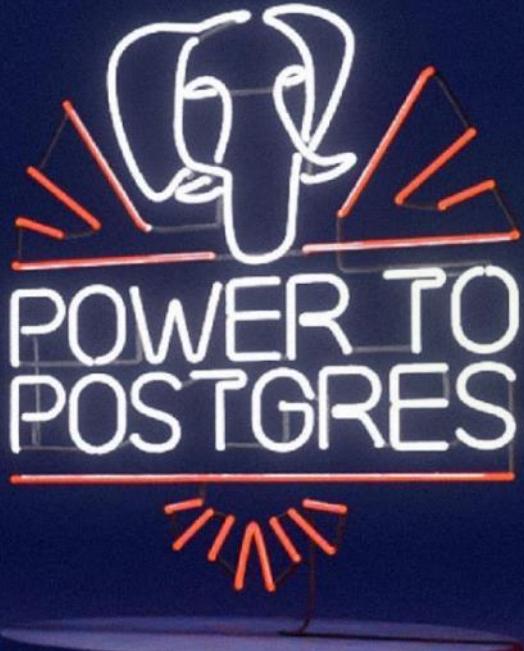


EPAS Essentials



Course Agenda

- Introduction and Architectural Overview
- System Architecture
- EDB Postgres Advanced Server Installation
- User Tools - Command Line Interfaces
- Database Clusters
- Database Configuration
- Data Dictionary
- Creating and Managing Database Objects
- Database Security
- Monitoring and Admin Tools Overview
- SQL Primer
- Backup and Recovery
- Routine Maintenance Tasks
- Data Loading
- Data Replication and High Availability



Module - 1

Introduction

Module Objectives

- EDB Portfolio
- Facts about PostgreSQL and EDB Postgres Advanced Server
- Major Features
- EDB Postgres Advanced Server Database Features
- General Database Limits
- Common Database Object Names



EDB Supported Databases



Postgres

Open source Postgres

- EDB continues to be committed to advancing features in collaboration with the broader community



Postgres Extended

EDB proprietary distribution for EDB Postgres Distributed use cases with Transparent Data Encryption



Postgres Advanced Server

EDB proprietary distribution with Transparent Data Encryption

- SQL compatible with Postgres, extended for stringent availability and advanced replication needs
- Transparent Data Encryption
- Formerly known as 2ndQPostgres

- SQL compatible with Oracle, reduces effort to migrate applications and data to Postgres
- Transparent Data Encryption
- Additional value-add enterprise features

PostgreSQL

The open source database of choice



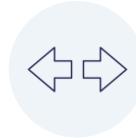
Performance

Handles enterprise workloads with 50% improvement in the last 4 years



Scalability

Multiple technical options for operating Postgres at scale



Extensibility

Supported by a wide array of extensions plus multiple SQL and NoSQL data models



Community-driven

Multiple companies and individuals contribute to the project and drive innovation



Facts about PostgreSQL

- The world's most advanced open source database
- Designed for extensibility and customization
- ANSI/ISO compliant SQL support
- Actively developed for more than 20 years
 - University Postgres (1986-1993)
 - Postgres95 (1994-1995)
 - PostgreSQL (1996-current)

PostgreSQL



EDB Postgres Extended Server



- **Replication Enhancements**

- Enables EDB Postgres Distributed functionality such as:
 - Group Commit, Commit at Most Once, and Eager all-node synchronous replication
 - Timestamp-based Snapshots
 - Estimates for Replication Catch-up times
 - Selective Backup of a Single Database
 - Hold back freezing to assist resolution of UPDATE/DELETE conflicts
 - Multi-node PITR
 - Application Assessment

- Only available for use with an additional subscription for Extreme HA



EDB Postgres Advanced Server



EDB Postgres Advanced Server

- **Oracle Compatibility** - Compatibility for schemas, data types, indexes, users, roles, partitioning, packages, views, PL/SQL triggers, stored procedures, functions, and utilities
- **Additional Security** - Password policy management, session tag auditing, data redaction, SQL injection protection, and procedural language code obfuscation
- **Developer Productivity** - Over 200 pre-packaged utility functions, user-defined object types, autonomous transactions, nested tables, synonyms, advanced queueing
- **DBA Productivity** - Throttle CPU and I/O at the process level, over 55 extended catalog views to profile all the objects and processing that occurs in the database
- **Performance** - Query optimizer hints, SQL session/system wait diagnostics
- **Replication Enhancements** - Enables EDB Postgres Distributed functionality such as Group Commit, Commit at Most Once and Eager all-node synchronous replication, timestamp-based snapshots, estimates for replication catch-up times, selective backup of a single database, hold back freezing to assist resolution of UPDATE/DELETE conflicts, multi-node PITR



Database Servers - High Level Overview

Database Server	PostgreSQL	EDB Postgres Extended Server	EDB Postgres Advanced Server: Berkeley	EDB Postgres Advanced Server: Redwood
SQL Compatibility	PostgreSQL	PostgreSQL	PostgreSQL +	Oracle
Binary Compatibility	Yes	No	No	No
Advanced PGD Features		✓	14+	14+
Transparent Data Encryption		15+	15+	15+
Advanced Security			✓	✓
Advanced SQL			✓	✓
Advanced Performance			✓	✓
Resource Manager			✓	✓
Bulk Data Loader			✓	✓
Oracle Compatibility				✓



Capabilities And Tools



Management/Monitoring

Postgres Enterprise Manager
pgAdmin



High Availability

EDB Postgres Distributed
Failover Manager
Repmgr
Patroni



Backup and Recovery

Barman
pgBackRest



Migration

Migration Portal
Migration Toolkit
Replication Server



Integration

Connectors
Foreign Data Wrappers
Connection Poolers



Kubernetes

EDB Postgres for Kubernetes
CloudNativePG



Major Features

- **Portable:**
 - Written in ANSI C
 - Supports Windows, Linux, Mac OS/X and major UNIX platforms
- **Reliable:**
 - ACID Compliant
 - Supports Transactions and Savepoints
 - Uses Write Ahead Logging (WAL)
- **Scalable:**
 - Uses Multi-version Concurrency Control
 - Table Partitioning and Tablespaces
 - Parallel Sequential Scans, DDL(Table and Index Creation)



Major Features(continued)

- **Secure:**
 - Employs Host-Based Access Control
 - Provides Object-Level Permissions and Row Level Security
 - Supports SSL Connections and Logging
 - Transparent Data Encryption - TDE
- **Recovery and Availability:**
 - Streaming Replication, Logical Replication and Replication Slots
 - Replication Slots, Sync or Async Options
 - Supports Hot-Backup, pg_basebackup and Point-in-Time Recovery
- **Advanced:**
 - Supports Triggers, Functions and Procedures
 - Supports Custom Procedural Languages
 - Upgrade using pg_upgrade
 - Unlogged Tables and Materialized Views
 - Just-in-Time (JIT) Compilation



Postgres for Big Data

- Postgres enables you to support a wider range of workloads with your relational database
 - An Object-relational design and decades of proven reliability make Postgres the most flexible, extensible and performant database available
 - Document store capabilities: XML, JSON, PLV8; HStore (key-value store); non-durable storage; full text indexing
 - Foreign Data Wrappers enable read/write integration with other database technologies



General Database Limits

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250-1600 (Depending on Column types)
Maximum Indexes per Table	Unlimited



Common Database Object Names

Industry Term	Postgres Term
Table or Index	Relation
Row	Tuple
Column	Attribute
Data Block	Page (when block is on disk)
Page	Buffer (when block is in memory)



Lab Setup Guidelines

- All the instructor demos and labs are based on Linux
- CentOS 7 machine or virtual machine with at least 1 GB RAM and 10 GB storage space is recommended
- Participants using Linux must follow instructor during the installation module and install EDB Postgres Advanced Server



Module Summary

- EDB Portfolio
- Facts about PostgreSQL and EDB Postgres Advanced Server
- Major Features
- EDB Postgres Advanced Server Database Features
- General Database Limits
- Common Database Object Names



Module - 2

System Architecture

Module Objectives

- Architectural Summary
- Process and Memory Architecture
- Utility Processes
- Connection Request-Response
- Disk Read Buffering
- Disk Write Buffering
- Background Writer Cleaning Scan
- Commit and Checkpoint
- Statement Processing
- Physical Database Architecture
- Data Directory Layout
- Installation Directory Layout
- Page Layout

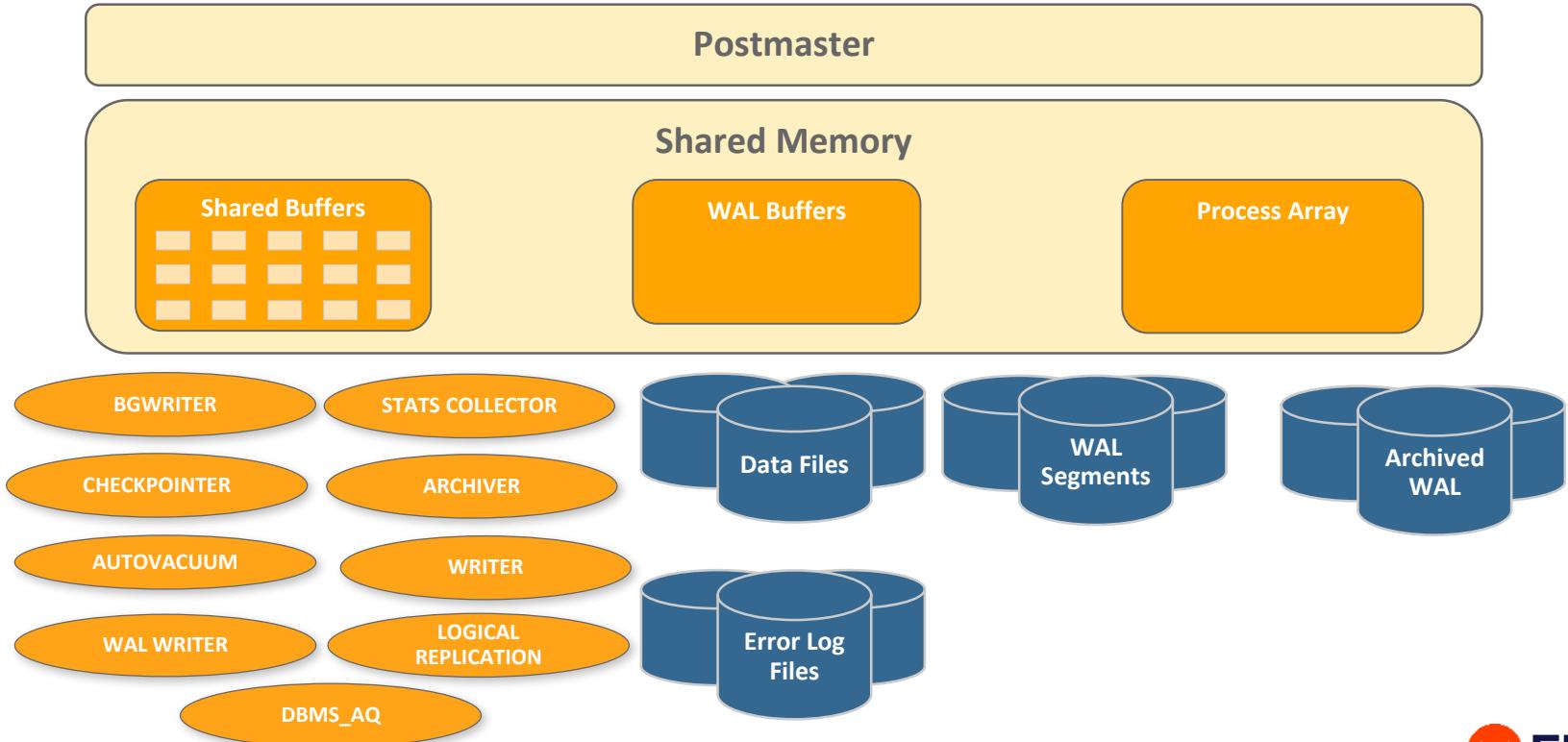


Architectural Summary

- EDB Postgres Advanced Server uses processes, not threads
- The postmaster process acts as a supervisor
- Several utility processes perform background work
 - postmaster starts them, restarts them if they die
- One backend process per user session
 - postmaster listens for new connections



Process and Memory Architecture



Utility Processes

- **Background writer**
 - Writes dirty data blocks to disk
- **WAL writer**
 - Flushes write-ahead log to disk
- **Checkpointer**
 - Automatically performs a checkpoint based on config parameters
- **Autovacuum launcher**
 - Starts Autovacuum workers as needed
- **Autovacuum workers**
 - Recover free space for reuse



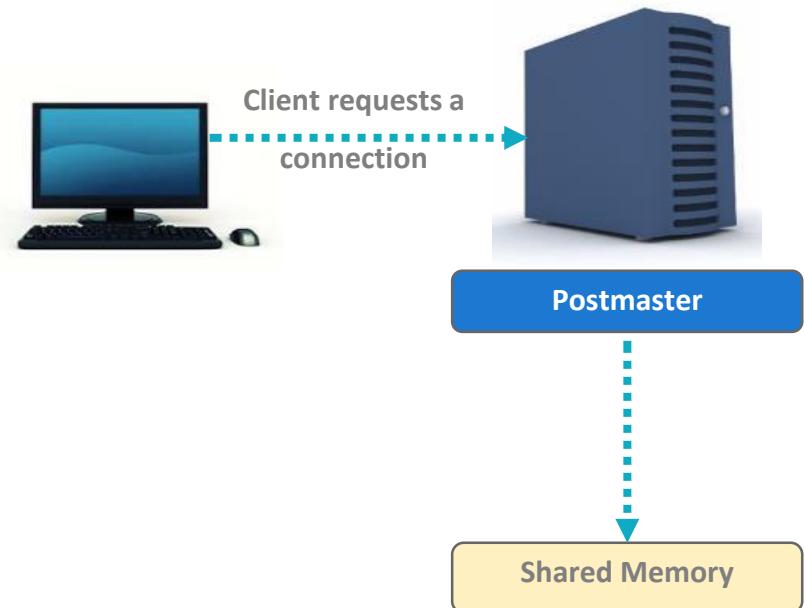
More Utility Process

- **Logging collector**
 - Routes log messages to syslog, eventlog, or log files
- **Stats collector**
 - Collects usage statistics by relation and block
- **Archiver**
 - Archives write-ahead log files
- **Logical replication launcher**
 - Starts logical replication apply process for logical replication
- **Dbms_aq launcher**
 - Collects information for queueing functionality of advanced server



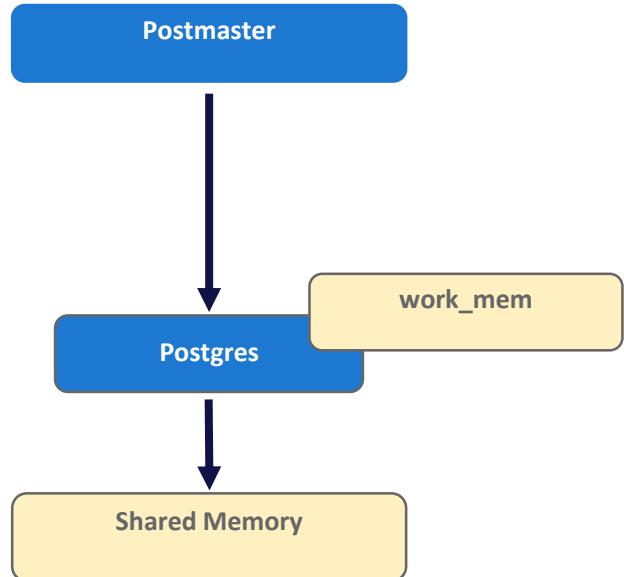
Postmaster as Listener

- Postmaster is the main process called `postgres`
- Listens on 1, and only 1, tcp port
- Receives client connection requests



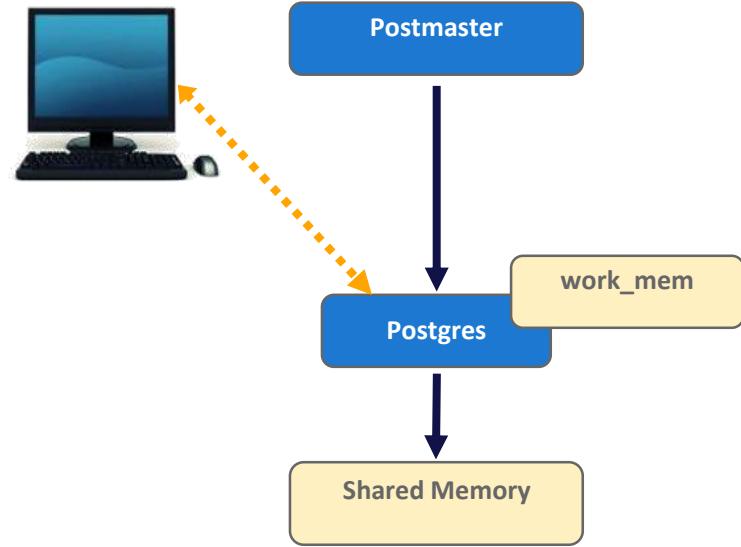
User Backend Process

- Postmaster process spawns a new server process for each connection request detected
- Communication is done using semaphores and shared memory
- Authentication - IP, user and password
- Authorization - Verify permissions



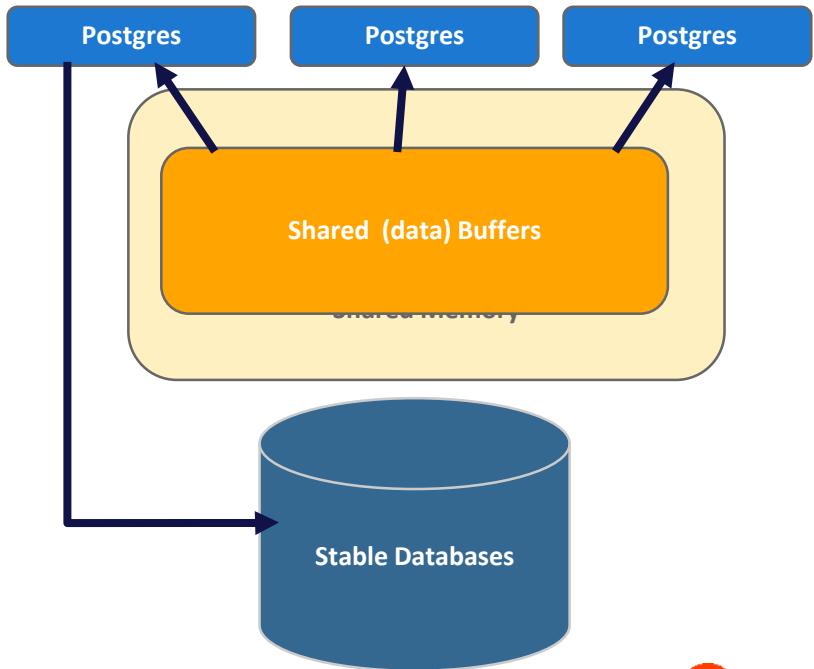
Respond to Client

- User backend process called postgres
- Callback to client
- Waits for SQL
- Query is transmitted using plain text



