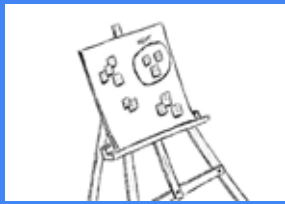




PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024



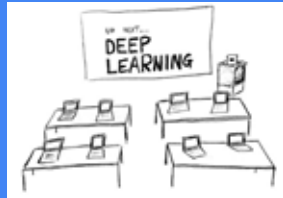
Consulting



Support



Services



Training



We Work With Data

We Work With
Big Data & Data Science

We are Data
Scientists



ailton.moreira@openup.pt

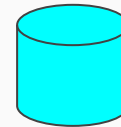
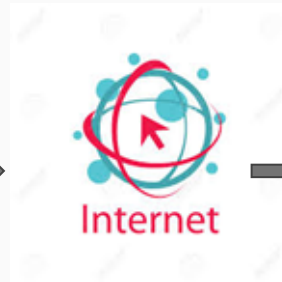
- PhD Candidate in Technology and Information Systems;
- Invited Assistant Professor at the University of Minho;
- Work as Researcher and Information Systems Engineer at ALGORITMI Centre;
- Worked with many RDBMS like OracleDB, MySQL, PostgreSQL, MS SQL Server, Redis, Cassandra, Neo4j, etc;
- Worked at Bosch Car Multimedia – Braga as researcher, developer and data engineer;
- Work in different Data Science projects.

- Briefly introduce yourself:
 - Name
 - Title or position
 - Company
 - Experience with database systems
 - Expectations

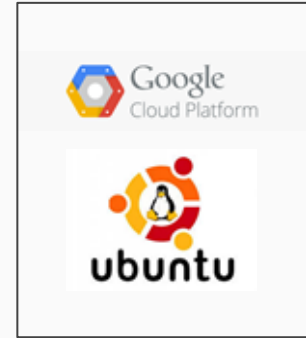
- Presences
 - <https://skillstep.pt/Presencas>
 - Autentication with
 - Course reference
 - Your name (from list)
 - Internal presence controll
- Evaluations (last day)
 - <https://skillstep.pt/Avaliacoes>
 - Autentication with
 - Course reference
 - Your name (from list)



Your laptop



PostgreSQL 16



VM Server



Container

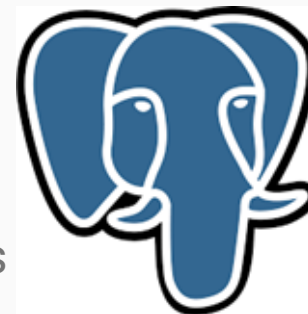
- Introduction
 - Introduction to Databases
 - Data Normalization
 - Indexes, Relationships and Keys
 - Introduction to RDBMS
 - Introduction to PostgreSQL 16



- Installation and Configuration
 - Downloading PostgreSQL 16 (Enterprise vs Open Source)
 - Installation and Service Configuration
 - Using the Command Line Client
- Install Admin Tools
 - pgAdmin
 - phpPgAdmin



- Administration and Security
 - Using pgAdmin
 - Accepting External Connections
- Databases, Tables, Schemas, Tablespaces, Other Objects
 - Creating Databases
 - Creating Tables
 - Inserting, Viewing and Deleting Data
- Users and Roles
 - Roles - Creating and Deleting Users
 - Roles - Assigning Permissions
- pgaudit extension



- Performance & Tuning
- Backing Up and Restoring Databases
- Replication
 - Master Server Setup
 - Replication - Slave Server Setup
- Database Clustering

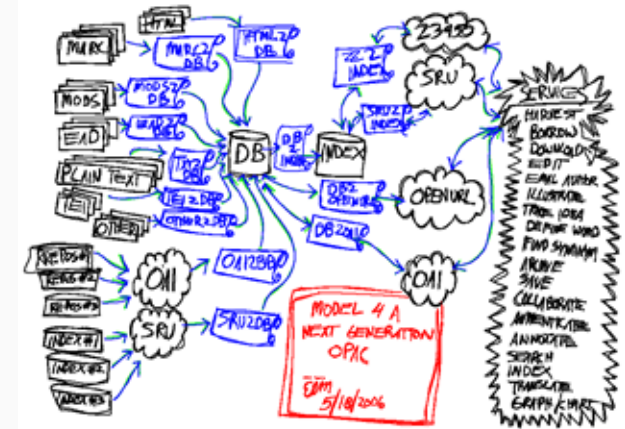


- Datatypes
 - Introduction to Datatypes
 - Character Datatype
 - Numeric Datatype
 - Serial Datatype
 - Boolean Datatype
 - Special Datatype - NULL
 - Date Datatype
 - Time Datatype
 - Timestamps



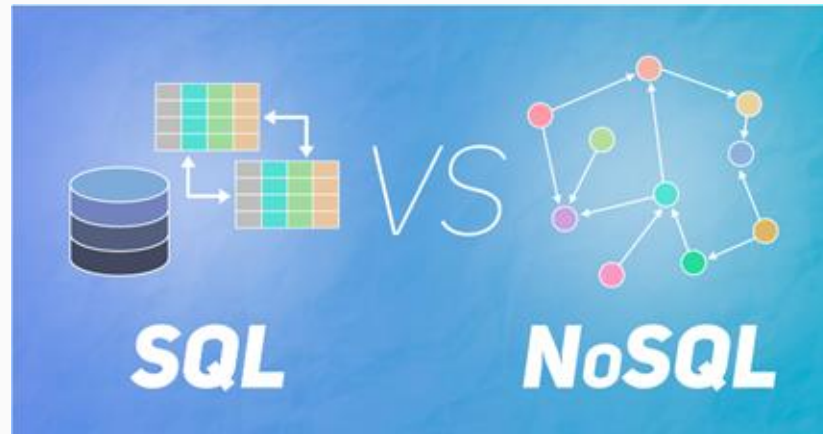
- Conclusion
 - NoSQL Use Cases and PostgreSQL
 - Big Data Processing



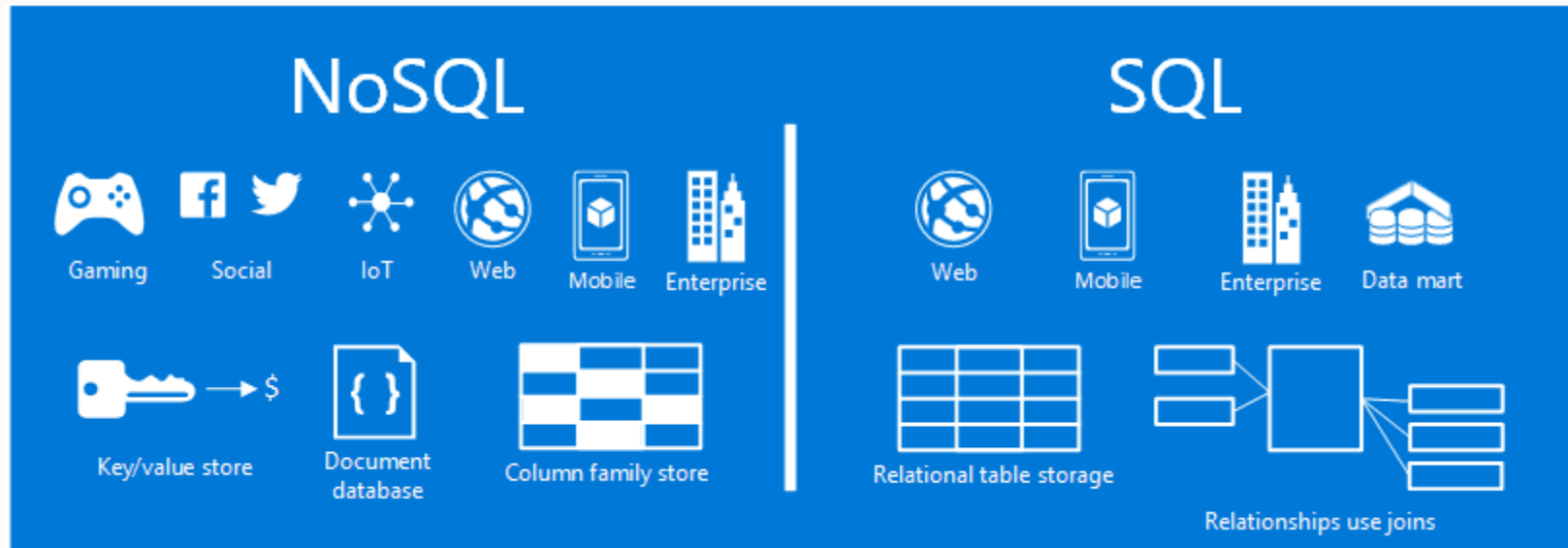


- Types of Databases, examples

- Relational Database
- NoSQL Database
- Spatial Database
- Unstructured Database
- Data Warehouse



- NoSQL vs SQL Databases



- What is Data Normalization
 - The process of structuring a relational database in order to reduce or eliminate redundancy of data in a database and improve data integrity;

2nd Normal Form - All attributes (Non-Key Columns) dependent on the key



3rd Normal Form - All Fields (columns) can be determined Only by the Key in the table and no other column

- What is an index
 - A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.
 - Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed

Composite Index

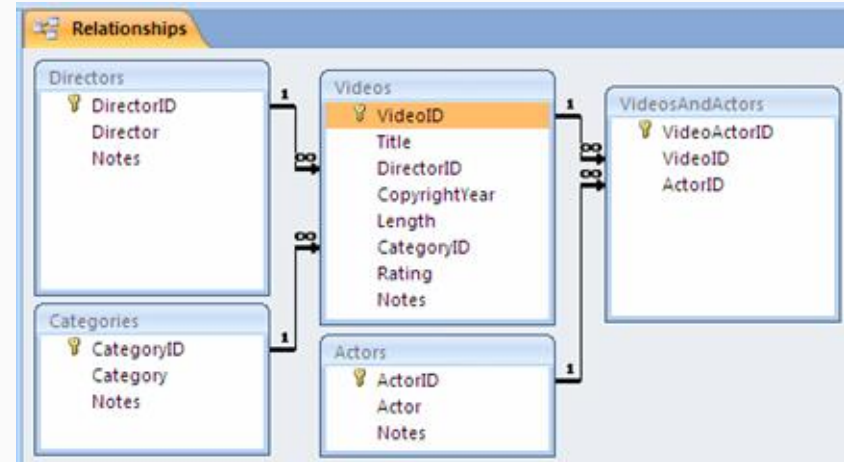
Allan, Ethen
Allan, John
Cooper, Stephen
Cooper, Thomas
Faust, Liz
Greenburg, Dale
Greenburg, Mike
Greenburg, Simon
-
Wu, Ellen

A composite index contains more than one column in a specific order. In this case, Last Name comes first.

