970s



INGRES

1980s



POSTGRES

1995



Postgres95

1996

1

Name changed
to PostgreSQL

2

PostgreSQL
Global
Development
Group

3

PostgreSQL 6.0
released in
January 1997

PostgreSQL

- No tiers with a *very* permissive license

PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2011, PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Core Team and Contributors



- > Set release dates
- > Confidential project matters
- > Act as spokespeople for the project
- > Arbitrate community decisions
- > Code contributions
- > Patch reviewers
- > Mailing list moderators
- > Active community members
- > Core team
- > Major contributors

<https://www.postgresql.org/community/contributors/>

COMMITFEST



Five each year



Dictates what patches
are included in a release



Mailing-list and
volunteer led

Commitfest 2022-11

Redgate

Home / Commitfest 2022-11 / Activity log / Log in

Commitfest 2022-11

Search/filter Shortcuts ▾ New patch

Status summary: Needs review: 158. Waiting on Author: 60. Ready for Committer: 23. Committed: 38. Withdrawn: 8. Rejected: 1. Returned with Feedback: 4. Total: 292.

Active patches

Patch	Status	Ver	Author	Reviewers	Committer	Num cfs	Latest activity	Latest mail
Bug Fixes								
pgbench: using prepared BEGIN statement in a pipeline could cause an error	Needs review	15	Yugo Nagata (yugo.nagata)	Kyotaro Horiguchi (horiguti), Fabien Coelho (calvin), Daniel Gustafsson (dege)		7	2022-10-12 05:10	2022-09-30 01:07
Error "initial slot snapshot too large" in create replication slot	Waiting on Author		Kyotaro Horiguchi (horiguti), Dilip Kumar (dilip.kumar)			5	2022-10-12 05:10	2022-10-12 05:10
Fix checkpoint sync request queue problems	Waiting on Author		Thomas Munro (macdice)			3	2022-10-12 05:11	2022-03-16 03:04
Nonreplayable XLog records by means of overflows and >MaxAllocSize lengths	Waiting on Author	stable	Matthias van de Meent (mmeent)	Michael Paquier (michael-kun)	michael-kun	3	2022-10-05 07:46	2022-10-05 07:46
Fix dsa_free() to re-bin segment	Needs review		Dongming Liu (inferyes)			3	2022-10-12 05:07	2022-04-06 07:10
Fix recovery conflict SIGUSR1 handling	Needs review		Thomas Munro (macdice)	Michael Paquier (michael-kun)		3	2022-10-06 06:41	2022-07-26 23:22
Data is copied twice when specifying both child and parent table in publication	Needs review		wei wang (wangwei), Yu Shi (shiy.fnst)	Amit Kapila (amitkapila), Takamichi Osumi (tatatapp), Hou Zhijie (houzj)		3	2022-10-18 02:31	2022-10-21 09:01

<https://commitfest.postgresql.org/>



PostgreSQL Variants



Timescale



AlloyDB



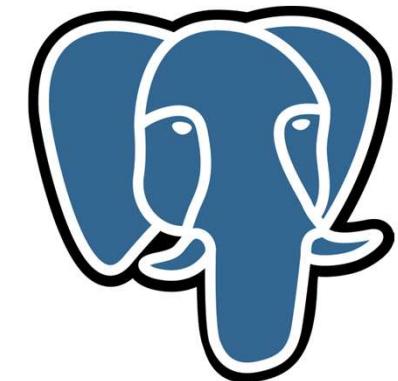
?