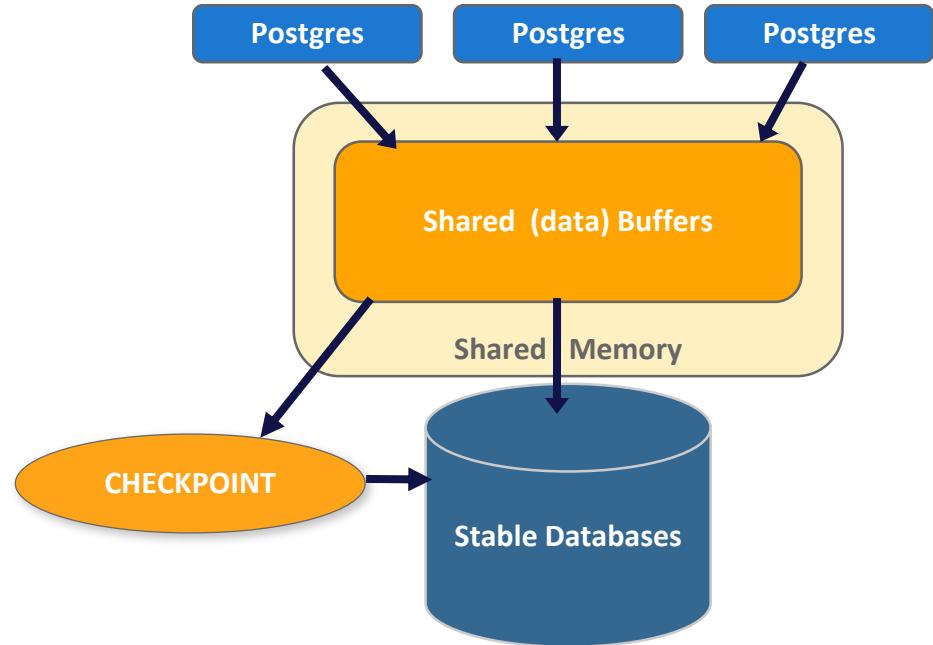
Disk Read Buffering

- EDB Postgres buffer cache (`shared_buffers`) reduces OS reads
- Read the block once, then examine it many times in cache



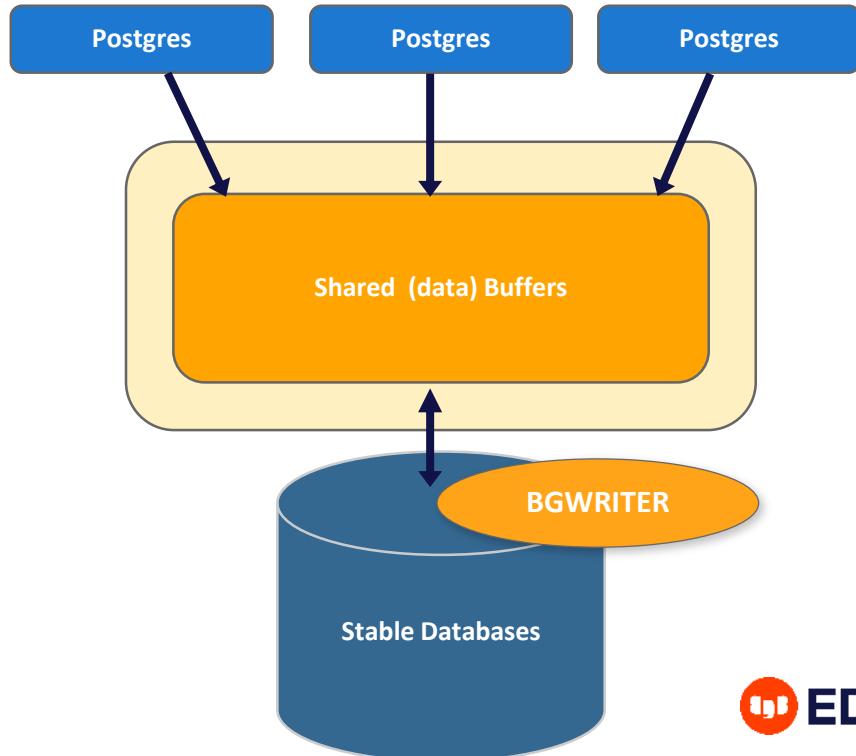
Disk Write Buffering

- Blocks are written to disk only when needed:
 - To make room for new blocks
 - At checkpoint time



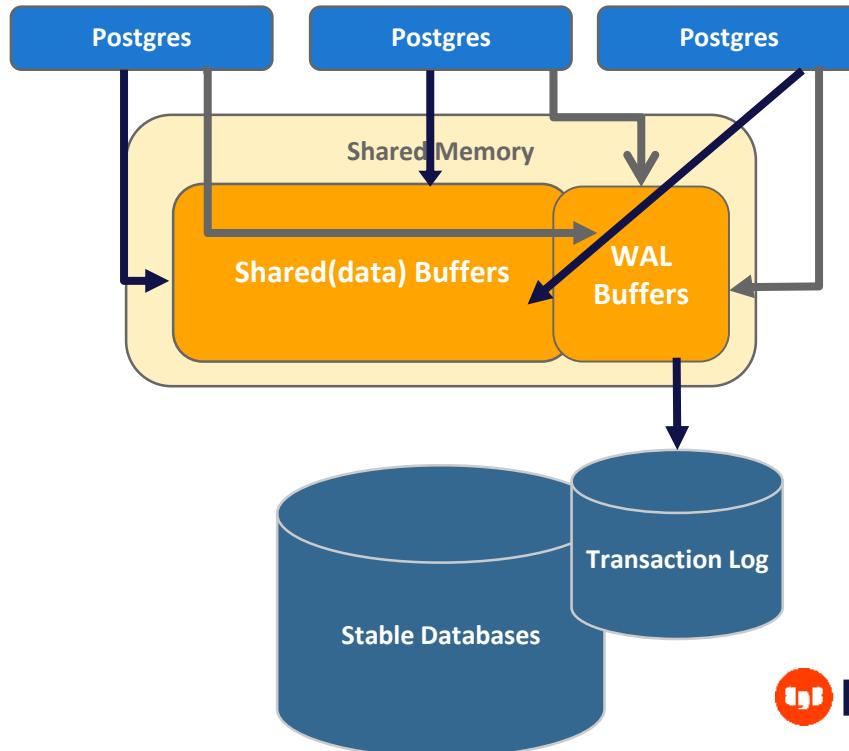
Background Writer Cleaning Scan

- Background writer scan attempts to ensure an adequate supply of clean buffers
- Back end write dirty buffers as need



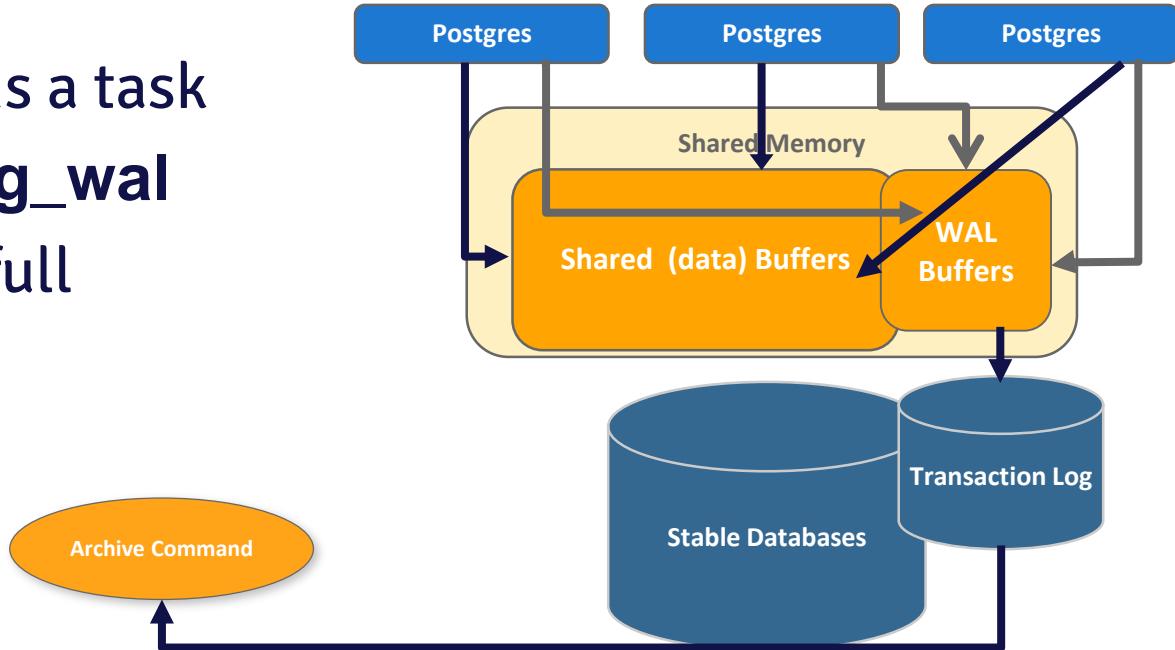
Write Ahead Logging (WAL)

- Back end write data to WAL buffers
- Flush WAL buffers periodically (WAL writer), on commit, or when buffers are full
- Group commit



Transaction Log Archiving

- Archiver spawns a task to copy away **pg_wal** log files when full

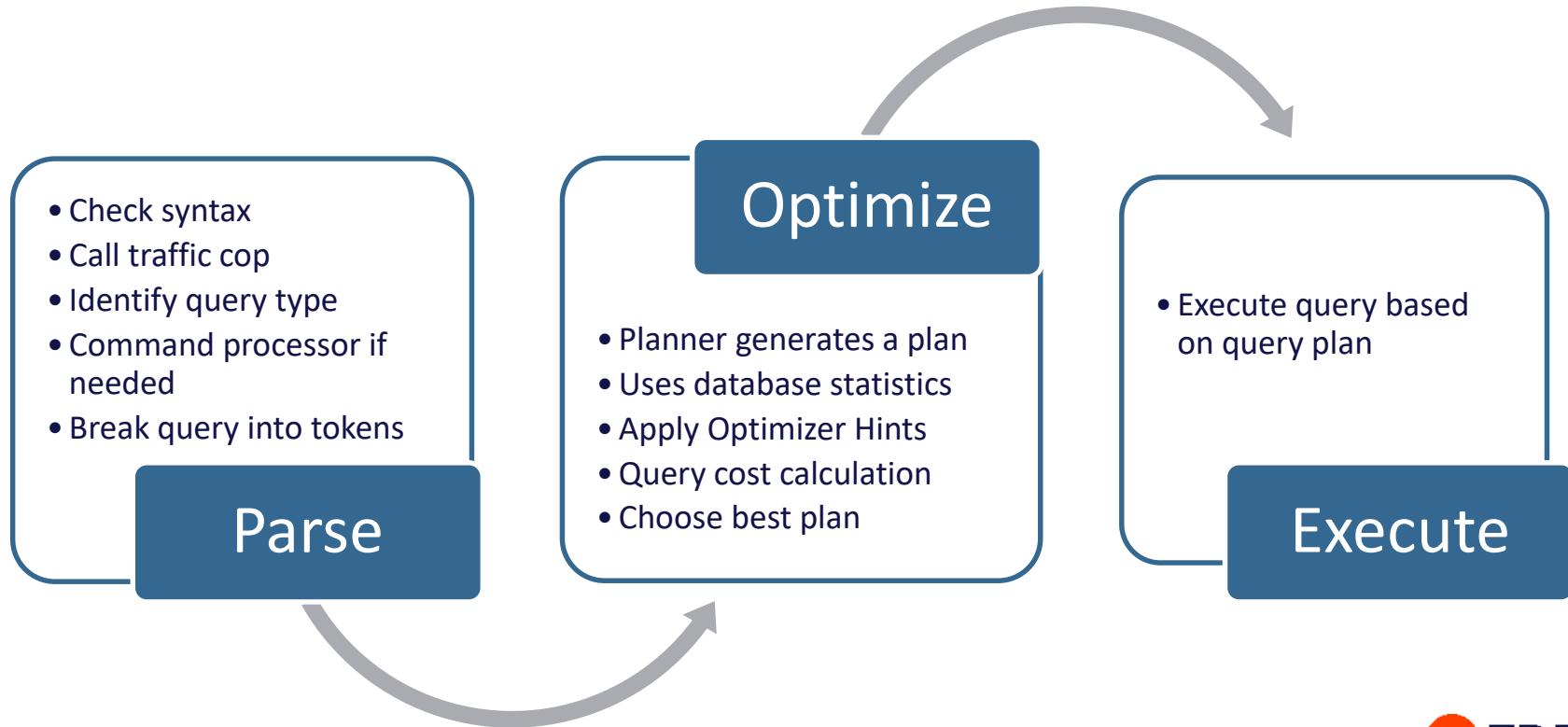


Commit and Checkpoint

- Before commit
 - Uncommitted updates are in memory
- After commit
 - WAL buffers are written to the disk (write-ahead log file) and shared buffers are marked as committed
- After checkpoint
 - Modified data pages are written from shared memory to the data files



Statement Processing



Physical Database Architecture

- Database cluster is a collection of databases managed by single server instance
- Each cluster has a separate
 - Data directory
 - TCP port
 - Set of processes
- A cluster can contain multiple databases

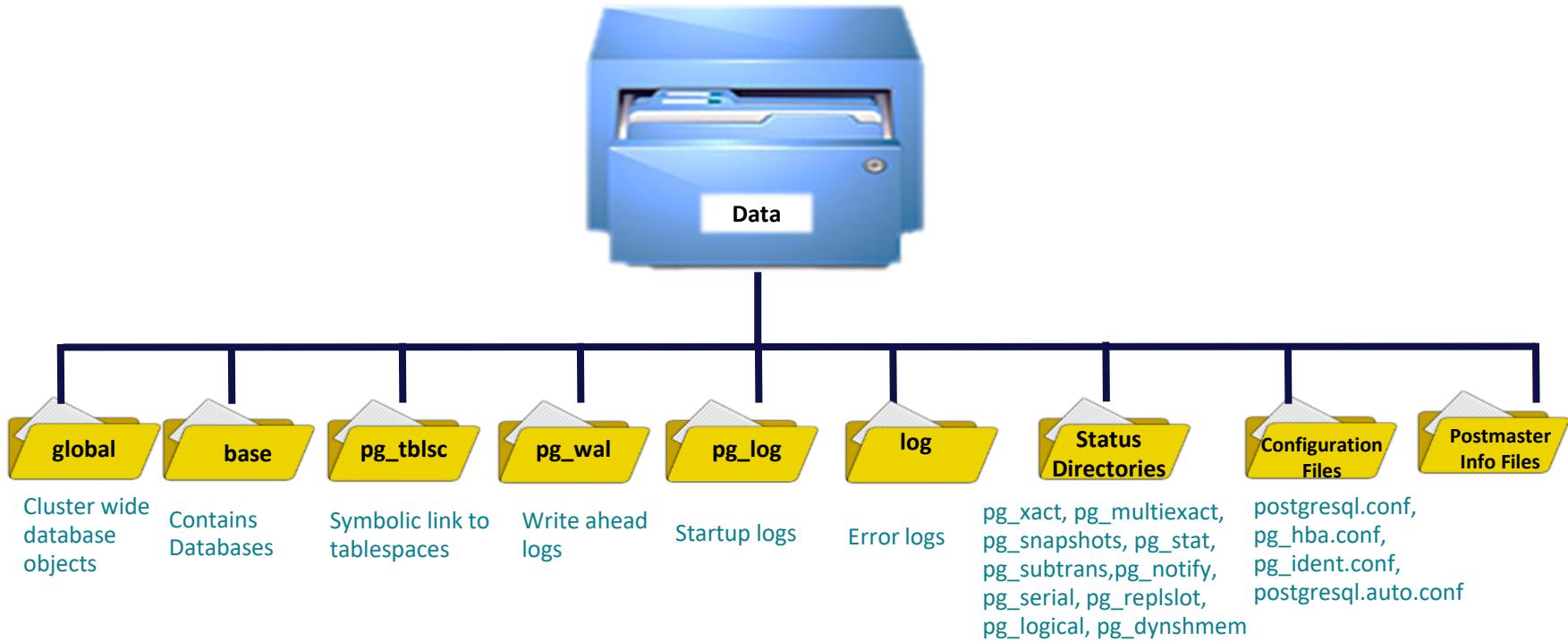


Installation Directory Layout

- **Default Installation Directory Location:**
 - **Linux** – /usr/edb/as15
 - **Windows** – C:\Program Files\edb\as15
- **bin** – Programs
- **share** – Shared data
- **include** – Header files
- **lib or lib64** – Libraries



