PostgreSQL for the SQL Server Professional

CodeMash 2025

Ryan Booz

redgate

redgate | About me

Ryan Booz

PostgreSQL & DevOps Advocate

@ryanbooz

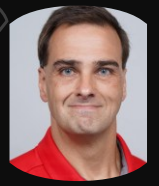
/in/ryanbooz

www.softwareandbooz.com

youtube.com/@ryanbooz





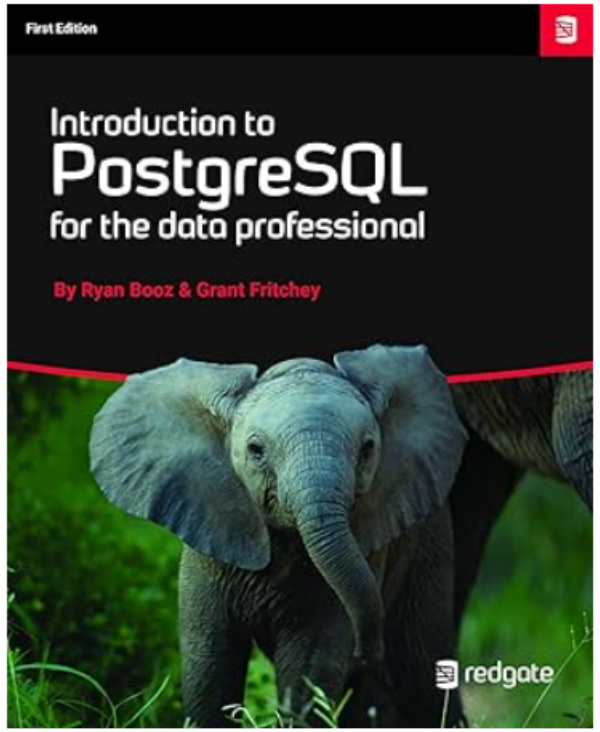
redgate
Simple Talk

A technical journal and community hub

- ✓ Blogs, videos, events, and more
- ✓ Contributions by experts
- ✓ Unbiased, multi-database content

Join the community at Simple-Talk.com

Keep learning after today...



Follow the author

Ryan Booz Follow

Introduction to PostgreSQL for the data professional. Kindle Edition

by Ryan Booz (Author), Grant Fritchey (Author) | Format: Kindle Edition

See all formats and editions

Adoption and use of PostgreSQL is growing all the time. From mom-and-pop shops to large enterprises, more data is being managed by PostgreSQL. In turn, this means that more data professionals need to learn PostgreSQL even when they have experience with other databases. While the documentation around PostgreSQL is detailed and technically rich, finding a simple, clear path to learning what it is, what it does, and how to use it can be challenging. This book seeks to help with that challenge.

We set about giving you **all the basics of how to get started within PostgreSQL**. From getting your first instance, or cluster, up and running, to backups, to server configurations, we cover all the basics. You'll learn about processing languages, monitoring, indexes and more. Every chapter provides fundamental knowledge and guidance so that you can more easily get started working with PostgreSQL. Getting started with PostgreSQL doesn't have to be a challenge and this book will help.

Print length	Language	Publisher	Publication date	File size
452 pages	English	Redgate Books	February 3, 2025	32025 KB

See all details

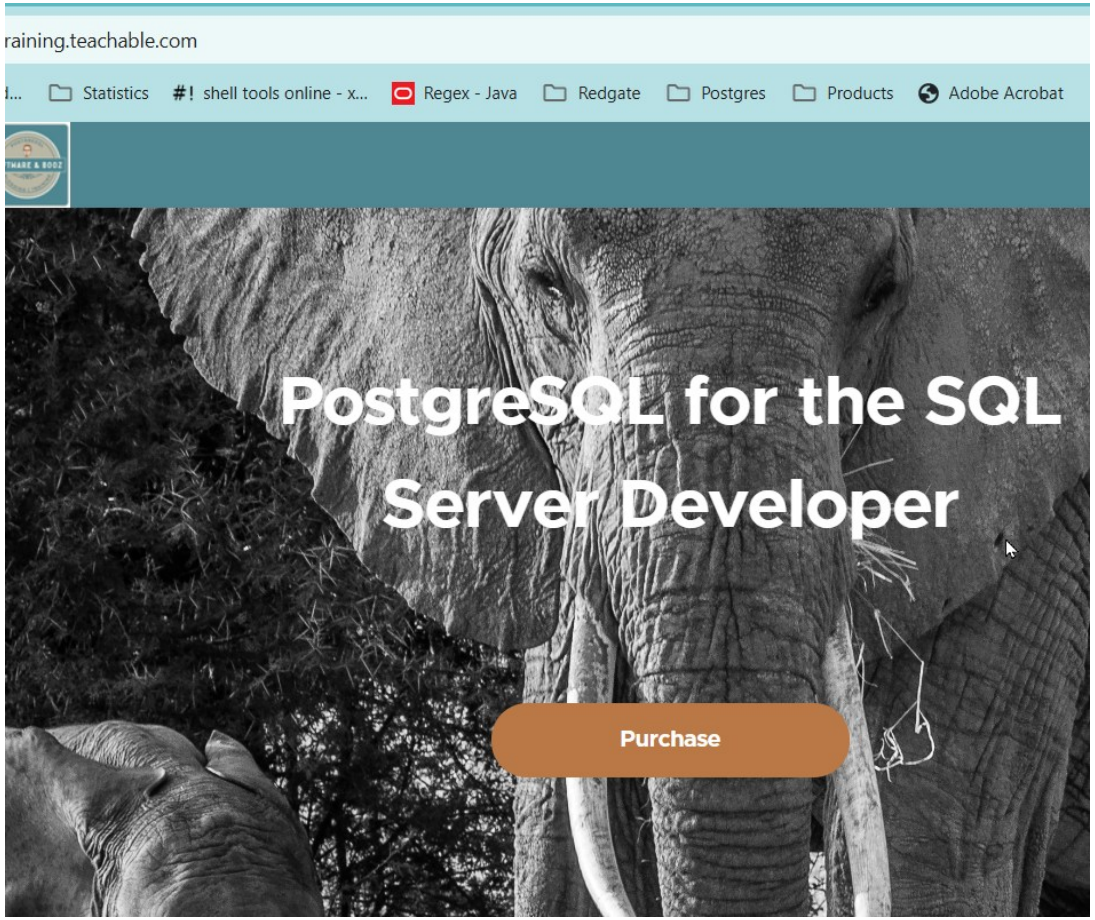
Due to its large file size, this book may take longer to download