02/10

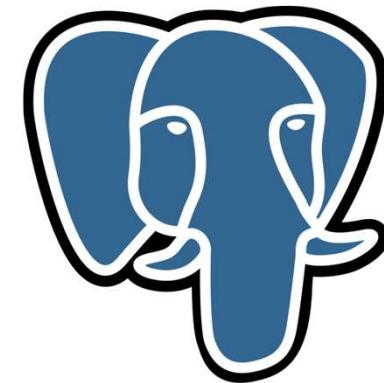
PostgreSQL Feature Comparison

Customizable with Extensions





Native JSON/JSONB Datatype



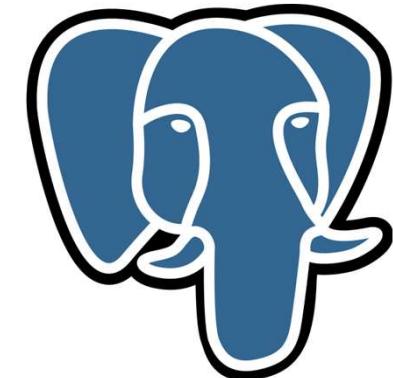
Multiple Index Types



2

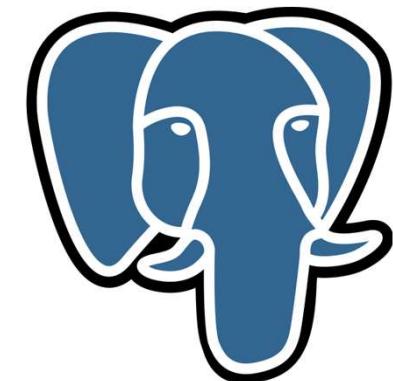


2

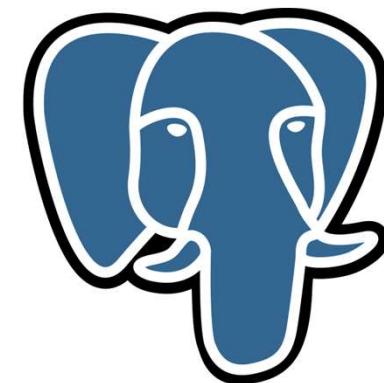


6+

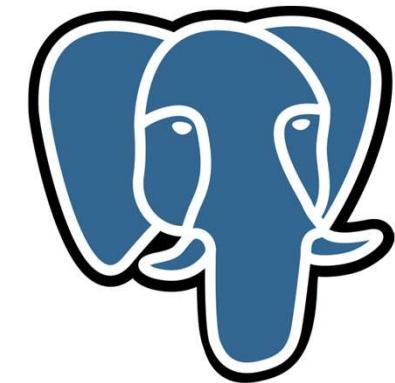
Native Partitioning



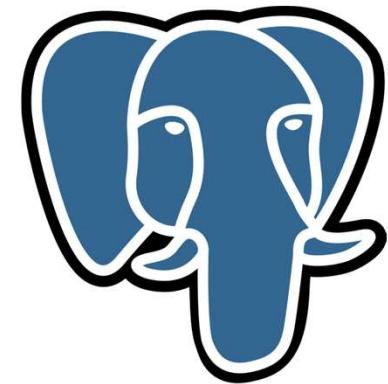
Function/Procedure Multi-language Support



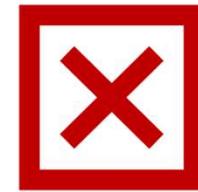
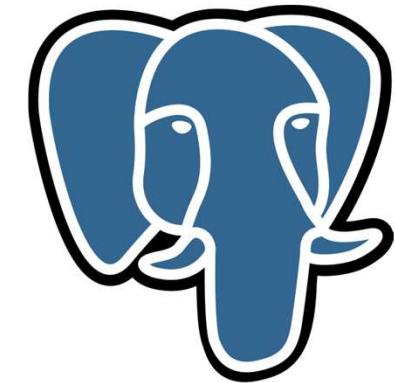
Transactional DDL