Database Cluster Data Directory Layout

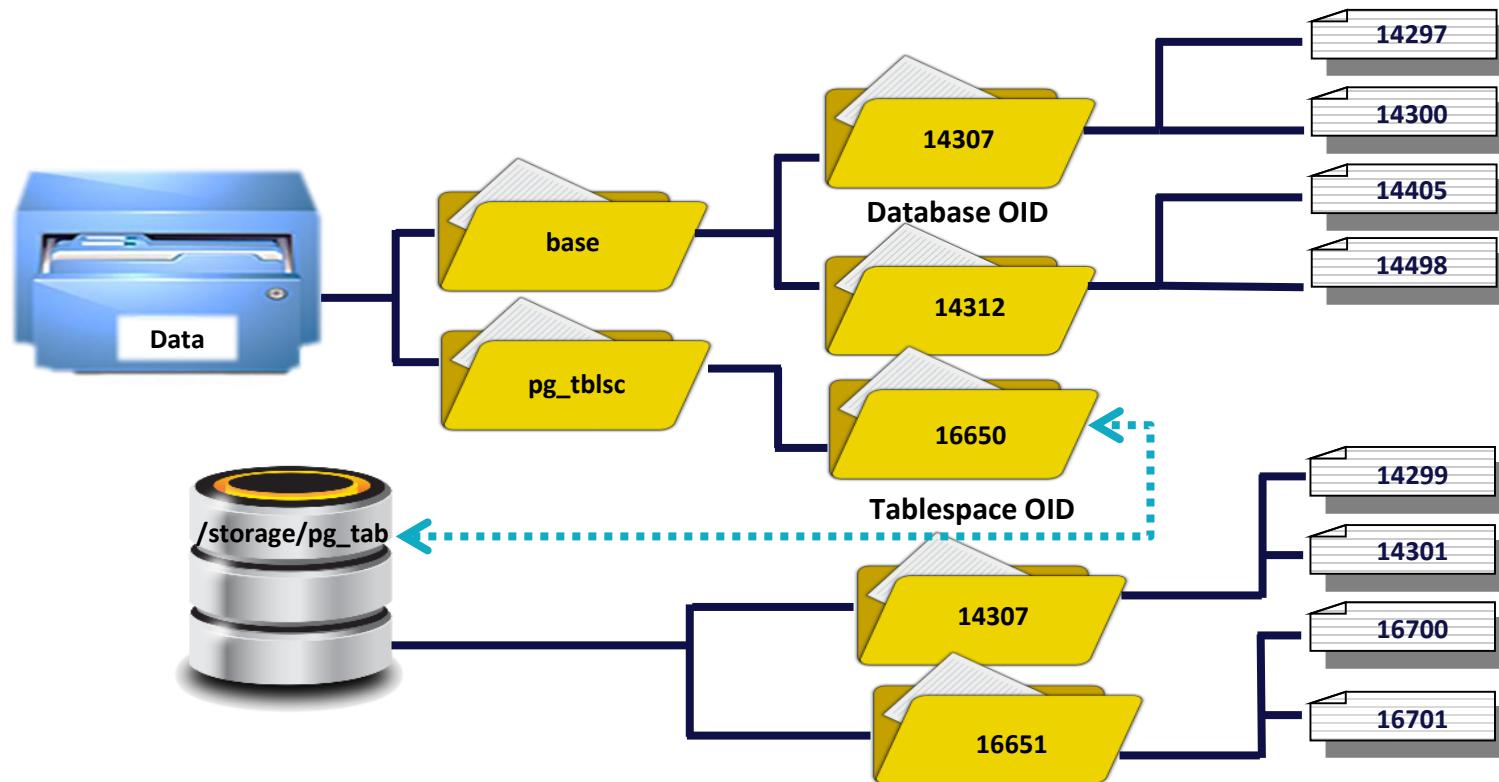


Physical Database Architecture

- File-per-table, file-per-index
- A table-space is a directory
- Each database that uses that table-space gets a subdirectory
- Each relation using that table-space/database combination gets one or more files, in 1GB chunks
- Additional files are used to hold auxiliary information (free space map, visibility map)
- Each file name is a number (see **pg_class.relfilenode**)



Sample - Data Directory Layout

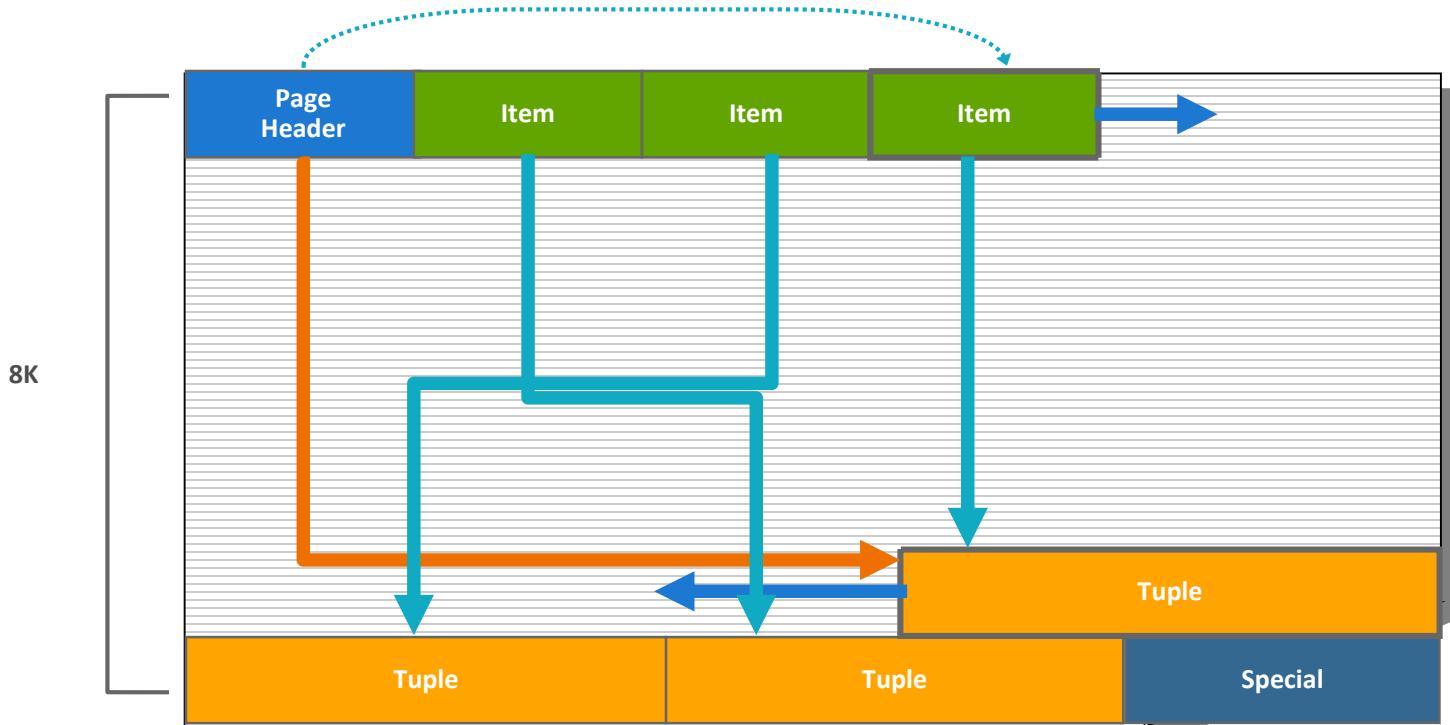


Page Layout

- **Page header**
 - General information about the page
 - Pointers to free space
 - 24 bytes long
- **Row/index pointers**
 - Array of offset/length pairs pointing to the actual rows/index entries
 - 4 bytes per item
- **Free space**
 - Unallocated space
 - New pointers allocated from the front, new rows/index entries from the rear
- **Row/index entry**
 - The actual row or index entry data
- **Special**
 - Index access method specific data
 - Empty in ordinary tables



Page Structure



Module Summary

- Architectural Summary
- Shared Memory
- Inter-processes Communication
- Statement Processing
- Utility Processes
- Disk Read Buffering
- Disk Write Buffering
- Background Writer Cleaning Scan
- Commit and Checkpoint
- Physical Database Architecture
- Data Directory Layout
- Installation Directory Layout
- Page Layout



Module - 3

EDB Postgres Advanced Server Installation

Module Objectives

- Deployment Options
- OS User and Permissions
- Package Installation
- Installation of EDB Postgres Advanced Server
- Setting Environmental Variables



Deployment Options

- Deployment methods for EDB Postgres Advanced Server and supported Tools:
 - **BigAnimal:** Fully managed database-as-a-service with built-in Oracle compatibility.
 - **EDB PostgreSQL for Kubernetes:** Operator designed for managing PostgreSQL workloads on Kubernetes clusters.
 - **Native packages or installers:** EDB Repository can be used for YUM and RPM based installation



OS User and Permissions

- EDB Postgres Advanced Server runs as a daemon (Unix / Linux) or service (Windows)
- The EDB Postgres Advanced Server Installation requires superuser/admin access
- All processes and data files must be owned by a user in the OS
- During installation an `enterprisedb` locked user will be created on Linux
- On Windows a password is required
- SELinux must be set to permissive mode on systems with SELinux



The enterpriseDB User Account

- It is advised to run EDB Postgres Advanced Server under a separate user account
- This user account should only own the data directory that is managed by the server
- The `useradd` or `adduser` Unix command can be used to add a user
- The user account named `enterpriseDB` is used throughout this training

```
[root@Base ~]# useradd enterpriseDB
[root@Base ~]# passwd enterpriseDB
Changing password for user enterpriseDB.
New password:
Retype new password:
passwd:
all authentication tokens updated successfully.
```



Package Installation Options

Wizard Installer

- Interactive Method
- Graphical or Command Line Mode, available for Windows
- Easy Download from www.enterprisedb.com

RPM Installer

- Preferred Installation Method on Linux
- Access to EnterpriseDB's rpm Repository is required
- Dependencies are resolved manually

YUM Installer

- Attempt to Install required package dependencies
- Can be used to install EDB Postgres in Isolated Environments





YUM Installation

Configure EDB Repositories

- An EDB account is required to access our software repositories and downloads
- Setup EDB Repositories: <https://www.enterprisedb.com/repos-downloads>
- Ensure EPEL repositories are setup:
`sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm`
- Check Instructions for various operating systems:
<https://www.enterprisedb.com/docs/epas/latest/installing/>



YUM Installation Overview

- YUM command can be used to install EDB Postgres Advanced Server:
`# yum install epel-release edb-as15-server`
- Configure a Package Installation using service configuration file
`# /usr/lib/systemd/system/edb-as-15.service`
- Create a database cluster and start the cluster using services:
`# /usr/edb/as15/bin/edb-as-15-setup initdb`
`# systemctl start edb-as-15`



Example – EDB Postgres Server Installation

- Step 1 – Login as root user and add user enterpriseDb using adduser or useradd command and set its password:

```
# useradd enterpriseDb  
# passwd enterpriseDb
```

- Step 2 – Install EPEL using yum:

```
# yum install epel-release
```

- Step 3 – Configure EDB Repos 2.0 using your secure token:

```
# curl -1sLf 'https://downloads.enterprisedb.com/<secure  
token>/enterprise/setup.rpm.sh' | sudo -E bash
```



Example – EDB Postgres Advanced Server Installation (continued)

- Step 4 – Install EDB Postgres Advanced server using yum command

```
# yum install edb-as15-server
```

- Step 5 - Create a database cluster and start the cluster using services:

```
# /usr/edb/as15/bin/edb-as-15-setup initdb  
# systemctl start edb-as-15  
# systemctl enable edb-as-15
```





After Installation

Example – Environmental Variables setup

```
[enterprisedb@pgsrv1 ~]$ vi .bash_profile
```

Edit User Profile

```
PATH=/usr/edb/as15/bin/:$PATH:$HOME/.local/bin:$HOME/bin
```

```
export PATH  
export PGDATA=/var/lib/edb/as15/data/  
export PGUSER=enterprisedb  
export PGPORT=5444  
export PGDATABASE=enterprisedb
```

Logoff and Login

```
[enterprisedb@pgsrv1 ~]$ exit
```

```
logout
```

```
[root@pgsrv1 ~]# su - enterprisedb
```

```
[enterprisedb@pgsrv1 ~]$ which psql
```

```
/usr/edb/as15/bin/psql
```

Verify
Environmental
Settings

```
[enterprisedb@pgsrv1 ~]$ pg_ctl status
```

```
pg_ctl: server is running (PID: 66406)
```

```
/usr/edb/as15/bin/edb-postgres "-D" "/var/lib/edb/as15/data"
```



Module Summary

- OS User and Permissions
- Installation Options
- Installation of EDB Postgres Advanced Server
- Setting Environmental Variables



Lab Exercise - 1

- Choose the platform on which you want to install EDB Postgres Advanced Server
- Download the EDB Postgres Advanced Server installer from the EnterpriseDB website for the chosen platform
- Prepare the platform for installation
- Install EDB Postgres Advanced Server
- Connect to EDB Postgres Advanced Server using psql



Module - 4

User Tools - Command Line Interfaces

Module Objectives

- Introduction to psql
- Connecting to Database
- psql Command Line Parameters
- psql Meta-Commands
- Conditional and Information Commands
- EDB*Plus
- Installing and Starting EDB*Plus
- EDB*Plus Commands





psql(edb-psql)

Introduction to psql

- **psql** is a command line interface (CLI) to Postgres
- Can be used to execute SQL queries and **psql** meta commands

```
[enterprisedb@Base ~]$ psql -h /tmp -p 5444 -U enterprisedb -d edb
Type "help" for help.
edb=# \q
```



Connecting to a Database

psql Connection Options:

- -d <Database Name>
- -h <Hostname>
- -p <Database Port>
- -U <Database Username>

Environmental Variables

- PGDATABASE, PGHOST, PGPORT and PGUSER



Conventions

- **psql** has its own set of commands, all of which start with a backslash (\).
- Some commands accept a pattern. This pattern is a modified regex. Key points:
 - * and ? are wildcards
 - Double-quotes are used to specify an exact name, ignoring all special characters and preserving case



On Startup...

- **psql** will execute commands from \$HOME/.psqlrc, unless option **-X** is specified
- **-f FILENAME** will execute the commands in FILENAME, then exit
- **-c COMMAND** will execute COMMAND (SQL or internal) and then exit
- **--help** will display all the startup options, then exit
- **--version** will display version info and then exit



Entering Commands

- psql uses the command line editing capabilities that are available in the native OS. Generally, this means:
 - Up and Down arrows cycle through command history
 - On UNIX, there is tab completion for various things, such as SQL commands



History and Query Buffer

- `\s` will show the command history
- `\s FILENAME` will save the command history
- `\e` will edit the query buffer and then execute it
- `\e FILENAME` will edit FILENAME and then execute it
- `\w FILENAME` will save the query buffer to FILENAME



Controlling Output

- `psql -o FILENAME` or meta command `\o FILENAME` will send query output (excluding STDERR) to FILENAME
- `\g FILENAME` executes the query buffer sending output to FILENAME
- `\watch <seconds>` can be used to run previous query repeatedly



Advanced Features - Variables

- **psql** provides variable substitution
- Variables are simply name/value pairs
- Use `\set` meta command to set a variable

```
=> \set city Edmonton
```

```
=> \echo :city
```

Edmonton

- Use `\unset` to delete a variable

```
=> \unset city
```



Advanced Features - Special Variables

- Settings can be changed at runtime by altering special variables
- Some important special variables include:
 - AUTOCOMMIT, ENCODING, HISTFILE, ON_ERROR_ROLLBACK, ON_ERROR_STOP, PROMPT1 and VERBOSITY
- Example:

```
=# \set AUTOCOMMIT off
```

- Once AUTOCOMMIT is set to off use COMMIT/ROLLBACK to complete the running transaction



Conditional Commands

- Conditional commands primarily helpful for scripting
 - \if EXPR begin conditional block
 - \elif EXPR alternative within current conditional block
 - \else final alternative within current conditional block
 - \endif end conditional block



Information Commands

- `\d[(i|s|t|v|b|S)] [+] [pattern]`
 - List of objects (indexes, sequences, tables, views, tablespaces and dictionaries)
- `\d[+] [pattern]`
 - Describe structure details of an object
- `\l [ist] [+]`
 - Lists of databases in a database cluster



Information Commands (continued)

- `\dn+ [pattern]`
 - Lists schemas (namespaces)
 - + adds permissions and description to output
- `\df [+] [pattern]`
 - Lists functions
 - + adds owner, language, source code and description to output



Common psql Meta Commands

- `\q` or `^d` or `quit` or `exit`
 - Quits the psql program
- `\cd [directory]`
 - Change current working directory
 - Tip - To print your current working directory, use `\! pwd`
- `\! [command]`
 - Executes the specified command
 - If no command is specified, escapes to a separate Unix shell (CMD.EXE in Windows)



Help

- `\conninfo`
 - Current connection information
- `\?`
 - Shows help information about psql commands
- `\h [command]`
 - Shows information about SQL commands
 - If command isn't specified, lists all SQL commands
- `psql --help`
 - Lists command line options for psql





EDB*Plus

EDB*Plus

- EDB*Plus is a command line user interface to the EDB Postgres Advanced Server
- EDB*Plus accepts SQL commands, SPL anonymous blocks, and EDB*Plus commands
- EDB*Plus commands are compatible with Oracle SQL*Plus commands
- `edb-asXX-edbplus` package is available to install EDB*Plus using yum command and requires java runtime



EDB*Plus Features

- EDB*Plus can be used for:
 - Querying certain database objects
 - Executing stored procedures
 - Formatting output from SQL commands
 - Executing batch scripts
 - Executing OS commands
 - Recording output



Installing EDB*Plus

- You can also use yum package manager for installing EDB*Plus
- **EnterpriseDB Repos** must be installed prior to installing EDB*Plus packages
- Login as root or sudo user and run:
 - `yum install -y edb-edbplus`



Starting EDB*Plus

- The EDB*Plus program can be invoked by running `edbplus` from the `edbplus` subdirectory
- Syntax:

```
edbplus: [ -S[ILENT] ] [ login | /NOLOG ] [ @scriptfile[.ext] ]  
login: username[/password] [@{connectstring | variable} ]  
connectstring: host[:port] [/dbname] ]
```

- Example:

```
[centos@epas ~]$ /usr/edb/edbplus/edbplus.sh  
      User: enterprisedb  
Enter Password:  
Connected to EnterpriseDB 15.2.0 (localhost:5444/edb) AS enterprisedb  
  
edb*Plus (Build 41.1.0)  
Copyright (c) 2008-2023, EnterpriseDB Corporation. All rights reserved.  
  
SQL> [green box]
```



EDB*Plus SET Command

- Sets a session level variable to control certain aspects of EDB*Plus behavior as follows:

```
SET AUTO [COMMIT] {ON | OFF | IMMEDIATE | statement_count}
```

- EDB*Plus always autocommits ddl statements
- IMMEDIATE has the same effect as ON
- statement_count causes EDB*Plus to issue a commit after N successful SQL statements
- SET ECHO {ON | OFF} Determines whether SQL and EDB*Plus script statements are shown to the screen as they are executed. The default is OFF

- Type HELP SET in the EDB*Plus terminal to view the entire list of special variables



