Introduction to PostgreSQL

Ryan Booz & Grant Fritchey

redgate | About me

Ryan Booz

Redgate Advocate

@ryanbooz

/in/ryanbooz

www.softwareandbooz.com

youtube.com/@ryanbooz

redgate | About me

Grant Fritchey

DevOps Advocate

Microsoft PostgreSQL MVP
AWS Community Builder

@gfritchey

Grant.Fritchey@Red-Gate.com

/in/gfritchey

scarydba.com

Why PostgreSQL?

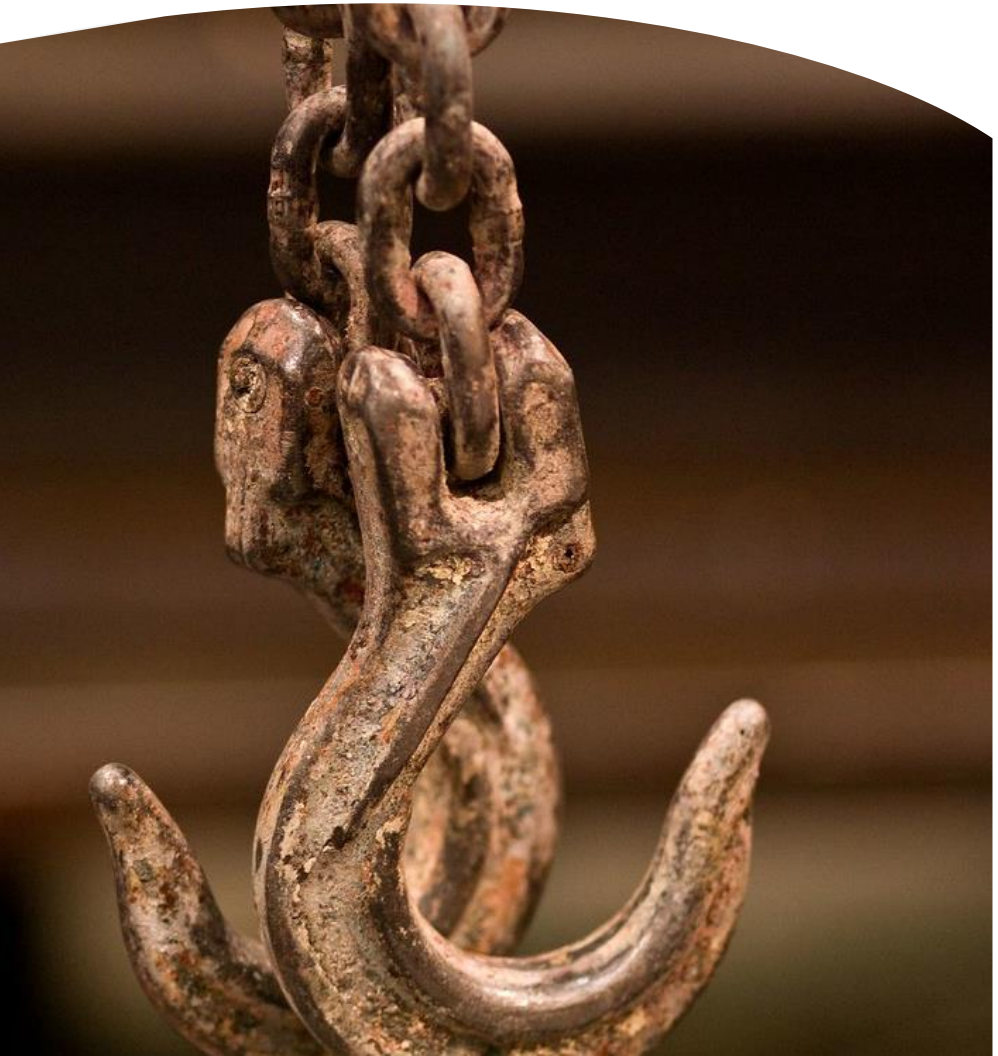
Licensing

- 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

Lay the Foundation

Instance = Cluster

SQL Server

PostgreSQL

User = Role

SQL Server PostgreSQL

redgate

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)**

redgate

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`

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;
```

redgate

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();
```

redgate

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
```

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
```