Report an issue with this product or seller

Demo Scripts

https://bit.ly/pg-codemash2025

https://software-and-booz-training.teachable.com/



- Full online course coming soon!
- Enter your email for a discount code when it's released in February

redgate

My Journey

Work

Hobby

1999 **2004** **2018** **2020** **2022**

Agenda

- Why PostgreSQL?
- Feature Comparison
- SQL Differences to know about
- IDEs
- Backup and Restore
- Roles and Privileges

redgate

Why PostgreSQL?

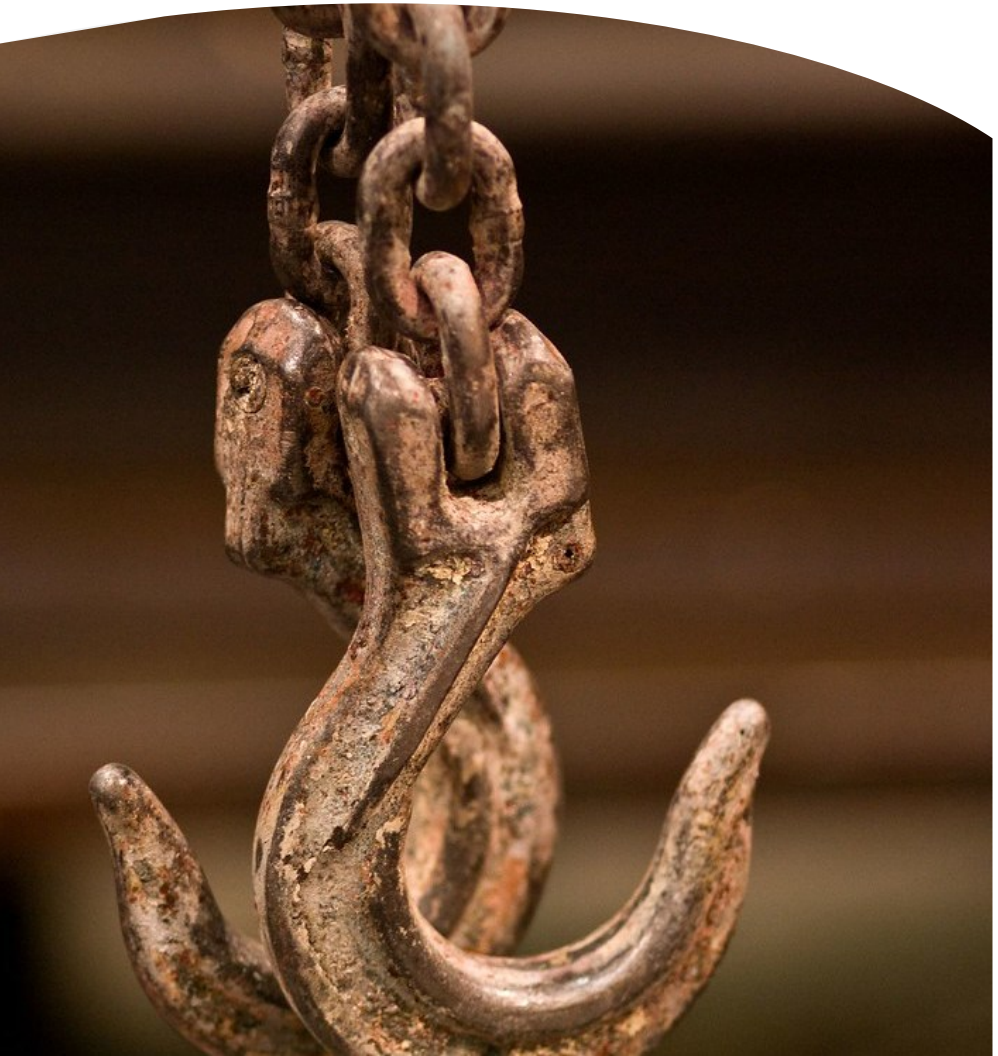
Licensing

- Extremely permissive, open-source license
- Core features can never be paywalled
- Most extensions piggyback on the PostgreSQL License