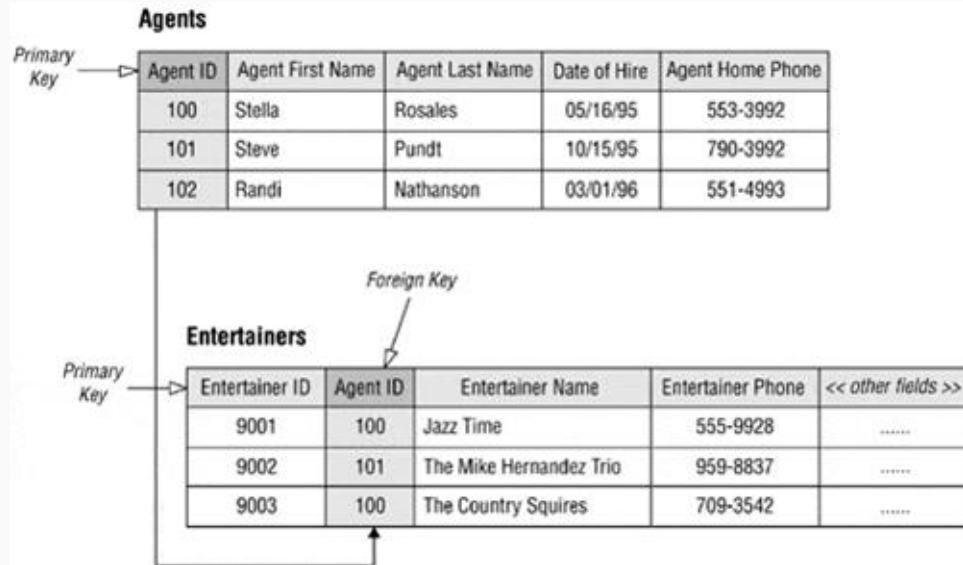
Table

1	Dale	Greenburg
2	Ellen	Wu
3	Ethen	Allan
4	John	Allan
5	Liz	Faust
6	Mike	Greenburg
7	Simon	Greenburg
8	Stephen	Cooper
-	Thomas	Cooper
26	-	-

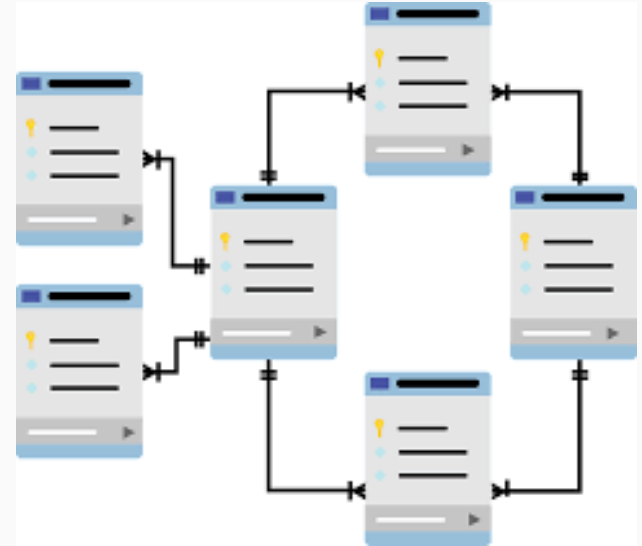
- What are Data Relationships
 - A relationship is established between two database tables when one table uses a foreign key that references the primary key of another table.
 - This is the basic concept behind the term relational database.



- What are Keys
 - A DBMS key is an attribute or set of an attribute which helps you to identify a row in a relation. They allow you to find the relation between two tables.
 - Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

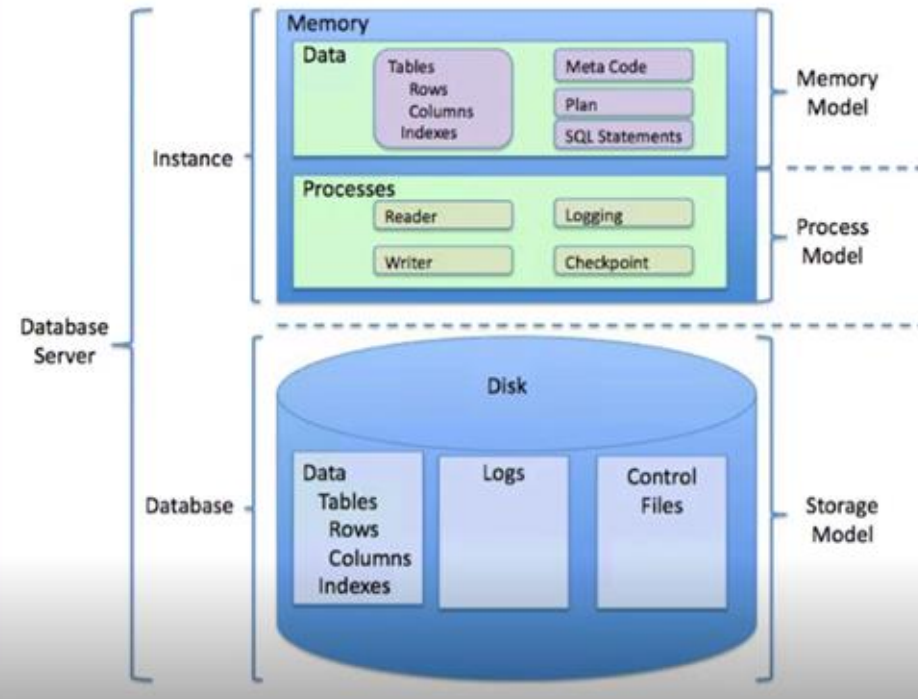


- What is an RDBMS
 - Stands for "Relational Database Management System." An RDBMS is a DBMS designed specifically for relational databases. Therefore, RDBMSes are a subset of DBMSes.
 - A relational database refers to a database that stores data in a structured format, using rows and columns.



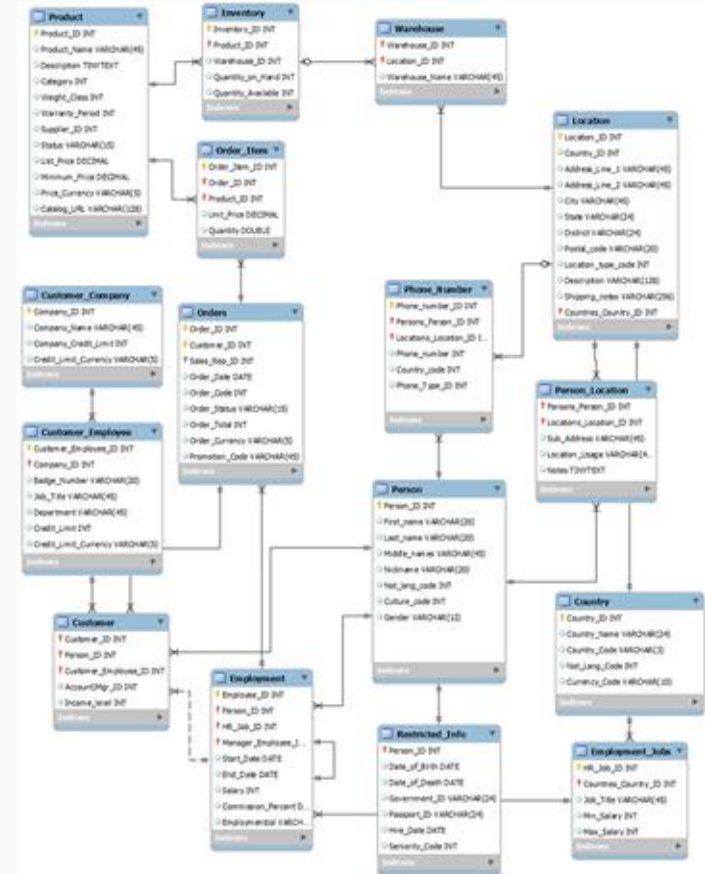
- What is an RDBMS
 - Data is assumed to have relationships with other data in the database;
 - The usage of indexes and keys is generally ubiquitous inside a RDBMS;

Basic Structure



- RDBMS Key Concepts

- Data is organized into tables of rows and columns with a unique key for each row;
- Each entity type the database has is own table;
- Rows in a table can be linked to rows in other tables through the storage of the unique key of the row it needs to be linked to (the foreign key);
- SQL is used as the language for querying and maintaining the database.





PostgreSQL

- What is PostgreSQL ?
 - It is a RDBMS similar to Oracle, MySQL, MariaDB, SQL Server, and others;
 - Comes in an Enterprise version with additional functions, features and support; and also an Open Source version freely downloadable with community support;
 - Supports both SQL (relational) and JSON (non-relational) querying.



- PostgreSQL Features

- All RDBMS SQL
 - Primary Keys
 - Foreign Keys
 - Transactional Integrity
 - etc
- Plus some NoSQL features
 - Document Database-JSON
 - Key-Value Pairs - HSTORE
 - JSONB – Binary JSON
 - Combining ANSI SQL Queries and JSON Queries



- PostgreSQL has many advanced features that other enterprise-class database management systems offer, such as:

User-defined types

Table inheritance

Sophisticated locking mechanism

Foreign key referential integrity

Views, rules, subquery

Nested transactions (savepoints)

Multi-version concurrency control (MVCC)

Asynchronous replication



- The recent versions of PostgreSQL support the following features:
 - Native Microsoft Windows Server version
 - Tablespaces
 - Point-in-time recovery
- And more new features are added in each new release.



- Lots of Management Tools
 - PgAdmin
 - phpPgAdmin
 - DbVisualizer
 - pgaccess
 - pgShark



- Supported Operating Systems

- Linux - several distributions
- Mac OSX
- MS Windows
- HP UX
- AIX
- Solaris/Sun OS
- Cygwin - Linux environment in MS Windows





We Work With Data

Thank you!

ailton.moreira@openup.pt



PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

Advantages of an Audit Log



Improve security

by providing records of all
IT activity



Prove compliance

for common regulations like
HIPAA and PCI DSS



Gain insight

to view user activity and
improve security



Manage risks

and demonstrate your
success to stakeholders

The options we have in PostgreSQL regarding audit logging are the following:

- By using exhaustive logging (`log_statement = all`)
- By writing a custom trigger solution
- By using standard PostgreSQL tools provided by the community, such as
audit-trigger 91plus
(<https://github.com/2ndQuadrant/audit-trigger>)
pgaudit extension
(<https://github.com/pgaudit/pgaudit>)



Session Auditing

Session auditing allows administrators to choose classes of statements to log:

- **READ** (SELECT and COPY when the source is a relation or a query)
- **WRITE** (INSERT, UPDATE, DELETE, TRUNCATE, and COPY when the destination is a relation)
- **FUNCTION** (Functions and DO blocks)
- **ROLE** (GRANT, REVOKE, CREATE/ALTER/DROP ROLE)
- **DDL** (All DDL not included in ROLE)
- **MISC** (DISCARD, FETCH, CHECKPOINT, VACUUM)



Session Auditing

Session auditing allows administrators to choose classes of statements to log:

```
AUDIT: SESSION,1,1,READ,SELECT,,SELECT * FROM account,  
,<not logged>
```

Diagram labels for the first statement:

- CLASS: points to 'SESSION'
- COMMAND: points to 'READ'
- STATEMENT: points to 'SELECT * FROM account'

```
AUDIT: OBJECT,1,1,READ,SELECT,TABLE,public.account,SELECT *  
FROM account,<not logged>
```

Diagram labels for the second statement:

- OBJECT_TYPE: points to 'OBJECT'
- OBJECT_NAME: points to 'public.account'





[12-Lab -Install and use PGaudit](#)



We Work With Data

Thank you!

ailton.moreira@openup.pt



PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

PostgreSQL provides various methods for authenticating users:

- **Trust** - which simply trusts that users are who they say they are.
- **Password** - which requires that users send a password.
 - **scram-sha-256** - The method scram-sha-256 performs SCRAM-SHA-256 authentication, as described in RFC 7677. This is the most secure of the currently provided methods, but it is not supported by older client libraries.
 - **md5** - It prevents password sniffing and avoids storing passwords on the server in plain text but provides no protection if an attacker manages to steal the password hash from the server. Also, the MD5 hash algorithm is nowadays no longer considered secure against determined attacks.
 - **password** - The method password sends the password in clear-text and is therefore vulnerable to password “sniffing” attacks. It should always be avoided if possible.