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();
```

DEMO!

redgate

PASS Session

The Accidental PostgreSQL Database Administrator


Wednesday, Nov. 6


5:00-6:00


Room 447

redgate

PASS Session

MSSQL and PostgreSQL Similarities and Differences for the SQL Server DBA


Friday, Nov. 1


1:00-2:00


Room 445-446

redgate

Installation

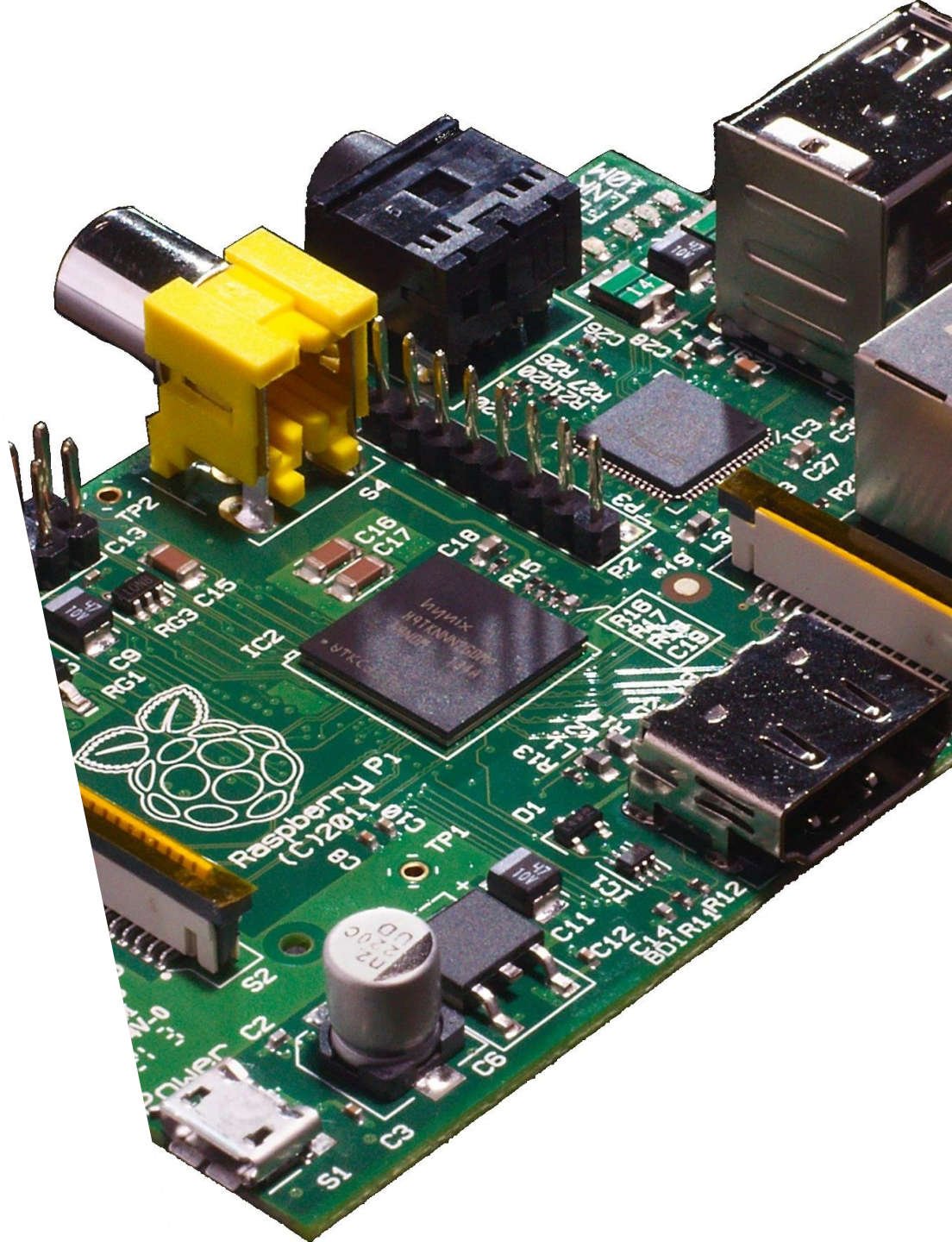
Supported Operating Systems

- Linux
- Windows
- Containers
- Cloud

redgate

Minimum Hardware

- Chip sets
  - x86 family
  - ARM
  - SPARC
  - PowerPC
  - S/390
  - MIPS
  - RISC-V
  - PA-RISC
- Recommended Minimums
  - > 1ghz speed CPU
  - > 2gb memory
  - > 50kb storage



Packaged Bits

- Core
  - For the “hackers”
- Packages
  - OS
  - Special functions

Containers

- Easiest way to get started
- Get in the habit of setting up volumes
- Multiple forks available

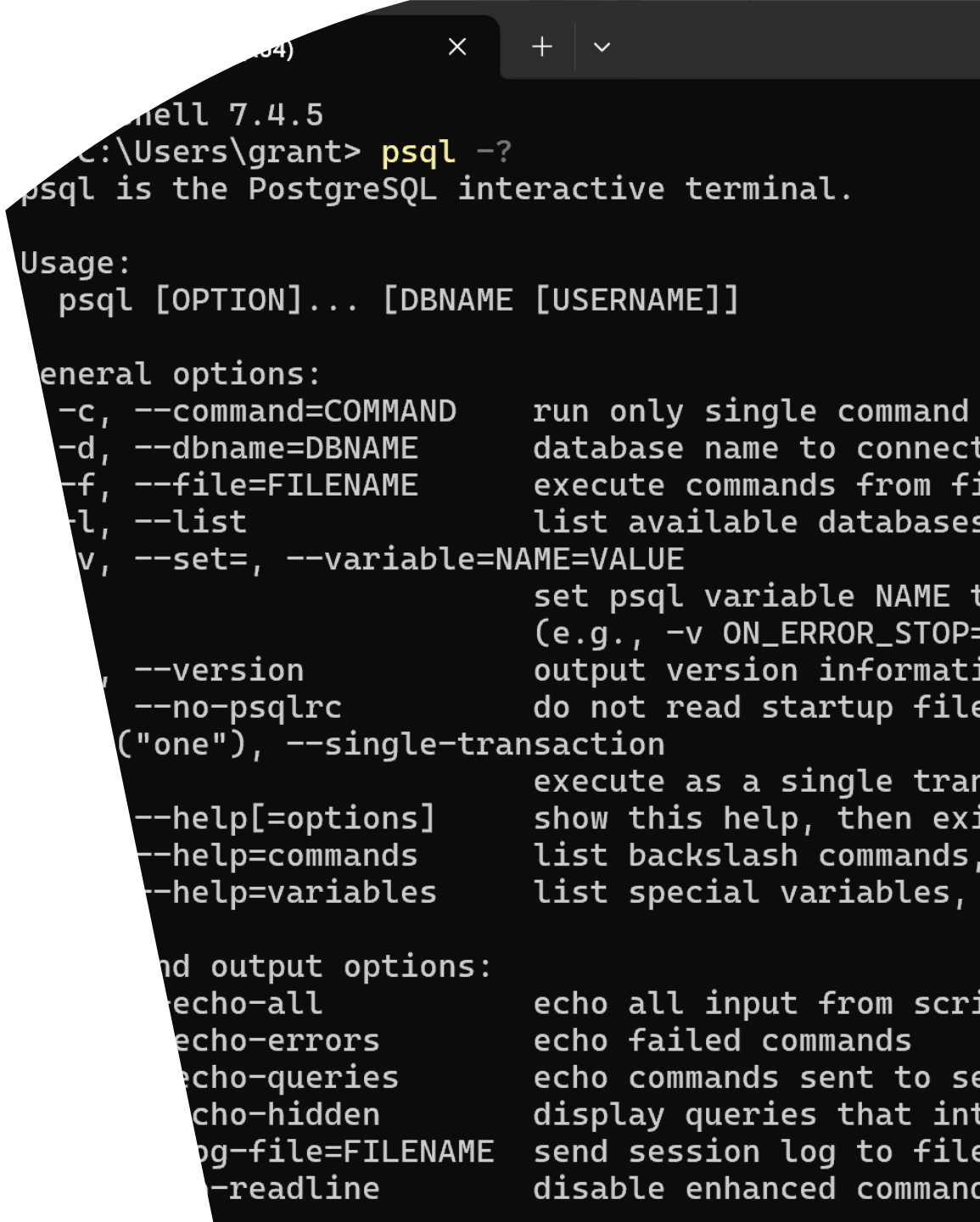
DEMO!

redgate

PostgreSQL Tools

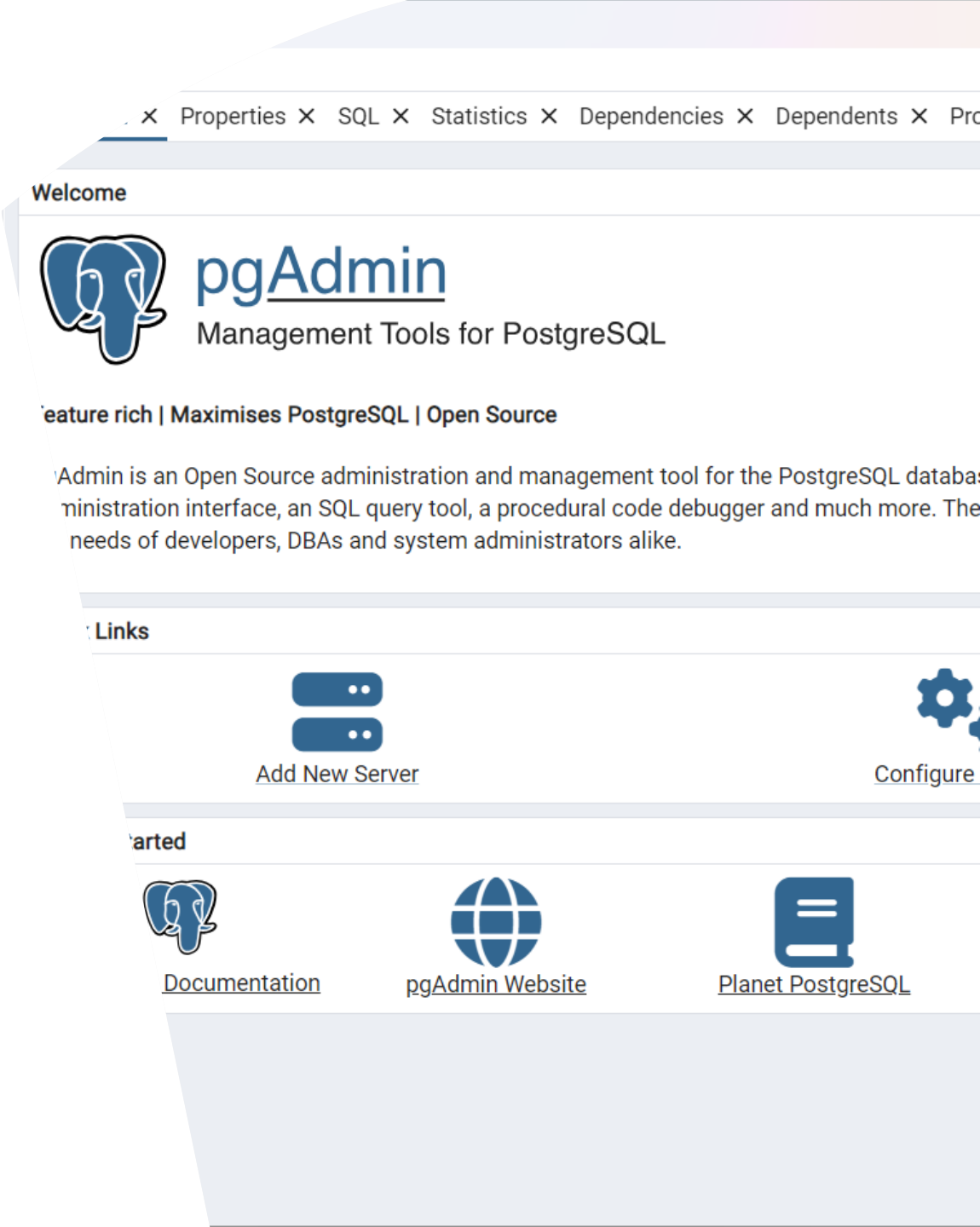
psql

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



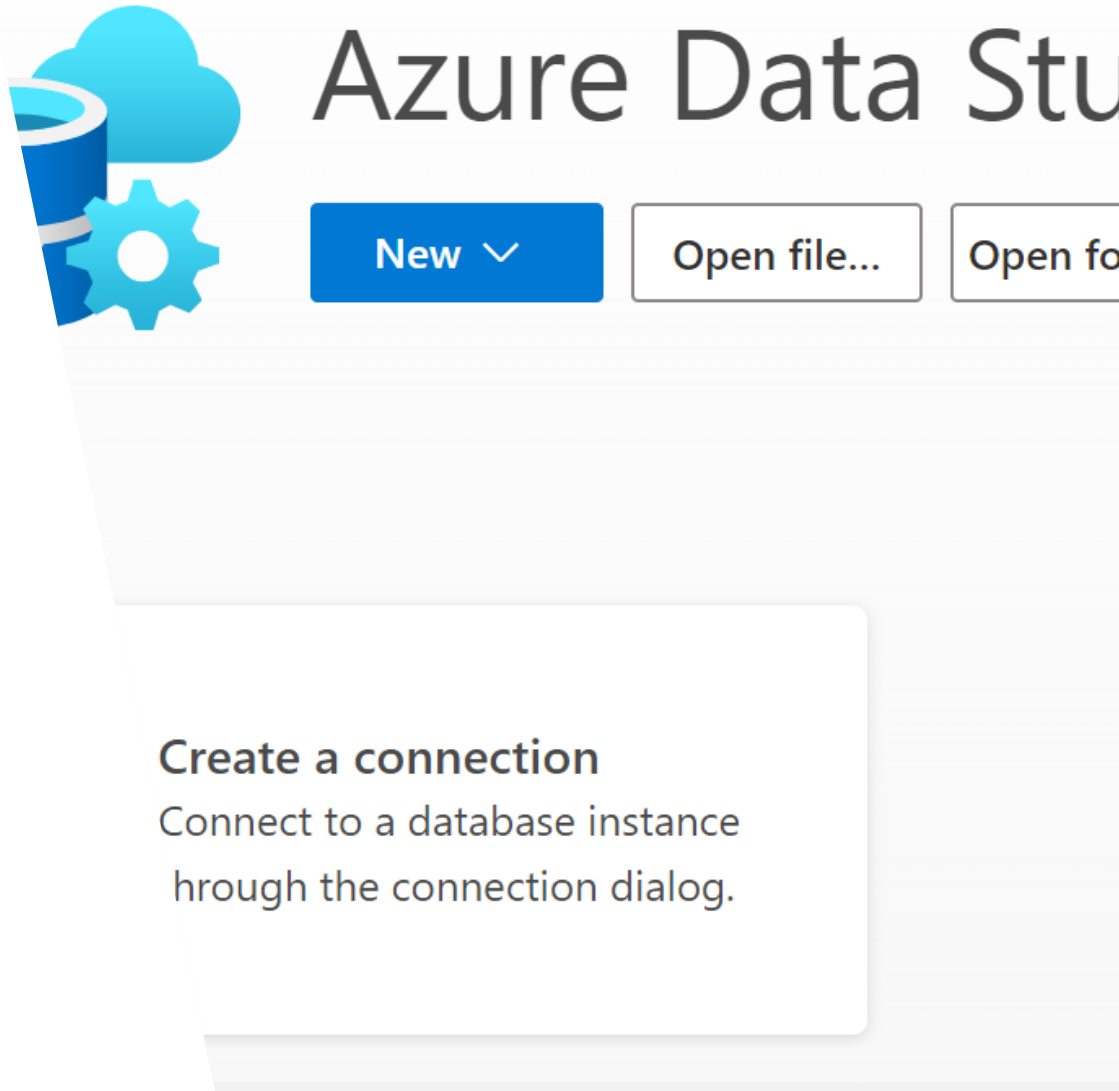
pgAdmin

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



Azure Data Studio/VS Code

- PostgreSQL Plugin
- Also works in VSCode
- Good for basics
- Good integration with source control



DBeaver

- Open source
  - Also paid
- Multi-platform
  - Good and bad
- Extensions
- Some code completion

DEMO!

redgate

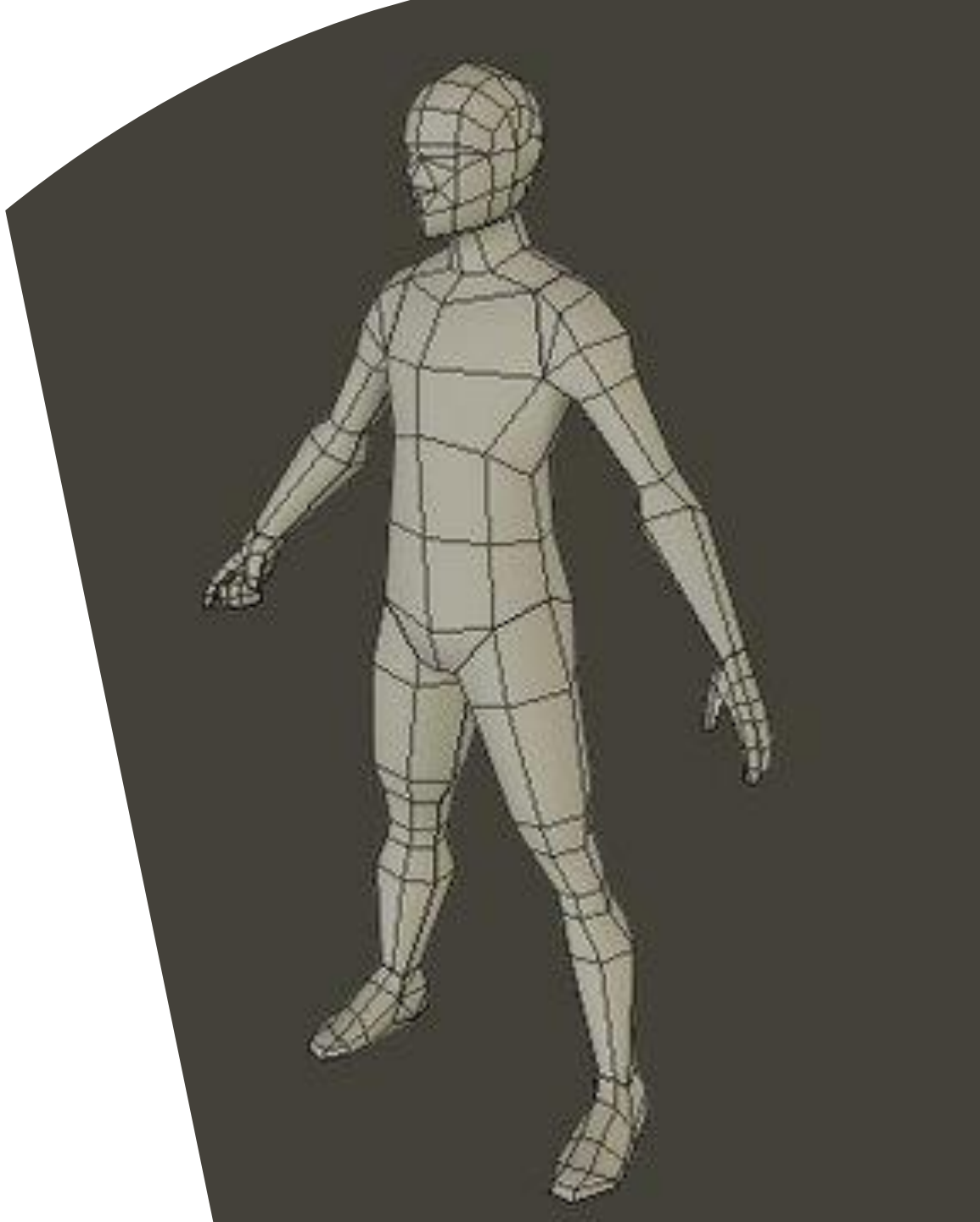
Databases

What Are Databases?

- Primary means of organizing storage, access and behavior
- Equivalent to SQL Server databases
- Control storage
- 3 by default
  - postgres
  - template0
  - template1

Templates

- Think, model db
- template0 – leave this alone
- template1 – customizable
- Custom templates
- You can choose the template



CREATE DATABASE

- Defaults to template1
- You can pick the template
- You can define the owner

ALTER DATABASE

- Change database behaviors
- Change database properties
- Modify behaviors of code and objects within database

DROP DATABASE

- Boom!
- IF EXISTS is a good idea
- WITH (FORCE) is probably not

DEMO!

redgate

Restores and Backups

“You are only as good as your last restore”

- Kimberly Tripp

Backups

- pg_dump
- pg_dumpall
- pg_basebackup
- Restore from dump
- pg_restore

redgate

pg_dump

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

redgate

Common pg_dump Behaviors

- -F – provides a file format, common to compress in tar file
- -c – adds drop commands so you don’t have to drop & recreate the db (sometimes)
- -a – data only
- -s – schema only
- -C – creates a database from dump

redgate

pg_basebackup

- Backs up the server
- Starting point for point in time recovery

redgate

WAL Archiving

- Write Ahead Log (WAL)
- Log backups for point in time recovery

redgate

WAL Archiving

- WAL level
  - Replica
  - Logical
- Archive mode enabled
- Define archive command
  - File copy, but more

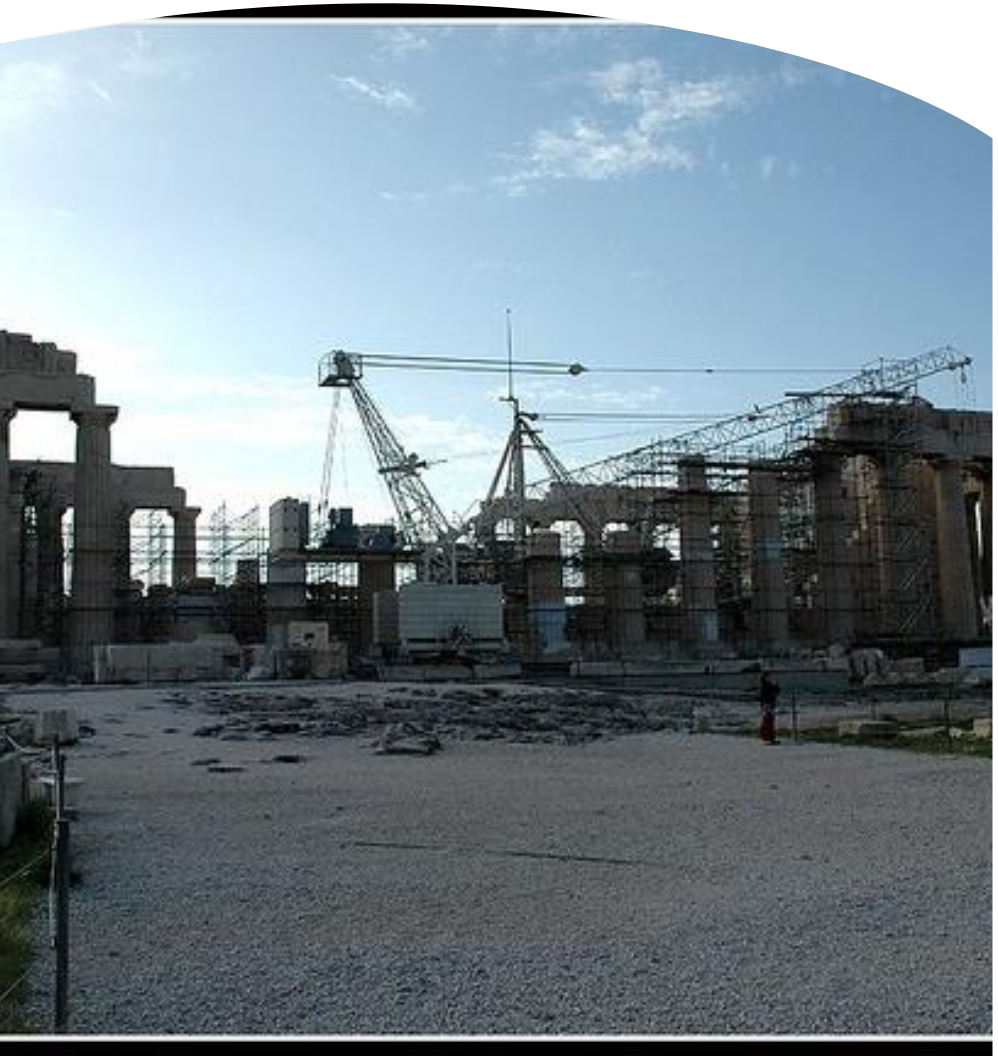
redgate

https://github.com/ryanbooz/bluebox

- GitHub - ryanbooz/bluebox: An updated sample database for PostgreSQL, building off of the Pagila database

redgate

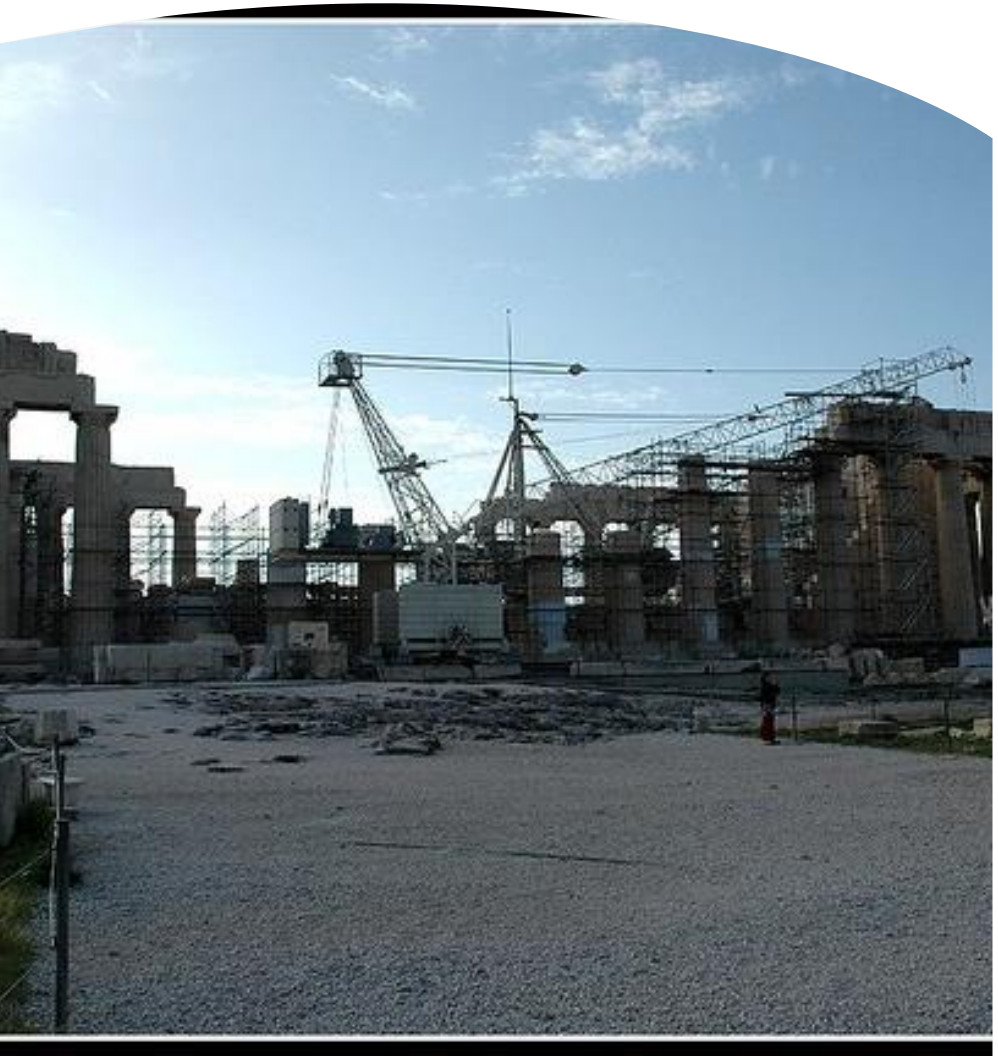
Restore



- “Replay” a dump
- pg_restore

redgate

Restore to Point In Time



1. Run `pg_switch_wal`
2. Stop the server
3. Clean data directory or make a new one
4. Restore the Base Backup files
5. Remove all files from pg_wal/
6. Edit postgresql.conf for restore command and point in time
7. Create a file called recovery.signal in the cluster data directory
8. Start the service

redgate

DEMO!

redgate

Database Objects

Data Types

- Twenty (20) Categories of data types
- Depending on how you count, ~125 data types
- We won’t be listing them all

Common Data Types

- Numeric
  - Integers, decimals & math
- Character
  - No need for Unicode
- Date/Time
- Binary
- Enums
- Geometric
- JSON
- Array

Data Type Behaviors

- Fixed or variable length, depending on type
- Most can be NULL
- DEFAULT values can be defined

Tables and Columns

- CREATE TABLE
- Columns have defined data types
- SERIAL vs. IDENTITY
- Belongs to a schema
- Owned by the creator, unless specified otherwise

Constraints

- NOT NULL
- UNIQUE
- Primary key
- Foreign key
- Check
- Exclusion

Primary Key

- Almost identical to a unique constraint
- No NULL values
- Defined by column, or by altering table

Foreign Key

- Created against primary key or unique constraint
- Enforces referential integrity
  - But NULL values can be equal
- ON DELETE RESTRICT
- CASCADE

Check Constraints

- Enforced on FALSE evaluation
  - Which means NULL comparisons count as TRUE
- Created with columns or ALTER TABLE

Exclusion Constraints

- Similar to UNIQUE
- Involves more complex comparisons
- A strict equality comparison would be the same as UNIQUE

Indexes

- PostgreSQL tables are heaps
- No concept of a clustered index
- CREATE and DROP CONCURRENTLY
- Must be table owner to add indexes

Index Types

- B-Tree
- Hash
- GiST
- SP-GiST
- GIN
- BRIN

Views

- Just a query
- Treated as a table in SQL
- Supports materialized views

DEMO!

redgate

Procedural Languages

Trusted vs. Untrusted Languages

- **Trusted** languages preserve security by preventing certain operations
  - DB internal processes
  - OS-level access
  - Non-privileged users can safely use

redgate

Trusted vs. Untrusted Languages

- **Untrusted** languages do not provide security safeguards
- Creator is responsible for security
- Only superusers can create functions in untrusted languages

redgate

Core Procedural PostgreSQL Languages

- SQL* (T)
- PL/pgSQL (T)
- PL/Python (U)
- PL/TcL (T/U)
- PL/Perl (T/U)

redgate

Lanouges Available by Extension

- PL/Java (T/U)
- pl/dotnet (U)
- PL/Rust (T/U)
- PL/R (U)
- Pl/v8js (T)

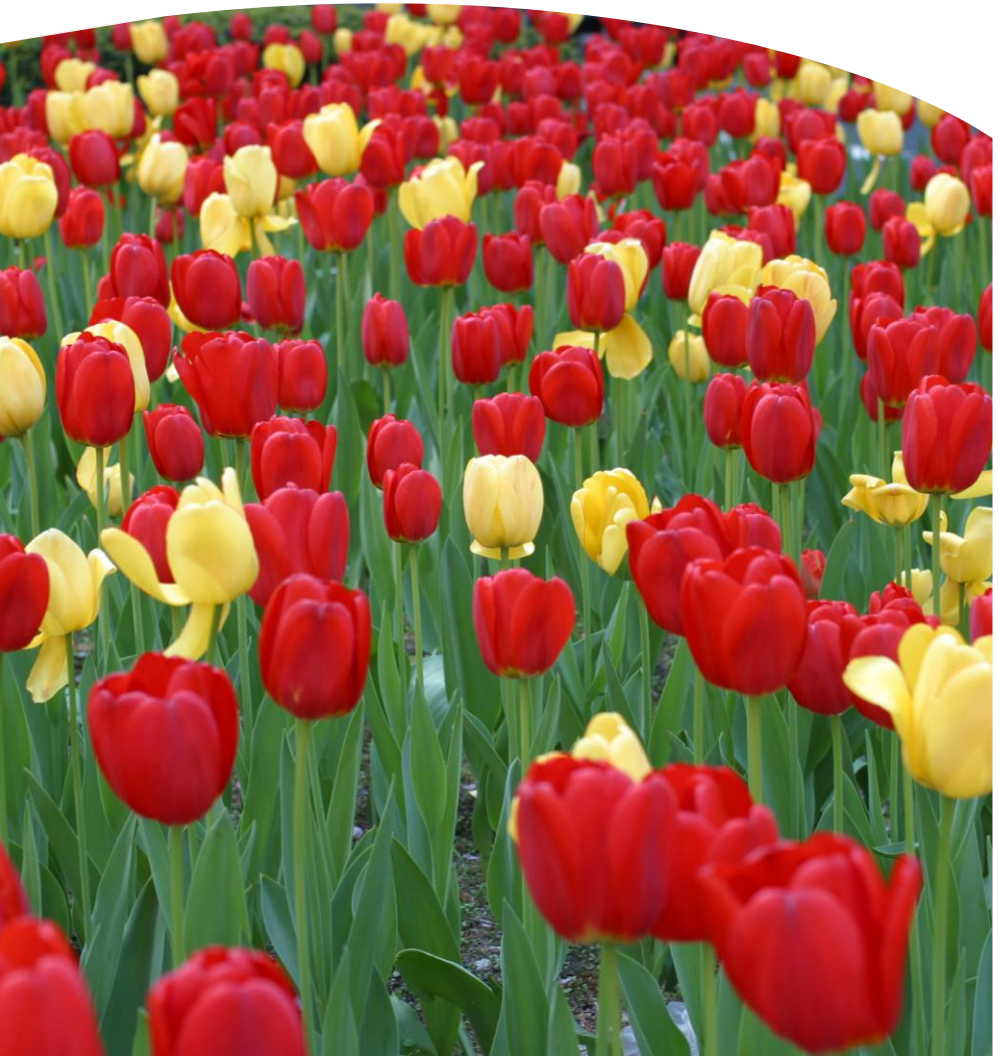
redgate

Core Functionality of Procedural Languages

- Blocks
- Variables
- Error handling
- If/Then
- Case
- Functions
- Procedures

redgate

Functions



- Implemented for returning data from PostgreSQL database
- Similar to Oracle functions or SQL Server procedures

redgate

Function Basics

- Use CREATE FUNCTION command
- Provide a name
- Define result set, value or record/tuple
- Create a code block
- Define language used

redgate

Function Behaviors

- Code blocks
  - Dollar quoting, $$
- Parameters
- Variables

redgate

Function Example