Amazon RDS

redgate

Extensibility



- >30 hooks available to extensions
- Any supported language, although SQL, C, and Rust are most popular

redgate

Extensibility

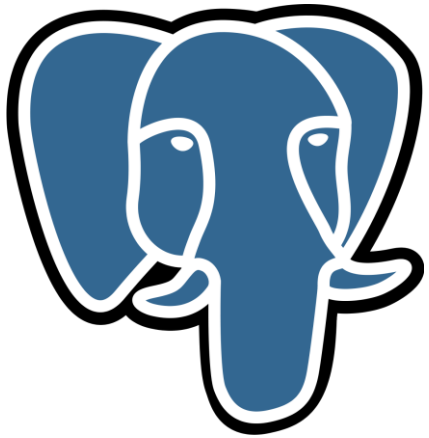
- Not just for extending functionality
- Package commonly used SQL functions/procedures
- Versioning required and helpful

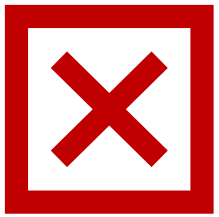
redgate

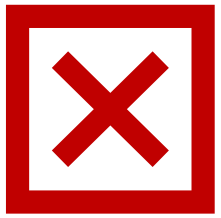
Feature Comparison

redgate

Customizable with Extensions

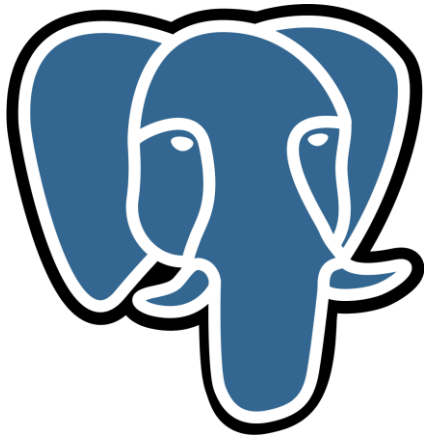


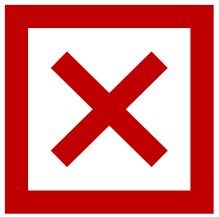




redgate

Native JSON/JSONB Datatype



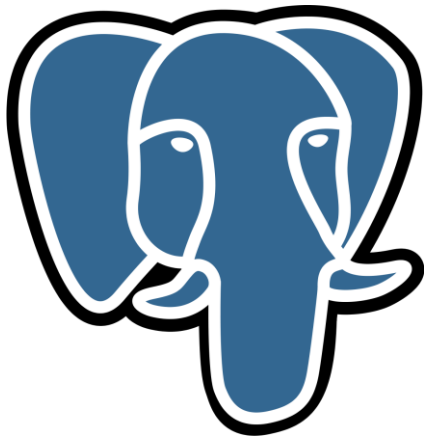


redgate

Multiple Index Types

2

2



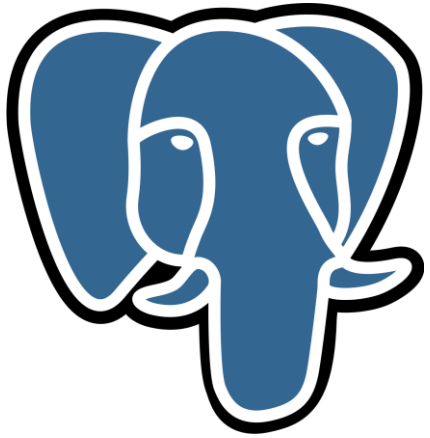
6+

redgate

Native Partitioning

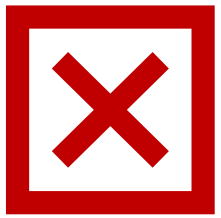
Microsoft® SQL Server®

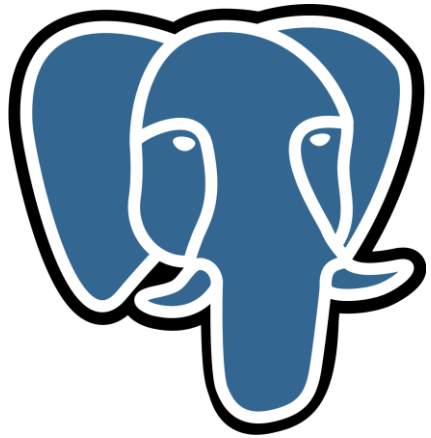
MySQL®



redgate

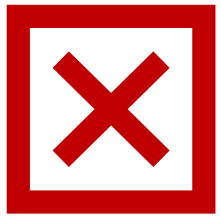
Function/Procedure Multi-language Support