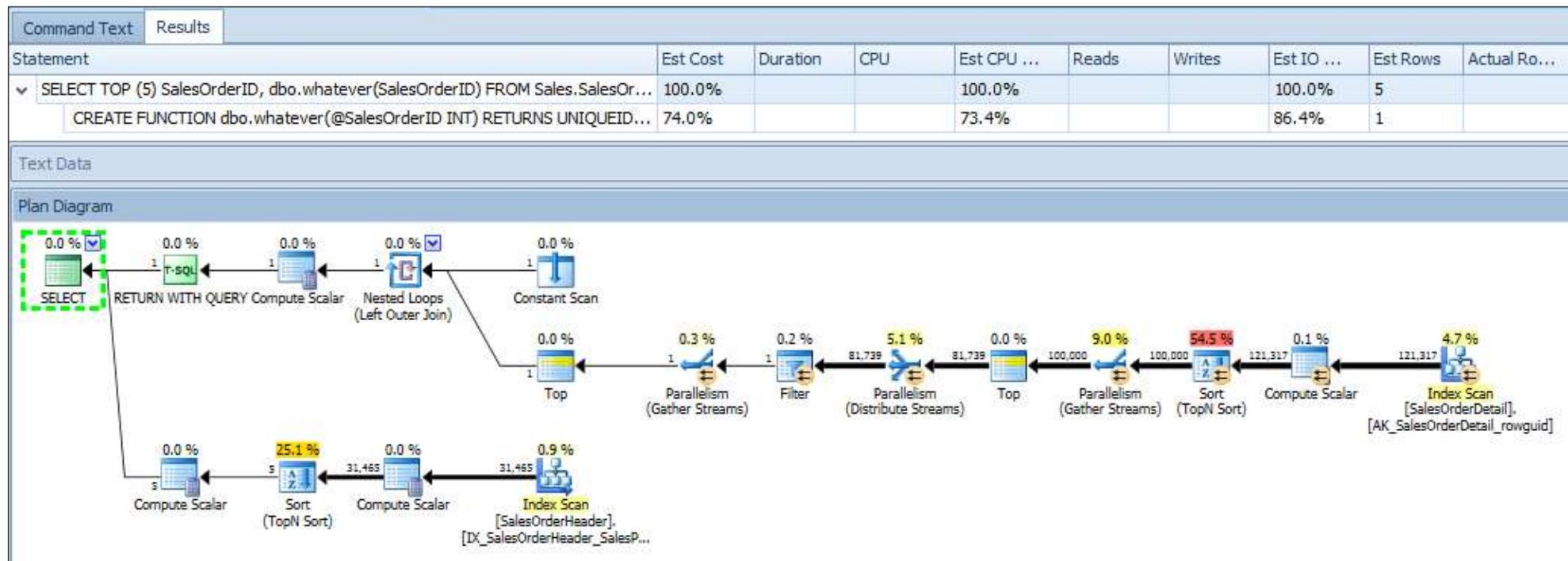


Query Hints

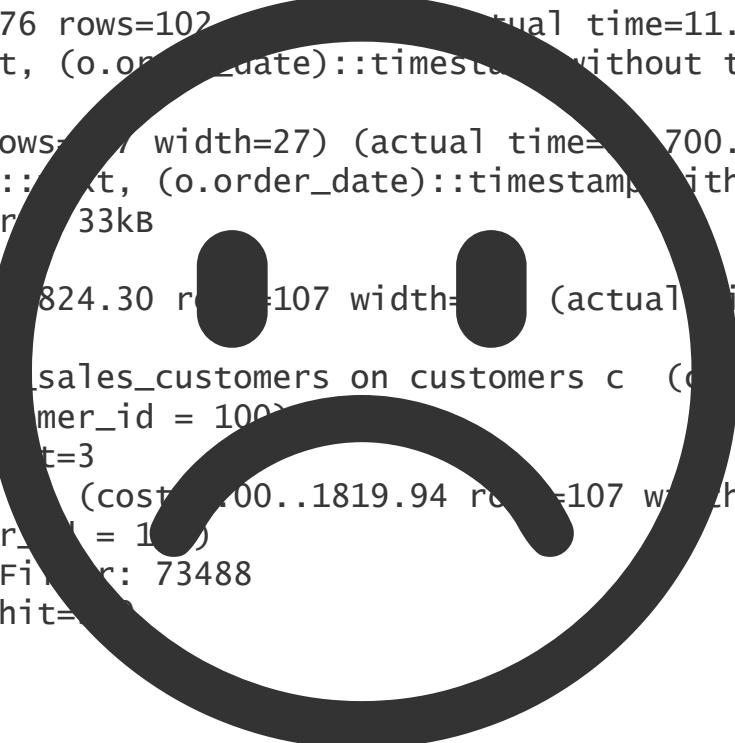


Visual Query Plan Explorer





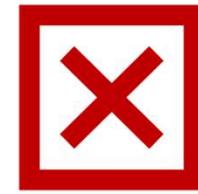
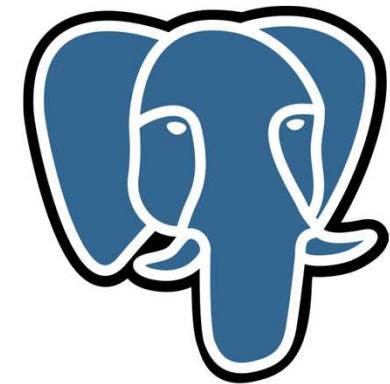
```
GroupAggregate (cost=1827.91..1830.76 rows=102 width=27) (actual time=11.956..13.754 rows=4 loops=1)
  Group Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer), c.customer_id
  Buffers: shared hit=903
->  Sort (cost=1827.91..1828.18 rows=107 width=27) (actual time= 700..12.686 rows=107 loops=1)
      Sort Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer) DESC,...
```



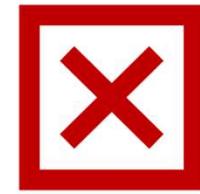
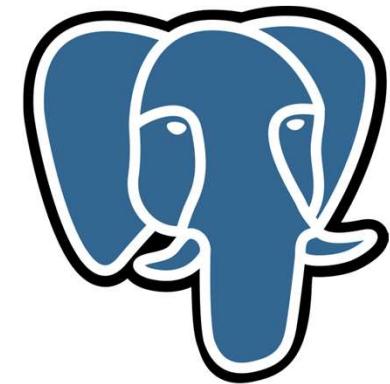
```
      Sort Method: quicksort Memory: 33kB
      Buffers: shared hit=903
->  Nested Loop (cost=0.28..824.30 rows=107 width=27) (actual time=0.113..10.649 rows=107 loops=1)
      Buffers: shared hit=903
        -> Index Scan using idx_sales_customers on customers c (cost=0.28..2.49 rows=1 width=27)
            Index Cond: (customer_id = 100)
            Buffers: shared hit=3
        -> Seq Scan on orders o (cost=400..1819.94 rows=107 width=8) (actual time=0.053..8.413...)
            Filter: (customer_id = 100)
            Rows Removed by Filter: 73488
            Buffers: shared hit=903
```

Planning Time: 0.267 ms
Execution Time: 13.934 ms

Query Cache and Monitor



Built-in Job Scheduler



03/10

PostgreSQL Terminology

Instance = Cluster

SQL Server

PostgreSQL



User = Role

SQL Server

PostgreSQL



Transaction Log ≈ Write Ahead Log

SQL Server

PostgreSQL



Row = Row \in Tuple

SQL Server

PostgreSQL

PostgreSQL



[Table One] = “Table One”

SQL Server

PostgreSQL



BULK INSERT = COPY

SQL Server

PostgreSQL



TOAST

The Oversized-Attribute Storage Technique



04/10

Architecture Highlights

PostgreSQL is a heap-based relational database
with Multi-version
Concurrency Control (MVCC)

MVCC is about concurrency
control through transaction
isolation levels

Transaction Isolation

- Read Committed
- Repeatable Read
- Serializable Read
- **Read Uncommitted (not supported)**