```
CREATE FUNCTION genre_list ()
RETURNS TABLE (genre_id int, genre_name TEXT)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        fg.genre_id,
        fg.name
    FROM
    public.film_genre AS fg;
END
$$ LANGUAGE plpgsql;
```

Using Functions

- Treated like a table
- SELECT
  - *
  - Column list
  - JOIN

redgate

Using Function Example

```
SELECT
   gl.genre_id,
   gl.genrename
FROM
genre_list() AS gl;
```

Function Query Planner Parameters

- **COST** (100): estimated execution cost of the function
- **ROWS** (1,000): estimated number of rows from the function

redgate

Function Query Planner Parameters

- **SET**: modify session configuration
  - search_path is often set
- **PARALLEL {UNSAFE|SAFE}**
  - Unsafe (default) forces serial execution plan
  - Safe allows the planner to consider parallel plans

redgate

Function Example

```
CREATE FUNCTION genre_list ()
RETURNS TABLE (genre_id int, genre_name TEXT)
COST 10000
ROWS 100000
AS $$
BEGIN
    RETURN QUERY
    SELECT
        fg.genre_id,
        fg.name
    FROM
    public.film_genre AS fg;
END
$$ LANGUAGE plpgsql;
```

redgate

Function Example

```
CREATE FUNCTION genre_list ()
RETURNS TABLE (genre_id int, genre_name TEXT)
SET search_path=bluebox,public
AS $$
BEGIN
    RETURN QUERY
    SELECT
        fg.genre_id,
        fg.name
    FROM
        film_genre AS fg;
END
$$ LANGUAGE plpgsql;
```

redgate

Procedures

- Procedures are meant to perform actions, as opposed to functions, which return data
- Similar to Oracle procedures, or SQL Server functions

redgate

Procedure Basics

- Use CREATE PROCEDURE command
- Provide a name
- Create a code block
- Define language used

redgate

Procedure Example

```
CREATE OR REPLACE PROCEDURE new_genre (genrename TEXT)
AS $$
BEGIN
    INSERT INTO bluebox.film_genre
       (name) VALUES (genrename);
END
$$
LANGUAGE plpgsql;
```

redgate

INOUT Procedure Example

```
CREATE OR REPLACE PROCEDURE
    genre_count(genreid int, INOUT genrecount int )
AS $$
BEGIN
    SELECT count(*)
    INTO genrecount
    FROM
       film AS f
    WHERE genreid = ANY(f.genre_ids);
END
$$
LANGUAGE plpgsql;
```

redgate

Using Procedures

- CALL
- INOUT parameters

redgate

Using Procedures Examples

```
CALL new_genre('Anime');

DO $$
DECLARE genre_counts int;
BEGIN
    CALL genre_count(genreid => 27, genrecount => genre_counts);
    RAISE NOTICE 'Films in genre: %', genre_counts;
END $$
```

redgate

Subtransactions

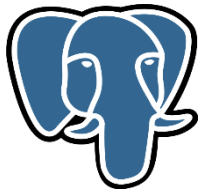
- Internal transactions are committed separately
- Helpful for processing large batches of data over time
  - Checking progress

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

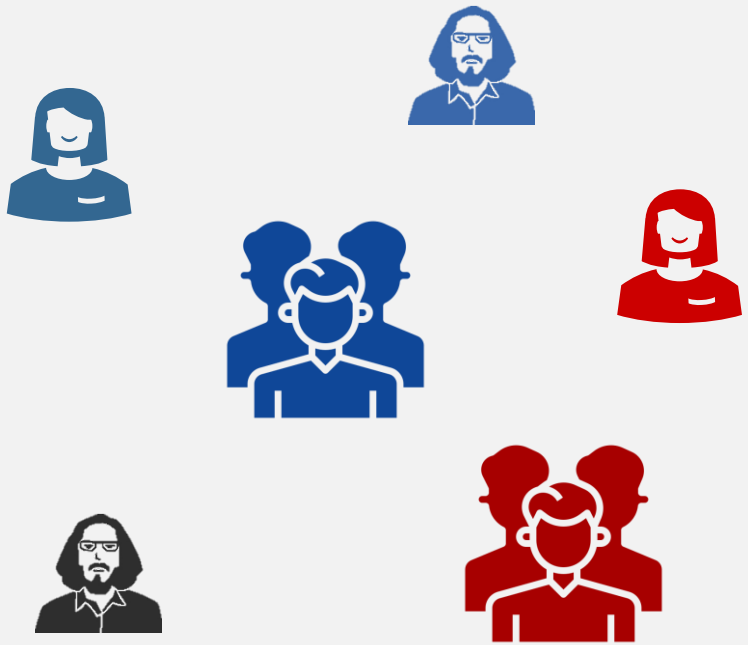
Cluster

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?

redgate

```
# 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

DEMO!

redgate

PASS Session

For Your Eyes Only: Roles, Privileges, and Security in PostgreSQL

Wednesday, Nov. 6

9:45-10:45

Room 445-446

https://www.youtube.com/watch?v=3bCWLe-dU58

redgate

MVCC and Vacuum

'MVCC Unmasked' by Bruce Momjian

Portions of this content were sourced from Bruce's seminal talk, "**MVCC Unmasked**", under the Creative Commons Attribution License.

https://momjian.us/main/writings/pgsql/mvcc.pdf

PostgreSQL documentation uses the word **tuple** for **row**.

In technical terms, a tuple is an instance of the row schema.

Multi-Version Concurrency Control (MVCC) is what allows PostgreSQL (and other databases) to support highly concurrent workloads with the goal that:

"readers never block writers,

and writers never block readers"

Multi-Version Concurrency Control

For the default **Read Committed Isolation Level**:

- Each query only sees transactions that were committed before it started (ID less than current transaction)
- At the beginning of a query, PostgreSQL records:
  - The transaction ID (xid) of the current command
  - All transaction IDs that are in process
- Multi-statement transactions also record any of their own completed statements from earlier in transaction

Assigning XIDs

In PostgreSQL, transaction IDs are assigned when the first command/statement is run within a transaction, **not** when **BEGIN** is executed.

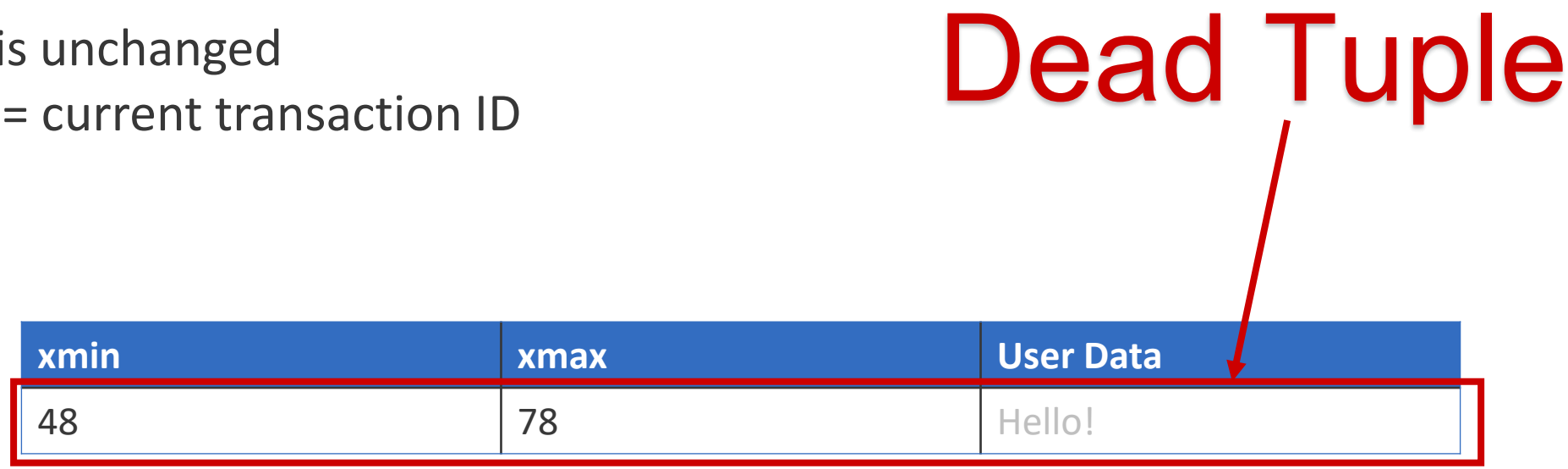
INSERT

- xmin = current transaction ID
- xmax = zero (invalid)

xmin	xmax	User Data
48	0	Hello!

DELETE

- xmin is unchanged
- xmax = current transaction ID



UPDATE

Original tuple:

- xmin is unchanged
- xmax = current transaction ID

New tuple:

- xmin = current transaction ID
- xmax = zero

Dead Tuple

xmin	xmax	User Data
48	78	Hello!
78	0	Hello World!

UPDATE (with the same value)

Original tuple:

- xmin is unchanged
- xmax = current transaction ID

New tuple:

- xmin = current transaction ID
- xmax = zero

Dead Tuple

xmin	xmax	User Data
48	78	Hello!
78	0	Hello!

Visible Tuples

A tuple is considered "visible" when creation XID (**xmin**):

- is a committed transaction
- is less than the XID stored at the start of the current query
- was not in-process at the start of the current query

Visible tuples must *also* have an expired XID (**xmax**) that:

- is blank *or* aborted *or*
- is greater than the transaction counter stored at query start *or*
- was in-process at query start

Tuple Visibility and Snapshots

Created rows only

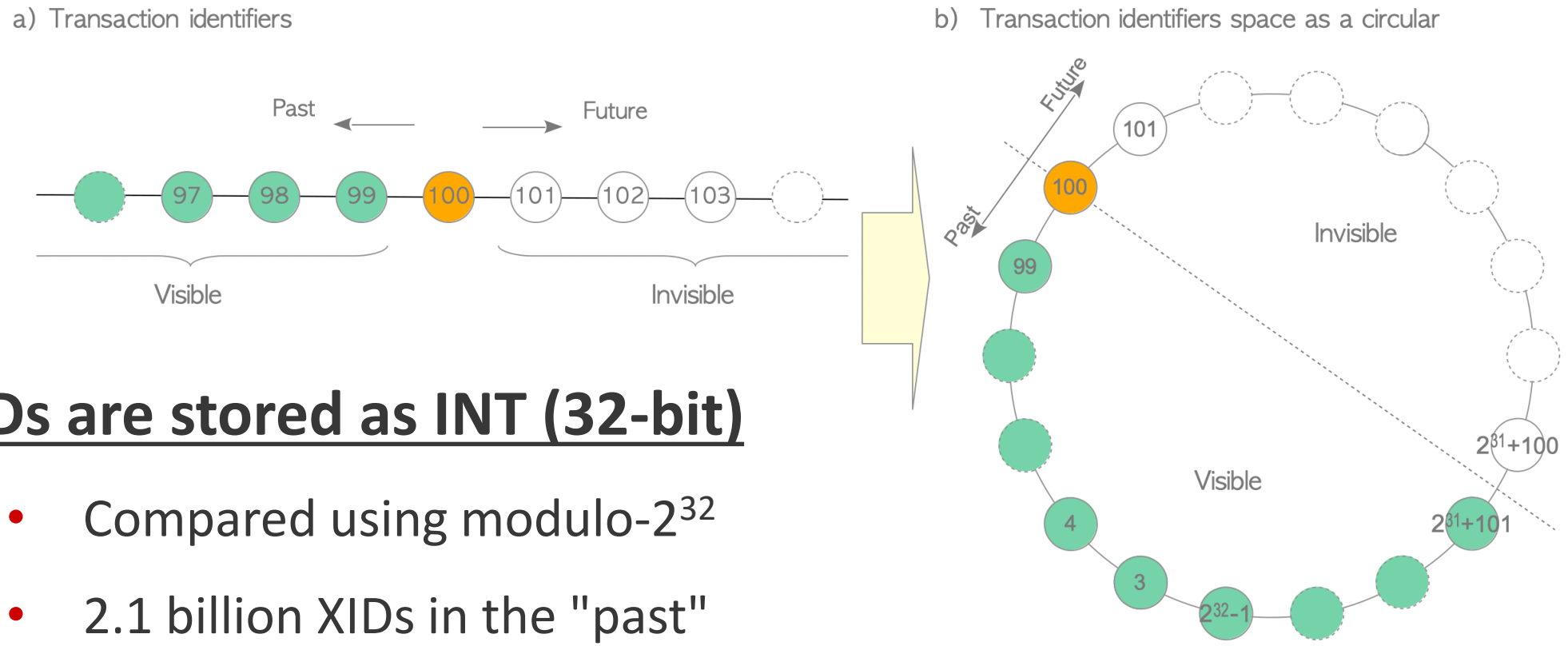
xmin	xmax	User Data	
10		Hello!	Visible
50		Velkommen	Not Visible
100		willkommen	Not Visible

Updated rows

xmin	xmax	User Data	
10	70	Hello!	Not Visible
40	50	Velkommen	Visible
48	100	willkommen	Visible

Snapshot

- Highest committed XID = 90
- Open transactions: 20,50,100
- Current XID = 110



XIDs are stored as INT (32-bit)

- Compared using modulo-2^{32}
- 2.1 billion XIDs in the "past"
- 2.1 billion XIDs in the "future"