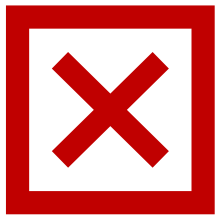


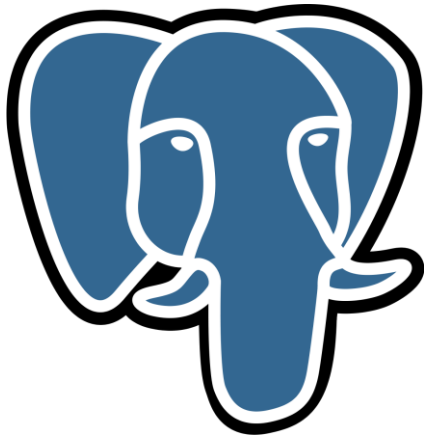


redgate

Transactional DDL

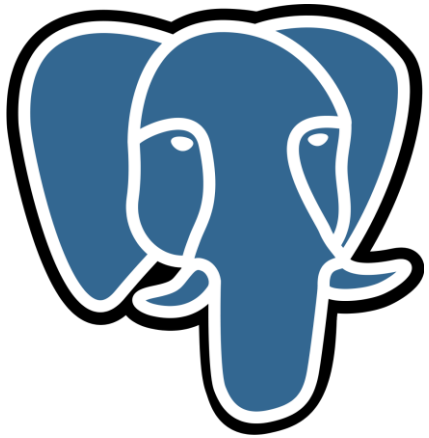






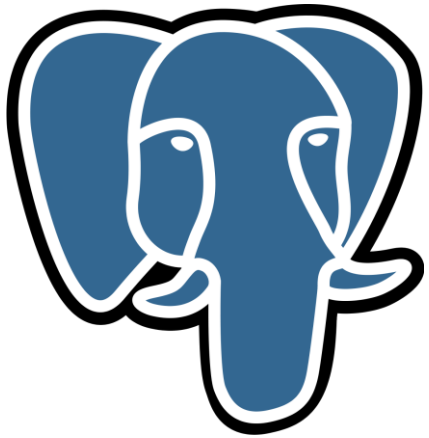
redgate

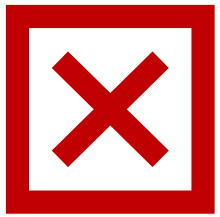
Query Hints



redgate

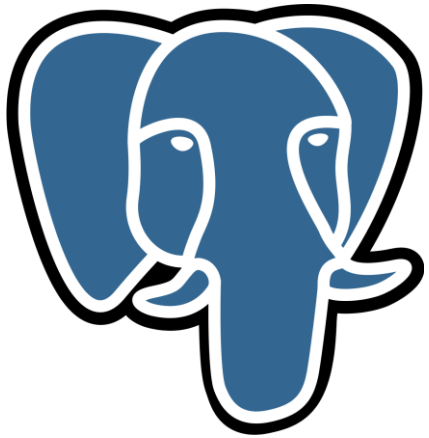
Visual Query Plan Explorer

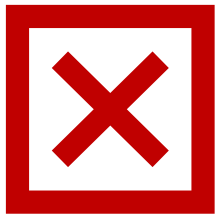




redgate

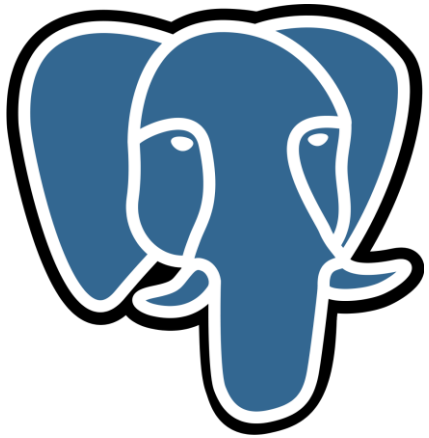
Query Cache and Monitor





redgate

Built-in Job Scheduler



redgate

Lay the Foundation

Instance = Cluster

SQL Server

PostgreSQL

User = Role

SQL Server PostgreSQL

Transaction Log ≈ Write Ahead Log

SQL Server

PostgreSQL

Row = Row ∈ Tuple

SQL Server PostgreSQL PostgreSQL

redgate

[Table One] = "Table One"

SQL Server

PostgreSQL

BULK INSERT ≈ COPY

SQL Server

PostgreSQL

TOAST

The **O**versized-**A**ttribute **S**torage **T**echnique

Transaction Isolation

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

PostgreSQL MVCC at a glance

- Readers don't block writers
- Tuples maintain on-row transaction visibility (xmin, xmax)
- 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)
- **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

Case Rules

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

Data Type Differences

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

🔥 ARRAY, RANGE, ENUM, GEOM 🔥

SQL Things

Case Rules

SQL Server

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

Postgres

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

redgate

Case Rules

SQL Server

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

Postgres

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

redgate

LIMIT ≈ TOP

SQL Server

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

Postgres

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

redgate

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;
```

redgate

SYSDATETIME() ≈ NOW()

SQL Server

```
SELECT SYSDATETIME();
```

Postgres

```
SELECT now();
```

Date Math & Intervals

SQL Server

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

redgate

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;
```

redgate

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
```

redgate

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;
$$
```

redgate

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!