MVCC at a glance

- Readers don't block writers
- Tuples maintain on-row transaction visibility values (xmin, xmax)
- INSERTS/UPDATES always create new rows
 - Heap-only Tuples (HOT) attempts to mitigate some growth
- Popular MVCC talk:

https://www.youtube.com/watch?v=gAE_MSQtqnQ



MVCC at a glance

- Readers don't block writers
- Tuples maintain on-row transaction visibility values (xmin, xmax)
- **INSERTS/UPDATES always create new rows**
 - Heap-only Tuples (HOT) attempts to mitigate some growth
 - Popular MVCC talk: https://www.youtube.com/watch?v=gAE_MSQtqnQ

Extensions

- Must be installed on server
- DBaaS services support different lists
- Added per-database
- Per-database impacts backup/restore
 - Extension has to be available on the server
 - Same version must also be available for successful restore

Extensions

pg_tgram
(Trigram indexes)

PostGIS
(Geospatial)

HypoPG
(Hypothetical index)

fdw_*
(Foreign data wrapers)

CitusDB
(Horizontal clustering)

ZomboDB
(Elasticsearch index)

TimescaleDB
(Relational time-series)

tablefunc
(Pivot & crosstab)

MORE...



Extensions

See available extensions

```
SELECT * FROM pg_available_extensions ORDER BY name;
```

Install

```
CREATE EXTENSION timescaledb;
```

Update

```
ALTER EXTENSION timescaledb UPDATE;
```

Uninstall

```
DROP EXTENSION timescaledb;
```



05/10

Getting Connected

psql

- ≈ mssql-cli on steroids
- Cross-platform
- Go-to interface for 90% of community and training
- Bottom line: you should learn at least some basic psql

Resources

- https://psql-tips.org/psql_tips_all.html
- <https://tomcam.github.io/postgres/>
- <http://postgresguide.com/utilities/psql.html>



- Cross-platform, Eclipse (Java) based
- Community and paid license
- Multi-database support
- Most similar feeling to SSMS thus far*



- Subscription fee with JetBrains
- Lots of great tooling and support
- (Almost) no-brainer if your team uses JetBrains tools



- Supports Postgres
- Notebooks work relatively well
- Postgres still not a first-class citizen, understandably



06/10

Configuration Highlights

Configuration

- *Every non-serverless Postgres should be tuned*



Amazon RDS



Helpful Tuning Resources

https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

<https://www.youtube.com/watch?v=IFIIXpm73qtk>

<https://postgresqlco.nf/>



postgresql.conf – The big 4

shared_buffers:

- data cache
- at least 25% of total RAM

work_mem:

- max memory for each plan operation
- JOINS, GROUP BY, ORDER BY can all have workers

maintenance_work_mem:

- memory for background tasks like VACUUM and ANALYZE

max_connections:

- exactly as it says

postgresql.conf (other)

wal_compression:

- Off (default)
- Default uses pglz
- PG15 can use lz4 or zstd

autovacuum:

- On (default)
- Do not turn off
- Might have to tune individual tables

autoanalyze:

- On (default)
- Do not turn off
- Might have to tune individual tables

SHOW & SET

- SHOW will display the current value
- SET allows settings to be modified (when possible)
- Examples:
 - SHOW ALL ; – page all settings
 - SHOW max_connections ; –configured connection limit

Individual Table Auto(vacuum/analyze)

- Autovacuum triggers at 20% data modification
- Autoanalyze triggers at 10% data modification
- Heavy table modifications may require tweaking individual table thresholds
- Analogous to TF-2731, but per table!

Analyze at 5% modification:

```
ALTER TABLE t SET (autovacuum_vacuum_scale_factor=0.05)
```

Analyze threshold above percentage:

```
ALTER TABLE t SET (autovacuum_vacuum_threshold=500)
```

**'If you think auto(vacuum/analyze)
is running too frequently, it's
probably not running frequently
enough'**

07/10

Create, Backup, and Restore

Create

- Default database = **postgres**
- Template database = **template1** ≈ **MODEL**
- “Super secret” template = **template0**
- Creating a database will use **template1** by default
 - If you want specific extensions, users, permissions, schemas – modify this template

Create database (with template1)

```
CREATE DATABASE myDB;
```



Backup

- **pg_basebackup** for cluster-wide, PITR (with WAL backup)
- **pg_dump** for *regular* backups
- Can be text-based (SQL script) or archive (binary)
- Only directory archives can be parallelized
- **CREATE EXTENSION** commands saved, but not versions

```
pg_dump -U postgres -Fd postgres -W -h localhost -j 2 -f testdump
```

-U = username -Fd = format (directory | custom | tar) -W = ask for password
-h = host -j = number of parallel jobs -f = filename

Restore

- Must always have a clean database for a fresh restore
 - The `--create` option doesn't do what you think
- To restore to a new name, empty database must be created
- Only directory archives can be restored in parallel

```
pg_restore -d test_restore -Fd testdump -U postgres -h localhost -W
```

`-U` = username `-Fd` = format (directory | custom | tar) `-W` = ask for password
`-h` = host `-d` = target database