Image Credit: https://www.interdb.jp/pg/pgsql05.html

redgate

PostgreSQL Heap Page Layout

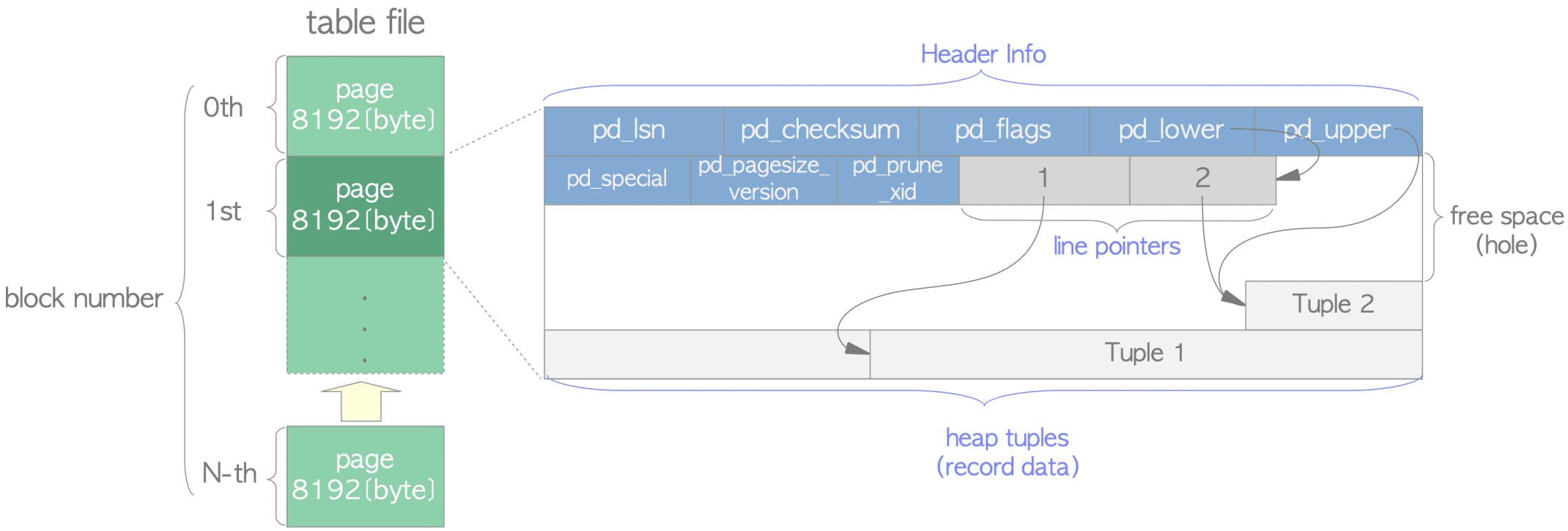


Image credit: https://www.interdb.jp/pg/pgsql01.html

redgate

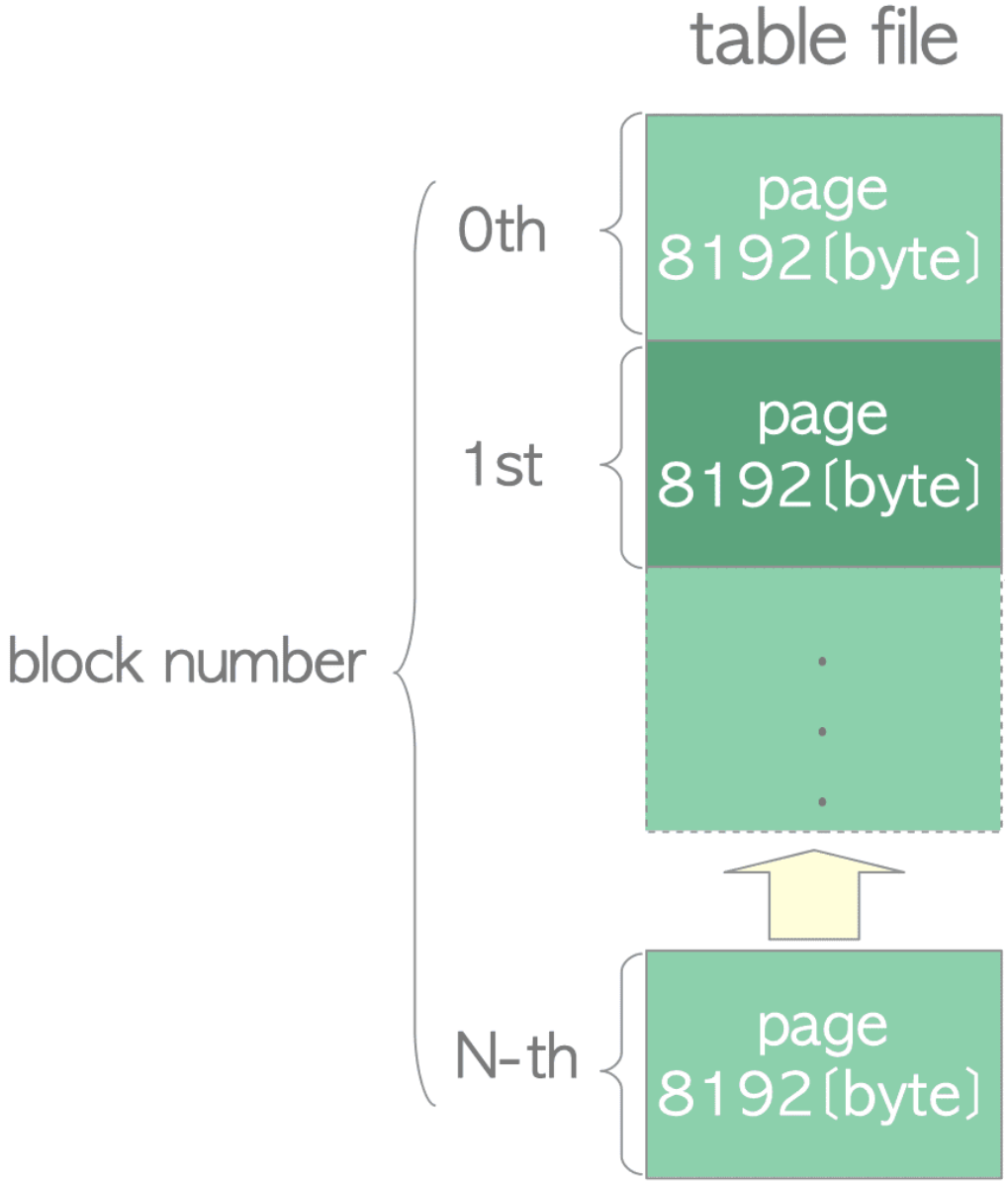
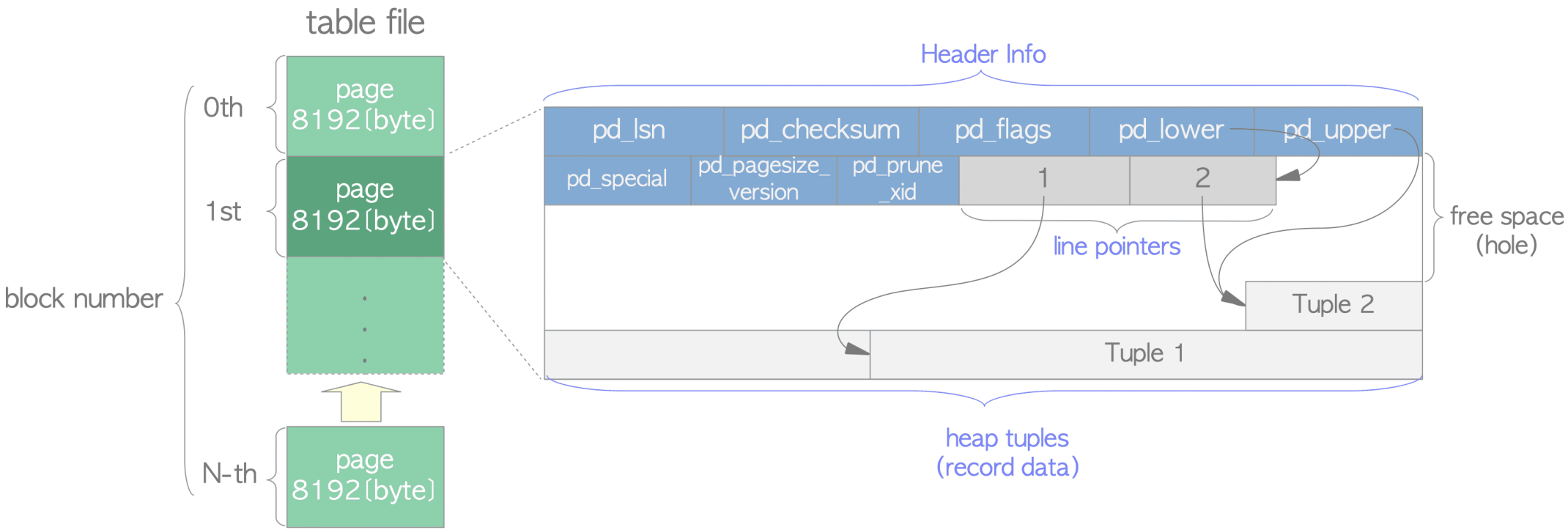
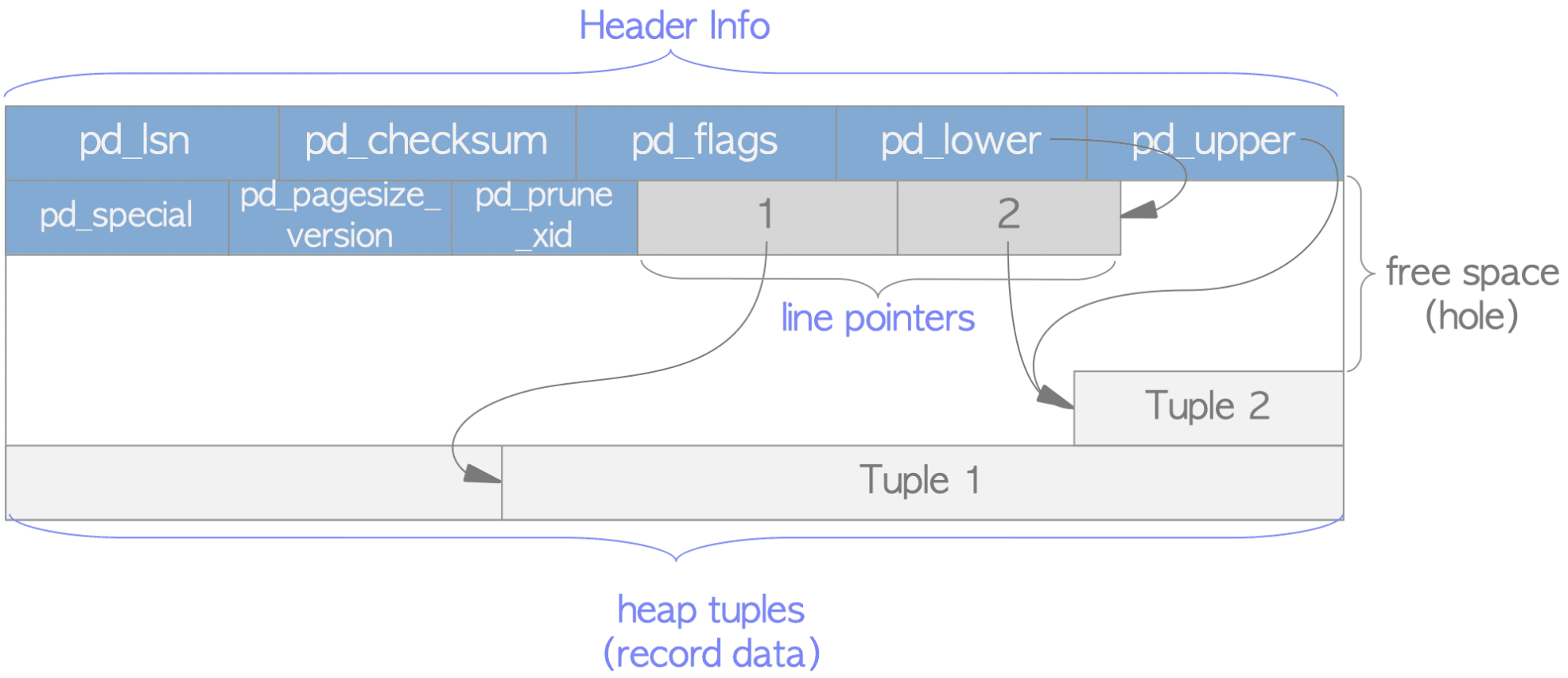


table file
block number
0th
1st
N-th
page
8192[byte]
page
8192[byte]
page
8192[byte]