EDB*Plus Commands

- EDB*Plus supports a variety of Oracle SQL*Plus compatible commands
- Example:

```
SQL> ACCEPT city
Enter value for city: Edmonton
SQL> DEFINE city
DEFINE CITY = "Edmonton"
```

CL[EAR] [BUFF[ER] | SQL | COL[UMNS] | SCR[EEN]]

- CLEAR can be used to clear the buffers, column settings and screen

CON[NECT] username[/password] [@{connectstring | variable}]

- CONNECT can be used to change the database connection



List of EDB*Plus Commands

```
[centos@epas ~]$ /usr/edb/edbplus/edbplus.sh
      User: enterprisedb
Enter Password:
Connected to EnterpriseDB 15.2.0 (localhost:5444/edb) AS enterprisedb

edb*Plus (Build 41.1.0)
Copyright (c) 2008-2023, EnterpriseDB Corporation. All rights reserved.

SQL> HELP INDEX

Type 'HELP [topic]' for command line help.

@          ACCEPT        APPEND        CHANGE
CLEAR      COLUMN        CONNECT       DEFINE
DEL         DESCRIBE     DISCONNECT   EDBPLUS
EDIT        EXIT          GET           HELP
HOST        INDEX         INPUT         LIST
PASSWORD    PAUSE         PRINT         PROMPT
QUIT        REMARK        SAVE          SET
SHOW        SPOOL         START         TRIMS
UNDEFINE    VARIABLE
```



Setup Login Variable

- The database login variable can be setup in login.sql
- login.sql can be created in current folder
- Syntax for login.sql file:

```
define <dbname>="host:port/databasename"
```

- Example:

```
[centos@epas ~]$ cat login.sql
define edb="localhost:5444/edb"
[centos@epas ~]$ /usr/edb/edbplus/edbplus.sh enterpriseDB/edb@edb
Connected to EnterpriseDB 15.2.0 (localhost:5444/edb) AS enterpriseDB

edb*Plus (Build 41.1.0)
Copyright (c) 2008-2023, EnterpriseDB Corporation. All rights reserved.

SQL> █
```



Calling a SQL Script

```
[centos@epas ~]$ vi edb.sql
[centos@epas ~]$ cat edb.sql
select * from dept;
[centos@epas ~]$ /usr/edb/edbplus/edbplus.sh enterpriseedb/edb@edb @edb.sql
Connected to EnterpriseDB 15.2.0 (localhost:5444/edb) AS enterpriseedb
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
edb*Plus (Build 41.1.0)
Copyright (c) 2008-2023, EnterpriseDB Corporation. All rights reserved.
```

SQL> █



Module Summary

- Introduction to psql
- Connecting to Database
- psql Command Line Parameters
- psql Meta-Commands
- Conditional and Information Commands
- EDB*Plus
- Installing and Starting EDB*Plus
- EDB*Plus Commands



Prepare Lab Environment

- In the training materials provided by EnterpriseDB there is a script file **edbstore.sql** that can be executed using `edb-psql` to create a sample `edbstore` database. Here are the steps:
 - Download the **edbstore.sql** file and place in a directory which is accessible to the `enterprisedb` user
 - Login as `enterprisedb` OS user
 - Run the `edb-psql` command with the `-f` option to execute the **edbstore.sql** file and install all the sample objects required for this training
- ```
$ edb-psql -p 5444 -f edbstore.sql -d edb -U enterprisedb
```
- Note - The above command will prompt for `edbuser` password. The password is `edbuser`



# Lab Exercise - 1

- In this lab exercise you will have a chance to practice what you have learned through using command line interfaces:
  1. Connect to a database using psql
  2. Switch databases
  3. Describe the customers table
  4. Describe the customers table including description
  5. List all databases
  6. List all schemas
  7. List all tablespaces
  8. Execute a sql statement, saving the output to a file
  9. Do the same thing, just saving data, not the column headers
  10. Create a script via another method, and execute from psql
  11. Turn on the expanded table formatting mode
  12. Lists tables, views and sequences with their associated access privileges
  13. Which meta command displays the SQL text for a function?
  14. View the current working directory



# Lab Exercise – 2

1. What is the use of EDB\*Plus?
2. Open EDB\*Plus and connect to the `edb` database
3. Write a statement to view the structure of the `orders` table
4. Configure your EDB\*Plus session so that a commit is issued after 10 successful statements
5. Write a statement to turn on autocommit
6. Write a statement to clear the buffers, column settings and screen
7. Setup a default login variable so that `edbplus` connects to the EDB Postgres Advanced Server database named `edbstore` running on host `localhost` and port 5444



# **Module - 5**

# **Database Clusters**

# Module Objectives

- Database Clusters
- Creating a Database Cluster
- Starting and Stopping the Server (pg\_ctl)
- Connecting to the Server Using psql



# Database Clusters

- A Database Cluster is a collection of databases managed by a single server instance
- Database Clusters are comprised of:
  - Data directory
  - Port
- Default databases are created named:
  - template0
  - template1
  - postgres
  - edb



# Creating a Database Cluster

- Choose the data directory location for new cluster
- Initialize the database cluster storage area (data directory) using the `initdb` utility
- `initdb` will create the data directory if it doesn't exist
- You must have permissions on the parent directory so that `initdb` can create the data directory
- The data directory can be created manually by superuser and the ownership can be given to **enterprisedb** user



# initdb Utility

```
$ initdb [OPTION]... [DATADIR]
```

- **Options:**
  - `-D, --pgdata` location for this database cluster
  - `-E, --encoding` set default encoding for new databases
  - `-U, --username` database superuser name
  - `-W, --pwprompt` prompt for a password for the new superuser
  - `-X, --waldir` location for the write-ahead log directory
  - `--wal-segsize` size of wal segments , in megabytes
  - `-k, --data-checksums` use data page checksums
  - `--no-redwood-compat` Do not install Oracle compatibility features
  - `--redwood-like` Use Oracle compatible behavior
  - `-?, --help` show this help, then exit
- If the data directory is not specified, the environment variable PGDATA is used



# Enabling Transparent Data Encryption



Transparent Data Encryption (TDE) can be used to encrypt data files, WAL and temporary files



Following initdb command options can be used to enable TDE:

```
-y, --data-encryption
--copy-key-from=<file>
--key-wrap-command=<command>
--key-unwrap-command=<command>
--no-key-wrap
```



# Example - initdb

```
[root@Base ~]# mkdir /edbstore
[root@Base ~]# chown enterprise:enterprise /edbstore
[root@Base ~]# su - enterprise

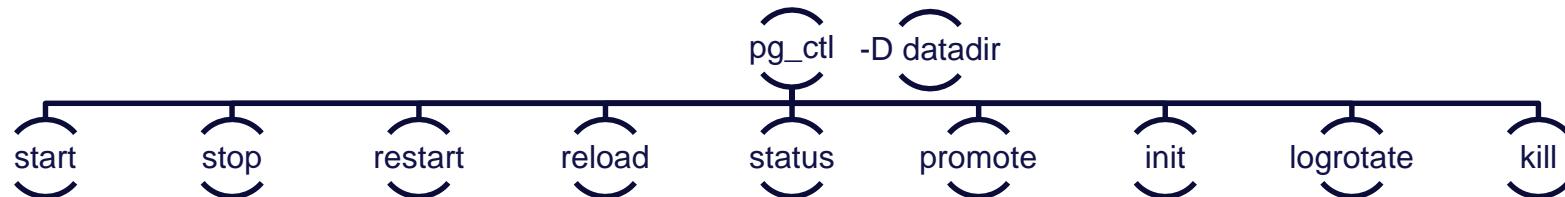
[enterprise@Base ~]$ initdb -D /edbstore --wal-segsize 1024 -W
```

- In the above example the database system will be owned by user enterprise
- The enterprise user is the database superuser
- The default server config file will be created in /edbstore named **postgresql.conf**
- --wal-segsize 1024 MB specifies the write-ahead log file segment size
- -W is used to force **initdb** to prompt for the superuser password



# pg\_ctl Utility

- **pg\_ctl** is a command line utility provided by Postgres to initialize, start, stop and control a Postgres instance
- It provides options for redirecting start log, controlled startup and shutdown
- `-D` option or environmental variable PGDATA can be used to specify cluster data directory



# Starting a Database Cluster

- After initializing a database cluster, a unique port must be assigned
- Choose a unique port for postmaster in **postgresql.conf**
- Start the database cluster using **pg\_ctl** utility
- Example:

```
[enterprisedb@Base ~]$ vi /edbstore/postgresql.conf
port = 5434

[enterprisedb@Base ~]$ pg_ctl -D /edbstore/ -l /edbstore/startlog start
waiting for server to start.... done
server started

[enterprisedb@Base ~]$ pg_ctl -D /edbstore/ status
pg_ctl: server is running (PID: 62239)
```



# Connecting To a Database Cluster

- The edb-psql and PEM clients can be used for connections

```
[enterprisedb@Base ~]$ edb-psql -p 5434 -d edb -U enterprisedb
Type "help" for help.

edb=# show port;
port

5434
(1 row)

edb=# show data_directory;
data_directory

/edbstore
(1 row)

edb=# \q
[enterprisedb@Base ~]$
```



# Reload a Database Cluster

- Some configuration parameter changes do not require a restart
- Changes can be reloaded using the `pg_ctl` utility
- Changes can also be reloaded using `pg_reload_conf()`
- Syntax:

```
$ pg_ctl reload [options]
```

  -D location of the database cluster's data directory

  -s only print errors, no informational messages



# Stopping a Database Cluster

- `pg_ctl` supports three modes of shutdown
  - smart quit after all clients have disconnected
  - Fast quit directly, with proper shutdown (default)
  - immediate quit without complete shutdown; will lead to recovery

- **Syntax:**

```
$ pg_ctl stop [-W] [-t SECS] [-D DATADIR] [-s] [-m SHUTDOWN-MODE]
```

- **Example:**

```
[enterprisedb@Base ~]$ pg_ctl -D /edbstore/ stop
waiting for server to shut down.... done
server stopped
```

```
[enterprisedb@Base ~]$ pg_ctl -D /edbstore/ status
pg_ctl: no server running
```



# **View Cluster Control Information**

- `pg_controldata` can be used to view the control information for a database cluster
  - It can be run with data directory as an option



# Module Summary

- Database Clusters
- Creating a Database Cluster
- Starting and Stopping the Server (pg\_ctl)
- Connecting to the Server Using psql



# Lab Exercise - 1

1. A new website is to be developed for an online music store.
  - Create a new cluster edbdata with ownership of enterpriseDB user
  - Start your edbdata cluster
  - Reload your cluster with pg\_ctl utility and using pg\_reload\_conf() function
  - Stop your edbdata cluster with fast mode



# **Module - 6**

# **Configuration**

# Module Objectives

- Server Parameter File - postgresql.conf
- Viewing and Changing Server Parameters
- Configuration Parameters - Security, Resources and WAL
- Configuration Parameters - Error Logging, Planner and Maintenance
- Viewing Compilation Settings
- Using File Includes



# Setting Server Parameters

- There are many configuration parameters that effect the behavior of the database system
- All parameter names are case-insensitive
- Every parameter takes a value of one of five types:
  - boolean
  - integer
  - floating point
  - string
  - enum
- One way to set these parameters is to edit the file postgresql.conf, which is normally kept in the data directory

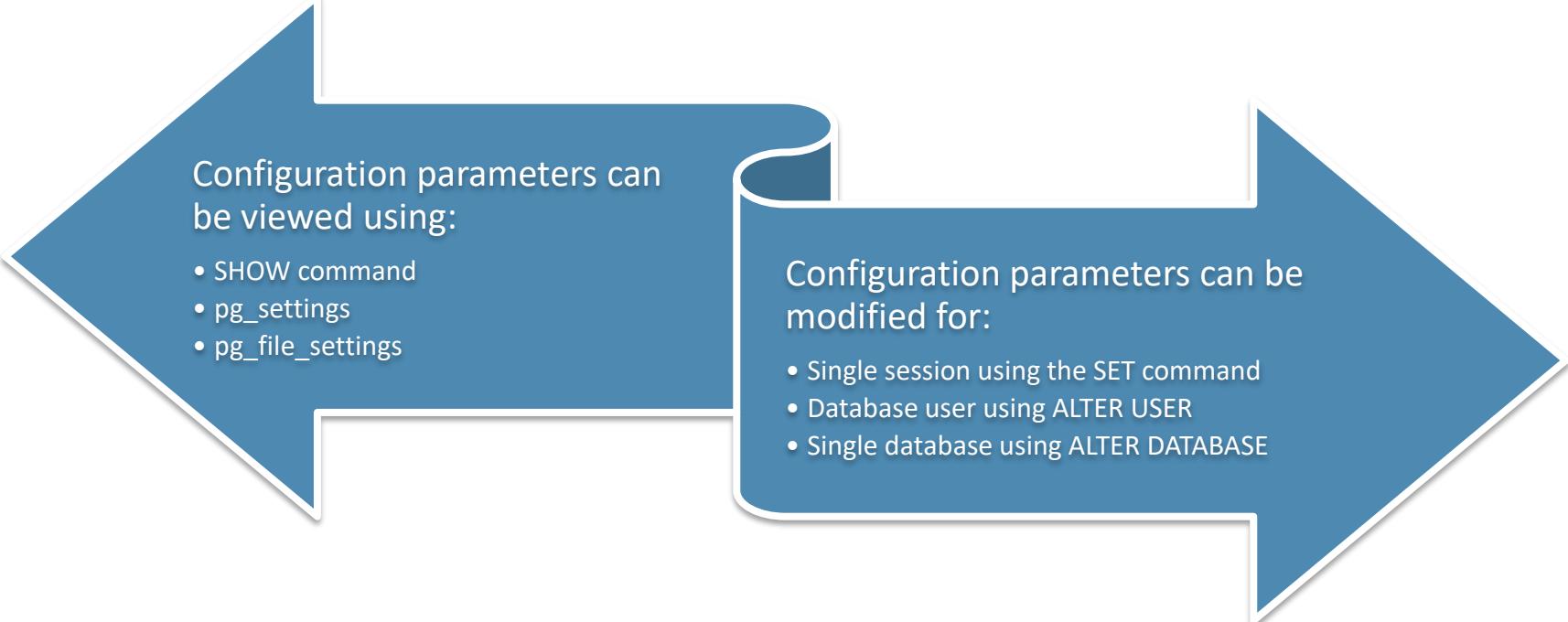


# The Server Parameter File - postgresql.conf

- Holds parameters used by a cluster
- Parameters are case-insensitive
- Normally stored in data directory
- `initdb` installs default copy
- Some parameters only take effect on server restart (`pg_ctl restart`)
- `#` used for comments
- One parameter per line
- Use include directive to read and process another file
- Can also be set using the command-line option



# Viewing and Changing Server Parameters



Configuration parameters can be viewed using:

- SHOW command
- pg\_settings
- pg\_file\_settings

Configuration parameters can be modified for:

- Single session using the SET command
- Database user using ALTER USER
- Single database using ALTER DATABASE



# Changing Configuration Parameter at Cluster Level

```
[enterprisedb@pgsrv1 ~] psql edb enterprisedb

edb=# ALTER SYSTEM SET work_mem=20480;
ALTER SYSTEM
edb=# SELECT pg_reload_conf();

edb=# ALTER SYSTEM RESET work_mem;
ALTER SYSTEM
edb=# SELECT pg_reload_conf();
```

Use ALTER SYSTEM command to edit cluster level settings without editing **postgresql.conf**

ALTER SYSTEM writes new setting to **postgresql.auto.conf** file which is read at last during server reload/restarts

Parameters can be modified using ALTER SYSTEM when required



# Connection Settings

- **listen\_addresses** (default \*) - Specifies the addresses on which the server is to listen for connections. Use \* for all
- **port** (default 5444) - The port the server listens on
- **max\_connections** (default 100) - Maximum number of concurrent connections the server can support
- **superuser\_reserved\_connections** (default 3) - Number of connection slots reserved for superusers
- **unix\_socket\_directory** (default /tmp) - Directory to be used for UNIX socket connections to the server
- **unix\_socket\_permissions** (default 0777) - access permissions of the Unix-domain socket



# Security and Authentication Settings

- **authentication\_timeout** (default is 1 minute) – Maximum time to complete client authentication, in seconds
- **row\_security** (default is on) – Controls row security policy behavior
- **password\_encryption** (default scram-sha-256) – Determines the algorithm to use to encrypt password
- **ssl** (default: off) - Enables SSL connections

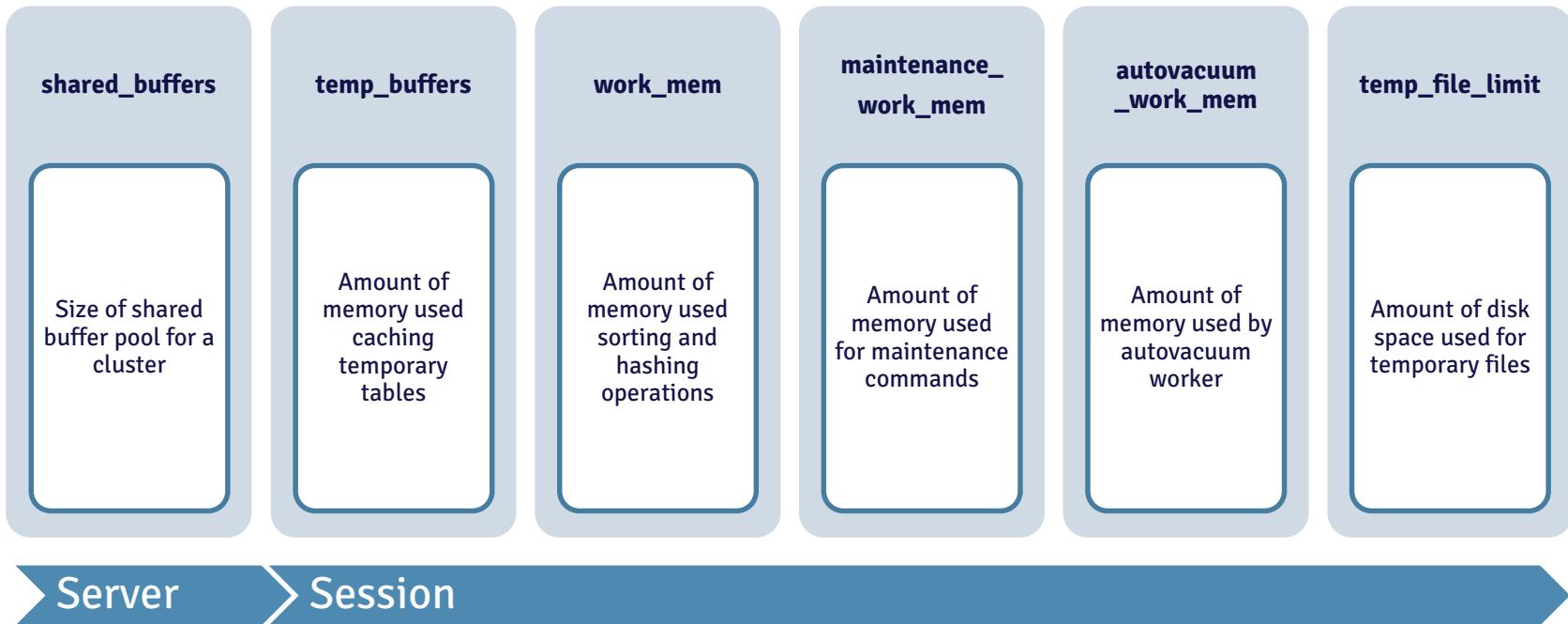


# SSL Settings

- **ssl\_ca\_file** - Specifies the name of the file containing the SSL server certificate authority (CA)
- **ssl\_cert\_file** - Specifies the name of the file containing the SSL server certificate
- **ssl\_key\_file** - Specifies the name of the file containing the SSL server private key
- **ssl\_ciphers** - List of SSL ciphers that may be used for secure connections
- **ssl\_dh\_params\_file** – Specifies file name for custom OpenSSL DH parameters