08/10
SQL Differences

Data Type Differences

SQL Server	PostgreSQL
BIT	BOOLEAN
VARCHAR/NVARCHAR	TEXT
DATETIME/DATETIME2	TIMESTAMP (kind of)
DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE
UUID	CHAR(16), <code>uuid-ossp</code> extension, PG14
VARBINARY	BYTEA

 ARRAY, RANGE, ENUM, GEOM 

Case Rules

- Case sensitive string comparison (LIKE/ILIKE)
- All object names are coerced to lower case
- Double quote to escape names
- Prefer `lower_snake_case`

Case Rules

SQL Server

```
SELECT [Name]
FROM [Purchasing].[Vendor]
WHERE BusinessEntityID = 1492
```

Postgres

```
SELECT "Name"
FROM "Purchasing"."Vendor"
WHERE "BusinessEntityID" = 1492;
```



Case Rules

SQL Server

```
SELECT [Name]  
      FROM [Purchasing].[Vendor]  
     WHERE BusinessEntityID = 1492
```

Postgres

```
SELECT name  
      FROM purchasing.vendor  
     WHERE business_entity_id = 1492;
```



LIMIT ≈ TOP

SQL Server

```
SELECT TOP(10) *
  FROM [Purchasing].[Vendor]
```

Postgres

```
SELECT *
  FROM purchasing.vendor
LIMIT 10;
```

LIMIT/OFFSET ≈ OFFSET/FETCH

SQL Server

```
SELECT *
  FROM [Purchasing].[Vendor]
 ORDER BY name
OFFSET 10 ROWS
FETCH 10 ROWS ONLY
```

Postgres

```
SELECT *
  FROM purchasing.vendor
 ORDER BY name
LIMIT 10 OFFSET 10;
```



ORDER/GROUP BY

SQL Server

```
SELECT SUBSTRING(name,1,1) nameCohort, SUM(amount) totalSales  
FROM [Purchasing].[Sales]  
GROUP BY SUBSTRING(name,1,1)  
ORDER BY nameCohort, SUM(amount)
```

Postgres

```
SELECT SUBSTRING(name,1,1) name_cohort, SUM(amount) total_sales  
FROM purchasing.sales  
GROUP BY name_cohort  
ORDER BY name_cohort, total_sales;
```



ORDER/GROUP BY

SQL Server

```
SELECT SUBSTRING(name,1,1) nameCohort, SUM(amount) totalSales  
FROM [Purchasing].[Sales]  
GROUP BY SUBSTRING(name,1,1)  
ORDER BY nameCohort, SUM(amount)
```

Postgres

```
SELECT SUBSTRING(name,1,1) name_cohort, SUM(amount) total_sales  
FROM purchasing.sales  
GROUP BY 1  
ORDER BY 1, 2;
```



DAY/MONTH/YEAR() ≈ DATE_PART()

SQL Server

```
SELECT COUNT(*), YEAR(OrderDate)
FROM Sales.Invoices
GROUP BY YEAR(OrderDate)
ORDER BY YEAR(OrderDate) DESC
```

Postgres

```
SELECT COUNT(*), DATE_PART('year',order_date) order_year
FROM sales.invoices
GROUP BY order_year
ORDER BY order_year DESC;
```



GETDATE() ≈ NOW()

SQL Server

```
SELECT GETDATE();
```

Postgres

```
SELECT now();
```

Date Math & Intervals

SQL Server

```
SELECT COUNT(*), YEAR(OrderDate)
FROM Sales.Invoices
WHERE OrderDate > DATEADD(MONTH,-1,GETDATE())
GROUP BY YEAR(OrderDate)
```

Postgres

```
SELECT COUNT(*), DATE_PART('year',order_date) order_year
FROM sales.invoices
WHERE order_date > NOW() - INTERVAL '1 MONTH'
GROUP BY order_year;
```



LATERAL ≈ APPLY

- CROSS/OUTER APPLY in T-SQL
 - Often used as an “optimization fence”
 - Outer query iterates an inner query
 - Helpful when JOIN isn’t possible
 - Retrieve top ordered row for each item in a list (ie. “most recent value for each item”)
 - Iterating a function that references the outer query
- LATERAL JOIN allows the similar functionality

LATERAL ≈ APPLY

SQL Server

```
SELECT id, name, dayMonth, total
  FROM Vendor v1
CROSS APPLY (
    SELECT TOP(1) dayMonth, SUM(amount) total
      FROM Sales
     WHERE customerID = v1.id
GROUP BY dayMonth
ORDER BY SUM(amount) DESC
) s1
```



LATERAL ≈ APPLY

Postgres

```
SELECT id, name, day_month, total
  FROM vendor v1
INNER JOIN LATERAL (
    SELECT day_month, SUM(amount) total
      FROM sales
     WHERE customer_id = v1.id
   GROUP BY day_month
  ORDER BY total DESC
  LIMIT 1
) on true s1;
```



Anonymous Code blocks

- Default query language is SQL
- Does not support anonymous code blocks
 - Variables
 - Loops
 - T-SQL like features
- Requires DO blocks, Functions, or SPROCs
 - Default language is pl/pgsql ≈ T-SQL