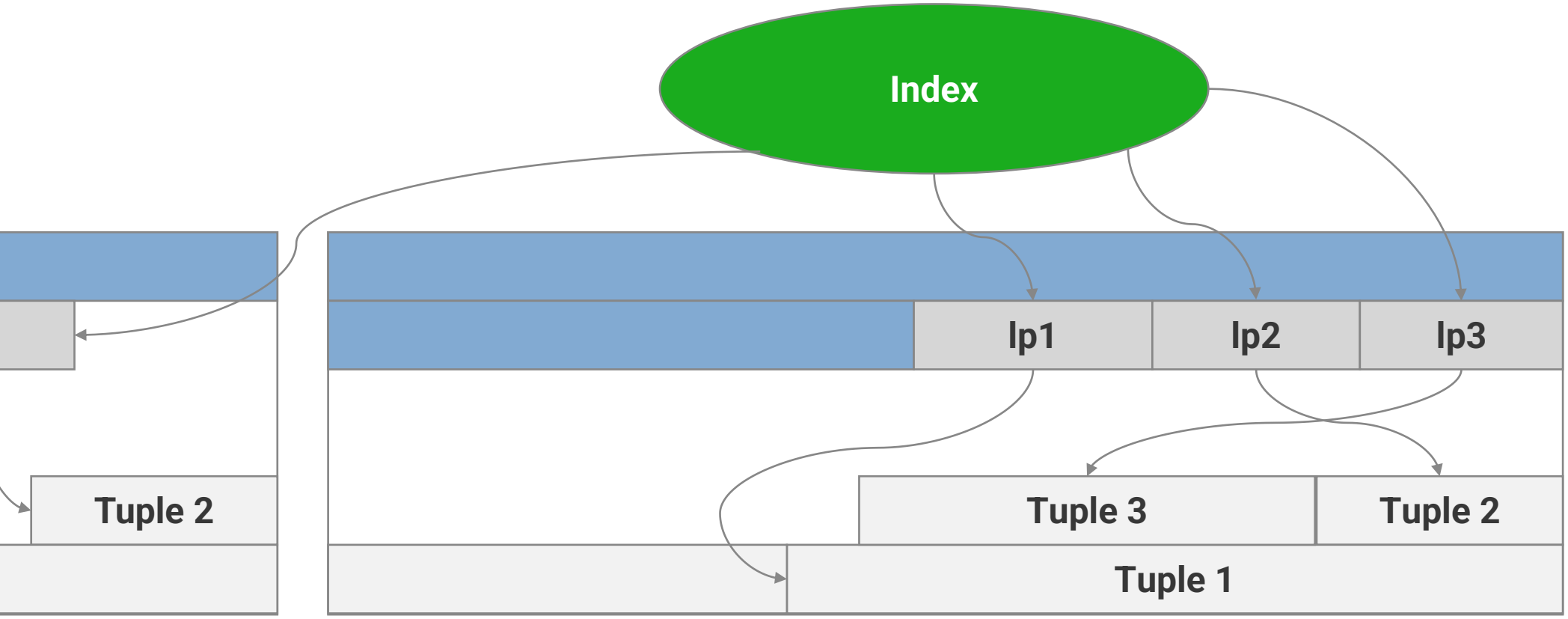
PostgreSQL Heap Page Layout



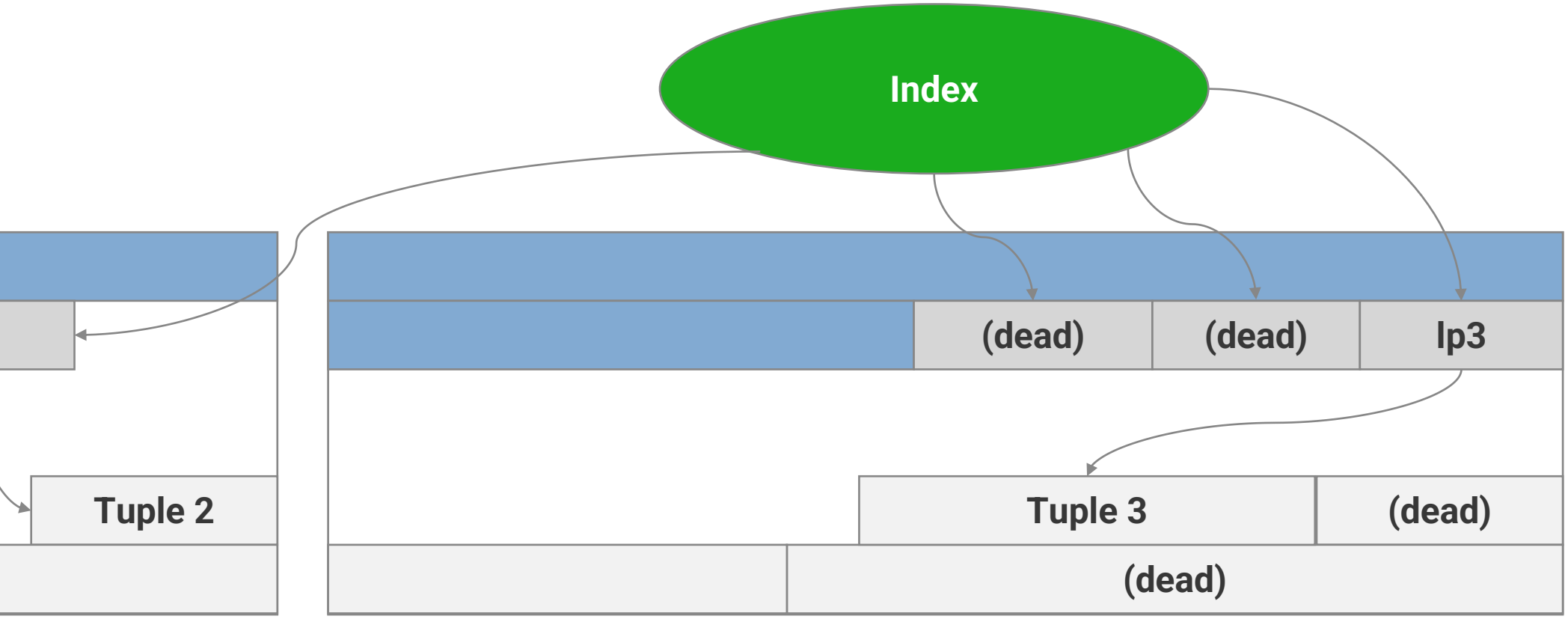
redgate

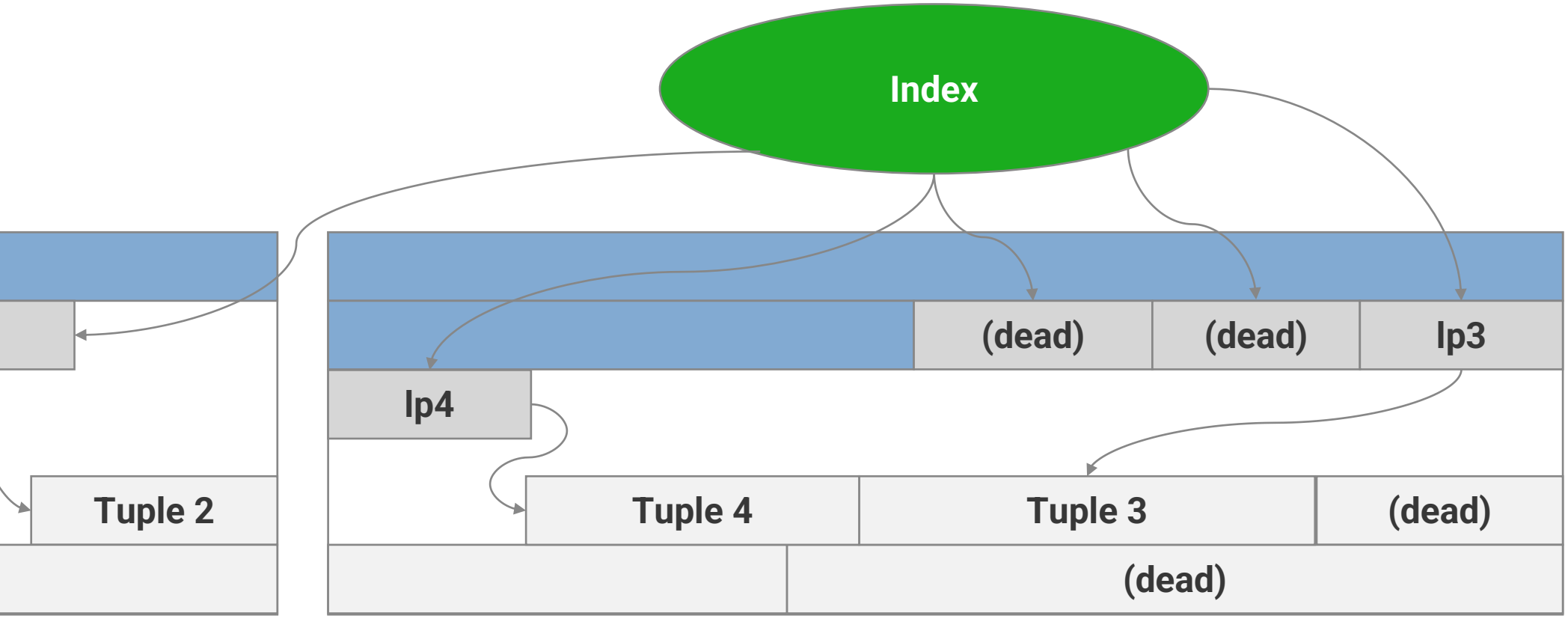

Header Info
pd_lsn
pd_checksum
pd_flags
pd_lower
pd_upper
pd_special
pd_pagesize_
version
pd_prune
_xid
1
2
line pointers
free space
(hole)
Tuple 2
Tuple 1
heap tuples
(record data)

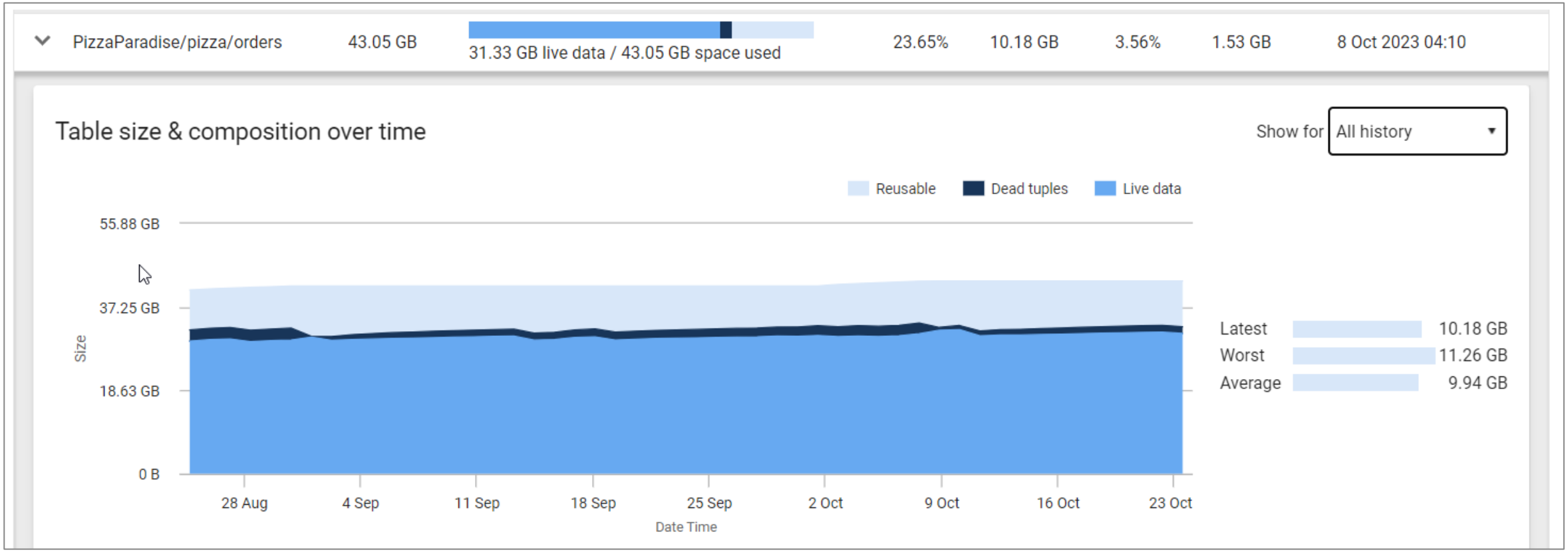
redgate



redgate


Index
(dead)
(dead)
lp3
Tuple 2
Tuple 3
(dead)
(dead)


Index
(dead)
(dead)
lp3
lp4
Tuple 2
Tuple 4
Tuple 3
(dead)
(dead)


PizzaParadise/pizza/orders
43.05 GB
31.33 GB live data / 43.05 GB space used
23.65%
10.18 GB
3.56%
1.53 GB
8 Oct 2023 04:10
Table size & composition over time
Show for
All history
Reusable
Dead tuples
Live data
55.88 GB
37.25 GB
18.63 GB
0 B
Size
28 Aug
4 Sep
11 Sep
18 Sep
25 Sep
2 Oct
9 Oct
16 Oct
23 Oct
Date Time
Latest
10.18 GB
Worst
11.26 GB
Average
9.94 GB

redgate

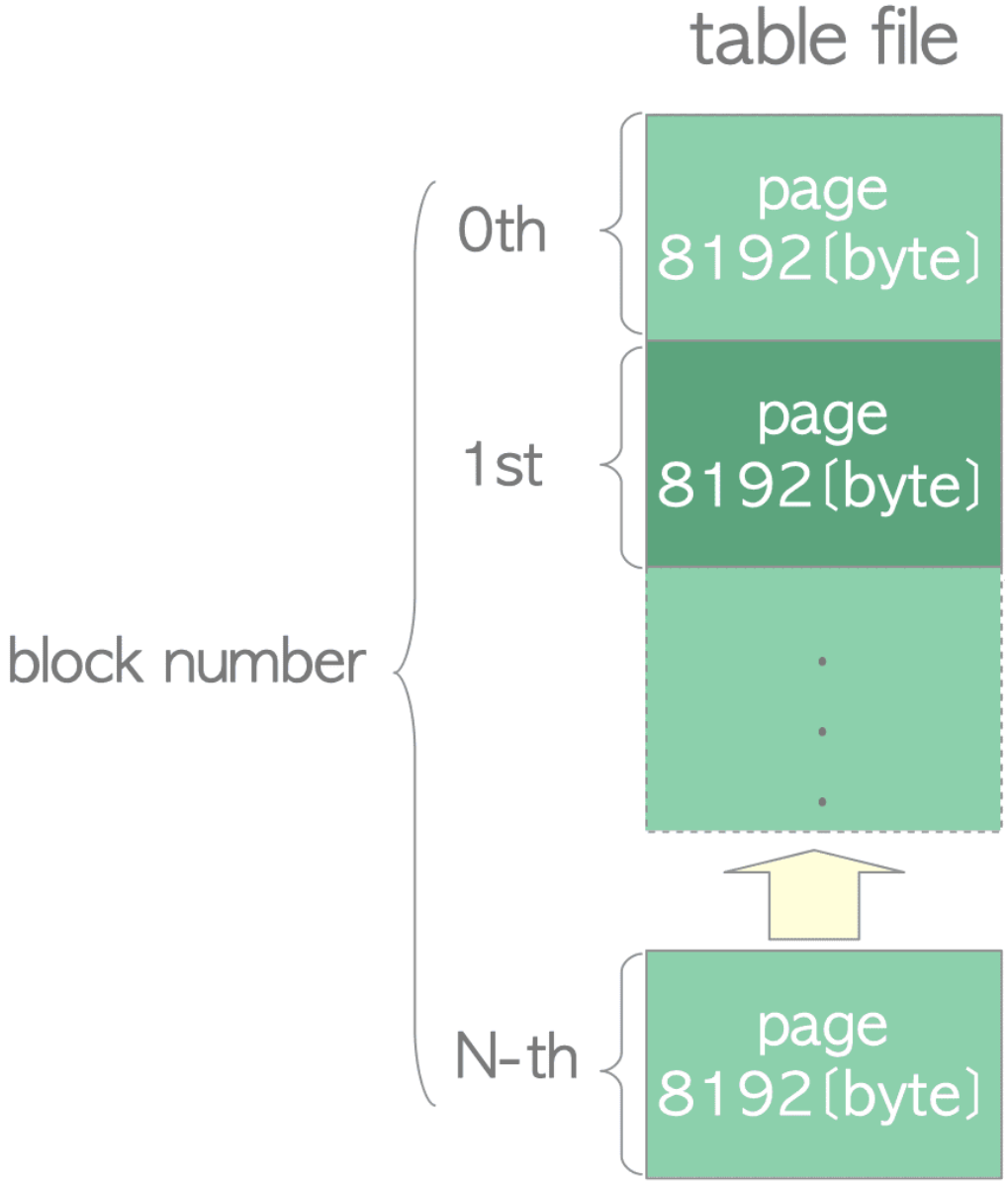
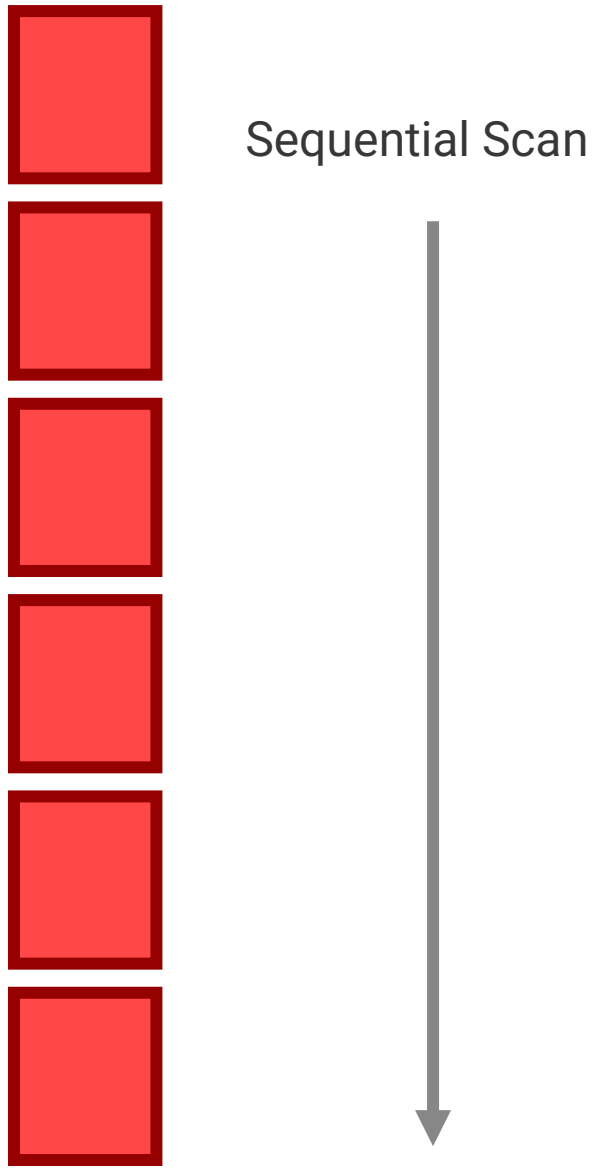
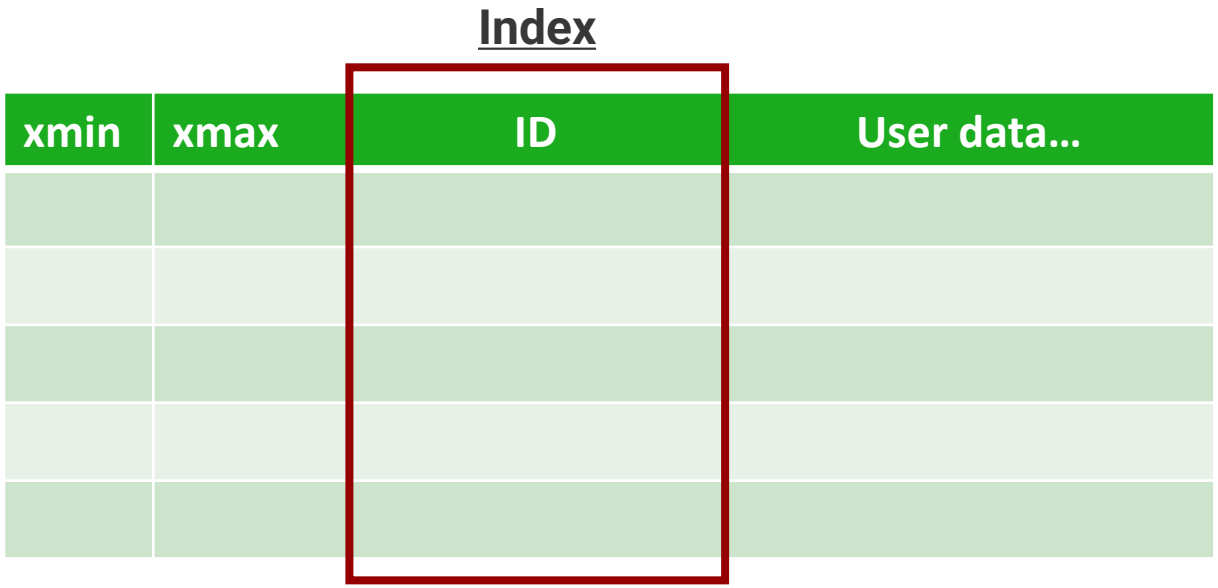


table file
block number
0th
1st
N-th
page
8192[byte]
page
8192[byte]
page
8192[byte]

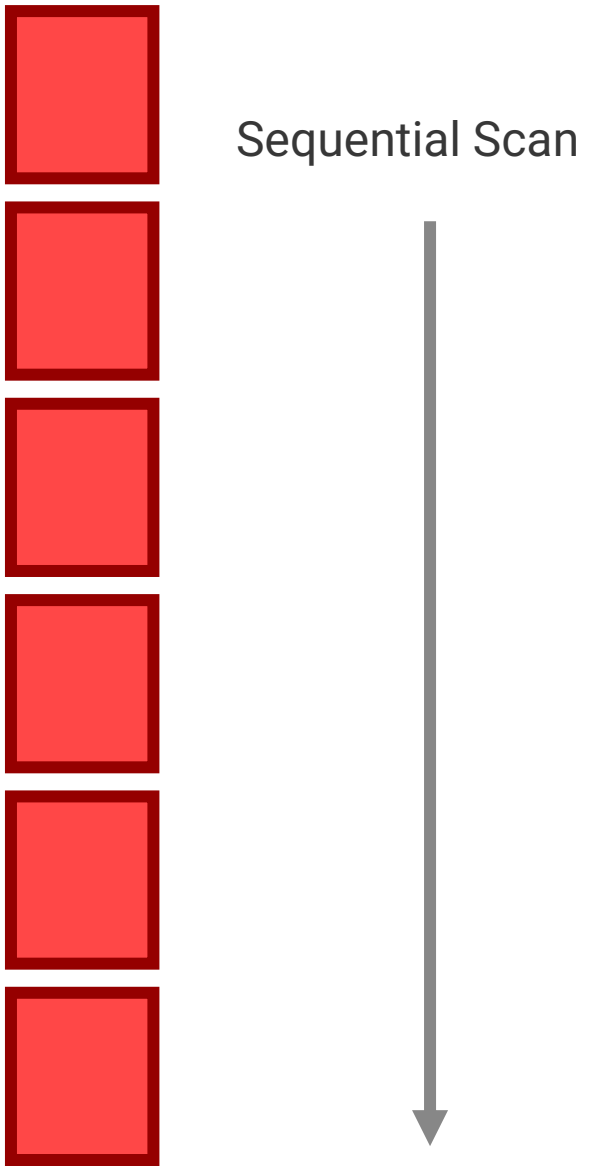
xmin	xmax	ID	User data…

SELECT a, b, c FROM t;