# Memory Settings



# Dynatune Dynamic Tuning

- Dynatune functionality allows Advanced Server to make optimal usage of the system resources
- Dynatune automatically configures various resource parameters at the time of startup
  - **edb\_dynatune** (0-100) - This parameter determines how Advanced Server allocates system resources
  - **edb\_dynatune\_profile** (oltp | reporting | mixed) - This parameter controls performance tuning aspects based on the type of work that the server performs



# EDB Resource Manager

- Prevents any single process from monopolizing resources to the detriment of other processes
- A resource group can limit the % of cpu or rate of dirty buffer IO
- Helps in managing multiple kinds of workload on your server

## `edb_max_resource_groups`

This parameter controls the maximum number of active resource groups

## `edb_resource_group`

Set per session to the name of the resource group in EDB Resource Manager



# Query Planner Settings

- **random\_page\_cost** (default 4.0) - Estimated cost of a random page fetch. May need to be reduced to account for caching effects
- **seq\_page\_cost** (default 1.0) - Estimated cost of a sequential page fetch.
- **effective\_cache\_size** (default 4GB) - Used to estimate the cost of an index scan.
- **enable\_hints** (default true) - Controls whether Optimizer hints embedded in SQL commands are utilized or not
- **plan\_cache\_mode** (default auto) – Controls custom or generic plan execution for prepared statements. Can be set to auto, force\_custom\_plan and force\_generic\_plan



# Write Ahead Log Settings

- **wal\_level** (default replica) - Determines how much information is written to the WAL. Other values are minimal and logical
- **fsync** (default on) – Force WAL buffer flush at each commit, Turning this off can cause lead to arbitrary corruption in case of a system crash
- **wal\_buffers** (default -1, autotune) - The amount of memory used in shared memory for WAL data. The default setting of -1 selects a size equal to 1/32nd (about 3%) of shared\_buffers
- **min\_wal\_size** (default 80 MB) – The WAL size to start recycling the WAL files
- **max\_wal\_size** (default 1GB) – The WAL size to start checkpoint. Controls the number of WAL Segments(16MB each) after which checkpoint is forced
- **checkpoint\_timeout** (default 5 minutes) - Maximum time between checkpoints
- **wal\_compression** (default off) – The WAL of Full Page write will be compressed and written



# Where To Log

**log\_destination** Controls logging type for a database cluster.

Can be set to stderr, csvlog, jsonlog, syslog, and eventlog

**logging\_collector** Enables logger process to capture stderr and csv logging messages

These messages can be redirected based on configuration settings

## Log File and Directory Settings

**log\_directory** - Directory where log files are written

**log\_filename** - Format of log file name (e.g. postgresql-%Y-%m-%d\_%H%M%S.log)

**log\_file\_mode** - permissions for log files

**log\_rotation\_age** - Used for file age based log rotation

**log\_rotation\_size** - Used for file size based log rotation



# When To Log

## Duration and sampling

|                                    |                                                                                                                              |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>log_min_messages</b>            | Messages of this severity level or above are sent to the server log                                                          |
| <b>log_min_error_statement</b>     | When a message of this severity or higher is written to the server log, the statement that caused it is logged along with it |
| <b>log_min_duration_statement</b>  | When a statement runs for at least this long, it is written to the server log                                                |
| <b>log_autovacuum_min_duration</b> | Logs any Autovacuum activity running for at least this long                                                                  |
| <b>log_statement_sample_rate</b>   | Percentage of queries(above log_autovacuum_min_duration) to be logged                                                        |
| <b>log_transaction_sample_rate</b> | Sample a percentage of transactions by logging statements                                                                    |



# What To Log

|                            |                                                                                                          |
|----------------------------|----------------------------------------------------------------------------------------------------------|
| <b>log_connections</b>     | Log successful connections to the server log                                                             |
| <b>log_disconnections</b>  | Log some information each time a session disconnects, including the duration of the session              |
| <b>log_temp_files</b>      | Log temporary files of this size or larger, in kilobytes                                                 |
| <b>log_checkpoints</b>     | Causes checkpoints and restart points to be logged in the server log                                     |
| <b>log_lock_waits</b>      | Log information if a session is waits longer then deadlock_timeout to acquire a lock                     |
| <b>log_error_verbosity</b> | How detailed the logged message is. Can be set to default, terse or verbose                              |
| <b>log_line_prefix</b>     | Additional details to log with each line. Default is '%m [%p]' which logs a timestamp and the process ID |
| <b>log_statement</b>       | Legal values are none, ddl, mod (DDL and all other data-modifying statements), or all                    |



# Auditing

- EDB Postgres Advanced Server allows you to track and analyze database activities using the EDB Audit Logging functionality
- The audit logs can be configured to record information such as:
  - When a role establishes a connection to the database
  - What database objects are created, modified or deleted by a role
  - When any failed authentication attempts have occurred
- The parameters can be specified in the **postgresql.conf** or **postgresql.auto.conf** files



# Where To Audit

- **edb\_audit** (default `none`) – enables or disables database auditing. Possible values are `none` or `xml` or `csv`
- **edb\_audit\_directory** (default `PGDATA/edb_audit`) – specifies where the audit log files will be created
- **edb\_audit\_destination** (default `file`) - Specifies whether the audit log information is to be recorded in the directory as given by the `edb_audit_directory` parameter or to the directory and file managed by the syslog process. Set to `syslog` to use the `syslog` process and its location as configured in the `/etc/syslog.conf` file
- **edb\_audit\_filename** (default `audit-%Y%m%d_%H%M%S`) – file name of the audit file where the auditing information will be stored



# What To Audit

- **edb\_audit\_connect** (default failed) – enables auditing of database connection attempts by users. Possible values are none, failed or all
- **edb\_audit\_disconnect** (default none) – enables auditing of disconnections by connected users. Possible values are all or none
- **edb\_audit\_statement** (default ddl,error) – specifies auditing of different categories of SQL statements. Various combinations of these values may be specified: none, dml, insert, update, delete, truncate, select, error, rollback, ddl, create, drop, alter, grant, revoke or all
- **edb\_audit\_tag** (default none) – specifies a string value that will be included in the audit log when the `edb_audit` parameter is set to csv or xml



# Controlling Audit Trail

- **edb\_audit\_rotation\_day** (default `every`) – specifies the day of the week on which to rotate the audit files. Possible values are `sun, mon, tue, wed, thu, fri, sat, sun, every` or `none`
- **edb\_audit\_rotation\_size** (default `0MB`) – specifies a file size threshold in MB when file rotation will be forced to occur
- **edb\_audit\_rotation\_seconds** (default `0`) – specifies the rotation time in seconds when a new logfile should be created



# Background Writer Settings

- **bgwriter\_delay** (default 200 ms) - Specifies time between activity rounds for the background writer
- **bgwriter\_lru\_maxpages** (default 100) - Maximum number of pages that the background writer may clean per activity round
- **bgwriter\_lru\_multiplier** (default 2.0) - Multiplier on buffers scanned per round. By default, if system thinks 10 pages will be needed, it cleans  $10 * \text{bgwriter\_lru\_multiplier}$  of 2.0 = 20
- Primary tuning technique is to lower **bgwriter\_delay**



# Statement Behavior

- **search\_path** - This parameter specifies the order in which schemas are searched. The default value for this parameter is "\$user", public
- **default\_tablespace** - Name of the tablespace in which objects are created by default
- **temp\_tablespaces** - Tablespaces name(s) in which temporary objects are created
- **statement\_timeout** - Postgres will abort any statement that takes over the specified number of milliseconds A value of zero (the default) turns this off
- **idle\_in\_transaction\_session\_timeout** – Terminates any session with an open transaction that has been idle for longer than the specified duration in milliseconds



# Parallel Query Scan Settings

- Advanced Server supports parallel execution of read-only queries
- Can be enabled and configured by using configuration parameters
  - **max\_parallel\_workers\_per\_gather (default 2)**: Enables parallel query scan
  - **parallel\_tuple\_cost (default 0.1)**: Estimated cost of transferring one tuple from a parallel worker process to another process
  - **parallel\_setup\_cost (default 1000)**: Estimates cost of launching parallel worker processes
  - **min\_parallel\_table\_scan\_size (default 8MB)**: Sets minimum amount of table data that must be scanned in order for a parallel scan
  - **min\_parallel\_index\_scan\_size (default 512 KB)**: Sets the minimum amount of index data that must be scanned in order for a parallel scan
  - **force\_parallel\_mode (default off)**: Useful when testing parallel query scan even when there is no performance benefit



# Parallel Maintenance Settings

- PostgreSQL supports parallel processes for creating an index
- Currently this feature is only available for btree index type
  - max\_parallel\_maintenance\_workers (default 2):** Enables parallel index creation

```
edb=# set max_parallel_maintenance_workers=0;
SET
Time: 0.399 ms
edb=# create index idx_test_i on test(i);
CREATE INDEX
Time: 6297.539 ms (00:06.298)
```

```
edb=# set max_parallel_maintenance_workers=4;
SET
Time: 0.151 ms
edb=# create index idx_test_i on test(i);
CREATE INDEX
Time: 4807.879 ms (00:04.808)
```



# Vacuum Cost Settings

- **vacuum\_cost\_delay** (default 0 ms) - The length of time, in milliseconds, that the process will wait when the cost limit is exceeded
- **vacuum\_cost\_page\_hit** (default 1) - The estimated cost of vacuuming a buffer found in the buffer pool
- **vacuum\_cost\_page\_miss** (default 10) - The estimated cost of vacuuming a buffer that must be read into the buffer pool
- **vacuum\_cost\_page\_dirty** (default 20) - The estimated cost charged when vacuum modifies a buffer that was previously clean
- **vacuum\_cost\_limit** (default 200) - The accumulated cost that will cause the vacuuming process to sleep



# Autovacuum Settings

- **autovacuum** (default on) - Controls whether the autovacuum launcher runs, and starts worker processes to vacuum and analyze tables
- **log\_autovacuum\_min\_duration** (default -1) - Autovacuum tasks running longer than this duration are logged
- **autovacuum\_max\_workers** (default 3) - Maximum number of autovacuum worker processes which may be running at one time
- **autovacuum\_work\_mem** (default -1, to use maintenance\_work\_mem) - Maximum amount of memory used by each autovacuum worker



# Just-in-Time Compilation

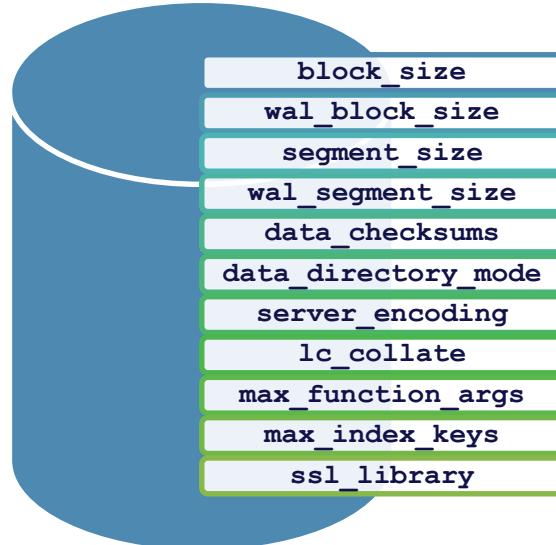
- Just-in-Time(JIT) is a core feature of EDB Postgres for accomplishing high performance
- `edb-as15-server-llvmjit` package must be installed
- JIT in EDB Postgres supports accelerating expression evaluation and tuple deforming
- JIT configuration parameters:

| NAME                                 | DEFAULT VALUE        |
|--------------------------------------|----------------------|
| <code>jit</code>                     | <code>on</code>      |
| <code>jit_above_cost</code>          | <code>100000</code>  |
| <code>jit_debugging_support</code>   | <code>off</code>     |
| <code>jit_dump_bitcode</code>        | <code>off</code>     |
| <code>jit_expressions</code>         | <code>on</code>      |
| <code>jit_inline_above_cost</code>   | <code>500000</code>  |
| <code>jit_optimize_above_cost</code> | <code>500000</code>  |
| <code>jit_profiling_support</code>   | <code>off</code>     |
| <code>jit_provider</code>            | <code>llvmjit</code> |
| <code>jit_tuple_deforming</code>     | <code>on</code>      |
| (10 rows)                            |                      |



# Read-only Parameters

- EDB Postgres sources are compiled using various setting.
- Various read-only configuration parameters can be used to view build settings



# Configuration File Includes

- The **postgresql.conf** file can now contain include directives
- Allows configuration file to be divided in separate files
- Usage in **postgresql.conf** file:
  - **include** 'filename'
  - **include\_dir** 'directory name'



# Module Summary

- Server Parameter File - postgresql.conf
- Viewing and Changing Server Parameters
- Configuration Parameters - Security, Resources and WAL
- Configuration Parameters - Error Logging, Planner and Maintenance
- Viewing Compilation Settings
- Using File Includes



# Lab Exercise - 1

1. You are working as a DBA. It is recommended to keep a backup copy of the **postgresql.conf** file before making any changes. Make the necessary changes in the server parameter file for the following settings:
  - Allow up to 200 connected users on the server
  - Reserve 10 connection slots for DBA users on the server
  - Maximum time to complete client authentication will be 10 seconds



# Lab Exercise - 2

1. Working as a DBA is a challenging job and to track down certain activities on the database server, logging has to be implemented. Go through the server parameters that control logging and implement the following:
  - Save all the error message in a file inside the `log` folder in your cluster data directory (e.g. `c:\edbdata` or `/edbdata`)
  - Log all queries which are taking more than 5 seconds to execute, and their time
  - Log the users who are connecting to the database cluster
  - Make the above changes and verify them



# Lab Exercise - 3

1. Perform the following changes recommended by a senior DBA and verify them. Set:
  - Shared buffer to 256MB
  - Effective cache for indexes to 512MB
  - Maintenance memory to 64MB
  - Temporary memory to 8MB



# Lab Exercise - 4

1. Vacuuming is an important maintenance activity and needs to be properly configured. Change the following **autovacuum** parameters on the production server. Set:
  - Autovacuum workers to 6
  - Autovacuum threshold to 100
  - Autovacuum scale factor to 0.3
  - Auto analyze threshold to 100
  - Autovacuum cost limit to 100



# **Module - 7**

# **Data Dictionary**

# Module Objectives

- The System Catalog Schema
- System Information Tables and Views
- System Information and Administration Functions
- Oracle-like Dictionaries
- Oracle Compatible built-in Packages



# The System Catalog Schema

- Stores information about table and other objects
- Created and maintained automatically in pg\_catalog schema
- pg\_catalog is always effectively part of the search\_path
- Contains:
  - System Tables like pg\_class etc.
  - System Function like pg\_database\_size() etc.
  - System Views like pg\_stat\_activity etc.