Inline T-SQL vs pl/pgsql

SQL Server

```
DECLARE @PurchaseName AS NVARCHAR(50)
SELECT @PurchaseName = [Name]
    FROM [Purchasing].[Vendor]
    WHERE BusinessEntityID = 1492
PRINT @PurchaseName
```



Inline T-SQL vs pl/pgsql

Postgres

```
DO $$  
    DECLARE purchase_name TEXT;  
  
BEGIN  
  
    SELECT name  
    FROM purchasing.vendor  
    WHERE business_entity_id = 1492 INTO purchase_name;  
  
    RAISE NOTICE '%', purchase_name;  
END;  
$$
```



Functions

SQL Server

- Scaler
- Inline Table Valued
- Multi-statement Table Valued
- No tuning hints for query planner
- TSQL Only

Postgres

- Scaler
- Table/Set Types
- Triggers
- Query planner tuning parameters
- Any procedural language, but typically PL/pgSQL

Stored Procedures

SQL Server

- Complex logic
- Multi-language
- Contained in calling transaction scope
- With some work, can return results

Postgres

- Complex logic
- Multi-language
- Can issue COMMIT/ROLLBACK and keep processing
- Can return a single INOUT value
- Introduced in PostgreSQL 11

Triggers

- Functions applied to tables
- INSERT/UPDATE/DELETE
- BEFORE/AFTER/INSTEAD OF
- ROW and STATEMENT
- Conditional
- NEW and OLD internal variables
- ...Reusable across tables!

Create Trigger Function

Postgres

```
CREATE OR REPLACE FUNCTION user_log()
RETURNS TRIGGER AS
$BODY$
BEGIN
    INSERT INTO UserLog (Username, Message) VALUES (NEW.Username, NEW.Message)
END;
$BODY$
LANGUAGE plpgsql;
```

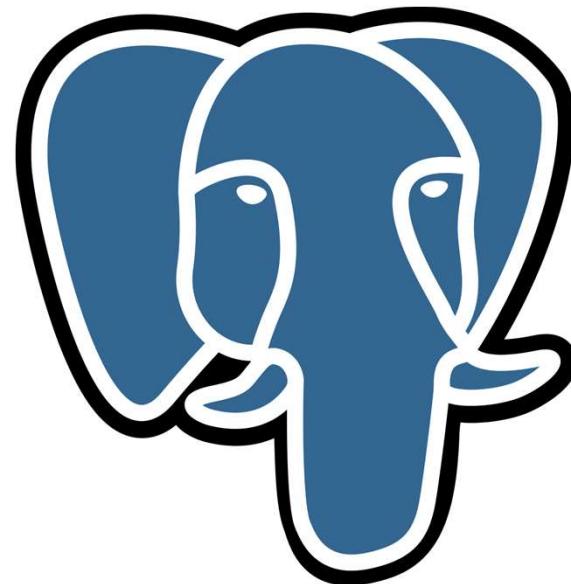
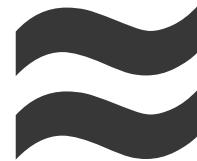


Apply Trigger

Postgres

```
CREATE TRIGGER tu_users
AFTER UPDATE
ON Users
FOR EACH ROW
WHEN (OLD.FirstName IS DISTINCT FROM NEW.FirstName)
EXECUTE PROCEDURE user_log();
```





09/10
Query Tuning

EXPLAIN in Practice

- EXPLAIN = Estimated plan
- EXPLAIN ANALYZE = Actual plan
- EXPLAIN (ANALYZE,BUFFERS)
 - Query plan with disk IO
- EXPLAIN (ANALYZE,BUFFERS,VERBOSE)
 - Additional details on columns, schemas, etc.

Always use
BUFFERS for
query tuning

EXPLAIN

- Inverted tree representation
- There is no built-in visual execution plan
- EXPLAIN provides textual plan
 - pgAdmin does do some visualization
- Websites for visualizations and suggestions
 - <https://www.pgmustard.com/>
 - <https://explain.depesz.com/>

Reading EXPLAIN



--> Nodes



Read inside-out



Join/Aggregate/Merge/Append at each level

EXPLAIN (ANALYZE, BUFFERS)

```
SELECT customer_name, count(*), date_part('year',order_date)::int order_year
FROM sales.orders o
INNER JOIN sales.customers c ON o.customer_id=c.customer_id
WHERE c.customer_id=100
GROUP BY customer_name,order_year
ORDER BY order_year DESC;
```

GroupAggregate (cost=1827.91..1830.76 rows=102 width=35) (actual time=11.956..13.754 rows=4 loops=1)
Group Key: ((date_part('year')::text, (o.order_date)::timestamp without time zone))::integer, c.customer_name
Buffers: shared hit=903

-> Sort (cost=1827.91..1828.18 rows=107 width=27) (actual time=11.700..12.686 rows=107 loops=1)
Sort Key: ((date_part('year')::text, (o.order_date)::timestamp without time zone))::integer DESC, c.customer_name
Sort Method: quicksort Memory: 33kB
Buffers: shared hit=903

-> Nested Loop (cost=0.28..1824.30 rows=107 width=27) (actual time=0.113..10.649 rows=107 loops=1)
Buffers: shared hit=903