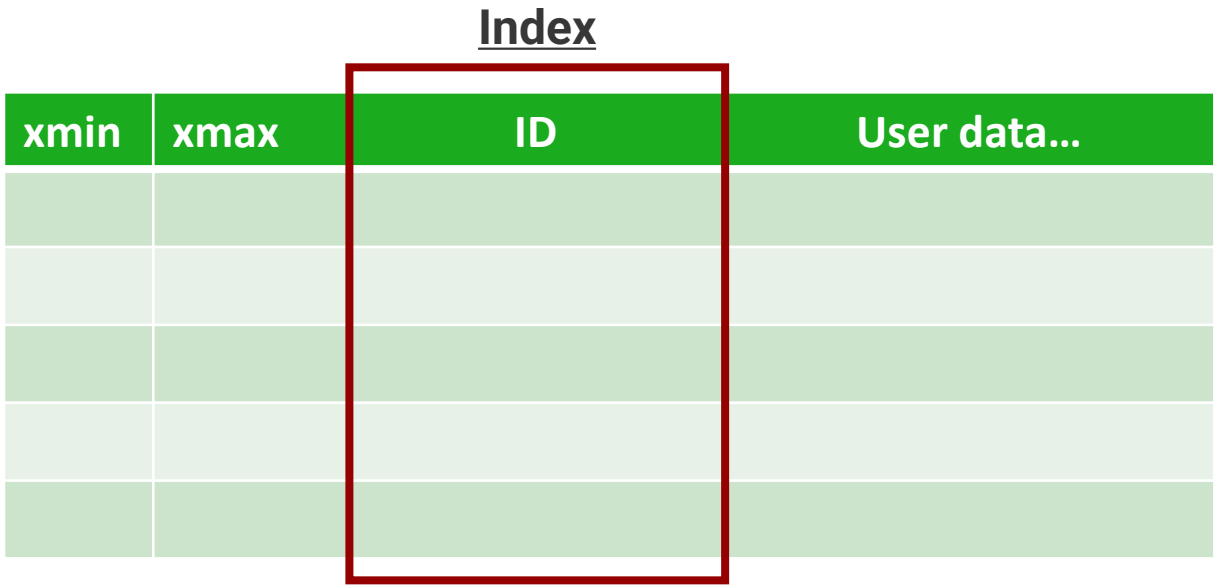




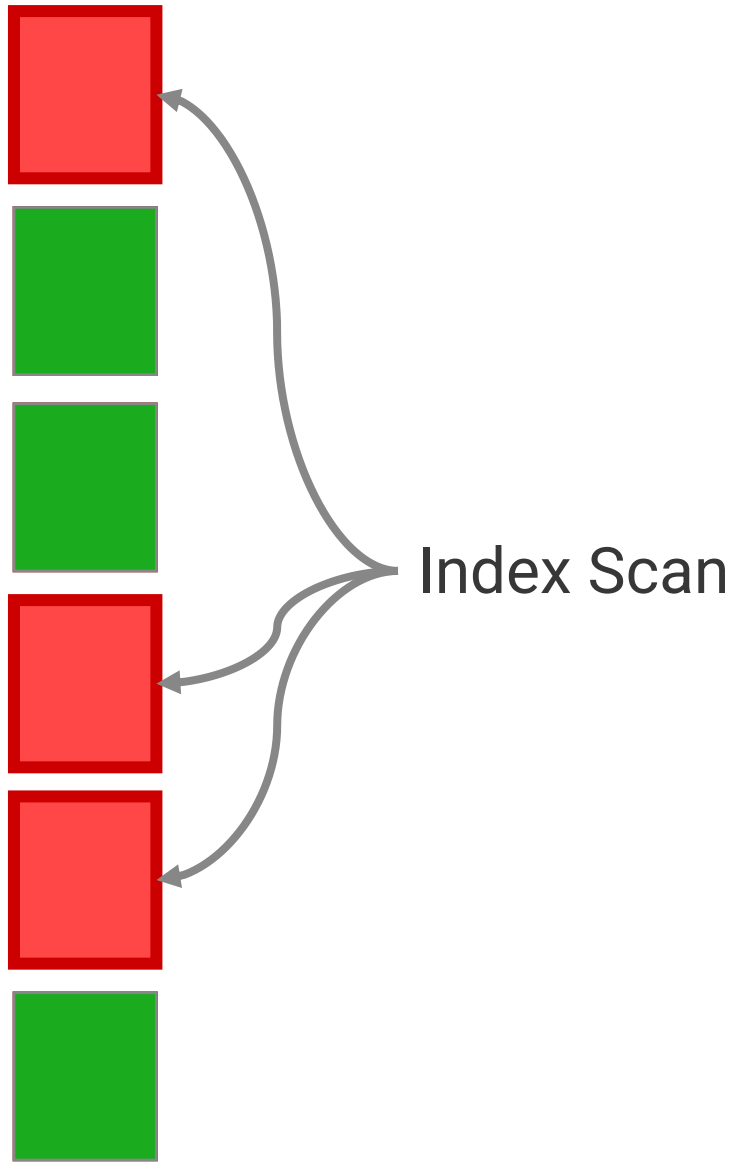
SELECT a, b, c FROM t
WHERE ID=10;



redgate




```
SELECT a, b, c FROM t
WHERE ID=10;
```



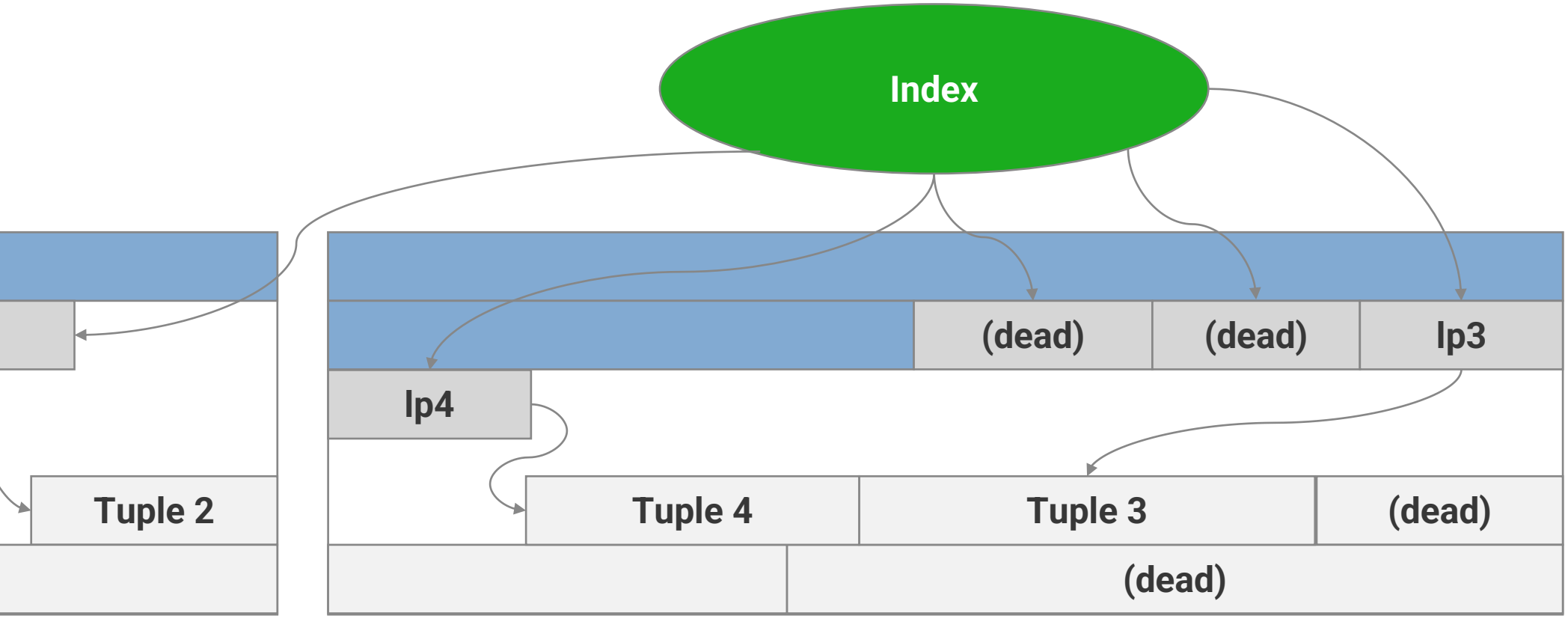
redgate


iRobot

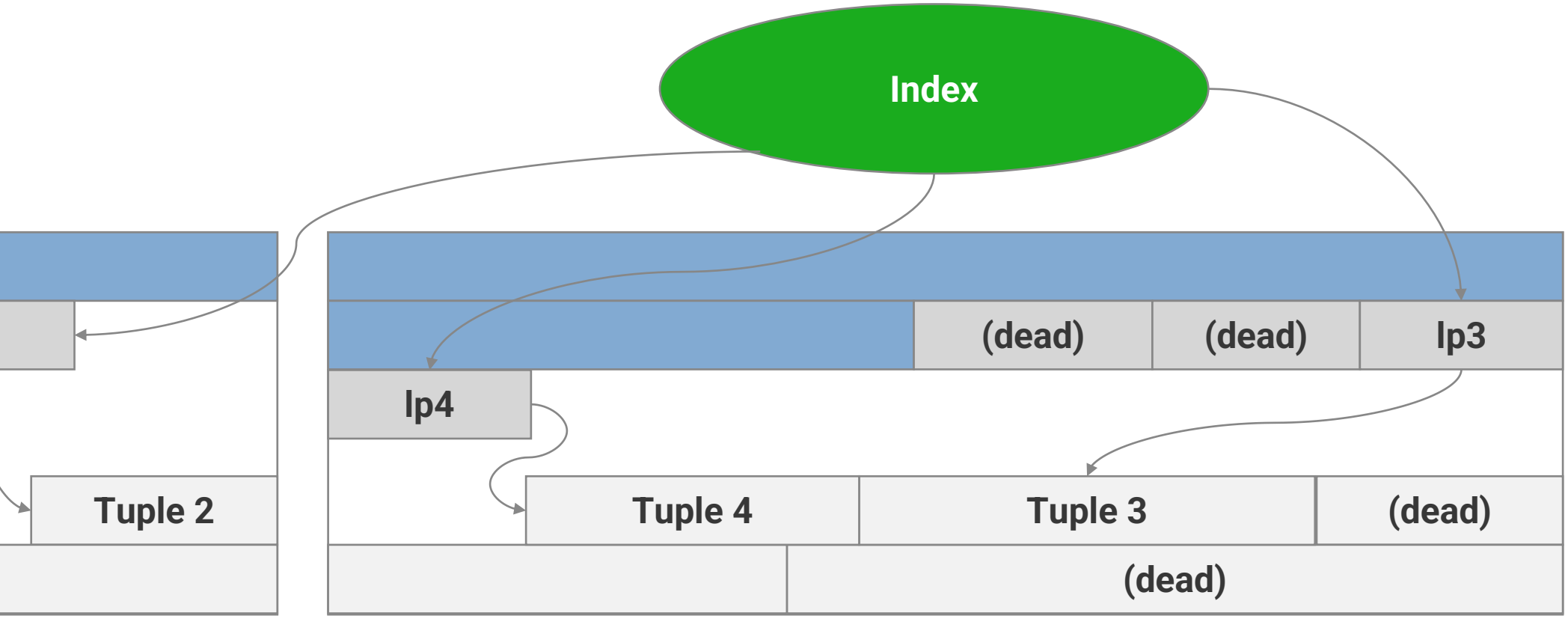
Vacuum!

- Serves two main purposes:
  - Free up space on pages used by dead tuples
  - Mark rows as "frozen" if they have a XID sufficiently older than the most recent XID, allowing the XID to be reused again in the future
  - Additionally, if all rows on a page are frozen, the page can be marked frozen to speed up scans