PostgreSQL provides various methods for authenticating users:

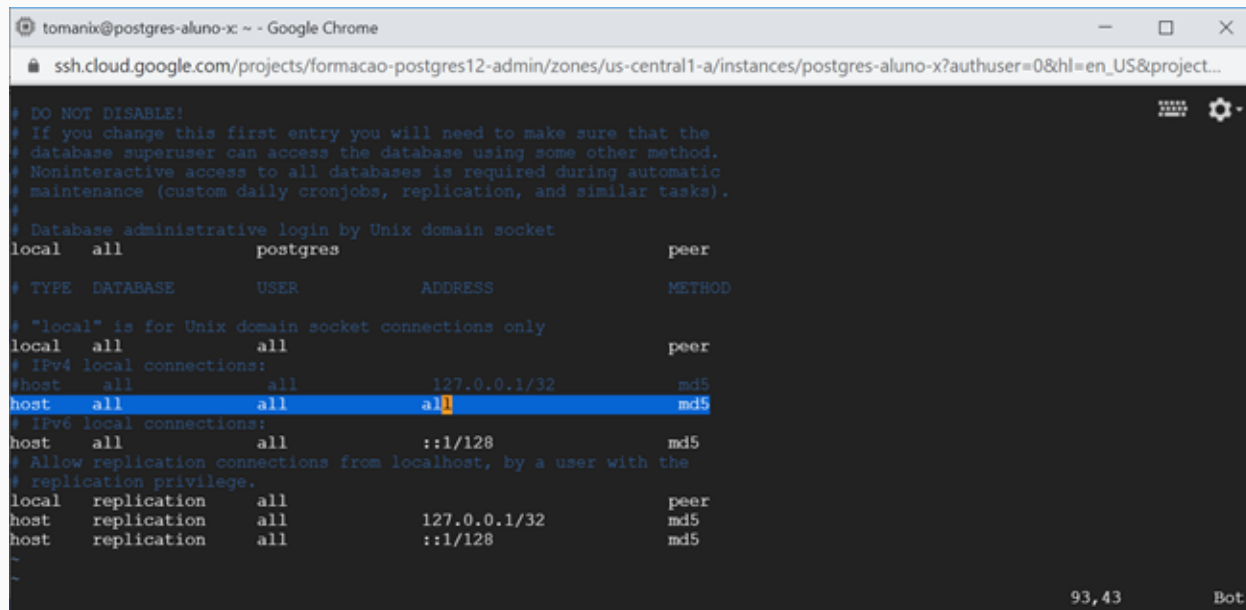
- **GSSAPI** - which relies on a GSSAPI-compatible security library. Typically this is used to access an authentication server such as a Kerberos or Microsoft Active Directory server.
- **SSPI** - which uses a Windows-specific protocol similar to GSSAPI.
- **Ident** - which relies on an “Identification Protocol” (RFC 1413) service on the client's machine. (On local Unix-socket connections, this is treated as peer authentication.)
- **Peer** - which relies on operating system facilities to identify the process at the other end of a local connection. This is not supported for remote connections.

PostgreSQL provides various methods for authenticating users:

- **LDAP** - which relies on an LDAP authentication server.
- **RADIUS** - which relies on a RADIUS authentication server.
- **Certificate** - which requires an SSL connection and authenticates users by checking the SSL certificate they send.
- **PAM** - which relies on a PAM (Pluggable Authentication Modules) library.
- **BSD** - which relies on the BSD Authentication framework (currently available only on OpenBSD).

Configuration file to set authentication methods:

/etc/postgresql/16/main/pg_hba.conf



```
tomanix@postgres-aluno-x: ~ - Google Chrome
ssh.cloud.google.com/projects/formacao-postgres12-admin/zones/us-central1-a/instances/postgres-aluno-x?authuser=0&hl=en_US&project...

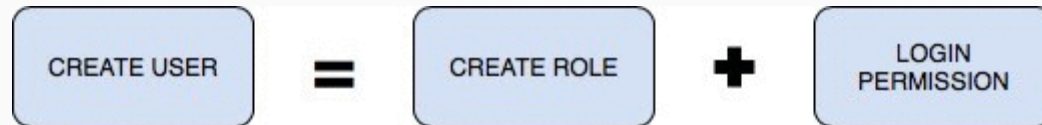
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local all postgres peer

# TYPE DATABASE USER ADDRESS METHOD

# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all all md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all peer
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
~
93, 43 Bot
```

PostgreSQL Authorization:

- **ROLE**
- **USER - ROLE + Login**





We Work With Data

Thank you!

ailton.moreira@openup.pt



PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

A

SQL Dump:

Extract database to a script or archive

- Restore anywhere
- Simple and flexible
- Current state only
- Large files

B

File System Level Backup:

Directly copy the files used to store data

- Faster
- Entire cluster
- No lock contention
- Near-instant recovery
- Requires shutdown
- Version dependent
- All or nothing
- Includes bloat

C

Continuous Archiving / PITR:

File System + WAL

- No snapshot needed
- Continuous backups
- Point-in-time
- Creates warm standby
- Most complicated
- Version dependent
- All or nothing
- Large archive storage

To backup one database, you can use the **pg_dump** tool. The `pg_dump` dumps out the content of all database objects into a single file.

Ex: `pg_dump -U postgres -W -F t dvdrental > /tmp/dvdrental.tar`

- **-U postgres**: specifies the user to connect to PostgreSQL database server. We used postgres in this example.
- **-W**: forces `pg_dump` to prompt for the password before connecting to the PostgreSQL database server.
- **-F**: specifies the output file format
 - **c**: custom-format archive file format
 - **d**: directory-format archive
 - **t**:tar
 - **p**: plain text SQL script file.

To backup all databases use the **pg_dumpall** tool that allows you to backup all databases at once. However, **it is not recommended to use this tool** because of the following reasons:

- The `pg_dumpall` program exports all databases, one after another, into a single script file, which prevents you from performing the parallel restore. If you backup all databases this way, the restore process will take more time.
- The processing of dumping all databases takes longer than each individual one so you do not know which dump of each database relates to a specific point in time.

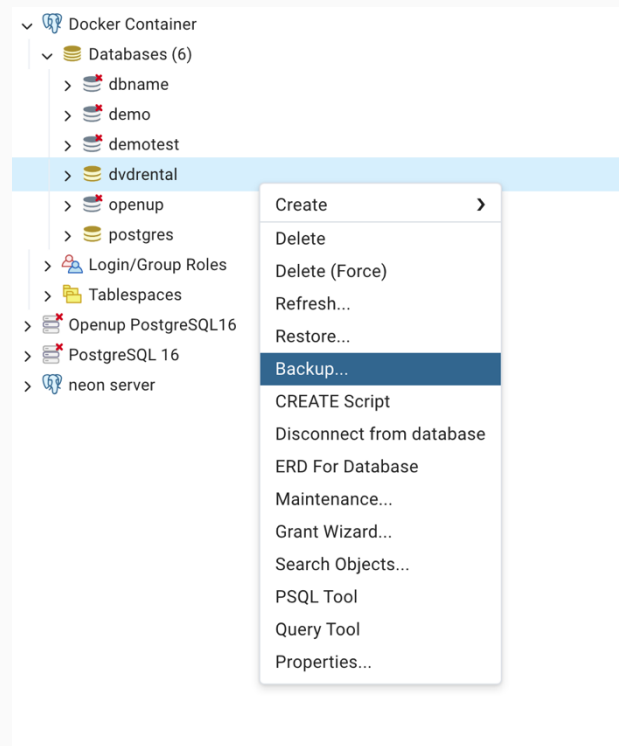
To backup only database object definitions so that you can restore the schema only:

- `pg_dumpall --schema-only > /tmp/definitiononly.sql`
- `pg_dumpall --roles-only > /tmp/allroles.sql`
- `pg_dumpall --tablespaces-only > /tmp/allroles.sql`

How to backup using pgAdmin



The pgAdmin provides an intuitive user interface that allows you to backup a database using pg_dump tool.



How to backup using pgAdmin



Backup (Database: dvdrental)

General

Data Options

Query Options

Table Options

Options

Objects

Filename

/Users/ailtonmoreira/Documents/dvdrental.tar

Format

Tar

Compression ratio

Encoding

Select an item...

Number of jobs

Role name

postgres

i

?

Close

Reset

Backup

Backup (Database: dvdrental)

General

Data Options

Query Options

Table Options

Options

Objects

Sections

Pre-data

Data

Post-data

Type of objects

Only data

Only schemas

Blobs

Do not save

Owner

Privileges

Tablespaces

Unlogged table data

Comments

Publications

Subscriptions

Security labels

i

?

Close

Reset

Backup

How to backup using pgAdmin



Backup (Database: dvdrental)

General

Data Options

Query Options

Table Options

Options

Objects

Use INSERT Commands

☐

Maximum rows per INSERT command

On conflict do nothing to INSERT command

☐

Include DROP DATABASE statement

☐

Include CREATE DATABASE statement

☐

Include IF EXISTS clause

☐

i

?

Close

Reset

Backup

Backup (Database: dvdrental)

General

Data Options

Query Options

Table Options

Options

Objects

📄

If Schema(s) is selected then it will take the backup of that selected schema(s) only

▼

public

>

✓

Tables

>

✓

Views

>

✓

1..3 Sequences

i

?

Close

Reset

Backup

The **psql** utility allows you to restore SQL script file generated by the pg_dump, pg_dumpall or any other tools that generates compatible backed up files.

Ex: `psql -U username -f backupfile.sql`

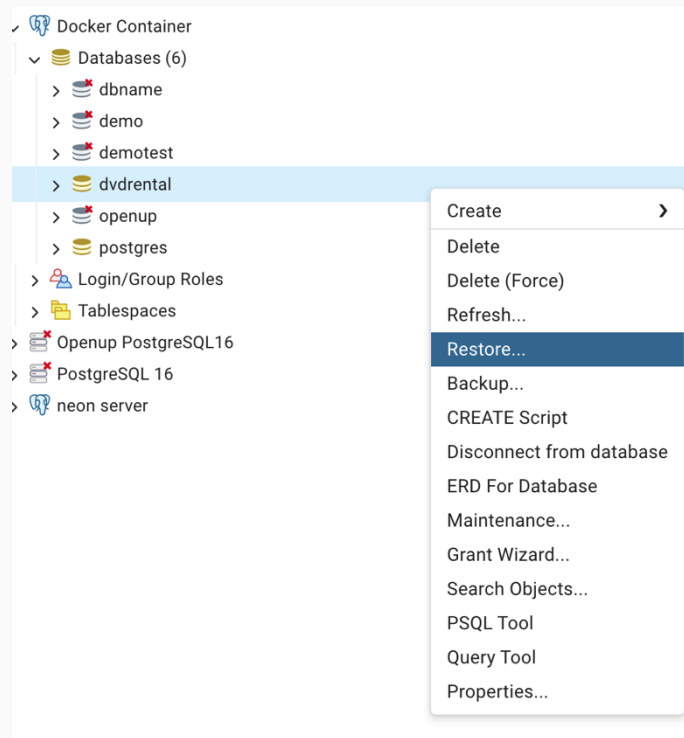
To restore database objects in a particular database

Ex: `psql -U username -d database_name -f objects.sql`

To restore a database backed up by the **pg_dump** or **pg_dumpall** we can use **pg_restore** program.