redgate

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;
```

redgate

Apply Trigger

Postgres

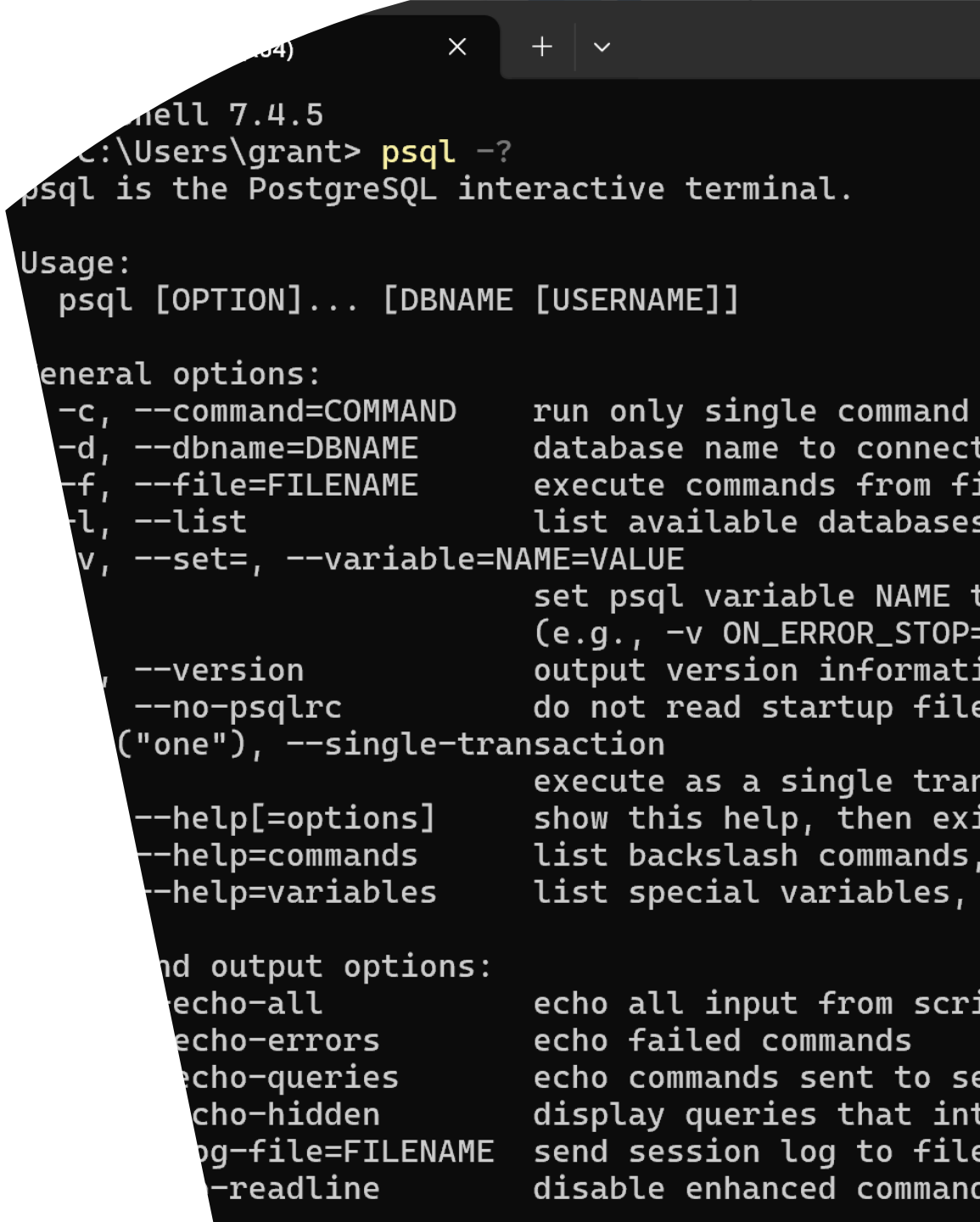
```
CREATE TRIGGER tu_users
AFTER UPDATE
ON Users
FOR EACH ROW
WHEN (OLD.FirstName IS DISTINCT FROM NEW.FirstName)
EXECUTE PROCEDURE user_log();
```

redgate

PostgreSQL IDEs

psql

- Always available (almost)
- Command line
- Can work remote



psql

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

Resources

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

redgate

psql

```
psql -h localhost -U postgres
```

```
psql postgres://joe:abc123@localhost:5432/postgres
```

Execute File or Command

-f to run file

- File is interpreted as starting after connecting
- Slash (meta) commands are 👍 (one per line)

-c to run SQL command

- One or more "-c" are 👍
- Cannot mix/match SQL and slash commands together
- Must be separate "-c" commands

redgate

.psqlrc

- `psql` configuration file typically found at ~/.psqlrc
- Example recommendation:

```
\set QUIET ON

\pset null '🦃'
\timing
\set HISTFILE ~/.psql_history- :HOST - :DBNAME
\set PROMPT1 '%M:%> %n@%/%R%#%x '
\set COMP_KEYWORD_CASE upper