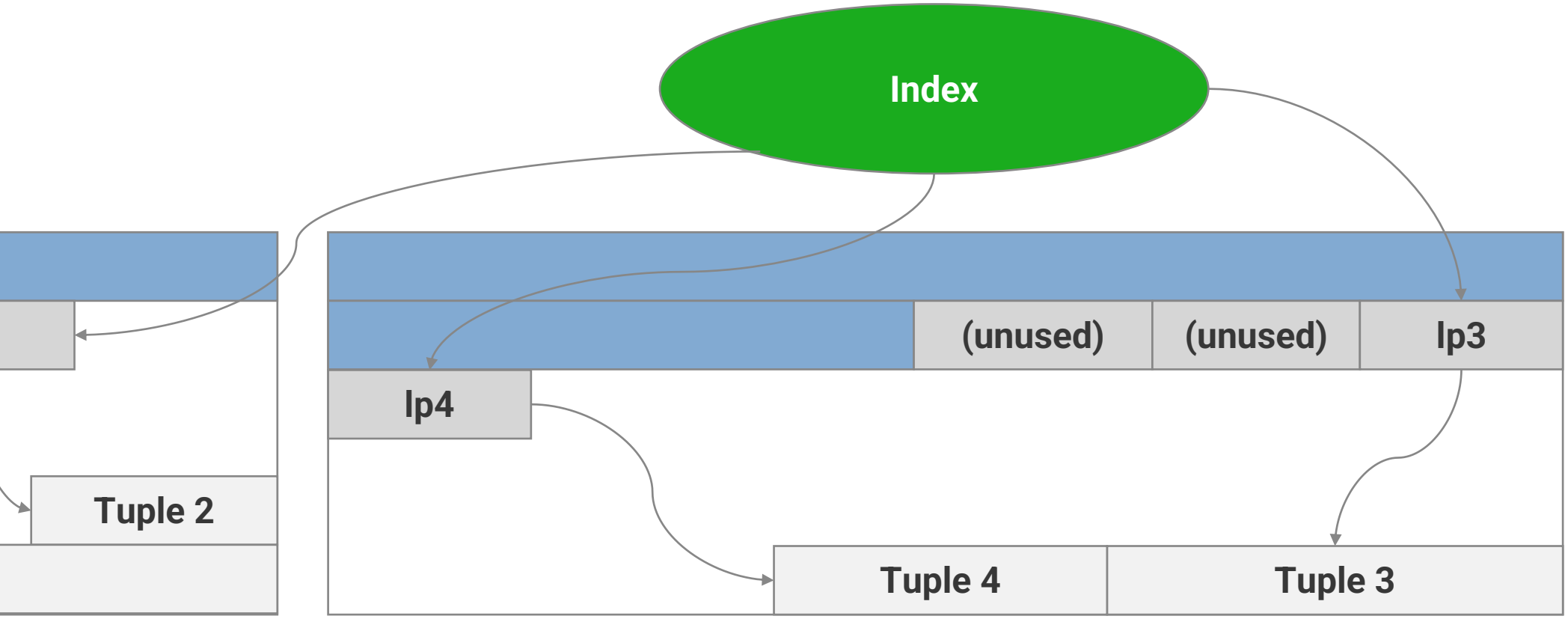
Free space used by dead tuples



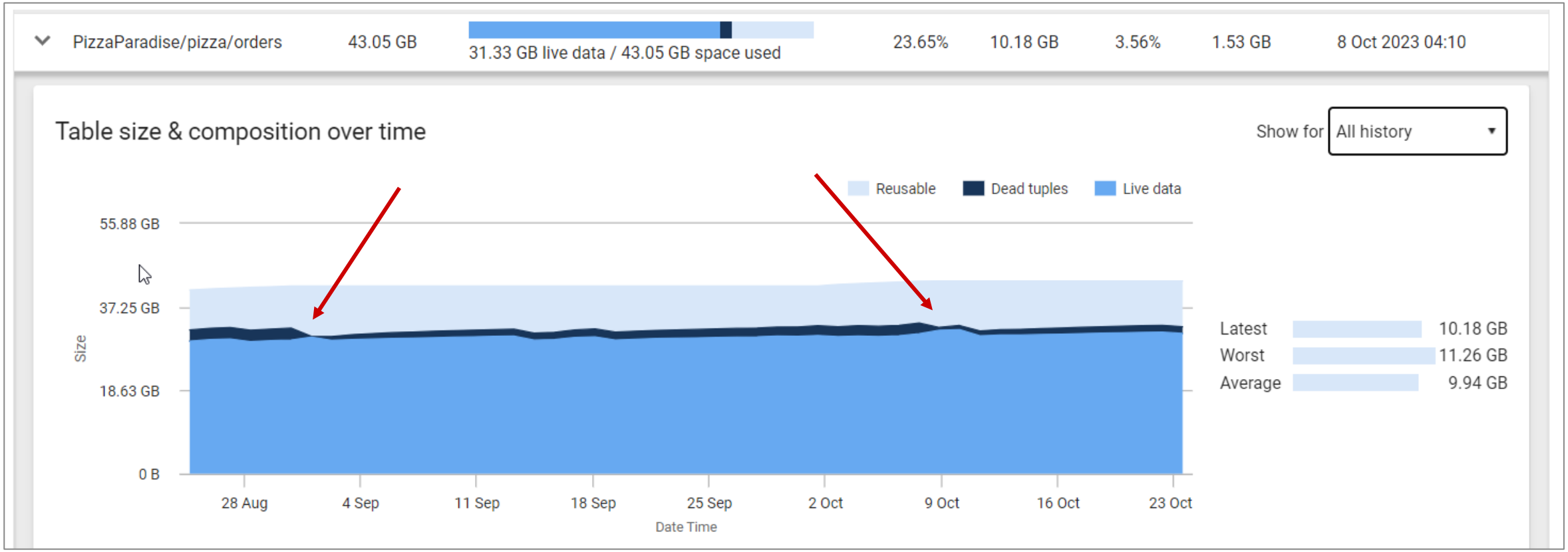
redgate



redgate



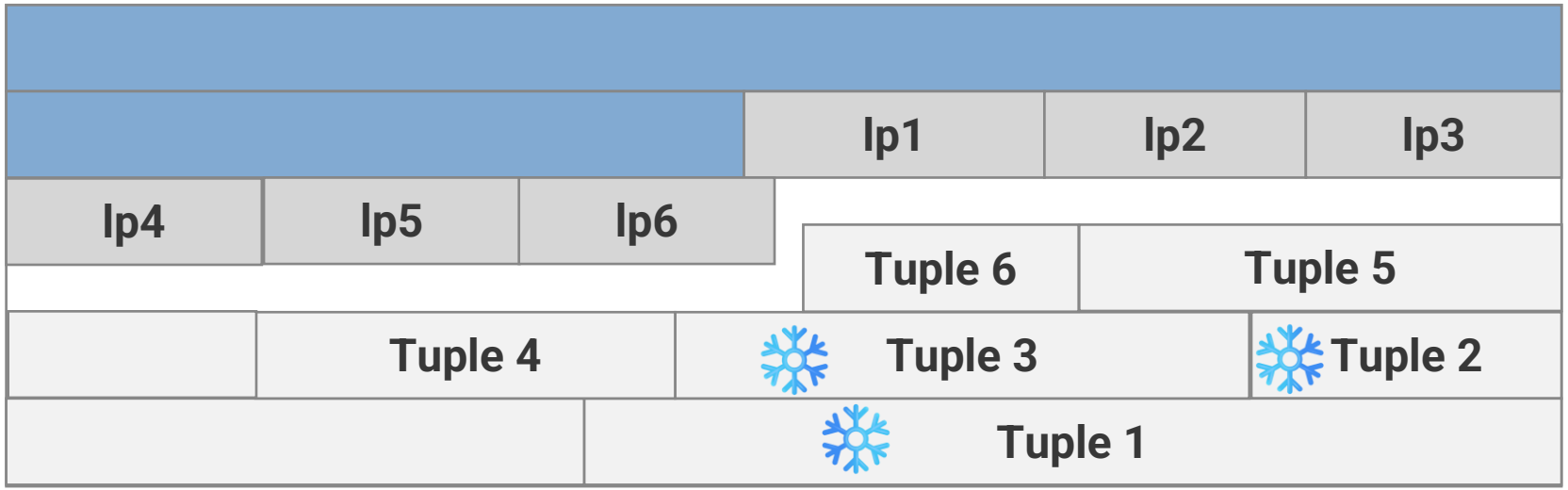
redgate


PizzaParadise/pizza/orders
43.05 GB
31.33 GB live data / 43.05 GB space used
23.65%
10.18 GB
3.56%
1.53 GB
8 Oct 2023 04:10
Table size & composition over time
Show for
All history
Reusable
Dead tuples
Live data
55.88 GB
37.25 GB
18.63 GB
0 B
Size
28 Aug
4 Sep
11 Sep
18 Sep
25 Sep
2 Oct
9 Oct
16 Oct
23 Oct
Date Time
Latest
10.18 GB
Worst
11.26 GB
Average
9.94 GB

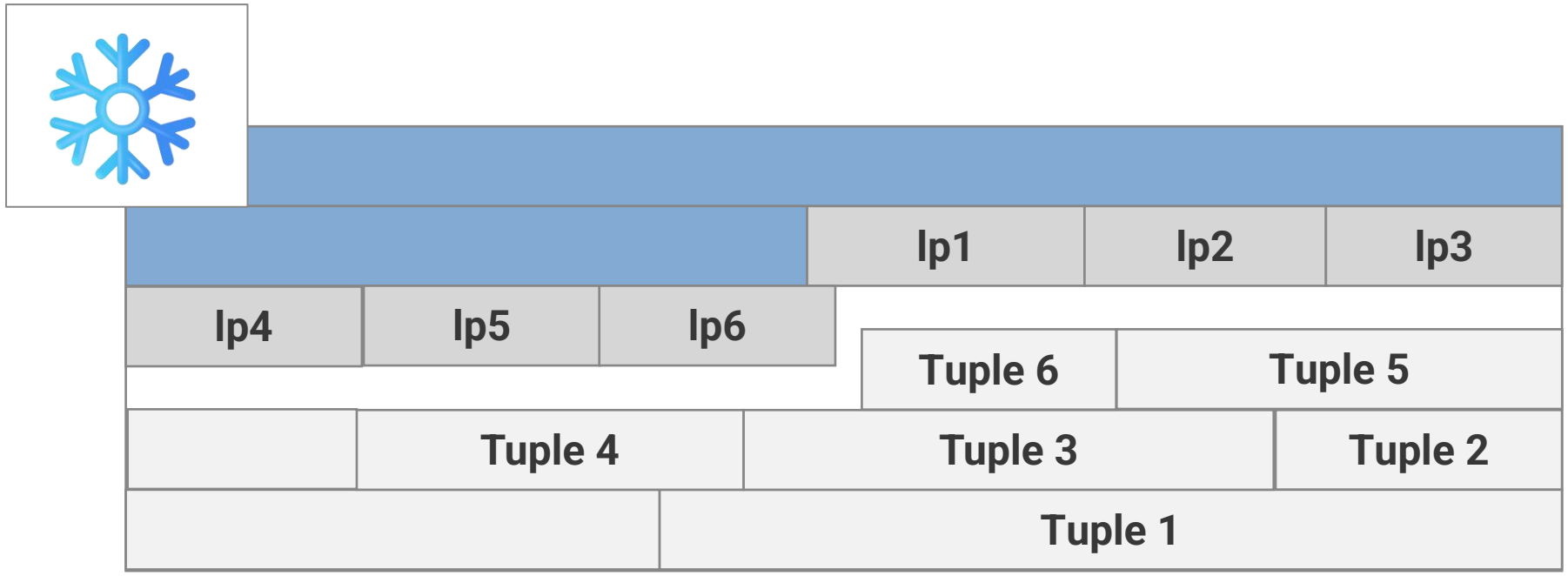
redgate

Freeze rows and pages

redgate


lp1
lp2
lp3
lp4
lp5
lp6
Tuple 6
Tuple 5
Tuple 4
Tuple 3
Tuple 2
Tuple 1

redgate


lp1
lp2
lp3
lp4
lp5
lp6
Tuple 6
Tuple 5
Tuple 4
Tuple 3
Tuple 2
Tuple 1

redgate

VACUUM Automation

- Vacuuming is so vital that there is an automated process called **autovacuum**
- Triggered based on a percentage of dead tuples/modified data (generally)
- Individual DELETE or UPDATE heavy tables can be tuned directly

Individual Table Auto(vacuum/analyze)

- Autovacuum triggers at 20% data modification
- Autoanalyze triggers at 10% data modification
- Analogous to TF-2731, but per table!

Global Autovacuum and Autoanalyze Settings

"Scale Factor" = Percentage of rows modified before process runs

"Threshold" = Added number of rows to add to the percent calculation.

(this prevents very small tables from taking resources)

```
ǎụʧộŵǎçụụn ŵǎçụụn şçǎľê ğǎçʧộs   . , ,.
ǎụʧộŵǎçụụn ŵǎçụụn ʧḥsêşḥộľđ   _.
ǎụʧộŵǎçụụn ǎŋǎľỳćê şçǎľê ğǎçʧộs   . , ,.
ǎụʧộŵǎçụụn ǎŋǎľỳćê ʧḥsêşḥộľđ   _.
```

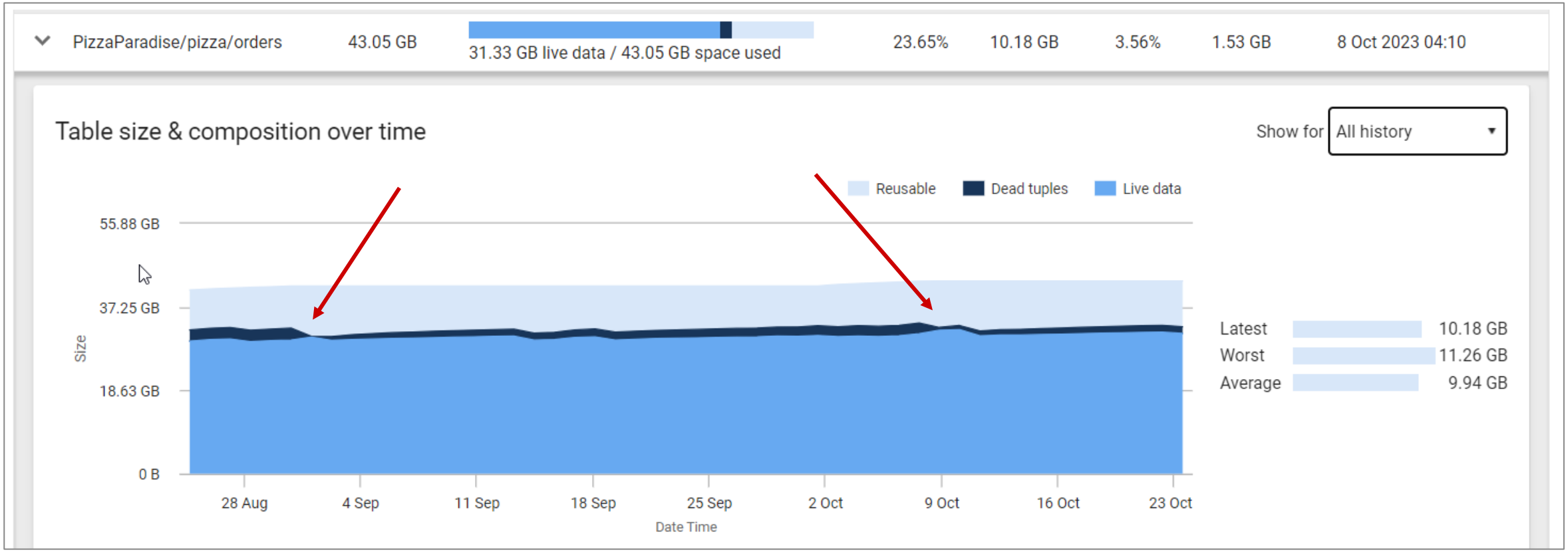
redgate

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)
```


PizzaParadise/pizza/orders
43.05 GB
31.33 GB live data / 43.05 GB space used
23.65%
10.18 GB
3.56%
1.53 GB
8 Oct 2023 04:10
Table size & composition over time
Show for
All history
Reusable
Dead tuples
Live data
55.88 GB
37.25 GB
18.63 GB
0 B
Size
28 Aug
4 Sep
11 Sep
18 Sep
25 Sep
2 Oct
9 Oct
16 Oct
23 Oct
Date Time
Latest
10.18 GB
Worst
11.26 GB
Average
9.94 GB

redgate

'If you think autovacuum/analyze is running too frequently, it's probably not running frequently enough'

PASS Session

Importance of PostgreSQL Vacuum Tuning to Optimize Database Performance

📅 Wednesday, Nov. 6
🕑 2:00-3:00
🏢 Room 447

redgate

Standard Configuration and Maintenance

redgate

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

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

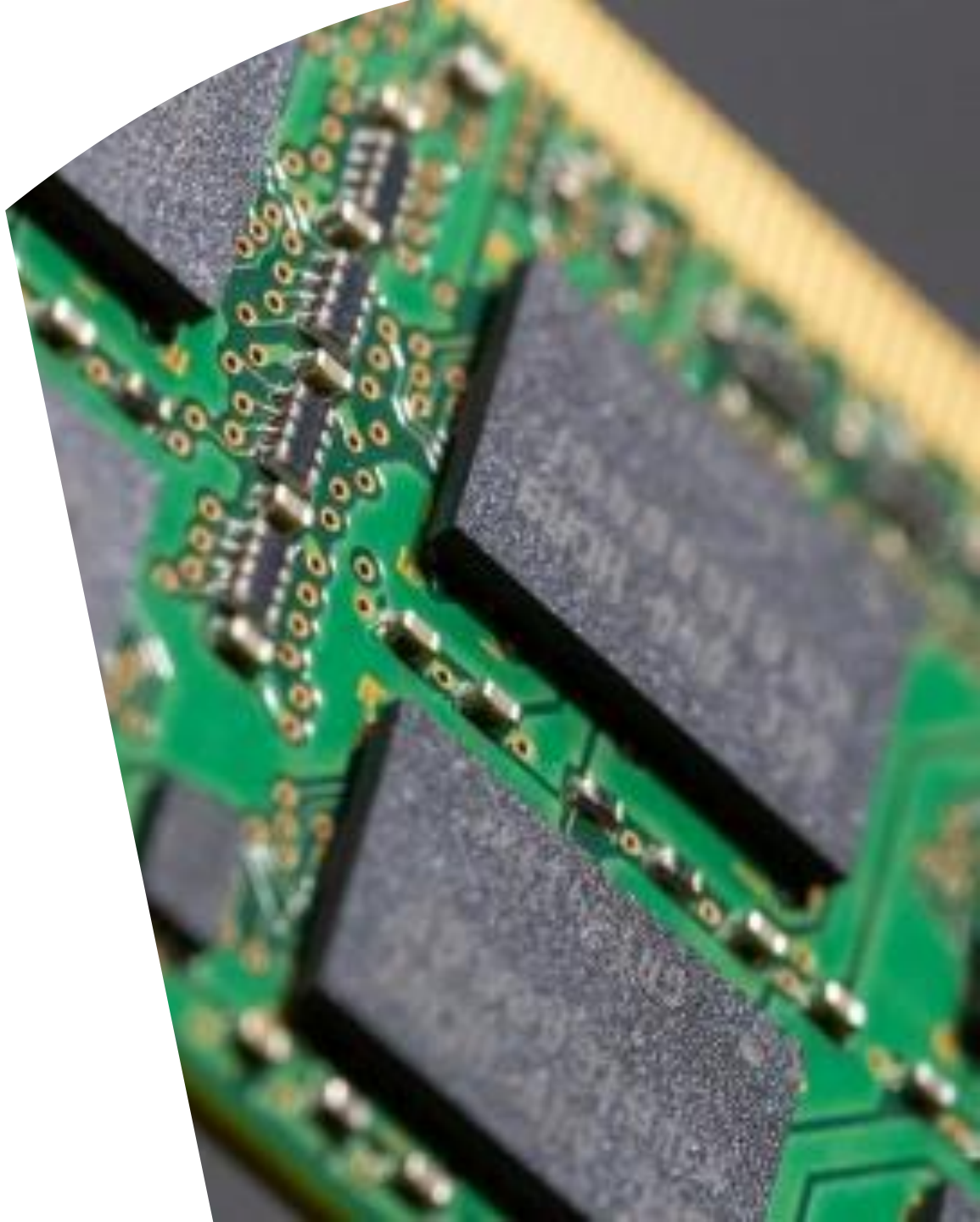
Cache Hit Ratio

```
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 1.0 in most environments and monitor

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

Other Maintenance Tasks

Reindexing

- Indexes can have bloat, too
- Reindexing consolidates and cleans index pages
- Requires access exclusive lock, unless CONCURRENTLY is used

redgate

Reindexing

- Maintains FILLFACTOR when reindexing
- Essential upkeep for some specific index types like BRIN

redgate

FILLFACTOR

- The target amount of free space left on each data page
- Defaults:
  - Tables = 100%
  - Indexes = 90%
- Very important consideration for update-heavy tables

FILLFACTOR

For indexes

- Higher value when indexed columns will not change (i.e. timestamps or counters)
- Lower value when data inserts will likely produce many page splits

For tables

- Higher value for insert-only workloads
- Lower value for update heavy tables
  - Remember, every update produces a new row version

Heap-only Tuples (HOT)

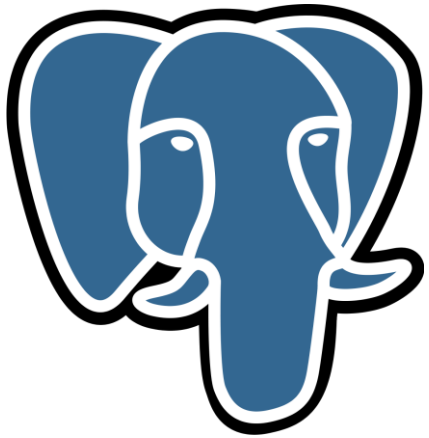
- HOT updates attempt to store new row version on the same data page
- Index pointer does not have to change
- Updated value cannot be referenced by an index
- Increase likelihood by decreasing FILLFACTOR
- Benefits update-heavy tables

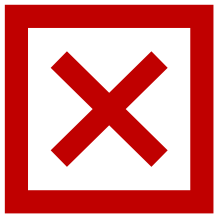
```
SELECT schemaname, relname,
       n_tup_upd, n_tup_hot_upd
FROM pg_stat_user_tables;
```

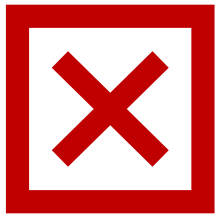
	schemaname	relname	n_tup_upd	n_tup_hot_upd
1	bluebox	rental	4,489	0
2	bluebox	inventory	4,461	442

Extensions

Customizable with Extensions







redgate

What are extensions?

- Non-core code that can be installed into a database
- Add new functions or features
- Modify PostgreSQL runtime based on **hooks** within the code
  - Query planner
  - Data retrieval
  - Storage engine
  - More…

redgate

What are extensions?

- Written in any supported language, including SQL
- Many popular extensions are written in C to gain the best performance
- The **pgrx** framework is a popular Rust-based toolchain for building extensions

Problems that extensions can solve

- Having a repeatable set of functions available in a database for DBAs within your company
- Providing access to server data that hasn't been exposed yet
- Modifying data storage
- Influencing the query planner

If you wish PostgreSQL could do something, it's probably possible with an extension... within reason

TimescaleDB

Citus

PostGIS

redgate

What are extensions?