- The `pg_restore` allows you to perform parallel restores using the `-j` option to specify the number of threads for restoration. Each thread restores a separate table simultaneously, which speeds up the process dramatically. Currently, the `pg_restore` support this option for the only custom file format.
- The `pg_restore` enables you to restore specific database objects in a backup file that contains the full database.
- The `pg_restore` can take a database backed up in the older version and restore it in the newer version.

If you want to run the **pg_restore** via an intuitive user interface instead of the command line, you can use the **pgAdmin** restore tool.



How to restore using pgAdmin



Restore (Database: dvdrental) X

General Data Options Query Options Table Options Options

Format

Filename

Number of jobs

Role name

[Info icon] [Help icon] X Close ↺ Reset ⬆ Restore

Restore (Database: dvdrental) X

General Data Options Query Options Table Options Options

Sections

Pre-data	☐	Data	☐
Post-data	☐		

Type of objects

Only data	☐	Only schema	☐
-----------	--------------------------	-------------	--------------------------

Do not save

Owner	☐	Privileges	☐
Tablespaces	☐	Comments	☐
Publications	☐	Subscriptions	☐
Security labels	☐	Table access methods	☐

[Info icon] [Help icon] X Close ↺ Reset ⬆ Restore

How to restore using pgAdmin

Restore (Database: dvdrental)

General

Data Options

Query Options

Table Options

Options

Disable

Triggers

☐

Miscellaneous / Behavior

Verbose messages

☒

Use SET SESSION AUTHORIZATION

☐

Exit on error

☐

Exclude schema

i

?

Close

Reset

Restore



We Work With Data

Thank you!

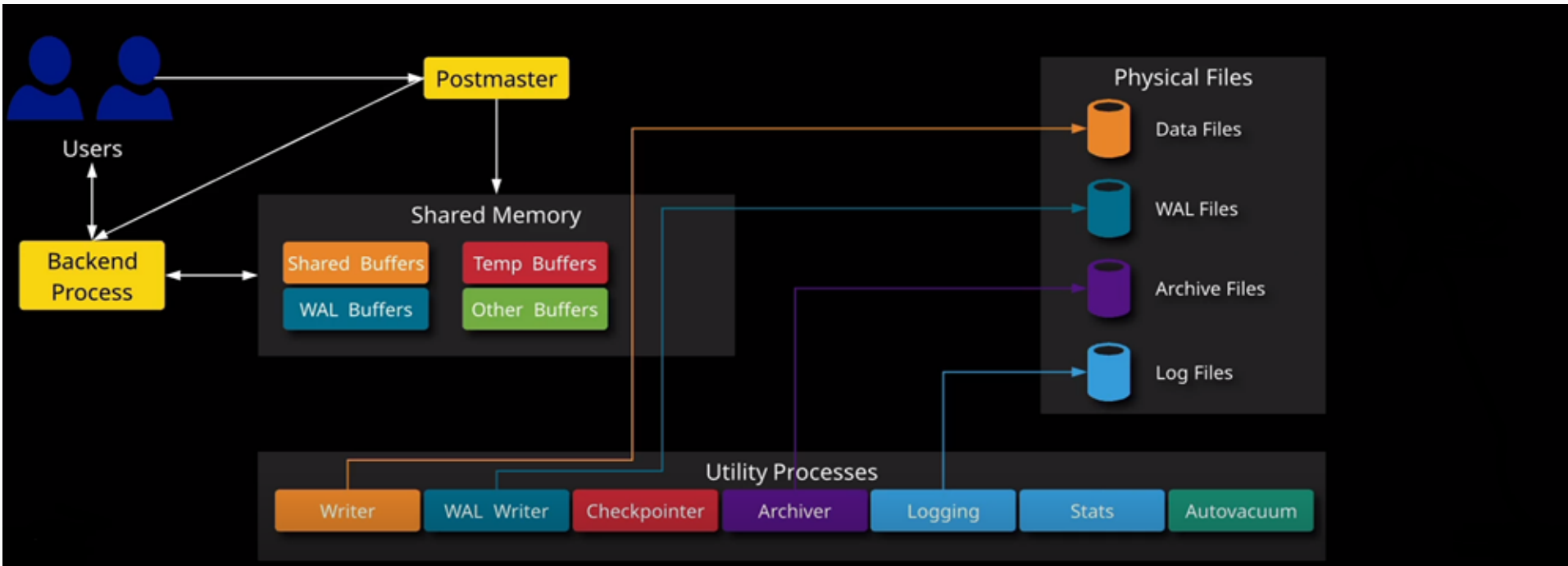
ailton.moreira@openup.pt

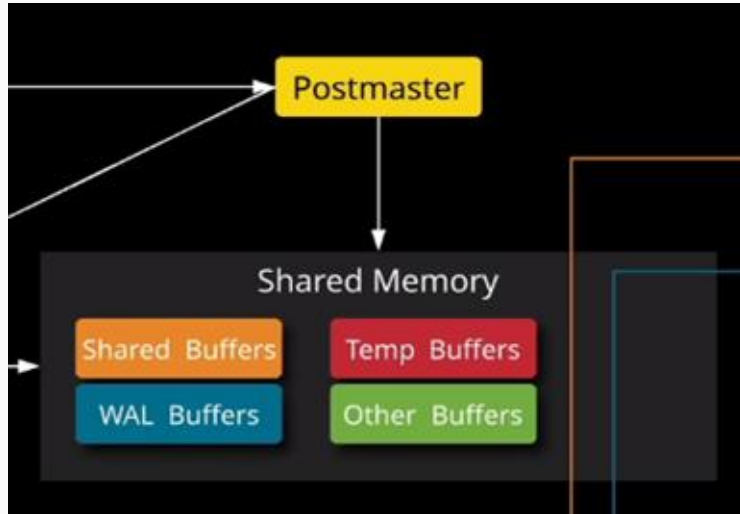


PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024





- **Postmaster** - is the first process started when you start PostgreSQL. At startup, performs recovery, initialize shared memory, and run background processes.
- **shared_buffers** - The purpose of Shared Buffer is to minimize DISK IO
- **temp_buffers** - session-local buffers used only for access to temporary tables
- **wal_buffers** - a buffer that temporarily stores changes to the database. The contents stored in the write ahead log (WAL) buffer are written to the WAL file at a predetermined point in time.

Physical Files

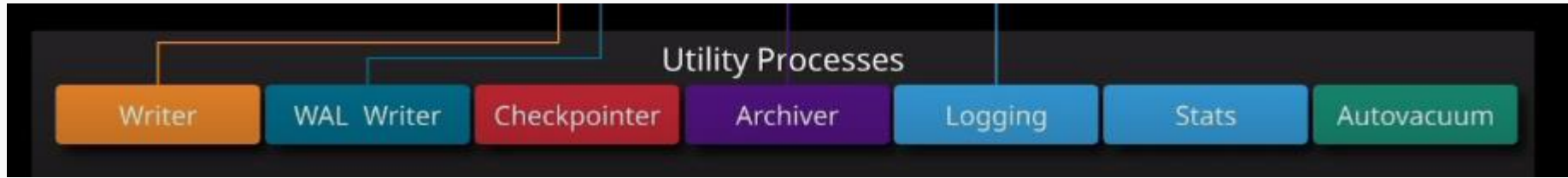
Data Files

WAL Files

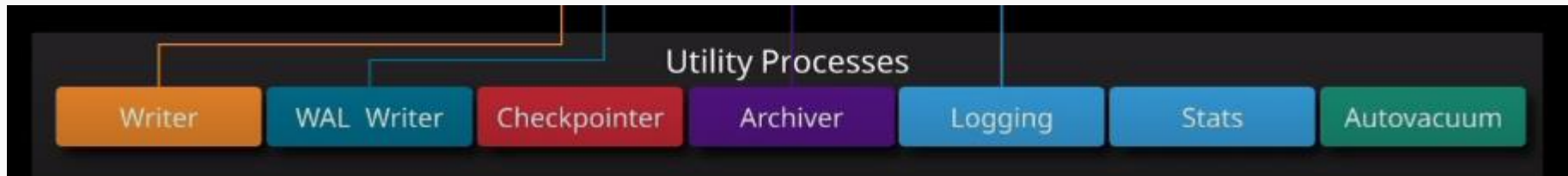
Archive Files

Log Files

- **Data Files** - Where the data is stored
- **WAL Files** - a write ahead log (WAL), records every change made to the database's data files.
- **Archive Files** - Copy, archive, of the WAL files
- **Log Files** - Log generated files



- **Writer** - Periodically writes the dirty buffer to a file.
- **WAL Writer** - Write the WAL buffer to the WAL file.
- **Checkpointer** - When a checkpoint occurs, the dirty buffer is written to the file.
- **Archiver** - When in Archive.log mode, copy the WAL file to the specified directory.
- **Logger** - Write the error message to the log file.
- **Stats Collector** - DBMS usage statistics such as session execution information (`pg_stat_activity`) and table usage statistical information (`pg_stat_all_tables`) are collected.
- **Autovacuum** - Fork autovacuum worker when autovacuum is enabled. It is the responsibility of the autovacuum daemon to carry vacuum operations on bloated tables on demand.



- **VACUUM** – garbage-collect and optionally analyze a database

VACUUM reclaims storage occupied by dead tuples. In normal PostgreSQL operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a VACUUM is done. Therefore, it's necessary to do VACUUM periodically, especially on frequently-updated tables.

- **ANALYZE** – collect statistics about a database

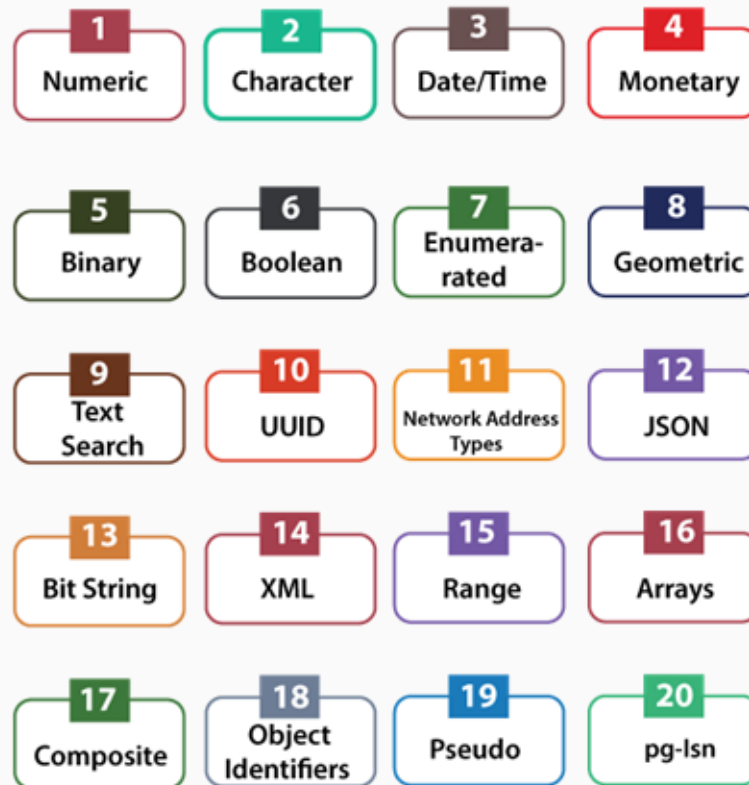
ANALYZE collects statistics about the contents of tables in the database, and stores the results in the `pg_statistic` system catalog. Subsequently, the query planner uses these statistics to help determine the most efficient execution plans for queries.