# System Information Tables

- `\ds` in `psql` prompt will give you the list of `pg_*` tables and views
- This list is from `pg_catalog` schema

|                |                       |
|----------------|-----------------------|
| pg_tables      | • list of tables      |
| pg_constraints | • list of constraints |
| pg_indexes     | • list of indexes     |
| pg_trigger     | • list of triggers    |
| pg_views       | • list of views       |



# More System Information Tables

`pg_file_settings`

- Provides summary of the contents of the server configuration file

`pg_policy`

- Stores row level security for tables

`pg_policies`

- Provides access to useful information about each row-level security policy in the database



# System Information Functions

`current_database()`

`current_schema[()]`

`pg_postmaster_start_time()`

`version()`

`current_user`

`current_schemas(boolean)`

`pg_current_logfile()`

`txid_status()`

`pg_conf_load_time()`

`pg_jit_available()`



# System Administration Functions

`current_setting, set_config`

- Return or modify configuration variables

`pg_cancel_backend`

- Cancel a backend's current query

`pg_terminate_backend`

- Terminates backend process

`pg_reload_conf`

- Reload configuration files

`pg_rotate_logfile`

- Rotate the server's log file

`pg_start_backup, pg_stop_backup`

- Used with point-in time recovery

`pg_ls_logdir()`

- Returns the name, size, and last modified time of each file in the log directory

`pg_ls_waldir()`

- Returns the name, size, and last modified time of each file in the WAL directory



# More System Administration Functions

`pg_*_size`

- Disk space used by a tablespace, database, relation or total\_relation (includes indexes and toasted data)

`pg_column_size`

- Bytes used to store a particular value

`pg_size.pretty`

- Convert a raw size to something more human-readable

`pg_ls_dir, pg_read_file`

- File operation functions. Restricted to superuser use and only on files in the data or log directories

`pg_blocking_pids()`

- Function to reliably identify which sessions block others



# System Information Views

pg\_stat\_activity

- details of open connections and running transactions

pg\_locks

- list of current locks being held

pg\_stat\_database

- details of databases

pg\_stat\_user\_\*

- details of tables, indexes and functions

pg\_stat\_archiver

- status of the archiver process

pg\_stat\_progress\_basebackup

- view pg\_basebackup progress

pg\_stat\_progress\_vacuum

- provides progress reporting for VACUUM operations

pg\_stat\_progress\_analyze

- provides progress details for ANALYZE operations

pg\_hba\_file\_rules

- provides a summary of the contents of the client authentication configuration file, pg\_hba.conf



# Built-in Packages

- EDB Postgres Advanced Server provides built-in packages compatible with Oracle
- Over 24 of most commonly used Oracle built-in packages are available in EDB Postgres Advanced Server
- These built-in packages provide administration and maintenance utilities

|               |              |             |              |                |
|---------------|--------------|-------------|--------------|----------------|
| DBMS_ALERT    | DBMS_AQ      | DBMS_AQADM  | DBMS_CRYPTO  | DBMS_JOB       |
| DBMS_LOB      | DBMS_LOCK    | DBMS_MVIEW  | DBMS_OUTPUT  | DBMS_PIPE      |
| DBMS_PROFILER | DBMS_RANDOM  | DBMS_RLS    | DBMS_SESSION | DBMS_SCHEDULER |
| DBMS_SQL      | DBMS.Utility | DBMS_REDACT | UTL_ENCODE   | UTL_FILE       |
| UTL_HTTP      | UTL_MAIL     | UTL_SMTP    | UTL_URL      | UTL_RAW        |



# System Catalog Views

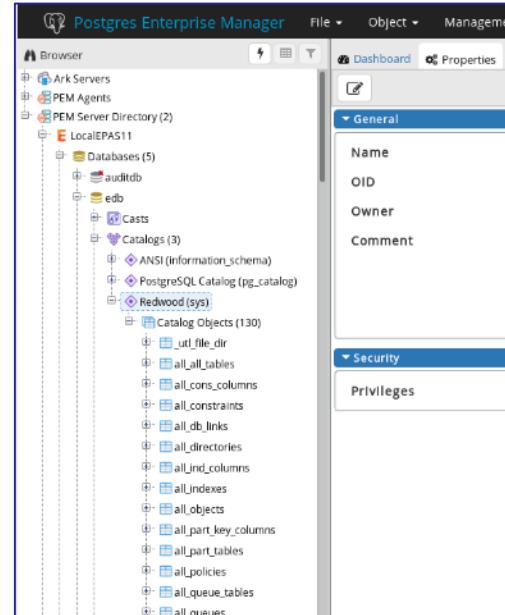
- EDB Postgres Advanced Server database stores user data in user tables
- EDB Postgres Advanced Server database stores internal information about all the objects and events in the catalog tables
- System catalog tables are kept in separate internal schemas
- EDB Postgres Advanced Server provides Oracle compatible catalog view (data dictionaries)
- These views can be queried to find:
  - Definition of schema objects
  - Users information
  - Privileges
  - Other general database information



# sys Schema

- sys schema contains Oracle compatible Catalog Views

| View   | Description                                                   |
|--------|---------------------------------------------------------------|
| user_* | User's view (what is in your schema; what you own)            |
| all_*  | Expanded user's view (what you can access)                    |
| dba_*  | Database administrator's view (what is in everyone's schemas) |
| edb\$  | Performance-related data                                      |



# Module Summary

- The System Catalog Schema
- System Information Tables and Views
- System Information and Administration Functions
- Oracle-like Dictionaries
- Oracle Compatible built-in Packages



# Lab Exercise - 1

1. You are working with different schemas in a database. After a while you need to determine all the schemas in your search path. Write a query to find the list of schemas currently in your search path.



## Lab Exercise - 2

1. You need to determine the names and definitions of all of the views in your schema. Create a report that retrieves view information - the view name and definition text.



## Lab Exercise - 3

1. Create a report of all the users who are currently connected. The report must display total session time of all connected users.
2. You found that a user has connected to the server for a very long time and have decided to gracefully kill its connection. Write a statement to perform this task.



# Lab Exercise - 4

1. Write a query to display the name and size of all the databases in your cluster. Size must be displayed using a meaningful unit.



# **Module - 8**

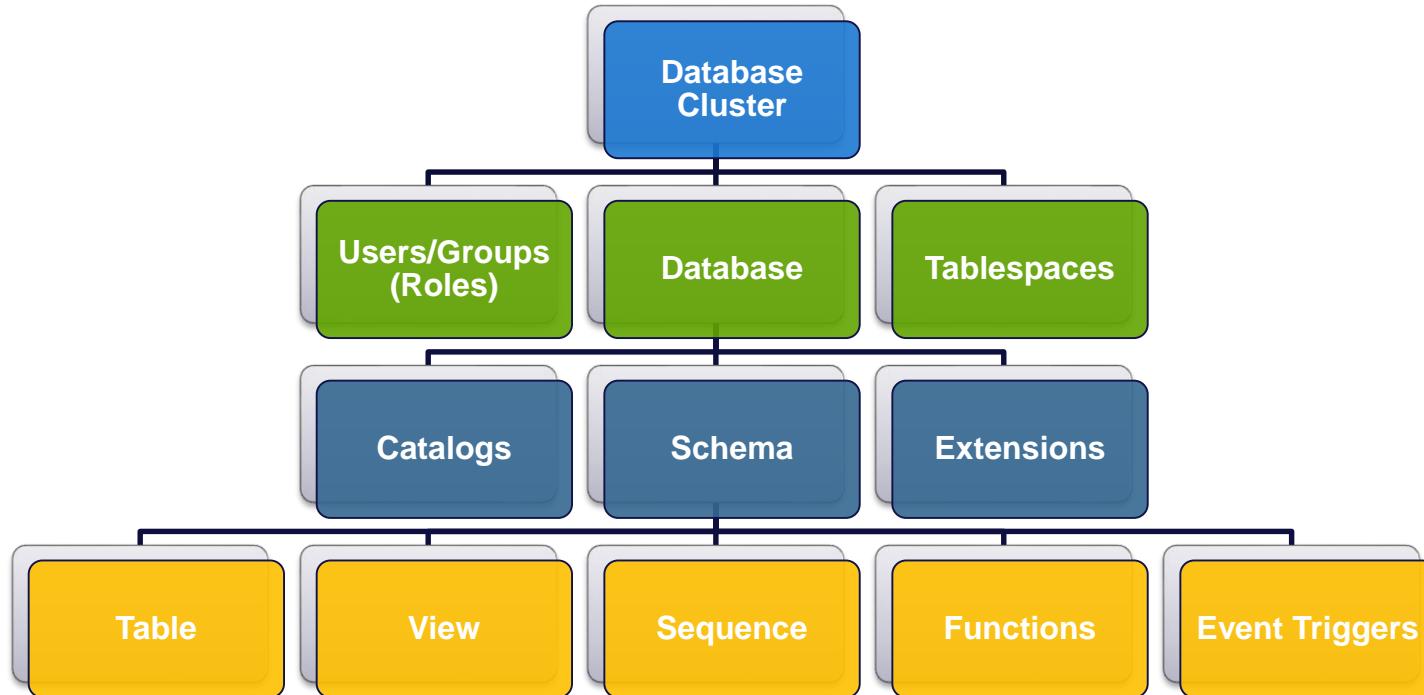
## **Creating and Managing Databases**

# Module Objectives

- Object Hierarchy
- Users and Roles
- Tablespaces
- Databases
- Access Control
- Creating Schemas
- Schema Search Path



# Object Hierarchy





# Users and Roles

# Database Users

- Are global across a database cluster
- Are not the operating system users
- Are used for connecting to a database
- Have a unique name not starting with pg\_
- enterpriseDB **is a predefined superuser**



# Creating Users Using psql

- How to create? CREATE USER sql command
- How to delete? DROP USER sql command
- superuser or createrole privilege is required for creating a database user

## Syntax:

```
CREATE USER name [[WITH] option [...]]
where option can be:
 SUPERUSER | CREATEDB | CREATEROLE | LOGIN |
 REPLICATION | BYPASSRLS CONNECTION LIMIT
 connlimit
 | PASSWORD 'password' | VALID UNTIL 'timestamp'
 | PROFILE profile_name
 | ACCOUNT { LOCK | UNLOCK }
 | LOCK TIME 'timestamp'
 | PASSWORD EXPIRE [AT 'timestamp']
```

## Example:

```
postgres=# CREATE USER rupi
postgres-# PASSWORD 'rupi123' SUPERUSER;
CREATE ROLE
postgres=#
```

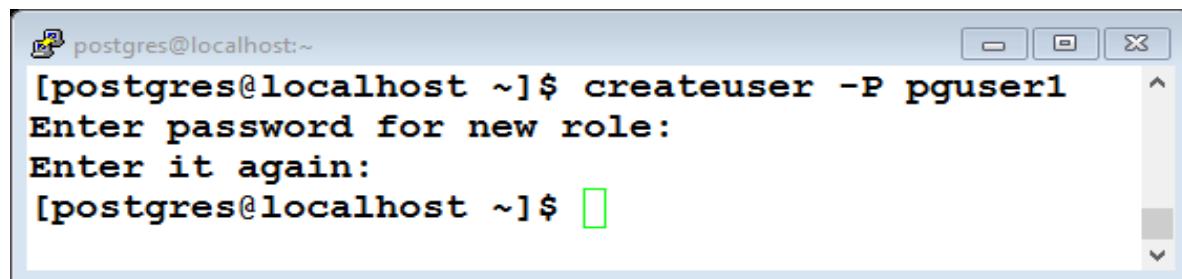


# Creating Users Using `createuser`

- The `createuser` utility can also be used to create a user
- Syntax:

```
$ createuser [OPTION]... [ROLENAME]
```

- Use `--help` option to view the full list of options available
- Example:



The screenshot shows a terminal window titled "postgres@localhost:~". The command entered is `createuser -P pguser1`. The terminal prompts the user to "Enter password for new role:" and "Enter it again:". The password is entered twice, and the terminal prompt changes to a green square.

```
postgres@localhost:~]$ createuser -P pguser1
Enter password for new role:
Enter it again:
[postgres@localhost ~]$
```



# User Profile Management

- A profile is a named set of password attributes
  - User profiles can be used to manage account status and password expiration
  - The default profile is assigned to all users
- 
- Password attributes:
    - FAILED\_LOGIN\_ATTEMPTS
    - PASSWORD\_LOCK\_TIME
    - PASSWORD\_LIFE\_TIME
    - PASSWORD\_GRACE\_TIME
    - PASSWORD\_REUSE\_TIME
    - PASSWORD\_REUSE\_MAX
    - PASSWORD\_VERIFY\_FUNCTION
    - PASSWORD\_ALLOW\_HASHED



# Creating A User Profile

- How to create? CREATE PROFILE command
- How to delete? DROP PROFILE command
- Example:

```
edb=# CREATE PROFILE dba_top LIMIT FAILED_LOGIN_ATTEMPTS 1;
CREATE PROFILE
edb=# CREATE USER user2 PASSWORD 'edb' SUPERUSER PROFILE dba_top;
CREATE ROLE
edb=#

```



# Roles

- Role is a collection of cluster and object level privileges
- Role makes it easier to manage multiple privileges
- How to create? CREATE ROLE statement
- How to assign? GRANT statement
- Who it can be assigned to? user or a group



# Default Roles

aq\_administrator\_role

pg\_checkpoint

pg\_database\_owner

pg\_execute\_server\_program

pg\_monitor

pg\_read\_all\_data

pg\_read\_all\_settings

pg\_read\_all\_stats

pg\_read\_server\_files

pg\_signal\_backend

pg\_stat\_scan\_tables

pg\_write\_all\_data

pg\_write\_server\_files

- Provide certain administrative capabilities using these default roles
- For Example, create a new user with read only access or a new user with access to view monitoring data only





# Tablespaces

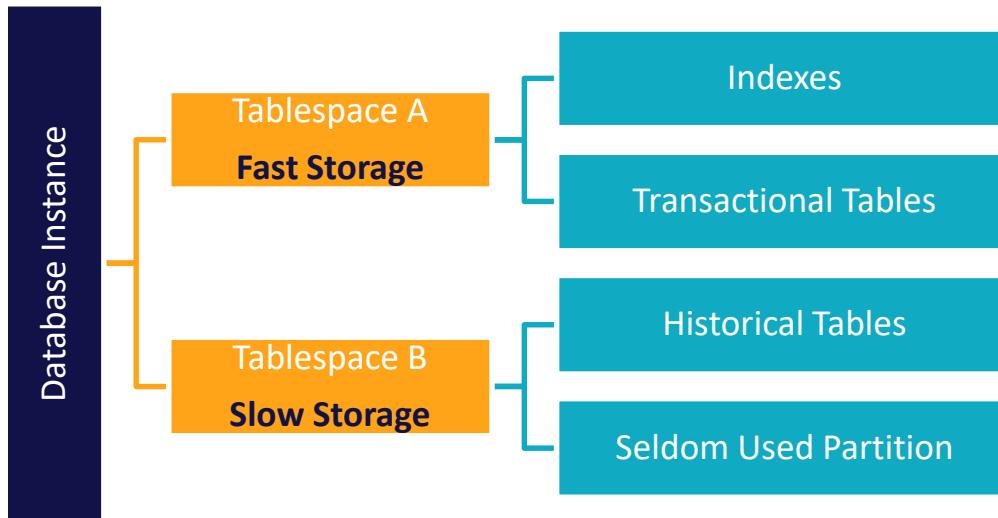
# Tablespaces and Data Files

- Data is stored logically in tablespaces and physically in data files
- **Tablespaces:**
  - Can belong to only one database cluster
  - Consist of multiple data files
  - Can be used by multiple databases
- **Data Files:**
  - Can belong to only one tablespace
  - Are used to store database objects
  - Cannot be shared by multiple tables (one or more per table)

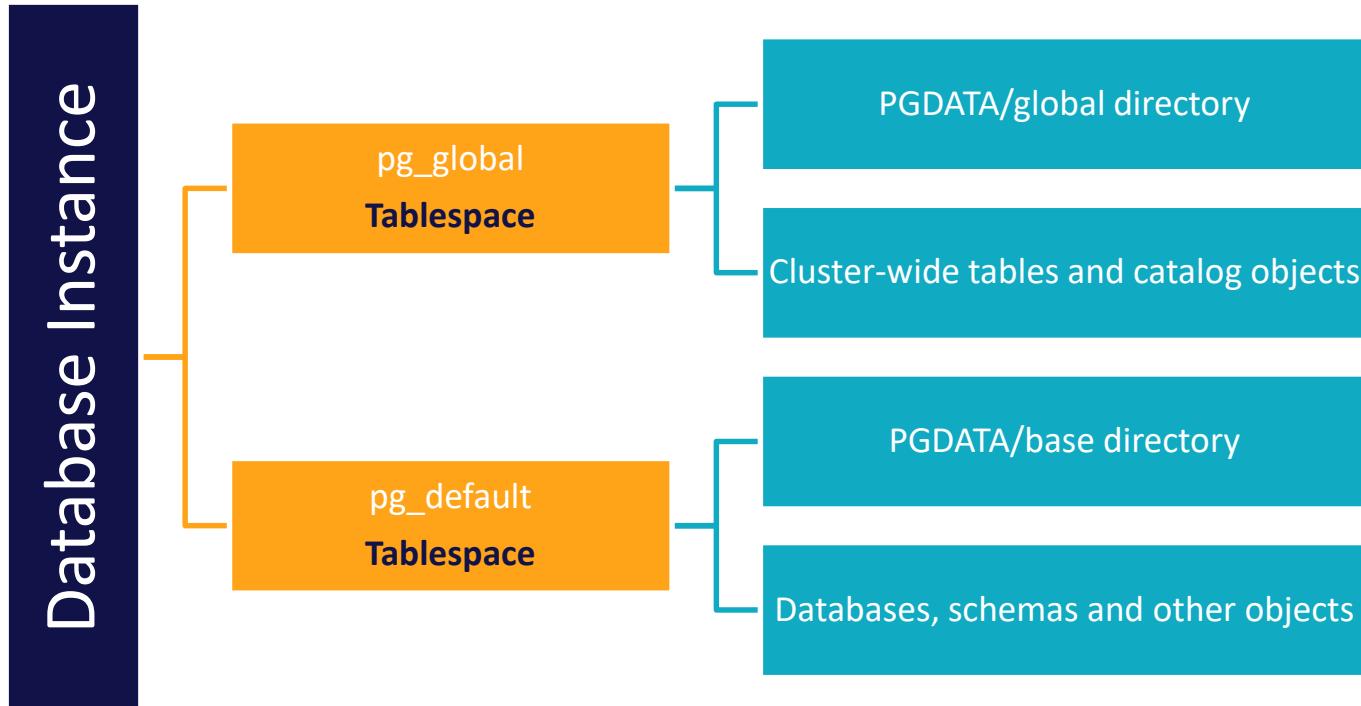


# Advantages of Tablespaces

- Control the disk layout for a database cluster
- Store indexes and data physically separated for performance

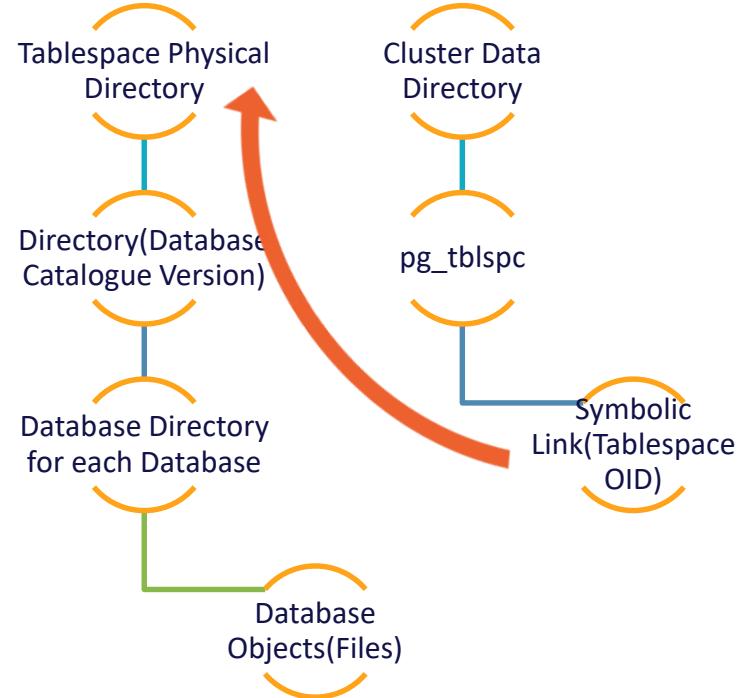


# Pre-Configured Tablespaces



# Creating Tablespaces

- How to create? CREATE TABLESPACE command
- The tablespace directory must be existing with permissions
- Syntax:
  - ```
CREATE TABLESPACE  
tablespace_name [ OWNER  
user_name ]  
LOCATION 'directory';
```