- Extensions must be installed **per-database**
  - There are no "server-level" extensions
- PostgreSQL ships with 'contrib' extensions
- Others must be installed on the server
  - Manually (source, RPM/YUM packages)
  - PostgreSQL Extension Network (PGXN)
- Trusted vs. Untrusted

Extensions

See available extensions

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

Install

```
CREATE EXTENSION pg_stat_statements;
```

Update

```
ALTER EXTENSION pg_stat_statements UPDATE;
```

Uninstall

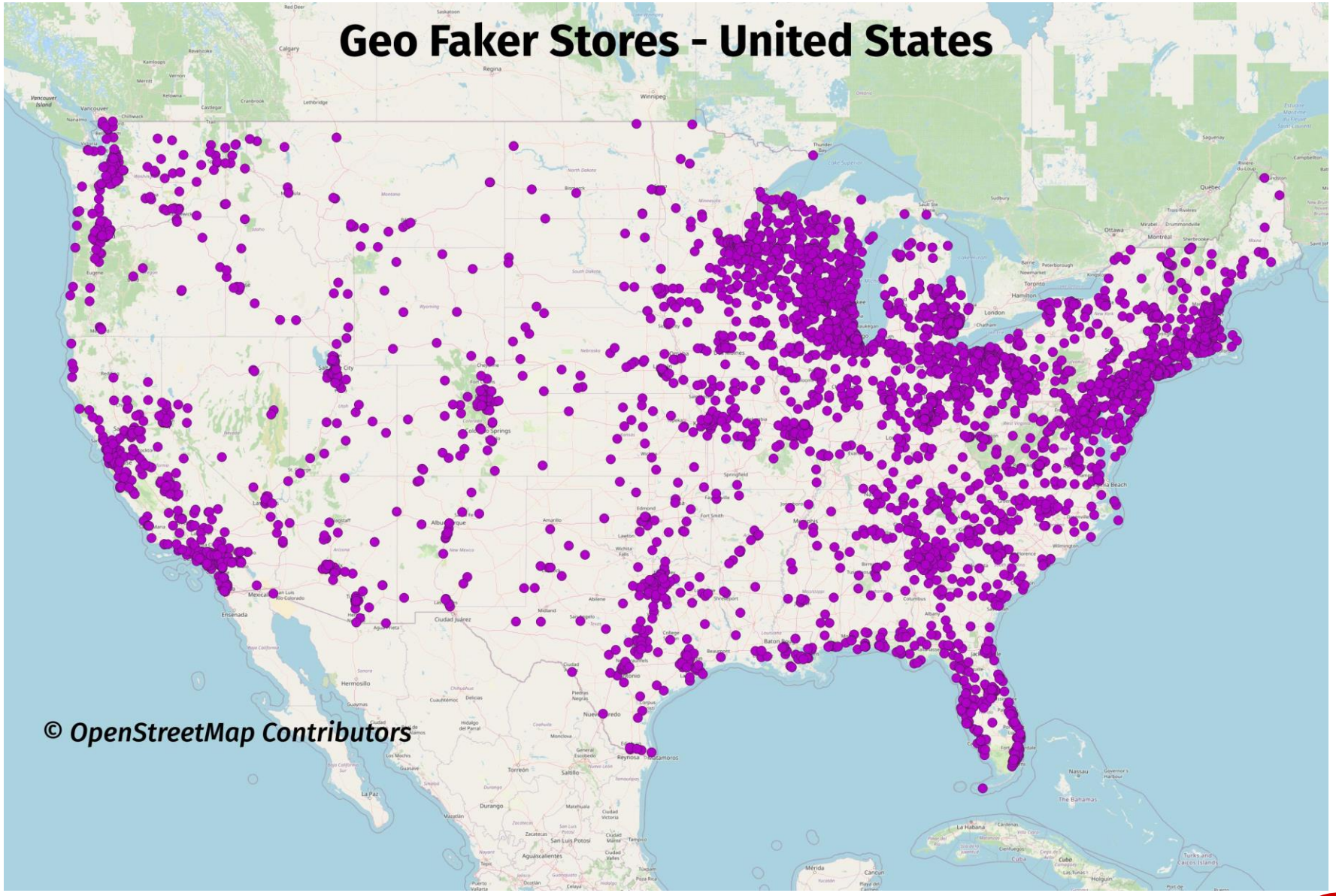
```
DROP EXTENSION pg_stat_statements;
```

Support varies among cloud providers

10 Extensions to know about

PostGIS

- Industry-standard, open-source, geographic database
- Adds geographic:
  - Types: point, line, polygon
  - Indexes: GIN, GIST, SPGIST, BRIN
  - Functions: hundreds of "ST_..." functions


Geo Faker Stores - United States
© OpenStreetMap Contributors

redgate

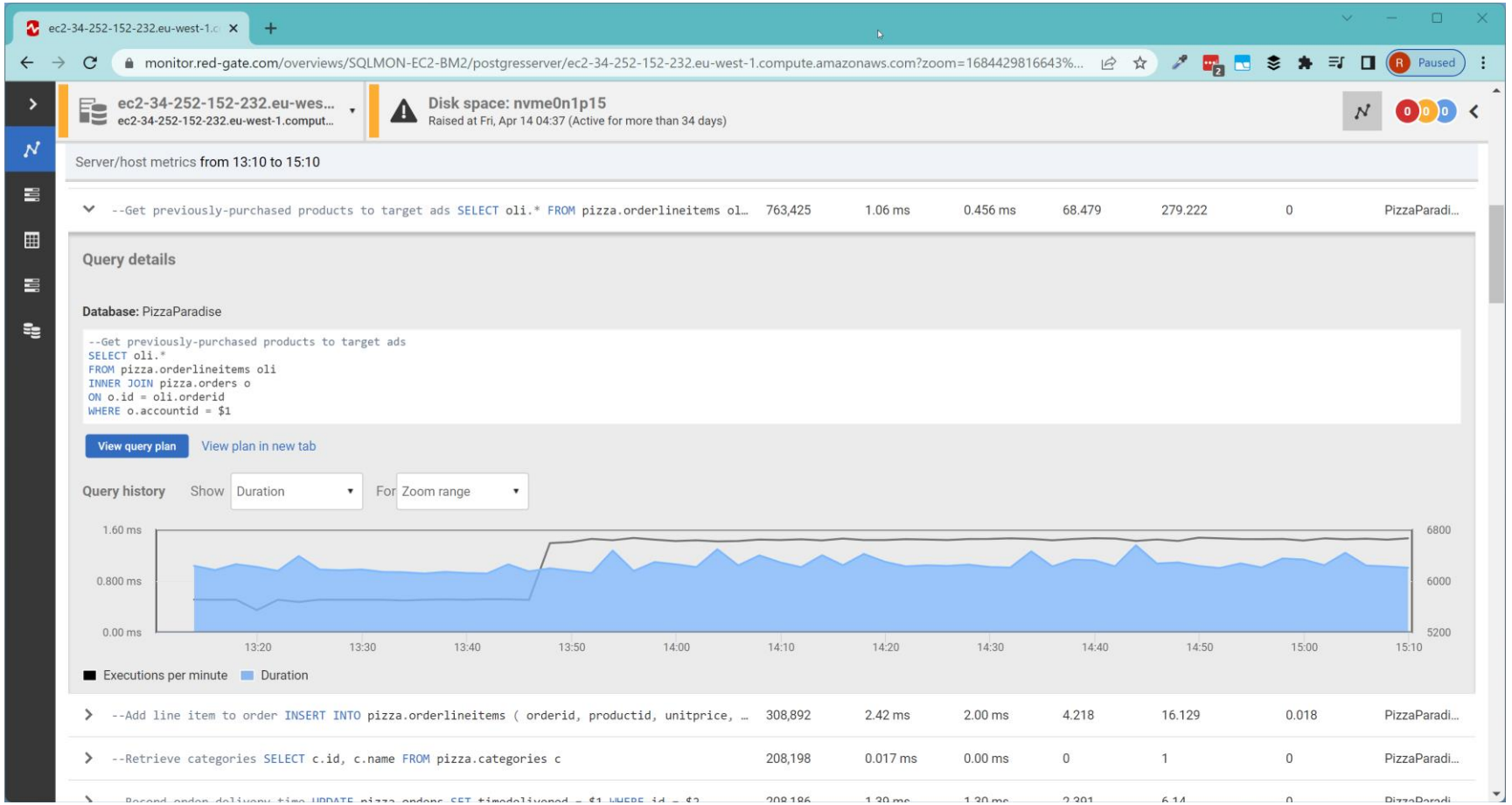
postgres_fdw

- Foreign Data Wrapper (FDW) is a standards-compliant connection infrastructure
- Initiates a connection to an external PostgreSQL database
- Necessary for cross-database *and* cross-instance queries
  - There is no fully-qualified notation to access other databases

pg_stat_statements

- One of many statistics views within PostgreSQL
- Data is stored by PostgreSQL, view provides access to the data
- Matches counts for parameterized versions of queries
- Cumulative only, requiring tooling to store snapshots over time
- Installed by default on all major cloud providers

SQL Monitor



redgate

pg_cron

- PostgreSQL does not have a built-in job scheduler
- pg_cron is a popular extension for scheduling and running jobs on a schedule in the background
- Other extensions will often recommend pg_cron for background work with that extension (e.g. pg_partman)
- Available in all major cloud providers

redgate

pg_trgm

- Provide a new Trigram index type
- Splits text into groups of three characters
- Indexes for faster searching and relevance
- Indexes are large and slow, by comparison, so use appropriately
- Use for smaller text fields, like names or people or items

hypopg

- Create an index that doesn't get materialized
- EXPLAIN the query to see if the planner would use the new index
- Supports many different types of index variations
- Extremely useful for large datasets
- YMMV

redgate

pg_partman

- Partition management in PostgreSQL
- Create partitions into the future before data arrives
- Archive/DROP older partitions for improved maintenance
- Use in conjunction with pg_cron for automated creation and maintenance over time

redgate

pg_vector

- New extension adding vector features to PostgreSQL
- Useful to working with LLM embeddings
- Lots of focus and development happening currently across multiple vector extensions

pg_hint_plan

- PostgreSQL doesn't support query hints or query plan caching
- Community extension that provides some ability to hint queries (similar to Oracle syntax)
- Won't solve all problems, but it can be helpful in a pinch

redgate

pgcrypto

- TDE is not currently supported in vanilla PostgreSQL
- **pgcrypto** provides cryptographic functions to:
  - Apply symmetric/public key encryption of data
  - Work with OpenPGP keys and payloads
  - Generate passwords
  - Hash/checksum content

TimescaleDB or Citus (BONUS)

- Examples of creating a "new" database on top of PostgreSQL using only extensions
- Automation of partitioning and sharding
- Various features for managing and querying large amounts of data across multiple nodes

Monitoring and Instrumentation

Server Metrics

- Error logs, off by default
- Cumulative Statistics System
  - Reset on crash, failover, or point in time restore
  - Can be manually reset
  - Aggregations only

redgate

Cumulative Statistics System

- pg_stat_activity – current activity on server
- pg_stat_database – activity & size of database
- pg_stat_*_tables – activity & size of table
  - all
  - sys
  - user
- pg_stat_*_indexes – guess
- pg_statio_*_tables – specific info on blocks & storage

redgate

Query Metrics

- EXPLAIN ANALYZE
- pg_stat_statements
  - Disabled by default

redgate

DEMO!

redgate

PASS Session

Professional PostgreSQL Monitoring Made Easy


Friday, Nov. 8


2:30-3:30


Room 447

redgate

Performance Tuning

Cost-based Optimizer

- Statistics determine plan choice
- Customizable per server, table, and column
- Asynchronous process maintains statistics
- Manually update with ANALYZE

Statistics

- Histogram of column values
- Defaults to 100 buckets
- Helpful views:
  - pg_catalog.pg_stats
  - pg_catalog.pg_stat_user_tables

EXPLAIN in Practice

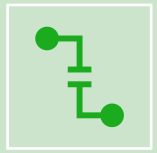
- EXPLAIN = Estimated plan
- EXPLAIN ANALYZE = Actual plan
- EXPLAIN (ANALYZE,BUFFERS)
  - Query plan with disk IO
- EXPLAIN (ANALYZE,BUFFERS,VERBOSE)
  - Additional details on columns, schemas, etc.

EXPLAIN

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

redgate

Reading EXPLAIN



--> Nodes

Read inside-out

Join/Aggregate/Merge/Append at each level

redgate

```
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
```

redgate

Primary Scan nodes

- Sequence scan (seq scan)
- Index scan
- Index-only scan

Primary join nodes

- Nested Loop
- Hash Join
- Merge Join

Explain Glossary: https://www.pgmustard.com/docs/explain

redgate

Other nodes

- Aggregate
- Append
- Sort
- Group
- Window Aggregate
- More…

Explain Glossary: https://www.pgmustard.com/docs/explain

redgate

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?”

redgate

pg_stat_statement statistics

- Execution Time (total/min/max/mean/stddev)
- Planning Time (total/min/max/mean/stddev)
- Calls (total)
- Rows (total)
- Buffers (shared/local/temp)
  - read/hit/dirtied/written
  - read/write time
- WAL

43 Columns of data (as of PG16)

redgate

```
Name                |Value
--------------------+-----------------------
userid              |16422
dbid                |16434
queryid             |-6155333619461995114
query               |SELECT id, name FROM...
plans               |0
total_plan_time     |0.0
min_plan_time       |0.0
max_plan_time       |0.0
mean_plan_time      |0.0
stddev_plan_time    |0.0
calls               |151
total_exec_time     |8.489053
min_exec_time       |0.013751
max_exec_time       |1.356096
mean_exec_time      |0.056218894039735096
stddev_exec_time    |0.11851139585068957
rows                |151
shared_blks_hit     |450
shared_blks_read    |3
```

All statistics are cumulative from the last restart*

*or reset by a superuser

redgate

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   | 2451.00000000000
Rows.        | 521
hit_percent  | 99.9873421
```

*Lots of good videos/tutorials, but also see the docs for more help and ideas.

redgate

PASS Session

Explaining the Postgres Query Planner


Wednesday, Nov. 6


2:00-3:00


Room 445-446

redgate

PASS Session

Can SQL Server DBAs/Developers Tune Queries in PostgreSQL

📅 Friday, Nov. 8

🕐 10:30-11:30

🏢 Room 447

redgate

PostgreSQL and the Cloud

PASS Session

PostgreSQL in the Cloud


Thursday, Nov. 7

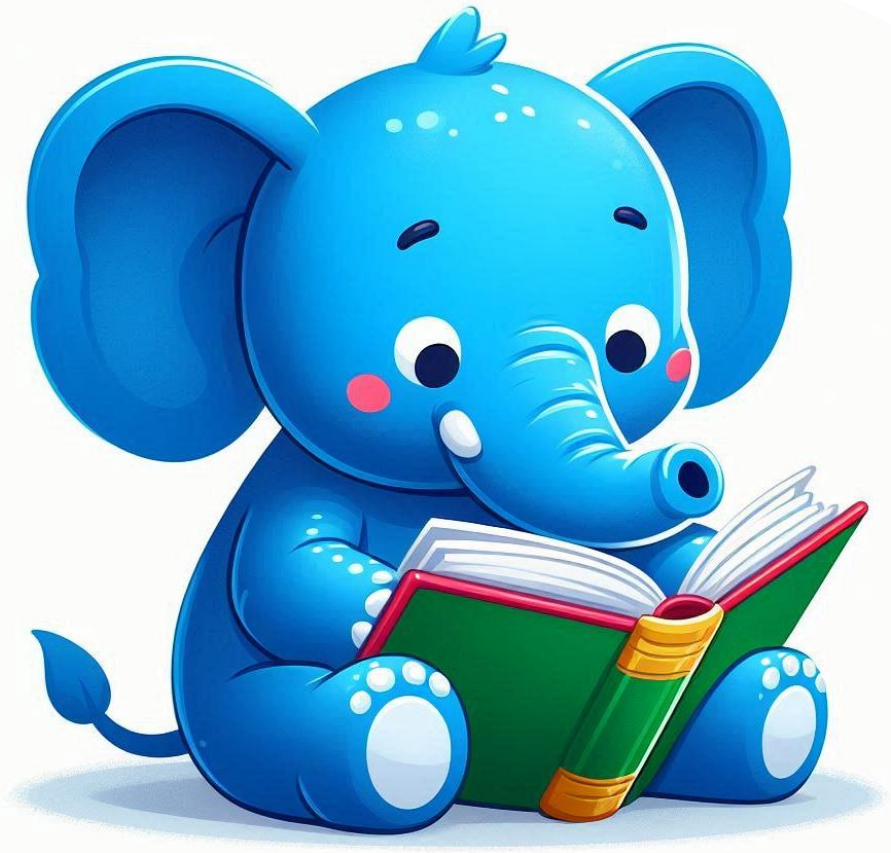

9:45-10:45


Room 445-446

redgate

Additional Resources

PostgreSQL Documentation



- https://www.postgresql.org/docs/
- Pay attention to this:
  - https://www.postgresql.org/docs/current/dml-delete.html

redgate

Books

- OURS!
- Art of PostgreSQL – Dimitri Fontaine
- Database Administration – Craig Mullins
- PostgreSQL Query Optimization - Henrietta Dombrovskaya, Boris Novikov and Anna Bailliekova

redgate

Other Events

- PostgreSQL User Groups (PUG)
- MeetUp
- PGDay
- pgConf (same link as PGDay)

redgate

Online

- Planet PostgreSQL
- PostgreSQL Slack
- Cooper Press Email List
- #pghelp on X
- Usual suspects

redgate

Calling All PostgreSQL Users

Whether you're a pro or just getting started, Redgate would love to hear your PostgreSQL insights and pain points

Together, we'll co-design the future of PostgreSQL tools and make development smoother for everyone!