PostgreSQL Data Types

[PostgreSQL 16 Datatypes](#)

<https://www.postgresql.org/docs/current/datatype.html>

PostgreSQL Data Types





We Work With Data

Thank you!

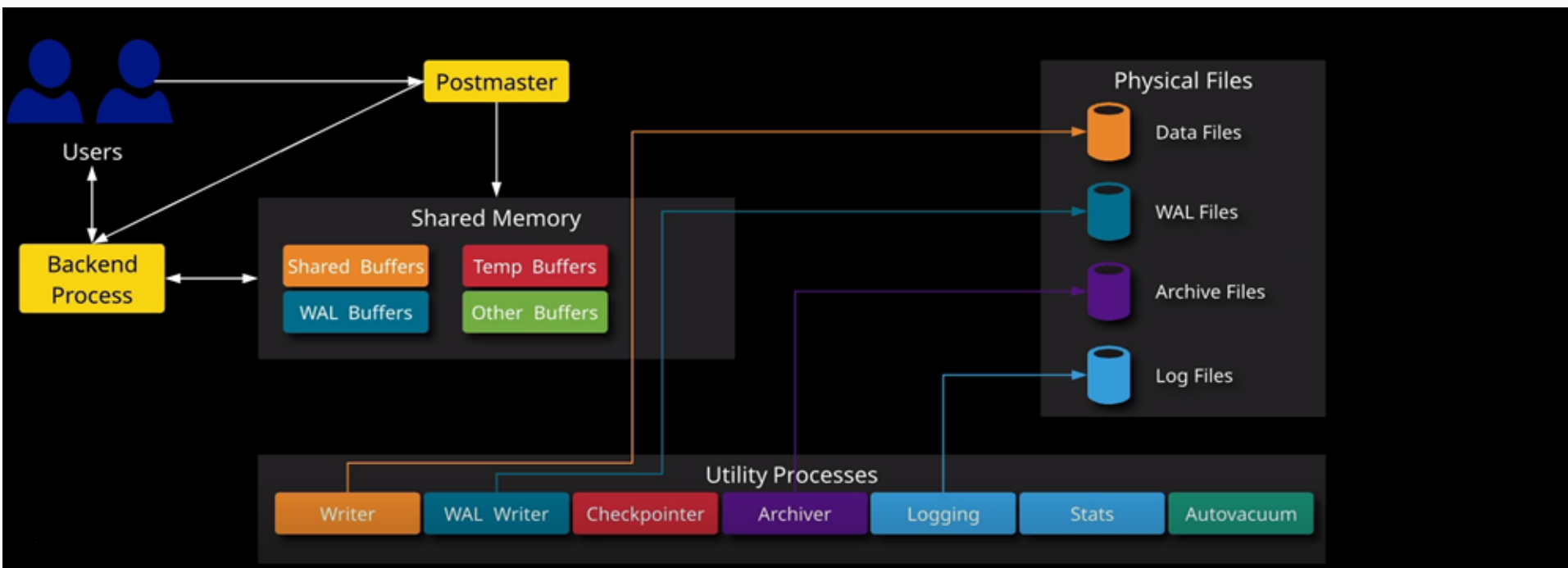
ailton.moreira@openup.pt



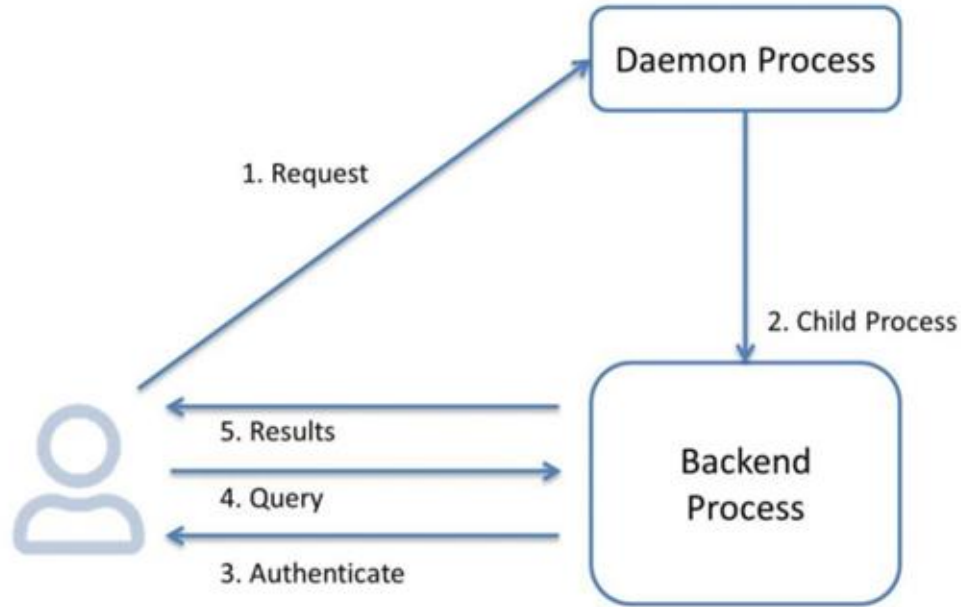
PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