-> Index Scan using pk_sales_customers on customers c (cost=0.28..2.49 rows=1 width=27)
Index Cond: (customer_id = 100)
Buffers: shared hit=3

-> Seq Scan on orders o (cost=0.00..1819.94 rows=107 width=8) (actual time=0.053..8.413 rows=107 loops=1)
Filter: (customer_id = 100)
Rows Removed by Filter: 73488
Buffers: shared hit=900

Planning Time: 0.267 ms
Execution Time: 13.934 ms



```
GroupAggregate (cost=1827.91..1830.76 rows=102 width=35) (actual time=11.956..13.754 rows=4 loops=1)
  Group Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer), c.customer_name
  Buffers: shared hit=903
-> Sort (cost=1827.91..1828.18 rows=107 width=27) (actual time=11.700..12.686 rows=107 loops=1)
  Sort Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer) DESC, c.customer_name
  Sort Method: quicksort Memory: 33kB
  Buffers: shared hit=903
-> Nested Loop (cost=0.28..1824.30 rows=107 width=27) (actual time=0.113..10.649 rows=107 loops=1)
  Buffers: shared hit=903
    -> Index Scan using pk_sales_customers on customers c (cost=0.28..2.49 rows=1 width=27)
      Index Cond: (customer_id = 100)
      Buffers: shared hit=3
    -> Seq Scan on orders o (cost=0.00..1819.94 rows=107 width=8) (actual time=0.053..8.413 rows=107 loops=1)
      Filter: (customer_id = 100)
      Rows Removed by Filter: 73488
      Buffers: shared hit=900
```

Planning Time: 0.267 ms

Execution Time: 13.934 ms



GroupAggregate

(cost=1827.91..1830.76 rows=102 width=35)
(actual time=11.956..13.754 rows=4 loops=1)

Estimate vs.
Actual

Group Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer), c.customer_name

Buffers: shared hit=903

Total 8kb Pages –
"Shared" = Memory / "Read" = Disk

Sort Method: quicksort Memory: 33kB

Memory or Disk Sort?

Buffers: shared hit=903

-> Nested Loop (cost=0.28..1824.30 rows=107 width=27) (actual time=0.113..10.045 rows=107 loops=1)

Buffers: shared hit=903

-> Index Scan using pk_sales_customers on customers c (cost=0.28..2.49 rows=1 width=27)

Index Cond: (customer_id = 100)

Buffers: shared hit=3

-> Seq Scan on orders o (cost=0.00..1819.94 rows=107 width=8) (actual time=0.053..8.413 rows=107 loops=1)

Filter: (customer_id = 100)

Rows Removed by Filter: 73488

Buffers: shared hit=900

Seq = All Rows
from table

Planning Time: 0.267 ms

High Planning
Time?

Execution Time: 13.934 ms

explain.depesz.com

PostgreSQL's explain analyze made readable

[new explain](#) [history](#) [help](#) [about](#) [contact](#)

[login](#)

Result: TXM1

[Settings](#)

[Add optimization](#)

#	exclusive	inclusive	rows x	rows	loops	node
1.	1.164	12.898	↑ 25.5		4	1 → GroupAggregate (cost=1,827.91..1,830.76 rows=102 width=35) (actual time=10.690..12.898 rows=4 loops=1) Group Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer), c.customer_name Buffers: shared hit=903
2.	2.829	11.734	↑ 1.0	107	1	1 → Sort (cost=1,827.91..1,828.18 rows=107 width=27) (actual time=10.442..11.734 rows=107 loops=1) Sort Key: ((date_part('year'::text, (o.order_date)::timestamp without time zone))::integer) DESC, c.customer_name Sort Method: quicksort Memory: 33kB Buffers: shared hit=903
3.	2.071	8.905	↑ 1.0	107	1	1 → Nested Loop (cost=0.28..1,824.30 rows=107 width=27) (actual time=0.096..8.905 rows=107 loops=1) Buffers: shared hit=903
4.	0.044	0.044	↑ 1.0	1	1	1 → Index Scan using pk_sales_customers on customers c (cost=0.28..2.49 rows=1 width=27) (actual time=0.023..0.044 rows=1 loops=1) Index Cond: (customer_id = 100) Buffers: shared hit=3
5.	6.790	6.790	↑ 1.0	107 - 73,488	1	1 → Seq Scan on orders o (cost=0.00..1,819.94 rows=107 width=8) (actual time=0.040..6.790 rows=107 loops=1) Filter: (customer_id = 100) Rows Removed by Filter: 73,488 Buffers: shared hit=900

Planning time : 0.423 ms

Execution time : 13.140 ms



`pg_stat_statement`

- MUST HAVE Extension for query analysis
- Installed in all/most DBaaS offerings
- Does not save plans or provide recommendations
- Allows you to do triage: “Is this a big query issue, or death by 1,000 small cuts?”

pg_stat_statement: total time w/ cache hit ratio

```
SELECT query, calls, total_time, rows,  
       100.0 * shared_blks_hit/nullif(shared_blks_hit + shared_blks_read, 0)  
    AS hit_percent  
FROM pg_stat_statements  
ORDER BY total_time DESC  
LIMIT 5;
```

-[RECORD 1]-----

query	SELECT * FROM sales.orders where customer_id = ?
calls	1000
total_time	2451.000000000000
Rows.	521
hit_percent	99.9873421

*Lots of good videos/tutorials, but also [see the docs](#) for more help and ideas.