\set QUIET OFF
```

Meta Commands

- \l – List cluster databases
- \c – Connect to specified database
- \d – Describe {object}
- \d+ - Describe {object} with extra details
- \x (on|off) – Display results in wide format
- \timing – Enable query timing output
- \pset – Output settings
- \set – Variable settings

Describe Commands

- \dn – Display user-defined schemas
- \dnS – Include system schemas
- \dt – Display user-defined tables
- \di – Display indexes
- \dv – Display views
- \dm – Display any materialized views
- \ds – Display any sequences
- \dg or \du – list user roles
- \dx – list installed extensions
- \df – Display user-defined functions
  - \dfn = normal functions only
  - \dfaS = aggregate functions only
  - \dfp = procedures only
  - \dft = trigger functions
  - \dfwS = window functions

Edit Commands

- \e – open current query buffer into editor
- \e {filename} – open file into editor
- \ef {function} – like "script for create"
- \ev {view} – like "script for create"

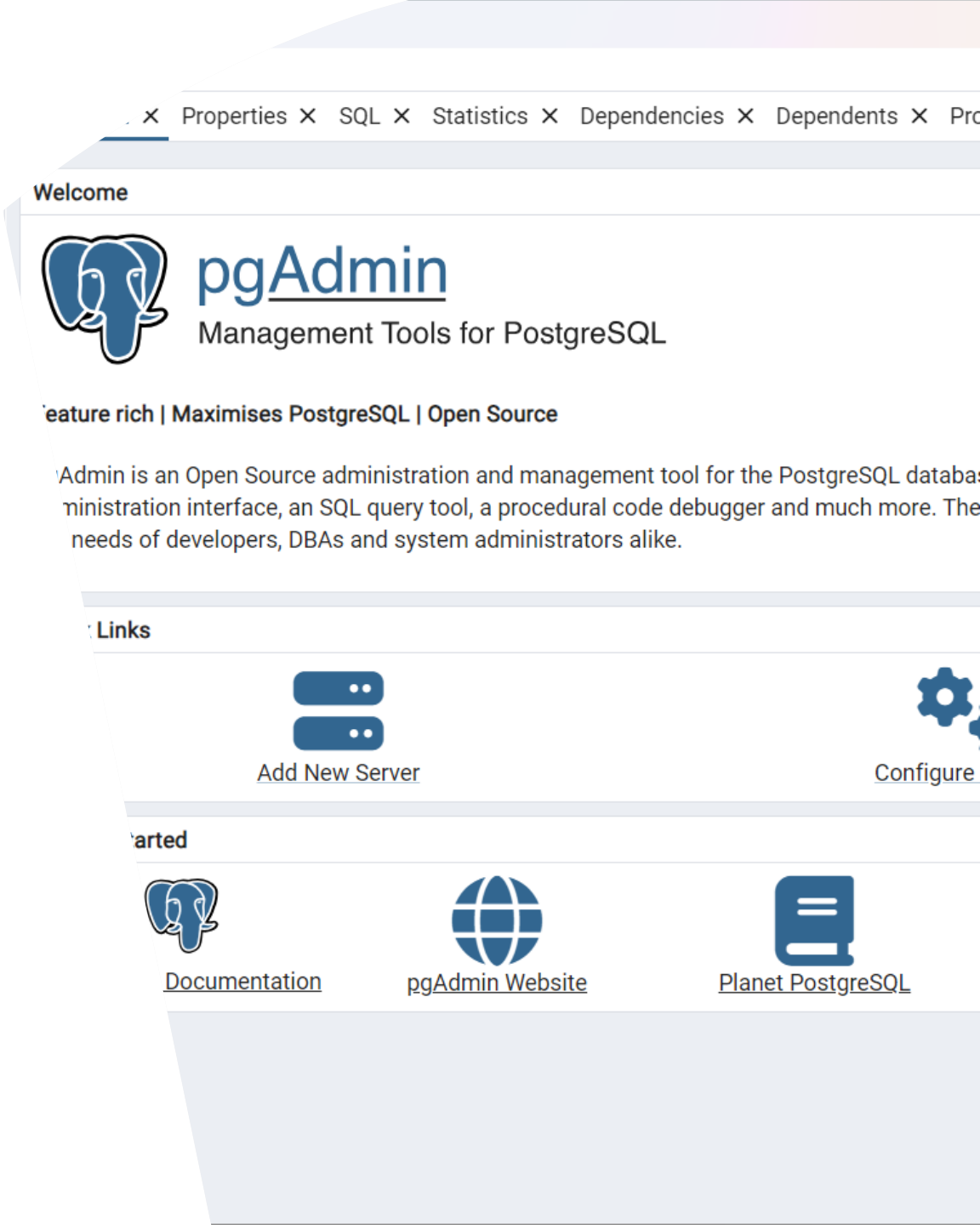
- Cross-platform, Eclipse (Java) based
- Community and paid license
- Multi-database support
- Most similar feeling to SSMS thus far*

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

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

pgAdmin

- Unofficial, "Official" PostgreSQL GUI
- Not installed by default
- Web version for remote work
- Extensions



Backups and Restores

“You are only as good as your last restore”

- Kimberly Tripp

Create

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

Create database (with template1)

```
CREATE DATABASE myDB;
```

pg_dump

- An export, not a true backup
- Not transactionally aware
- No point in time recovery
- One database at a time
- pg_dumpall

redgate

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 -Fc postgres -h localhost  -f test.dump
```

-U = username **-Fc** = format (directory | custom | tar

-h = host **-f** = filename

redgate

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 -Fc test.dump -U postgres -h localhost
```

-U = username **-Fc** = format (directory | custom | tar)

-h = host **-d** = target database

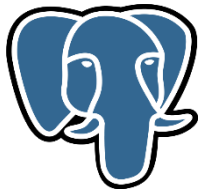
redgate

Roles and Privileges

The Basics

- **Roles –** The principle that can own objects and be granted permissions. E.g. Users & Groups
- **Privileges** – The set of permissions that a role can be granted to objects in a database or cluster
- **Ownership** – Every object in a cluster and database is owned by a role and carries a lot of power
- **Cluster** - The instance of PostgreSQL that's running, each containing one or more databases. A server can have multiple, independent clusters
- **Database** – The object we mostly care about. ;-) Also essential for database connections. Users connect to a database, not the cluster (specifically)

Server/Host (Firewall, Ports)



Cluster

Port: 5432

pg_hba.conf

Cluster

Port: 5433

pg_hba.conf

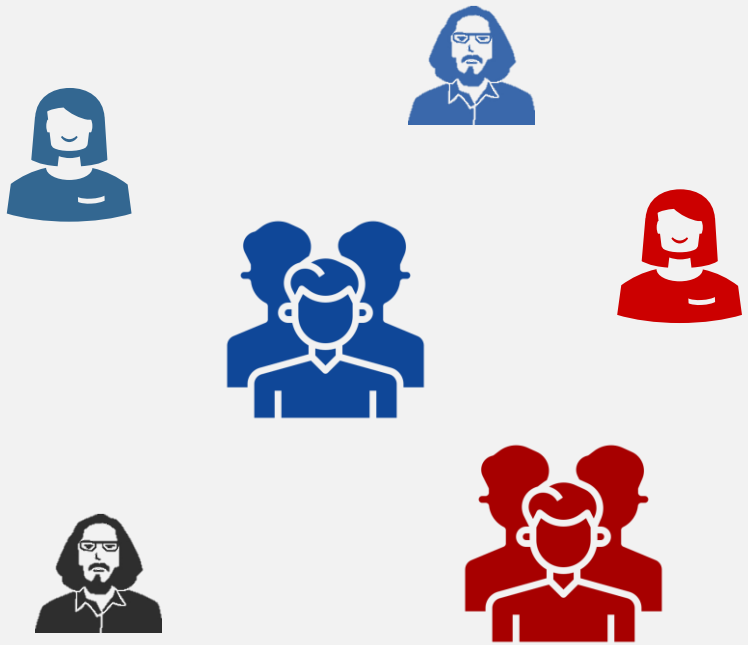
Cluster

Port: 5434

pg_hba.conf

Cluster

ROLES



Databases

pg_hba.conf

- First layer of authentication
- Similar to a firewall ruleset for PostgreSQL
- Cloud vendors largely manage this for you

Which hosts & roles, can connect to what databases, using what authentication method?