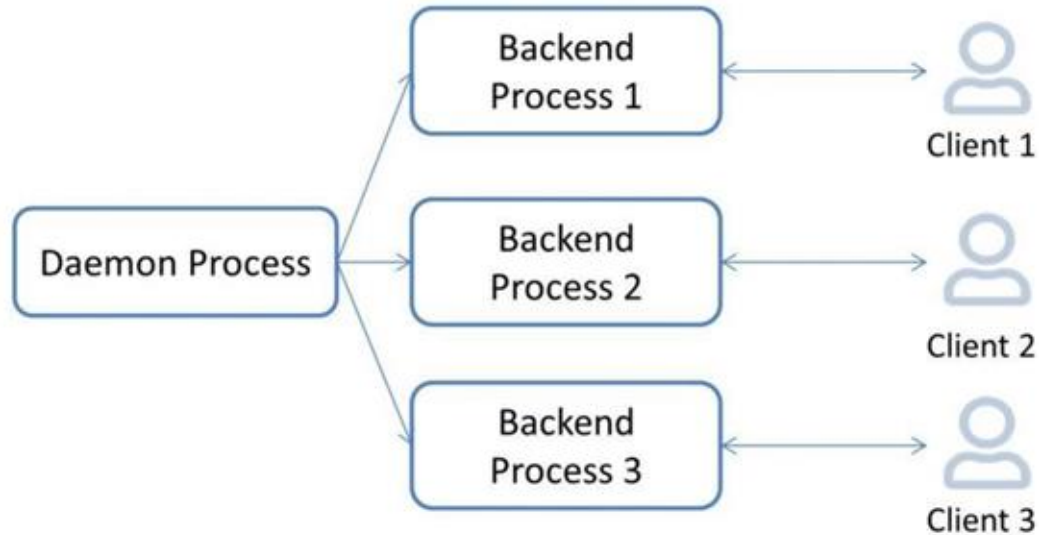


Process manager



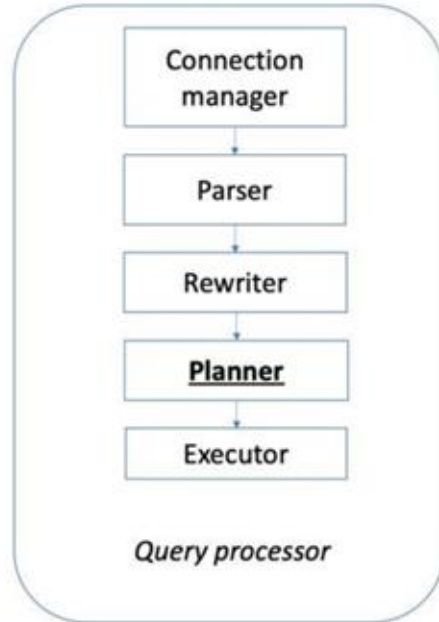
PostgreSQL query request overview

Process manager



PostgreSQL - forking the daemon process

Process manager



Cost Estimates

0.2



Use index

0.8



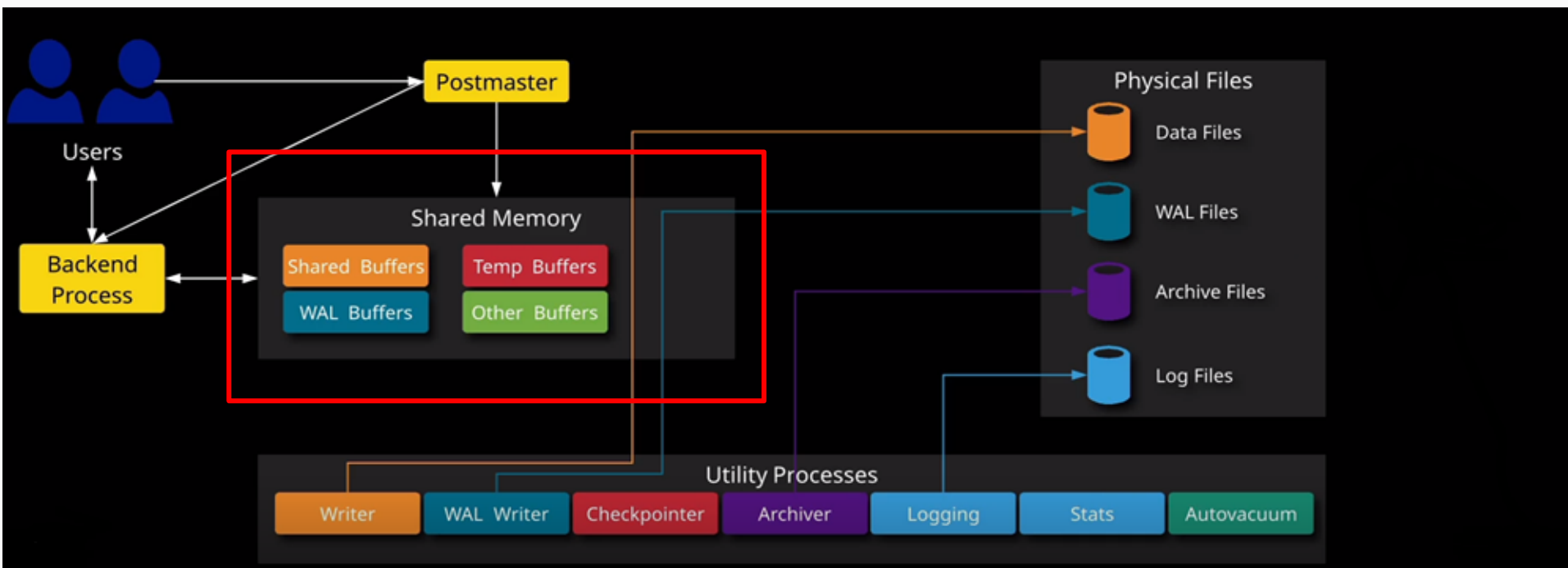
Use sequential scan

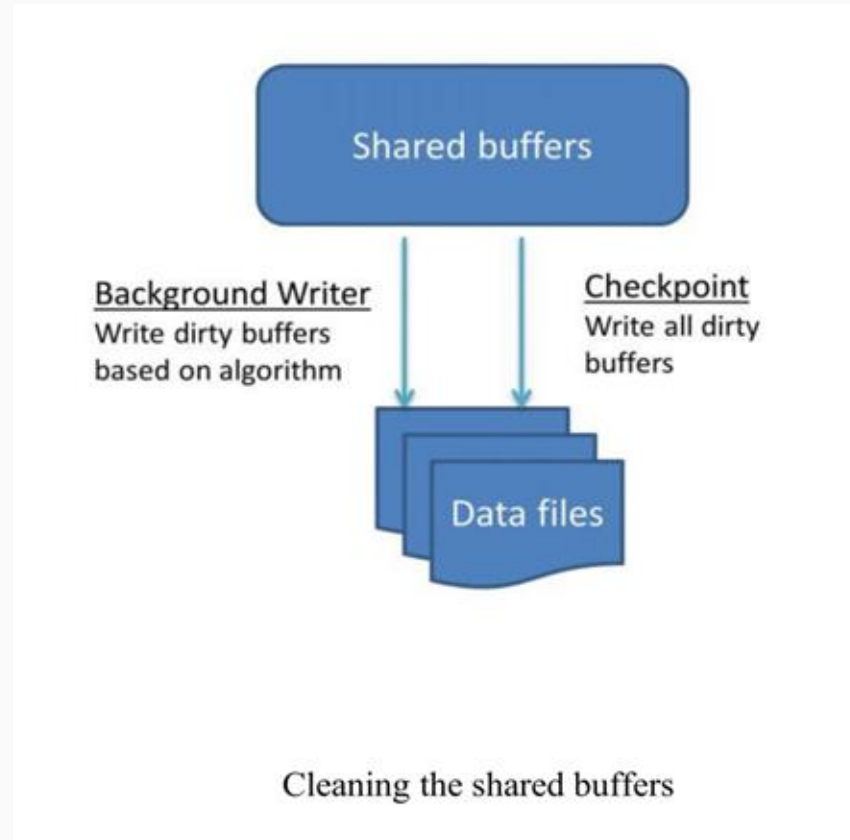
0.5



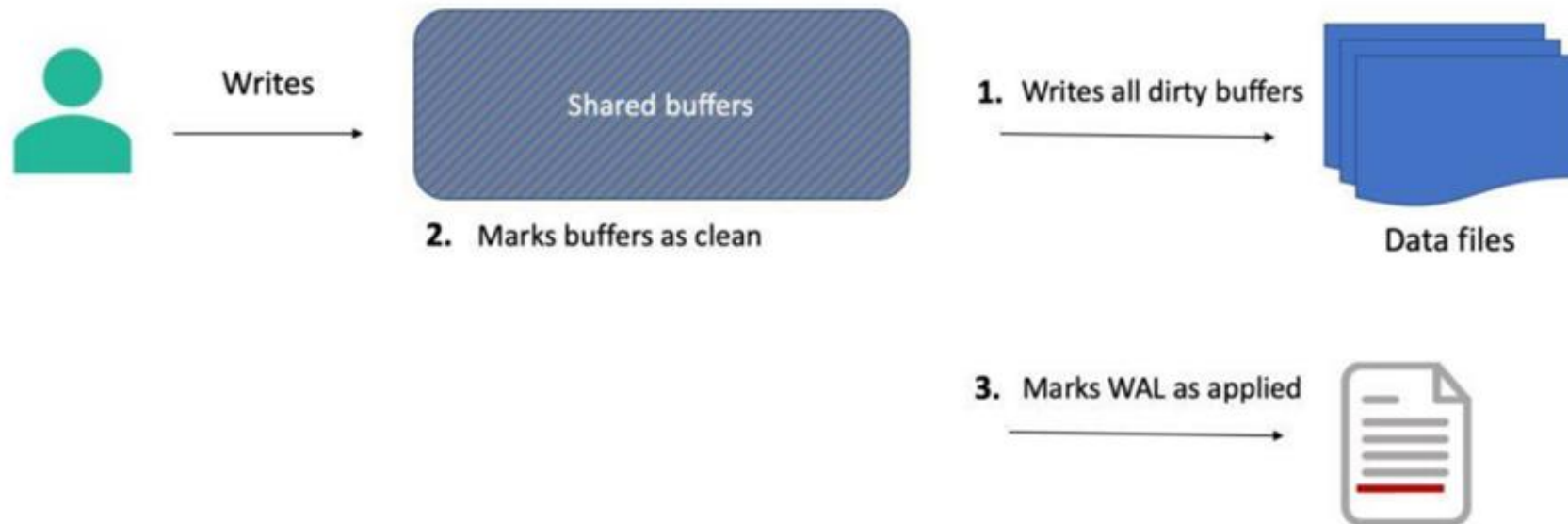
Use bitmap scan

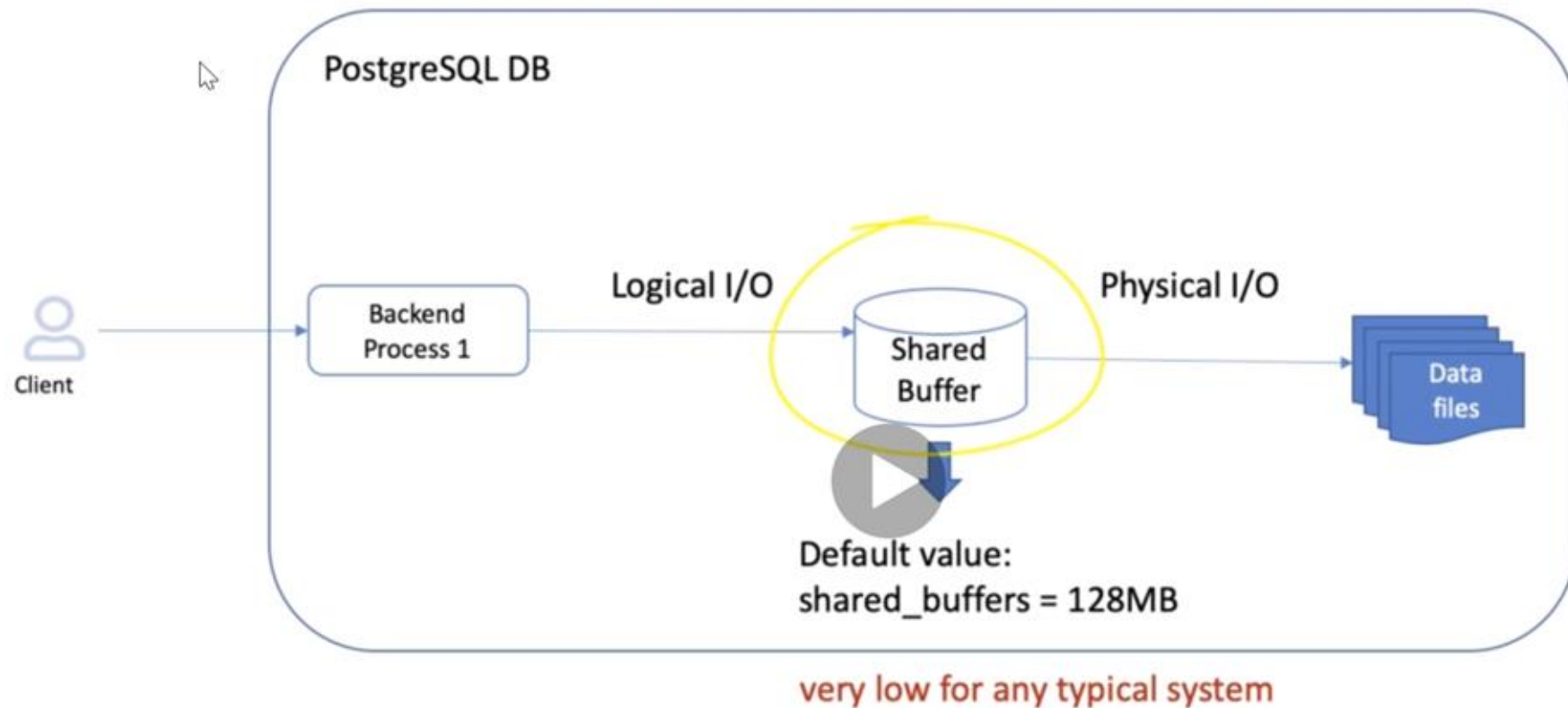
The Path of a Query

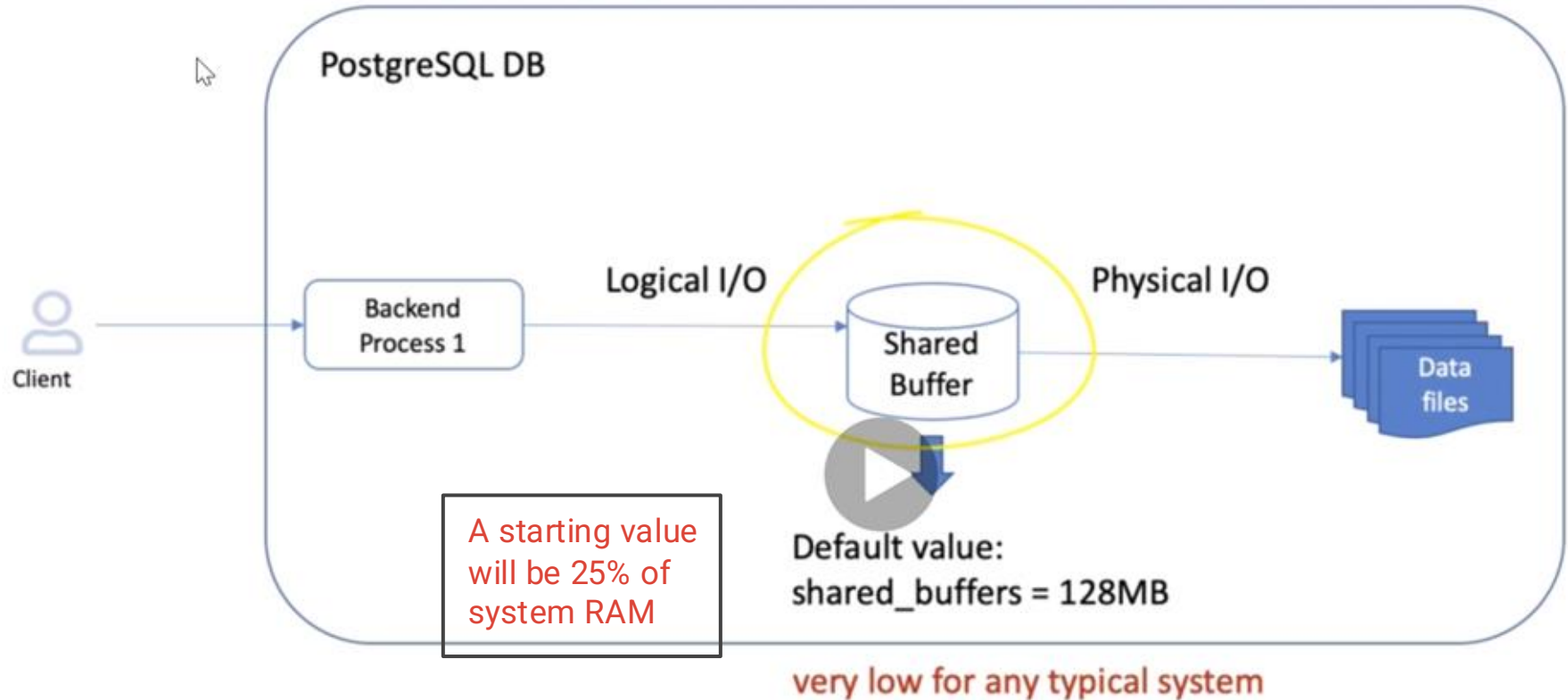




Checkpoint Process

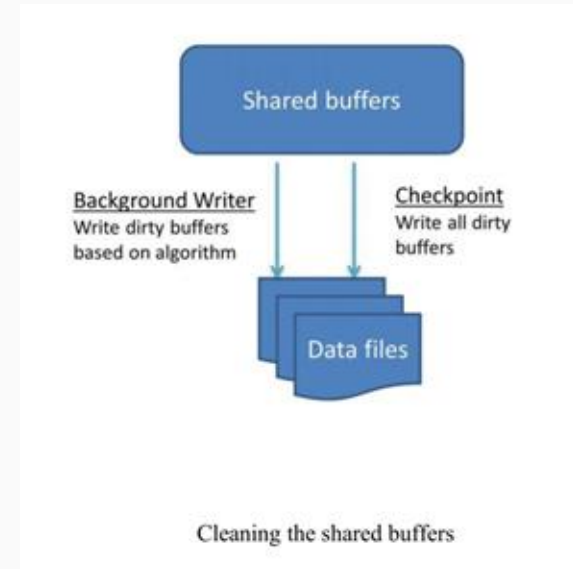






[Lab: Tuning the Shared Buffers using pgbench](#)