Example - CREATE TABLESPACE

```
[training@Base ~]$ sudo mkdir /newtab1
[training@Base ~]$ sudo chown enterprisebd:enterprisebd /newtab1
[training@Base ~]$ su - enterprisebd
[enterprisebd@Base ~]$ psql -p 5444 edb enterprisebd

edb=# CREATE TABLESPACE fast_tab LOCATION '/newtab1';
CREATE TABLESPACE
edb=# \db
      List of tablespaces
   Name    |    Owner    | Location
-----+-----+-----
 fast_tab | enterprisebd | /newtab1
 pg_default | enterprisebd |
 pg_global  | enterprisebd |
(3 rows)
```



Using Tablespaces

- Use the TABLESPACE keyword while creating databases, tables and indexes

```
edb=# CREATE TABLE account(acno NUMBER PRIMARY KEY,  
ac_hldr_fname VARCHAR2(20)) TABLESPACE fast_tab;  
CREATE TABLE
```



Default and Temp Tablespace

- `default_tablespace` server parameter sets default tablespace
- `default_tablespace` parameter can also be set using the `SET` command at the session level
- `temp_tablespaces` parameter determines the placement of temporary tables and indexes and temporary files
- `temp_tablespaces` can be a list of tablespace names

```
edb=# show default_tablespace;
default_tablespace
-----
(1 row)
edb=# show temp_tablespaces;
temp_tablespaces
-----
(1 row)
```



Altering Tablespaces

- `ALTER TABLESPACE` can be used to rename a tablespace, change ownership and set a custom value for a configuration parameter
- Only the owner or superuser can alter a tablespace
- The `seq_page_cost` and `random_page_cost` parameters can be altered for a tablespace



Example - Alter Tablespace

Syntax:

```
ALTER TABLESPACE name RENAME TO new_name  
ALTER TABLESPACE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }  
ALTER TABLESPACE name SET ( tablespace_option = value [, ...] )  
ALTER TABLESPACE name RESET ( tablespace_option [, ...] )
```

```
edb=# ALTER TABLESPACE fast_tab RENAME TO new_tab;
```

```
ALTER TABLESPACE
```

```
edb=# \db
```

```
      List of tablespaces
```

Name	Owner	Location
new_tab	enterprisedb	/newtab
pg_default	enterprisedb	
pg_global	enterprisedb	



Dropping a Tablespace

- `DROP TABLESPACE` removes a tablespace from the system
- Only the owner or superuser can drop a tablespace
- The tablespace must be empty
- If a tablespace is listed in the `temp_tablespaces` parameter, make sure current sessions are not using the tablespace
- `DROP TABLESPACE` cannot be executed inside a transaction





Databases

What Is a Database?

- A database is a named collection of SQL objects
- A running Postgres instance can manage multiple databases
- How to create? CREATE DATABASE command
- How to delete? DROP DATABASE command
- To determine the set of existing databases:
 - SQL - SELECT datname FROM pg_database;
 - psql META COMMAND - \l (backslash lowercase L)



Creating Databases

- Database can be created using:
 1. `createdb` utility program
 2. `CREATE DATABASE` SQL command
- SQL Command syntax:

```
CREATE DATABASE name [ [ WITH ] [ OWNER [=] user_name ]  
[ TEMPLATE [=] template ]  
[ ENCODING [=] encoding ]  
[ TABLESPACE [=] tablespace_name ]  
[ ALLOW_CONNECTIONS [=] allowconn ]  
[ CONNECTION LIMIT [=] connlimit ]
```



Example - Creating Databases

```
edb=# CREATE DATABASE prod OWNER enterpriseDb;
CREATE DATABASE
edb=# REVOKE CONNECT ON DATABASE prod FROM PUBLIC;
REVOKE
edb=# \connect prod
You are now connected to database "prod" as user "enterpriseDb".
prod=# SELECT datname FROM pg_database;
  datname
  -----
  postgres
  edb
  template1
  template0
  prod
(5 rows)
```



Accessing a Database

- **PEM Web Client** or **psql** can be used to access a database
- To use **psql**, open a terminal and execute:

```
$ psql -U postgres -d prod
```

Note: If PATH is not set you can execute psql command from the bin directory of postgres installation



Privileges

- Cluster level
 - Granted to a user during CREATE or later using ALTER USER
 - These privileges are granted by superuser
- Object Level
 - Granted to user using GRANT command
 - These privileges allow a user to perform particular actions on a database object, such as tables, views, or sequence
 - Can be granted by owner, superuser or someone who has been given permission to grant privileges (WITH GRANT OPTION)



GRANT Statement

- Grants object level privileges to database users, groups or roles
- GRANT can also be used to grant a role to a user
- How to view syntax and available privileges?
 - Type \h GRANT in psql



Example – GRANT Statement

```
edbstore=# GRANT CONNECT ON DATABASE edbstore TO user1;
GRANT
edbstore=# GRANT USAGE ON SCHEMA edbuser TO user1;
GRANT
edbstore=# GRANT SELECT, INSERT ON edbuser.emp TO user1;
GRANT
edbstore=# \c edbstore user1
Password for user user1:
You are now connected to database "edbstore" as user "user1".
edbstore=> SELECT * FROM edbuser.emp LIMIT 2;
empno | ename | job | mgr | hiredate | sal | comm | deptno
-----+-----+-----+-----+-----+-----+-----+-----+
 7499 | ALLEN | SALESMAN | 7698 | 20-FEB-81 00:00:00 | 1600.00 | 300.00 | 30
 7521 | WARD | SALESMAN | 7698 | 22-FEB-81 00:00:00 | 1250.00 | 500.00 | 30
(2 rows)
```



REVOKE Statement

- Revokes object level privileges from database users, groups or roles
- REVOKE [GRANT OPTION FOR] can be used to revoke only the grant option without revoking the actual privilege
- How to view syntax and available privileges?
 - Type \h REVOKE in psql



Example - REVOKE Statement

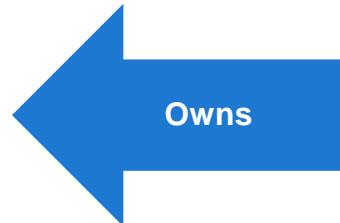
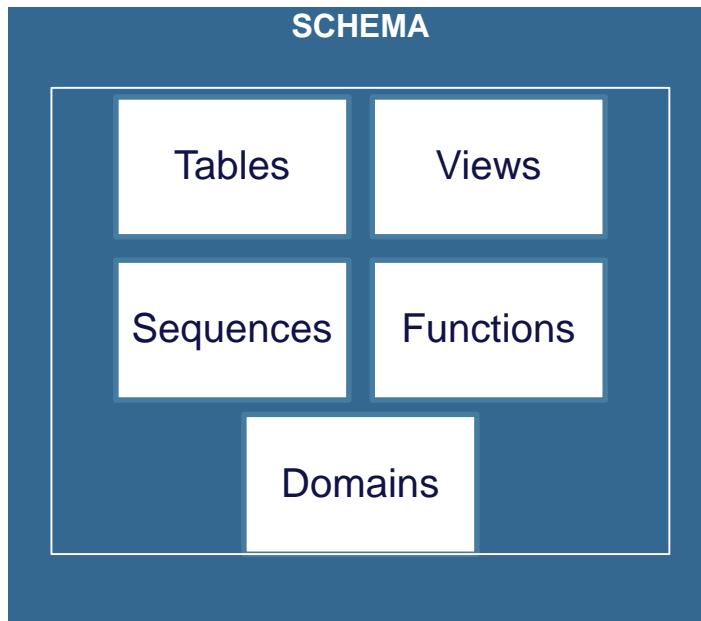
```
edbstore=# REVOKE SELECT, INSERT ON edbuser.emp FROM user1;
REVOKE
edbstore=# \c edbstore user1
Password for user user1:
You are now connected to database "edbstore" as user "user1".
edbstore=> SELECT * FROM edbuser.emp LIMIT 2;
ERROR: permission denied for relation emp
edbstore=> \c edbstore enterprise
Password for user enterprise:
You are now connected to database "edbstore" as user "enterprise".
edbstore=# REVOKE USAGE ON SCHEMA edbuser FROM user1;
REVOKE
edbstore=# REVOKE CONNECT ON DATABASE edbstore FROM user1;
REVOKE
edbstore=# REVOKE CONNECT ON DATABASE edbstore FROM public;
REVOKE
edbstore=# \c edbstore user1
Password for user user1:
FATAL: permission denied for database "edbstore"
DETAIL: User does not have CONNECT privilege.
Previous connection kept
edbstore=# ■
```





Database Schemas

What is a Schema



Benefits of Schemas

- A database can contain one or more named schemas
- By default, all databases contain a public schema
- There are several reasons why one might want to use schemas:
 - To allow many users to use one database without interfering with each other
 - To organize database objects into logical groups to make them more manageable
 - Third-party applications can be put into separate schemas so they cannot collide with the names of other objects



Creating Schemas

- Schemas can be added using the CREATE SCHEMA SQL command
- Syntax:

```
CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION  
role_specification ]
```

- Example:

```
prod=# CREATE SCHEMA rupi AUTHORIZATION rupi;  
CREATE SCHEMA  
prod=# GRANT CONNECT ON DATABASE prod TO rupi;  
GRANT  
prod=# \c prod rupi:  
Password for user rupi:  
You are now connected to database "prod" as user "rupi".  
prod=> CREATE TABLE city(cityname VARCHAR2);  
CREATE TABLE  
prod=> \dt  
      List of relations  
 Schema | Name | Type  | Owner  
-----+-----+-----+  
 rupi   | city  | table | rupi  
(1 row)  
  
prod=>
```



What is a Schema Search Path

- The schema search path determines which schemas are searched for matching table names
- Search path is used when fully qualified object names are not used in a query
- Example:

```
⇒ SELECT * FROM employee;
```

This statement will find the first employee table from the schemas listed in the search path

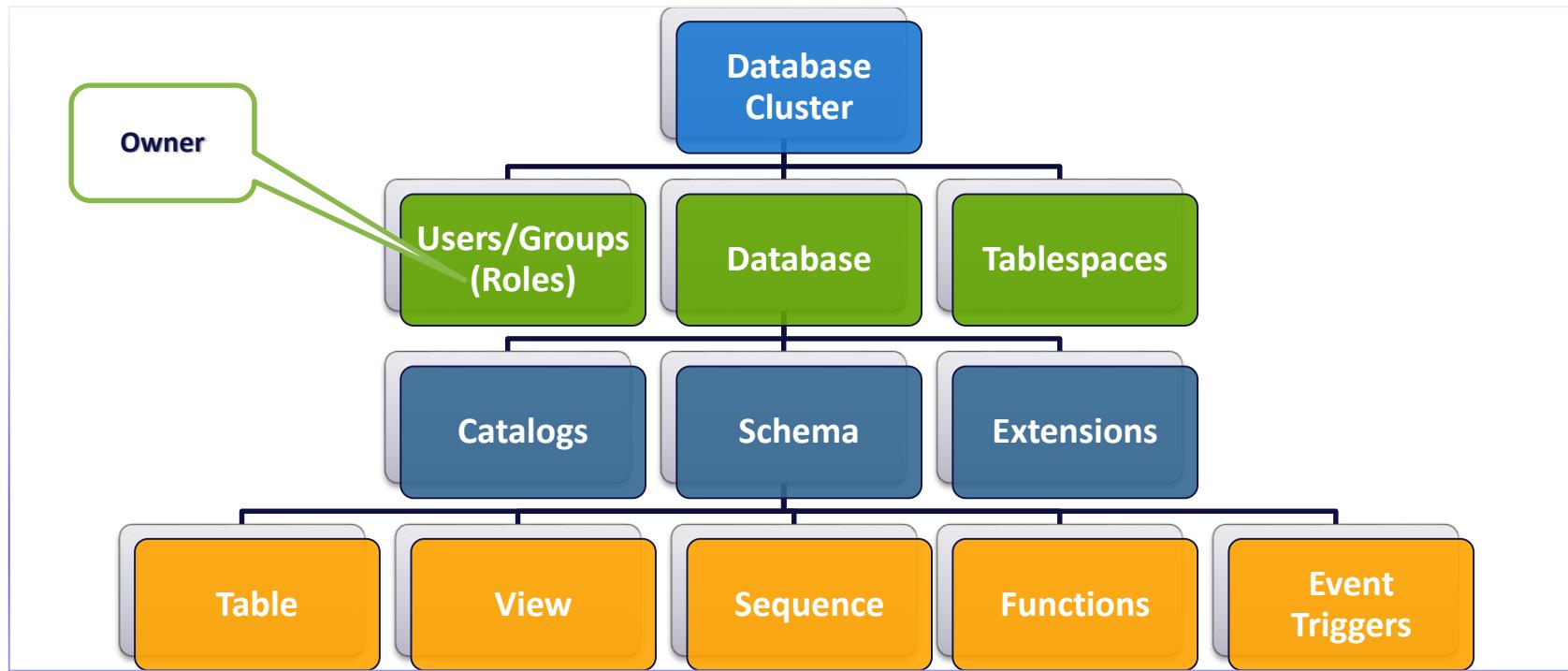


Determine the Schema Search Path

- To show the current search path, use the following command:
 - => SHOW search_path;
- Default search_path is "\$user", public
- Search path can be changed using SET command:
 - => SET search_path TO myschema, public;



Object Ownership



Module Summary

- Object Hierarchy
- Users and Roles
- Tablespaces
- Databases
- Access Control
- Creating Schemas
- Schema Search Path



Lab Exercise - 1

1. An e-music online store website application developer wants to add an online buy/sell facility and has asked you to separate all tables used in online transactions. Here you have suggested to use schemas.

Implement the following suggested options:

- Create an `ebuy` user with password ‘lion’
- Create an `ebuy` schema which can be used by user `ebuy`
- Login as the `ebuy` user, create a table `sample1` and check whether that table belongs to the `ebuy` schema or not



Lab Exercise - 2

1. Retrieve a list of databases using a SQL query
2. Retrieve a list of databases using the psql meta command
3. Retrieve a list of tables in the `edbstore` database and check which schema and owner they have



Module - 9

Database Security

Module Objectives

- Database Security Requirements and Protection Plan
- Levels of Security in Postgres
- Access Control using pg_hba.conf
- Introduction to Row Level Security
- Data Encryption
- Advanced Features - Data Redaction, SQL/Protect and EDB*Wrap
- General Security Recommendations



Why Database Security



- Databases are a core component of many computing systems
- Confidential data like SIN(Social Insurance Number), Healthcare, Banking details is stored and shared using databases
- It is very critical to protect stored information from hackers, insiders, and other groups who intend to steal valuable data
- Database Security is a mechanism to protect the data against threats



Data Security Requirements

- Stopping improper disclosure, modification and denial of access to information is very important
- Who wants an employee finding out boss's salary, changing his/her salary or stopping HR from printing paychecks
- Database Security Requirements includes:
 - Confidentiality
 - Integrity
 - Availability

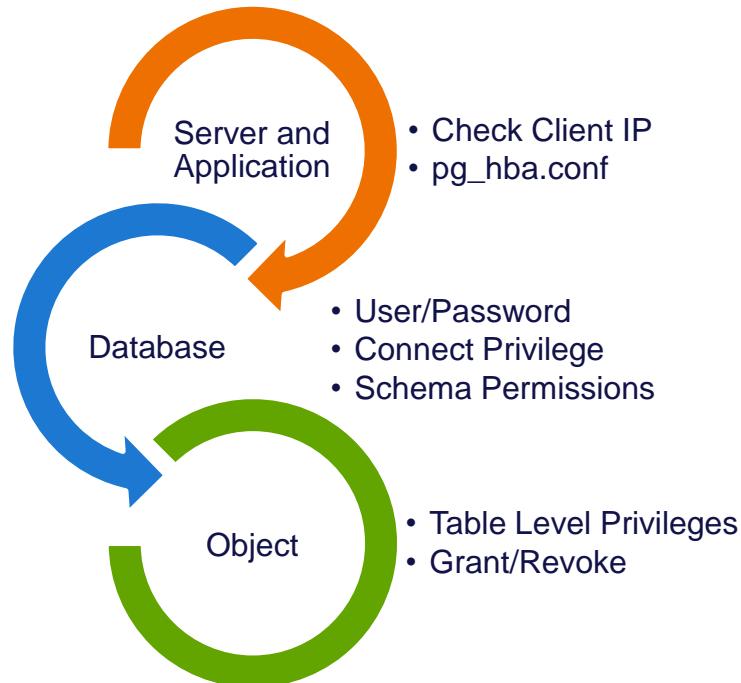


Protection Plan – We all need one

- **Access Control**
 - Authentication and Authorization
- **Data Control**
 - Views, Row Level Security, Encryptions
- **Network Control**
 - SSL Connections, Firewalls
- **Auditing**
 - Monitoring



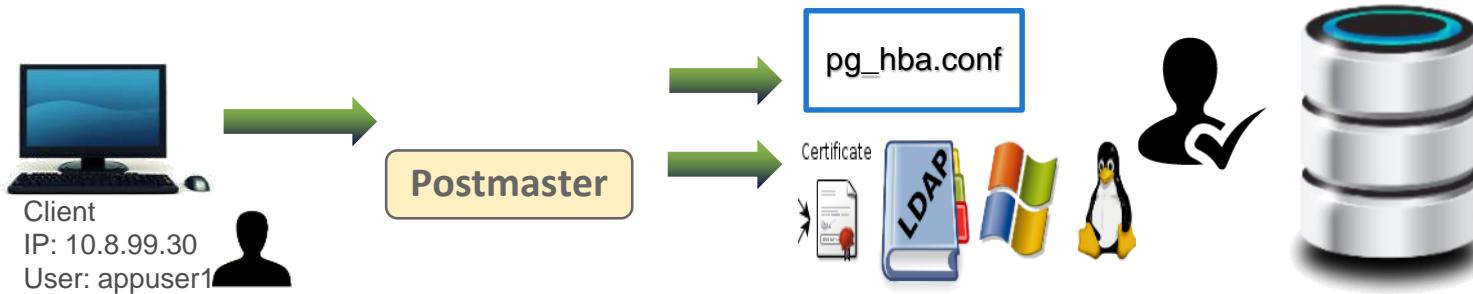
Levels of Security





Host Based Access Control

Host Based Access Control



- **pg_hba.conf** can be used to restrict the ability to connect to a database
- SSL can be forced for selected clients based on hostname or IP address
- Different authentication methods can be used
- Superuser access can be locked down to certain IPs using **pg_hba.conf**

pg_hba.conf - Access Control

- Host based access control file
- Located in the cluster data directory
- Read at startup, any change requires reload
- Contains a set of records, one per line
- Each record specifies a connection type, database name, user name, client IP and method of authentication
- Top to bottom read
- Hostnames, IPv6 and IPv4 supported
- Authentication methods - trust, reject, md5, password, gss, sspi, krb5, ident, peer, pam, ldap, radius, bsd, scram or cert



pg_hba.conf Example

#	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	ident
# IPv6 local connections:					
host	all	all		::1/128	ident
# Allow replication connections from localhost, by a user with the					
# replication privilege.					
local	replication	all			peer
host	replication	all		127.0.0.1/32	ident
host	replication	all		::1/128	ident



Authentication Problems

FATAL: no pg_hba.conf entry for host "192.168.10.23", user "edbstore", database "edbuser"

FATAL: password authentication failed for user "edbuser"

FATAL: user "edbuser" does not exist

FATAL: database "edbstore" does not exist

- Self-explanatory message is displayed
- Verify database name, username and Client IP in **pg_hba.conf**
- Reload Cluster after changing **pg_hba.conf**
- Check server log for more information





Row Level Security

Row Level Security (RLS)

- GRANT and REVOKE can be used at table level
- PostgreSQL supports security policies for limiting access at row level
- By default, all rows of a table are visible
- Once RLS is enabled on a table, all queries must go through the security policy
- Security policies are controlled by DBA rather than application
- RLS offers stronger security as it is enforced by the database

The diagram illustrates the concept of Row Level Security (RLS). It shows two user profiles on the left: one with a laptop icon and another with a smartphone icon. A blue line connects the laptop to a table on the right, and an orange line connects the smartphone to the same table. The table has two columns: 'ACCOUNT' and 'BALANCE'. The data rows are colored to represent different companies: Company J (light gray), Company M (orange), Company Z (light gray), Company L (blue), Company R (light gray), Company A (orange), Company T (blue), and Company Q (light gray). The 'BALANCE' column contains values such as '\$23,925', '\$133,007', '\$17,092', '\$997,654', '\$72,871', '\$0.0', '\$50,194', and '\$67,892'.