```
# Allow any user on the local system to connect to any database with
# any database user name using Unix-domain sockets (the default for local
# connections).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust

# The same using local loopback TCP/IP connections.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             127.0.0.1/32            trust

# Allow any user from host 192.168.12.10 to connect to database
# "postgres" if the user's password is correctly supplied.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.12.10/32        scram-sha-256
```

https://www.postgresql.org/docs/current/auth-pg-hba-conf.html

Roles

- Own databases, schemas, and objects
  - Tables, Functions, Views, Etc.
- Have cluster-level privileges (attributes)
- Granted privileges to databases, schemas, and objects
- Can possibly grant privileges to other roles

PostgreSQL 15 Attributes

LOGIN

SUPERUSER

CREATEROLE

CREATEDB

REPLICATION LOGIN

PASSWORD

INHERIT

BYPASSRLS

CONNECTION LIMIT

Users and Groups

- Semantically the same as roles
- By Convention:
  - User = `LOGIN`
  - Group = `NOLOGIN`
- PostgreSQL 8.2+ `CREATE (USER|GROUP)` is an alias

```
CREATE USER user1 WITH PASSWORD 'abc123' INHERIT;
```

```
CREATE GROUP group1 WITH INHERIT;
```

```
CREATE ROLE user1 WITH LOGIN PASSWORD 'abc123' INHERIT;
```

PostgreSQL Superuser

-  is created by default when the cluster is initialized
- Typically named `postgres` because the system process user initiates a `initdb`
- Bypasses all security checks except `LOGIN`
- Full privilege to do "anything"
- Treat superuser with care (like `root` on Linux)

redgate

Most cloud providers do not provide superuser access

redgate

Superuser-like

- Create a role with the right level of control
- Recommend adding `CREATEROLE` and `CREATEDB`
- Allows user management and database ownership
- May still limit some actions (e.g. installing extensions limited to superuser)

PostgreSQL 15+ Privileges

SELECT

INSERT

UPDATE

DELETE

TRUNCATE

REFERENCES

TRIGGER

CREATE

CONNECT

TEMPORARY

EXECUTE

USAGE

SET

ALTER SYSTEM

redgate

PUBLIC Role

- All roles are granted implicit membership to `PUBLIC`
- The public role cannot be deleted
- Granted `CONNECT`, `USAGE`, `TEMPORARY`, and `EXECUTE` by default
- <=PG14: `CREATE` on the public schema by default
- >=PG15: No `CREATE` on public schema by default

Granting Privileges

```
-- grant the ability to create a schema
GRANT CREATE ON DATABASE app_db TO admin1;

-- see and create objects in schema
GRANT USAGE,CREATE IN SCHEMA demo_app TO dev1;

-- allow some roles only some privileges
GRANT SELECT,INSERT,UPDATE
ON ALL TABLES IN SCHEMA demo_app TO jr_dev;
```

Granting Privileges

- Remember, explicit grants only effect existing database objects!