[Lab: Using pgbench to benchmark the top 10 queries](#)





We Work With Data

Thank you!

ailton.moreira@openup.pt



PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

- Installation
 - Lab: Install and Configure PostgreSQL 16
 - [3 - Lab: Install and Configure PostgreSQL 16](#)
 - [3 - Lab Solution - Install and Configure PostgreSQL 16](#)



- Installation
 - Lab: Configure PostgreSQL 16
 - [4 - Lab Configure PostgreSQL 16](#)
 - [4 - Lab Solution - Configure PostgreSQL 16](#)





We Work With Data

Thank you!

ailton.moreira@openup.pt



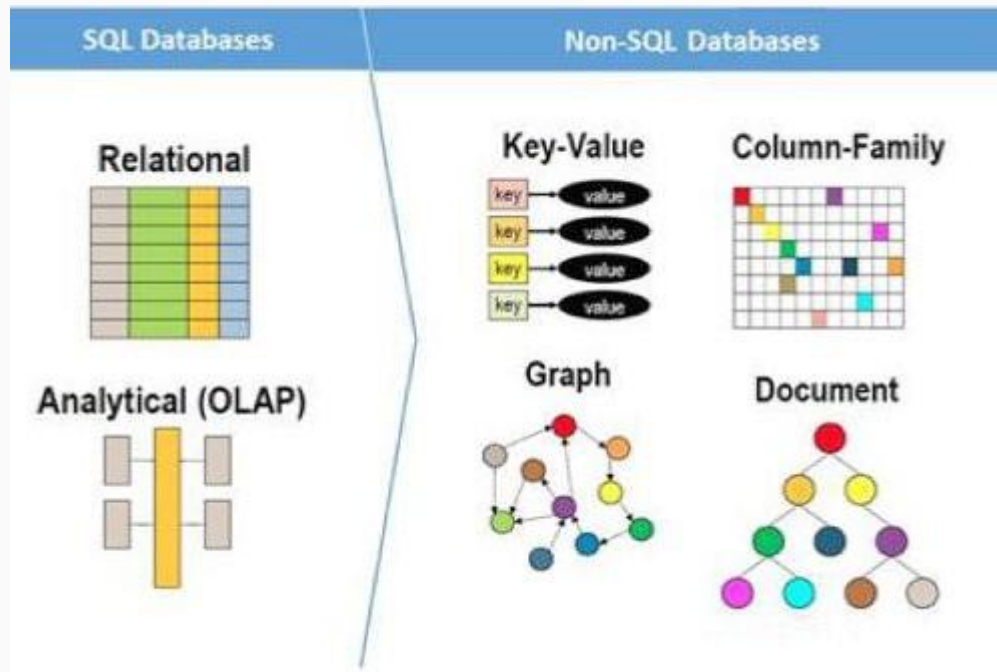
PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

One of the unique aspects of PostgreSQL is its **ability to function as both a relational and a NoSQL database**.

Advanced support for JSON objects and not only those, data types, making it a suitable choice for applications that require the flexibility of NoSQL with the reliability and integrity of a relational database.



1. JSON and JSONB Support

- Indexing: JSONB supports various indexing techniques, including GIN (Generalized Inverted Index) and B-tree, which significantly improve query performance.
- Functions and Operators: PostgreSQL offers a rich set of functions and operators to manipulate JSONB data, such as `->`, `->>`, `#>`, `#>>`, `@>`, `?`, and more.

2. Hstore

- Syntax: hstore data is represented as a series of key-value pairs.
- Indexing: Supports GIN and GiST (Generalized Search Tree) indexes for efficient querying.
- Operations: Provides functions and operators for manipulating and querying hstore data, such as `->`, `->>`, `#=`, and concatenation operators.

3. Arrays

- Syntax: Arrays can be of any data type, allowing for versatile usage.
- Operations: Provides a comprehensive set of functions and operators for array manipulation and querying, such as `array_append`, `array_prepend`, `unnest`, and more.

4. XML

- XML Functions: Provides functions to parse, query, and modify XML data, such as xpath, xmlparse, xmlserialize, and more.

5. Full-Text Search

- TSVector and TSQuery: These data types and associated functions enable efficient text search.
- Indexing: Supports GIN and GiST indexes for optimizing full-text search queries.
- Ranking: Provides tools for ranking search results based on relevance.

And many more...

Postgres: Document Store

JSON

- **Flexible data storage:** PostgreSQL's JSON type allows you to store JSON data directly in your database, preserving the text representation.
- **Hierarchical data support:** Ideal for applications that need to store hierarchical data structures.
- **Easy parsing and generation:** JSON data can be easily parsed and generated, making it straightforward to use with web APIs and applications.

JSONB

- **Binary storage:** JSONB stores JSON data in a binary format, making it more efficient for indexing and querying.
- **Performance:** Faster to process and allows more complex queries and indexing options compared to plain JSON.
- **Operators and functions:** Supports a rich set of operators and functions to query and manipulate JSONB data.

JSON Examples

- Creating a table with a JSONB field

```
CREATE TABLE json_data (data JSONB);
```

- Simple JSON data element:

```
{ "name": "Apple Phone", "type": "phone", "brand":  
  "ACME", "price": 200, "available": true,  
  "warranty_years": 1 }
```

- Inserting this data element into the table json_data

```
INSERT INTO json_data (data) VALUES  
  (' {  "name": "Apple Phone",  
        "type": "phone",  
        "brand": "ACME",  
        "price": 200,  
        "available": true,  
        "warranty_years": 1  
      } ')
```

Bridging between SQL and JSON

Simple ANSI SQL Table Definition

```
CREATE TABLE products (id integer, product_name text );
```

Select query returning standard data set

```
SELECT * FROM products;
```

```
id | product_name
----+-----
 1 | iPhone
 2 | Samsung
 3 | Nokia
```

Select query returning the same result as a JSON data set

```
SELECT ROW_TO_JSON(products) FROM products;
```

```
{"id":1,"product_name":"iPhone"}
{"id":2,"product_name":"Samsung"}
{"id":3,"product_name":"Nokia"}
```

JSON Data Types

- 1. Number:
 - Signed decimal number that may contain a fractional part and may use exponential notation.
 - No distinction between integer and floating-point
- 2. String
 - A sequence of zero or more Unicode characters.
 - Strings are delimited with double-quotation mark
 - Supports a backslash escaping syntax.
- 3. Boolean
 - Either of the values true or false.
- 4. Array
 - An ordered list of zero or more values,
 - Each values may be of any type.
 - Arrays use square bracket notation with elements being comma-separated.
- 5. Object
 - An unordered associative array (name/value pairs).
 - Objects are delimited with curly brackets
 - Commas to separate each pair
 - Each pair the colon ':' character separates the key or name from its value.
 - All keys must be strings and should be distinct from each other within that object.
- 6. null
 - An empty value, using the word null

JSON Data Type Example

```
{
  "firstName": "John",           -- String Type
  "lastName": "Smith",          -- String Type
  "isAlive": true,              -- Boolean Type
  "age": 25,                    -- Number Type
  "height_cm": 167.6,           -- Number Type
  "address": {                  -- Object Type
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [             // Object Array
    {                             // Object
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null                // Null
}
```

HSTORE Examples

- Create a table with HSTORE field

```
CREATE TABLE hstore_data (data HSTORE);
```

- Insert a record into hstore_data

```
INSERT INTO hstore_data (data) VALUES ('
    "cost"=>"500",
    "product"=>"iphone",
    "provider"=>"apple"');
```

- Select data from hstore_data

```
SELECT data FROM hstore_data ;
```

```
-----
"cost"=>"500", "product"=>"iphone", "provider"=>"Apple"
(1 row)
```



We Work With Data

Thank you!

ailton.moreira@openup.pt



PostgreSQL - Administration

Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

- List of useful tools

[Postgres Tools](#)

- Installation
 - Lab: Install Admin Tools
 - pgAdmin
 - phpPgAdmin





We Work With Data

Thank you!

ailton.moreira@openup.pt

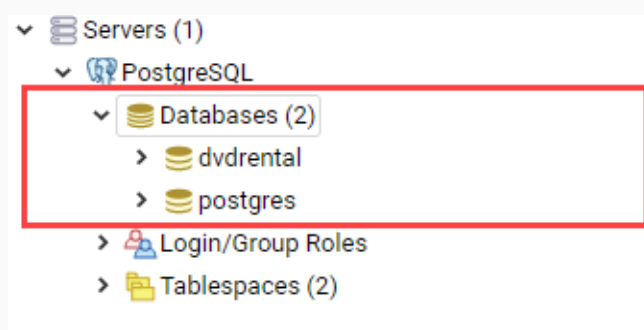


PostgreSQL - Administration

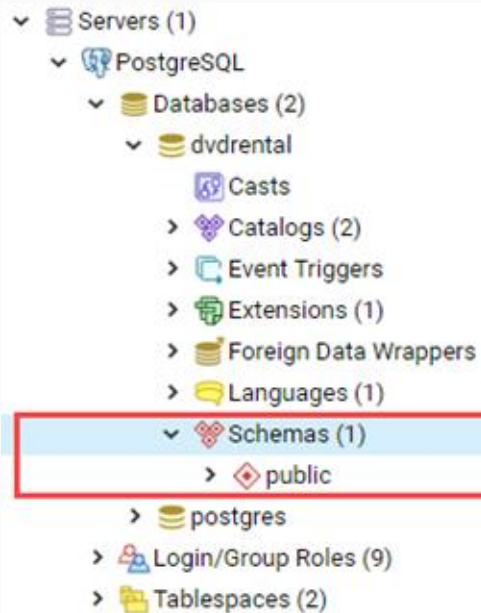
Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

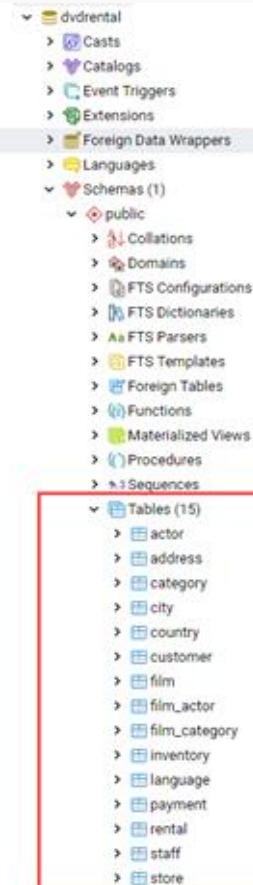
- Every instance of a running PostgreSQL server manages one or more databases. Databases are therefore the topmost hierarchical level for organizing SQL objects (“database objects”).
- A **database** is a container of other objects such as tables, views, functions, and indexes. You can create as many databases as you want inside a PostgreSQL server.