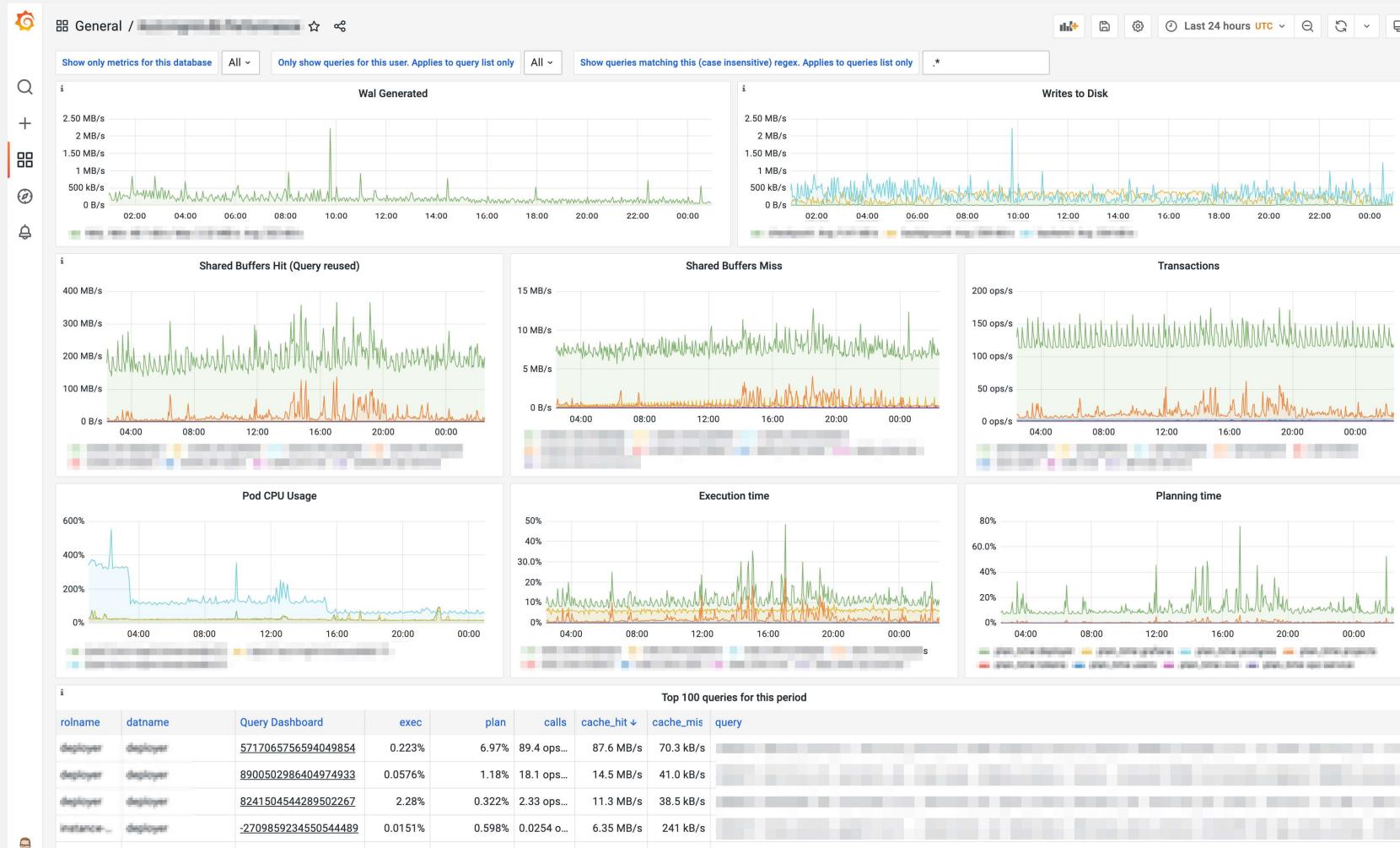


`pg_stat_statement`

- Statistics are cumulative 😞
- Use third party tools to help you monitor
- Install CRON extensions and query metrics periodically

Point-in-time metrics example: <https://www.youtube.com/watch?v=wTmKGyiqoCk>





10/10
Help and Community

Community & Help

- Postgres Slack
- Twitter (#postgres, #postgresql, #pghelp)
- StackOverflow (dba.stackexchange.com)
- Planet Postgres (blog aggregator)
- Mailing lists (postgresql.org/list)
- PostgreSQL Docs (postgresql.org/docs)

Community & Help

- Postgres Weekly (Cooper Press)
- Postgres.fm Podcast
- PGSQl Phriday monthly blog even



Books and Courses

- [The Art of PostgreSQL](#)
- [PostgreSQL Query Optimization \(Apress\)](#)
- [A Curious Moon \(Rob Conery\)](#)

What Questions do you have?





THANK YOU!

github.com/ryanbooz/presentations