```
-- This will only grant to existing objects
GRANT ALL TO ALL TABLES IN SCHEMA public TO dev1;
```

redgate

More Detail on GRANT and REVOKE

What the privileges mean:

https://www.postgresql.org/docs/current/ddl-priv.html

How to GRANT privileges:

https://www.postgresql.org/docs/current/sql-grant.html

How to REVOKE privileges:

https://www.postgresql.org/docs/current/sql-revoke.html

Standard Configuration and Maintenance

Over 300 configuration options

redgate

Configuration

- ***Every non-serverless Postgres should be tuned***
- Edit `postgresql.conf`

Helpful Tuning Resources

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

https://www.youtube.com/watch?v=IFIXpm73qtk

https://postgresqlco.nf/

redgate

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

Configuration – 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

shared_buffers

- Configurable memory reserved for data cache
- Start at 25% of total server memory
- Keep a watch on Cache Hit Ratio
  - <90% consistently, increase shared_buffers or increase instance resources

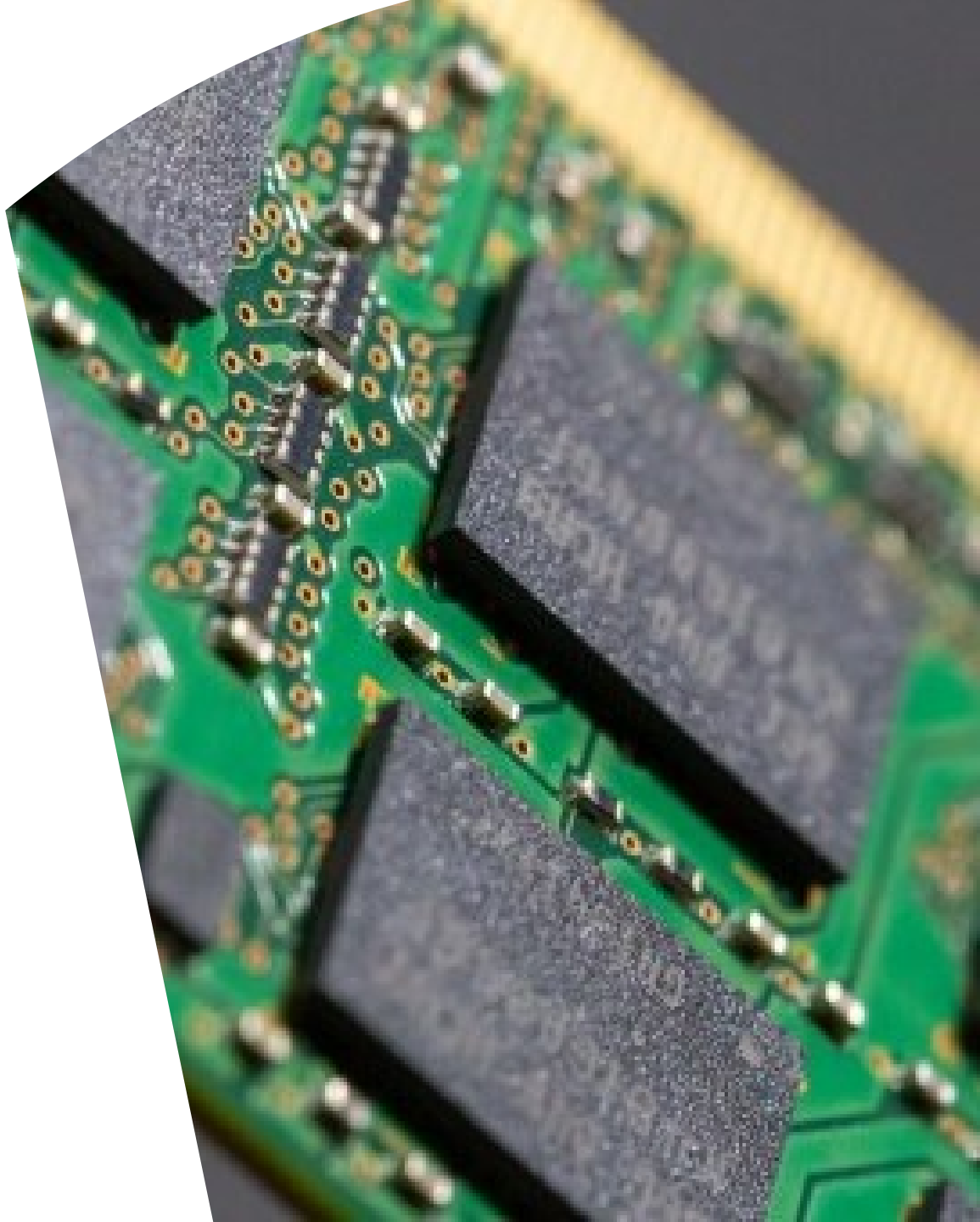
Cache Hit Ratio

```
SELECT
    sum(heap_blks_read) as heap_read,
    sum(heap_blks_hit) as heap_hit,
    sum(heap_blks_hit) / (sum(heap_blks_hit) +
        sum(heap_blks_read)) as ratio
FROM
pg_statio_user_tables;
```

redgate

work_mem

- Single most important setting for complex workloads*
- Memory used by query operations like sorting and hashing



*It Depends…

work_mem

- Postgres doesn't pre-allocate memory before running a query (i.e. memory grant)
- When you see "external disk" operations in query plans, work_mem likely needs to be increased
- 4MB default, typically tune higher (8/16MB)
- Set on individual query sessions when necessary

Configuration – The big 4

maintenance_work_mem:

- memory for background tasks like VACUUM and ANALYZE
- Index maintenance tasks

max_connections:

- Defaults to 100
- Many providers start at 200
- Total memory usage equals:

```
max_connections * work_mem * hash/sort operation
```

maintenance_work_mem

- `maintenance_work_mem` is the amount of RAM that a maintenance process like autovacuum can use to process pages of data. If it is not sufficient, vacuuming (and other essential maintenance processes) will take more time and fall behind.
- If autovacuum isn't completing it's work regularly, check this setting!

Default setting = 64MB

redgate

autovacuum_work_mem

- Amount of memory used by each autovacuum process
- Correlated to **autovacuum_max_workers**
- Tune if **maintenance_work_mem** is increased dramatically (>=1GB)

Default setting = maintenance_work_mem

redgate

Configuration (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

Configuration (other)

autoanalyze:

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

default_toast_compression:

- pglz (default)
- Consider **lz4** instead

Configuration (other)

random_page_cost:

- "scale" of cost for random page access
- Defaults to 4.0 – spinning rust
  - Increases likelihood of table scans
- Change to 1.1 in most environments and monitor