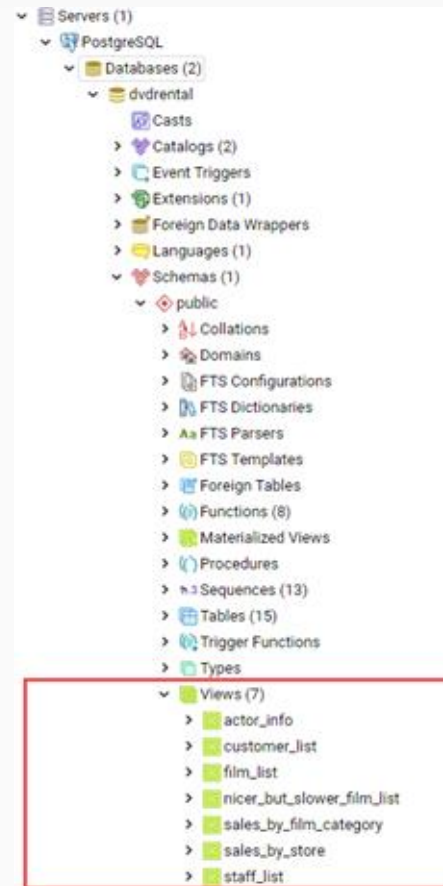
- A **schema** is a logical container of tables and other objects inside a database. Each PostgreSQL database may have multiple schemas.



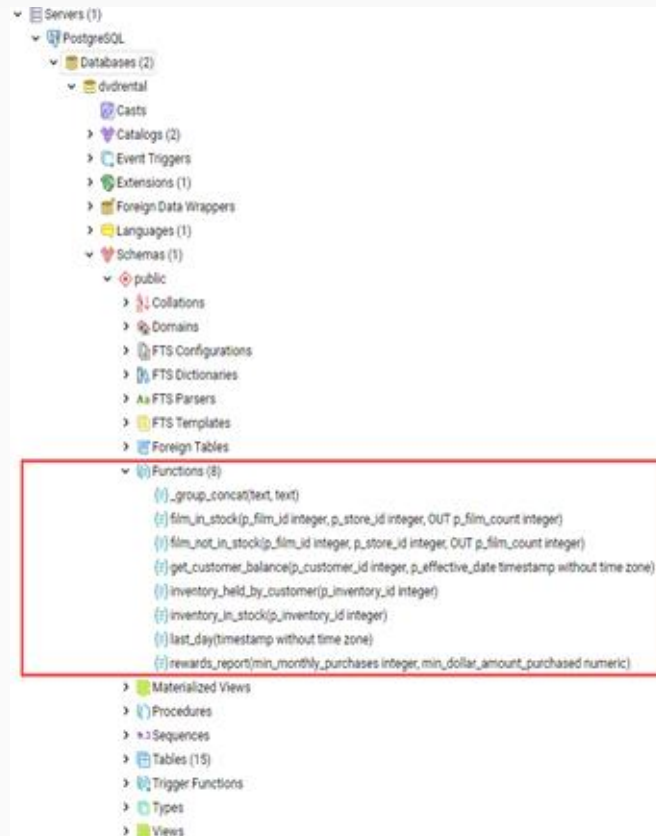
- **Tables** store data. A table belongs to a database and each database has multiple tables.



- **Views** are named query stored in the database. Besides the read-only views, PostgreSQL support updatable views.



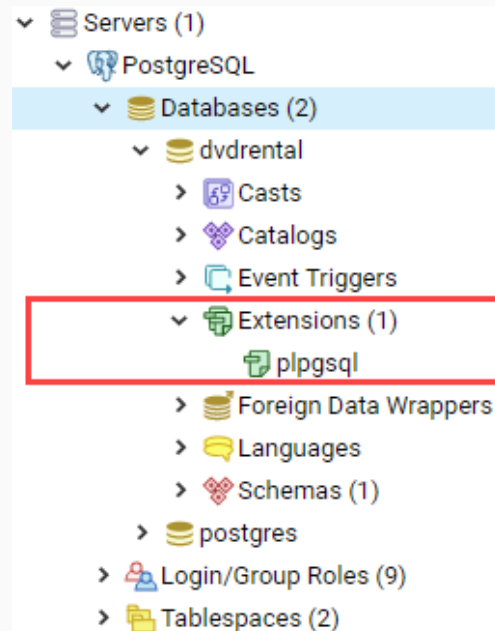
- A **function** is a reusable block of SQL code that returns a scalar value of a set of rows.



- **Sequences** are used to manage auto-increment columns that defined in a table as a serial column.



- PostgreSQL introduced **extension** concept since version 9.1 to wrap other objects including types, casts, indexes, functions, etc., into a single unit. The purpose of extensions is to make it easier to maintain.



- Identify Database Objects
 - [7-Use Admin Tools to Identify Database Objects](#)
- Managing Databases
 - [8-Create, Alter, Get Info, Delete Databases](#)





We Work With Data

Thank you!

ailton.moreira@openup.pt

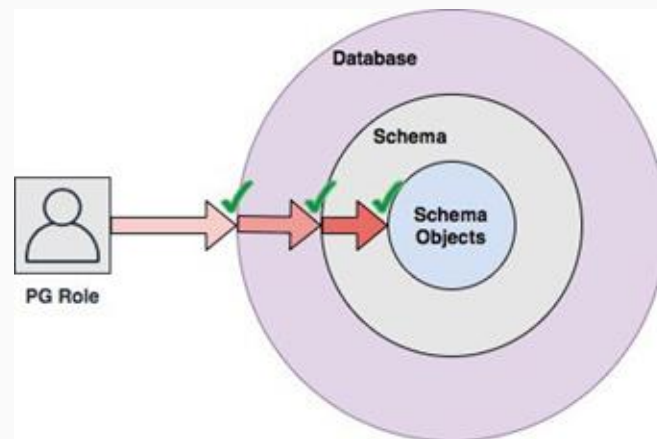


PostgreSQL - Administration

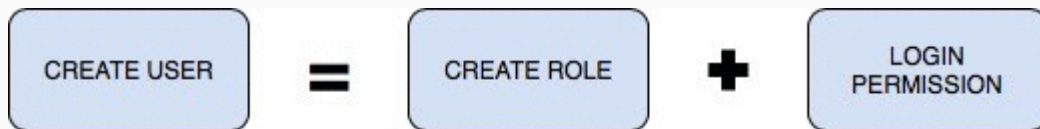
Ailton Moreira
ailton.moreira@openup.pt

VN Gaia 2024

- PostgreSQL uses the roles concept to manage database access permissions.
- A **role** can be a **user** or a **group**, depending on how you setup the role.
- A role that has login right is called user.
- A role may be a member of other roles, which are known as groups.



- **Users, groups, and roles** are the same thing in PostgreSQL, with the only difference being that users have permission to log in by default.
- The CREATE USER and CREATE GROUP statements are actually aliases for the CREATE ROLE statement.



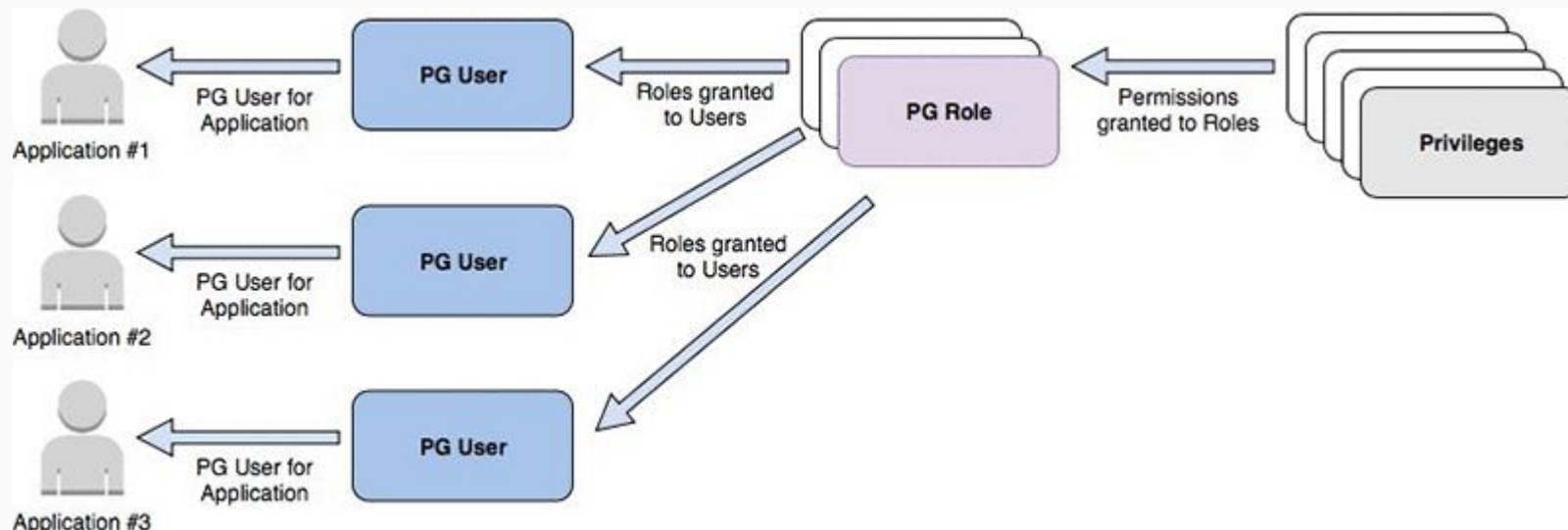
- Managing PostgreSQL users and roles
 - [9- Manage Users or Roles](#)



- PostgreSQL lets you grant permissions directly to the database users. However, as a good practice, it is recommended that you create multiple roles with specific sets of permissions based on application and access requirements. Then assign the appropriate role to each user.
- The roles should be used to enforce a **least privilege model** for accessing database objects.

- The recommended approach for setting up fine-grained access control in PostgreSQL is as follows:
 - Use the master user to create roles per application or use case, like **readonly** and **readwrite**.
 - Add permissions to allow these roles to access various database objects. For example, the **readonly** role can only run **SELECT** queries.
 - Grant the roles the least possible permissions required for the functionality.
 - Create new users for each application or distinct functionality, like **app_user** and **reporting_user**.
 - Assign the applicable roles to these users to quickly grant them the same permissions as the role. For example, grant the **readwrite** role to **app_user** and grant the **readonly** role to **reporting_user**.
 - At any time, you can remove the role from the user in order to revoke the permissions

- The recommended approach for setting up fine-grained access control in PostgreSQL is as follows:





We Work With Data

Thank you!

ailton.moreira@openup.pt