ACCOUNT	BALANCE
Company J	\$23,925
Company M	\$133,007
Company Z	\$17,092
Company L	\$997,654
Company R	\$72,871
Company A	\$0.0
Company T	\$50,194
Company Q	\$67,892

Example - Row Level Security

- For example, to enable row level security for the table accounts :

- Create the table first

```
edb=# CREATE TABLE accounts (manager text, company text, contact_email text);
```

- Then alter the table

```
edb=# ALTER TABLE accounts ENABLE ROW LEVEL SECURITY;
```

- Syntax:

```
CREATE POLICY name ON table_name  
    [ AS { PERMISSIVE | RESTRICTIVE } ]  
    [ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]  
    [ TO{ role_name | PUBLIC | CURRENT_USER | SESSION_USER} [, ...] ]  
    [ USING ( using_expression ) ]  
    [ WITH CHECK ( check_expression ) ]
```



Example - Row Level Security (continued)

- To create a policy on the accounts table to allow the managers role to view the rows of their accounts, the CREATE POLICY command can be used:

```
edb=# CREATE POLICY account_managers ON accounts TO managers USING  
(manager = current_user);
```

- To allow all users to view their own row in a user table, a simple policy can be used:

```
edb=# CREATE POLICY user_policy ON users USING (user =  
current_user);
```





Data Encryption

Transparent Data Encryption

- Transparent Data Encryption was introduced in EDB Postgres Advanced Server version 15
- Helps securing user data by encrypting data files, write-ahead logs and temporary files
- TDE is transparent to user and doesn't require any application-level change
- With TDE, Database Server and backup storage files are unintelligible for unauthorized users
- Does not encrypt transaction status metadata, data directory structure, foreign tables, logs and configuration files
- General overhead of AES is expected



Database Level Encryption

- Encrypting everything does not make data secure
- Resources are consumed when you query encrypted data
- `pgcrypto` provides mechanism for encrypting selected columns
- `pgcrypto` supports one-way and two-way data encryption
- Install `pgcrypto` using `CREATE EXTENSION` command
 - `CREATE EXTENSION pgcrypto;`

Emp_ID	EmpName	Salary_encrypt
1	Jim	0x003839E4BBC07047AAAA64145FA4967E010000003396414...
2	Cary	0x003839E4BBC07047AAAA64145FA4967E010000006910052...
3	Tom	0x003839E4BBC07047AAAA64145FA4967E010000006F82168...
4	Crouse	0x003839E4BBC07047AAAA64145FA4967E010000000BC83B...
5	Gary	0x003839E4BBC07047AAAA64145FA4967E010000000F51EF75...
8	Matts	0x003839E4BBC07047AAAA64145FA4967E010000009641E58...





Added Security in EDB Postgres Advanced Server

Added Security in EDB Postgres Advanced Server

- EDB Postgres Advanced Server supports all the security options available in PostgreSQL, plus:
 - Transparent Data Encryption
 - User Profiles and Password Policy Manager
 - SQL/Protect - Protection against SQL Injections
 - DBMS_CRYPTO, DBMS_RLS, DBMS_REDACT
 - Data redaction
 - EDB*Wrap - Obfuscate Source Code
 - Enhanced Auditing using `edb_audit`



Data Redaction

- Data Redaction can be used to conceal data values from selected users
- The redaction function is incorporated into redaction policy using CREATE REDACTION POLICY
- Data Redaction is controlled by `edb_data_redaction` configuration parameter
- Useful for compliance to GDPR, PCI and HIPAA standards

	Stored Data	Redacted Display
FULL	10/09/1992	01/01/2001
PARTIAL	987-65-4328	XXX-XX-4328
RED/EXP	first.last@example.com	[hidden]@example.com
RANDOM	5105105105105100	4012888888881881



SQL/PROTECT – Track, Warn and Protect

SQL firewall installed directly in the database server

Centrally managed layer of security

Screens incoming queries for common attack profiles



EDB*Wrap – Protects Your Code

- Safeguards sensitive code from prying eyes inside your firewall
- Protects critical algorithms, processes, seed values and more
- Restricts access to intellectual property on customer sites
- Additional layer of security beyond standard user ACLs



The screenshot shows a window titled "SQL pane" containing a PostgreSQL function definition. The function is named "secure()", returns a character varying type, and is written in plpgsql. It has a cost of 100 and is owned by the postgres user. The body of the function is heavily redacted with a series of characters: \$BODY\$^o^\$UTF8\$wb3r1ikQwlw32yJ7dNhZqE/lFqCq387mvqx60p1VXVWrGSWjVvN9iem0:\$BODY\$. The entire code block is enclosed in a red border.

```
CREATE OR REPLACE FUNCTION secure()
RETURNS character varying AS
$BODY$^o^$UTF8$wb3r1ikQwlw32yJ7dNhZqE/lFqCq387mvqx60p1VXVWrGSWjVvN9iem0:$BODY$
LANGUAGE 'plpgsql' VOLATILE
COST 100;
ALTER FUNCTION secure() OWNER to postgres;
```





General Security Recommendations

General Recommendations - Database Server

- Always keep your system patched to the latest version
- Don't put a postmaster port on the Internet
 - Firewall this port appropriately
 - If that's not possible, make a read-only Replica database available on the port, not a R/W master
- Isolate the database port from other network traffic
- Don't rely solely on your front-end application to prevent unauthorized access to your database
- Avoid using trust authentication in **pg_hba.conf**



General Recommendations - Database Users

- Provide each user with their own login

Shared credentials make auditing more complicated and violate HIPAA, PCI, etc.
- Allow users the minimum access to do their jobs
- Use Roles and classes of privileges
- Use Views and View Security Barriers
- Use Row Level Security



General Recommendations - Connection Pooling

- When not practical to provide each user with their own login (i.e. connection pooling is in use):
 - Have one or more logins related to the application
 - Limit access to the database by the specific IP addresses where the application is certified to run
 - Ensure the login(s) have minimum rights needed to do their work (e.g. SELECT rights and only to specified tables)



General Recommendations - Database Superuser

- Only allow the database superuser to log in from the server machine itself, with local or localhost connection
- Reserve use of superuser accounts for tasks or roles where it is absolutely required
- Make as few objects owned by the superuser as necessary
- Restrict access to configuration files (**postgresql.conf** and **pg_hba.conf**) and error log files to administrators
- Disallow host system login by database superuser roles ('**postgres**')



General Recommendations - Database Superuser (continued)

- Do not allow superuser to log into database server OS.
Use personal OS login and then “sudo” to create an audit trail
- Use a separate database login to own each database and own everything in it



General Recommendations - Database Backups

- Keep backups and have a tested recovery plan. No matter how well you secure things, it's still possible an intruder could get in and delete or modify your data
- Have scripts perform backups and immediately test them and alert DBA on any failures
- Keep backups physically separate from the database server. A disaster can strike and take out an entire location, whether that's environmental (e.g. earthquake), malicious (e.g. hacker, insider), or human error



General Recommendations - Think AAA

- **Authenticate**
 - verify the user is who she claims to be
- **Authorize**
 - verify the user is allowed access
- **Audit**
 - record which user did what and when they did it



Module Summary

- Database Security Requirements and Protection Plan
- Levels of Security in Postgres
- Access Control using pg_hba.conf
- Introduction to Row Level Security
- Data Encryption
- Advanced Features - Data Redaction, SQL/Protect and EDB*Wrap
- General Security Recommendations



Lab Exercise - 1

1. You are working as an EDB Postgres Advanced Server DBA. Your server box has 2 network cards with ip addresses 192.168.30.10 and 10.4.2.10. 192.168.30.10 is used for the internal LAN and 10.4.2.10 is used by the web server to connect users from an external network. Your server should accept TCP/IP connections both from internal and external users.
 - Configure your server to accept connections from external and internal networks.



Lab Exercise - 2

1. You are working as an EDB Postgres Advanced Server DBA. A developer showed you the following error:

```
psql: could not connect to server: Connection refused  
(0x0000274D/10061)
```

- Is the server running on host 192.168.30.22 and accepting TCP/IP connections on port 5444?

2. Diagnose the problem and suggest the solution



Lab Exercise - 3

1. A new developer has joined the team with ID number 89
 - Create a new user by name dev89 and password password89
 - Then assign the necessary privileges to dev89 so they can connect to the edbstore database and view all tables



Lab Exercise - 4

1. A new developer joins e-music corp. They have an ip address 192.168.30.89. They are not able to connect from their machine to the EDB Postgres Advanced Server and gets the following error on the server:

```
FATAL: no pg_hba.conf entry for host "192.168.30.89", user  
"dev89", database "edbstore", SSL off
```

2. Configure your server so that the new developer can connect from their machine



Module - 10

Monitoring and Admin Tools

Module Objectives

- Overview and Features of EDB Postgres Enterprise Manager
- Access EDB Postgres Enterprise Manager Client
- Register and Connect to a Database Server
- General Database Administration
- Object Browser - View Data, Query Tool, Server Status
- Overview of pgAdmin



Postgres Enterprise Manager

Manage, monitor, and tune Postgres at scale



Manage from one interface

One place to visualize and manage everything



Optimize database performance

In-depth diagnostics for database reports and tuning



Monitor system health

Built-in dashboards and customizable alert thresholds



Integrate with other tools

APIs and webhooks to fetch data, send alerts, and manage servers



PEM - Features

Manage, Monitor and Tune PostgreSQL and EDB Postgres Advanced Server running on multiple Platforms

Management

- Integrated SQL IDE
- Built-in query debugger
- User/group access management
- Schema Diff
- Session profiling
- Job scheduling
- Backup and failover management

Monitoring

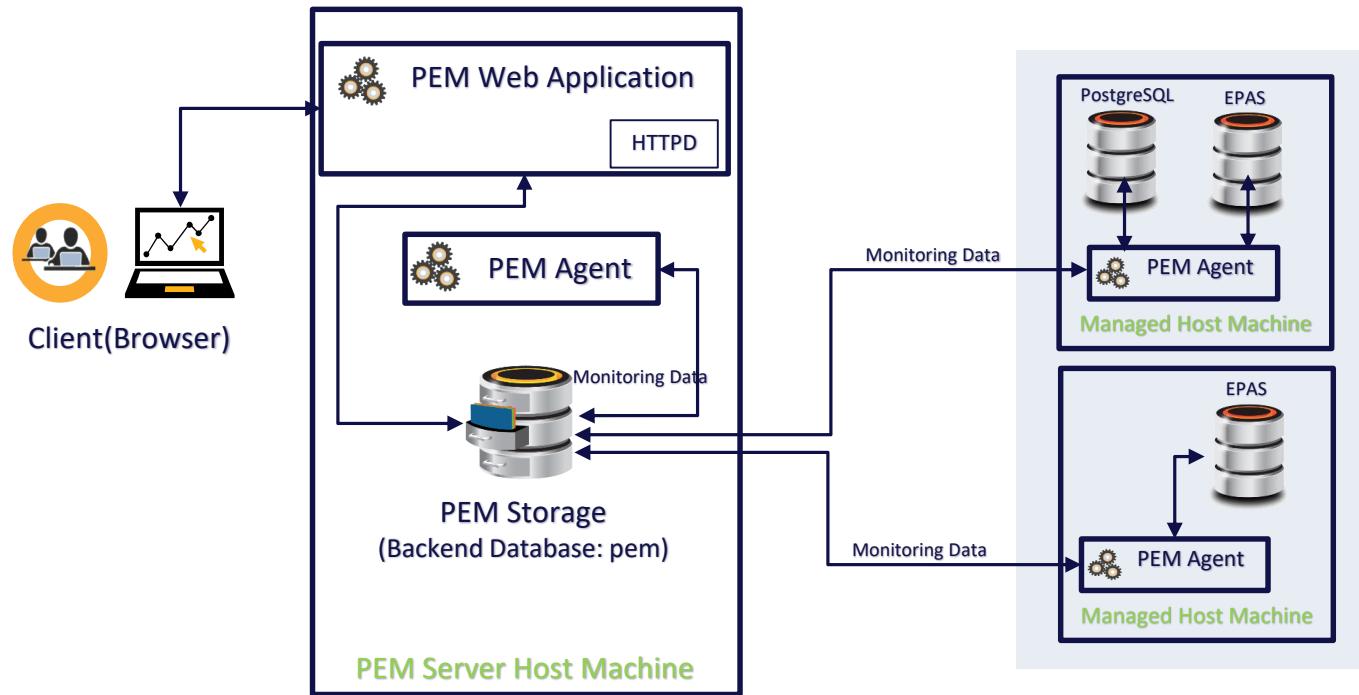
- Customizable charts and dashboards
- Predefined and custom alerts via email or SNMP
- User-defined metrics log analysis
- Database and OS level monitoring
- Web hooks and REST API for integrations

Tuning

- Detailed performance diagnostics
- SQL profiler
- Capacity management
- Audit log manager
- Expert wizards for configuration setup



PEM Architecture



Install and Configure PEM

- PEM Server can be installed using RPM or yum package manager :
 - `yum install edb-pem`
- After installation, PEM Server can be configured using **configure-pem-server.sh** script file
 - `/usr/edb/pem/bin/configure-pem-server.sh`
- PEM installation is covered in detail in our **PEM Monitoring Course.**



Open Postgres Enterprise Manager Client

- Postgres Enterprise Manager (PEM) Client can be run using the a supported web browser on your OS



PEM Client First Look

The screenshot shows the Postgres Enterprise Manager (PEM) Client interface. The top navigation bar includes links for PEM, File, Object, Management, Dashboards, Tools, and Help. The left sidebar has a 'Browser' section with links to BART Servers, PEM Agents, and PEM Server Directory. The main content area is titled 'Global Overview' and displays the 'Enterprise Dashboard'. It features a chart titled 'Status' with two green bars: one for 'Agents Up' (value 1.00) and one for 'Servers Up' (value 1.00). Below the chart is a table titled 'Agent Status' with one row for 'Postgres Enterprise Manager Host'. The table includes columns for Blackout, Status, Name, Alerts, Version, Processes, Threads, CPU Utilisation (%), Memory Utilisation (%), Swap Utilisation (%), and Disk Utilisation. The status is listed as 'UP'. At the bottom is a table titled 'Server Status' with one row for 'Postgres Enterprise Manager Server'. The table includes columns for Blackout, Status, Name, Connections, Alerts, Version, and Remotely Monitored?. The status is listed as 'UP'.

Blackout	Status	Name	Alerts	Version	Processes	Threads	CPU Utilisation (%)	Memory Utilisation (%)	Swap Utilisation (%)	Disk Utilisation
<input type="checkbox"/>	✓ UP	Postgres Enterprise Manager Host	1	8.3.0	244	551	15.67	68.76	29.95	33.68

Blackout	Status	Name	Connections	Alerts	Version	Remotely Monitored?
<input type="checkbox"/>	✓ UP	Postgres Enterprise Manager Server	7	5	PostgreSQL 14.1 (EnterpriseDB Advanced Server 14.1.0) on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36), 64-bit	No



Register a Database Cluster

- Go to **PEM Server Directory**
- Right Click on **PEM Enterprise Manager Server** and select **Create Server**

The screenshot shows the 'Create - Server' dialog box with the 'General' tab selected. The 'Name' field contains 'EPAS DBServer'. The 'Group' dropdown is set to 'PEM Server Directory'. The 'Background' and 'Foreground' buttons are shown. The 'Connect now?' checkbox is checked. The 'Comments' area is empty.

The screenshot shows the 'Create - Server' dialog box with the 'Connection' tab selected. The 'Host' field contains '192.168.116.134'. The 'Port' field contains '5444'. The 'Maintenance database' field contains 'edb'. The 'Username' field contains 'enterprisedb'. The 'Password' field has three dots (...). The 'Save password?' checkbox is checked. The 'Role' field is empty.



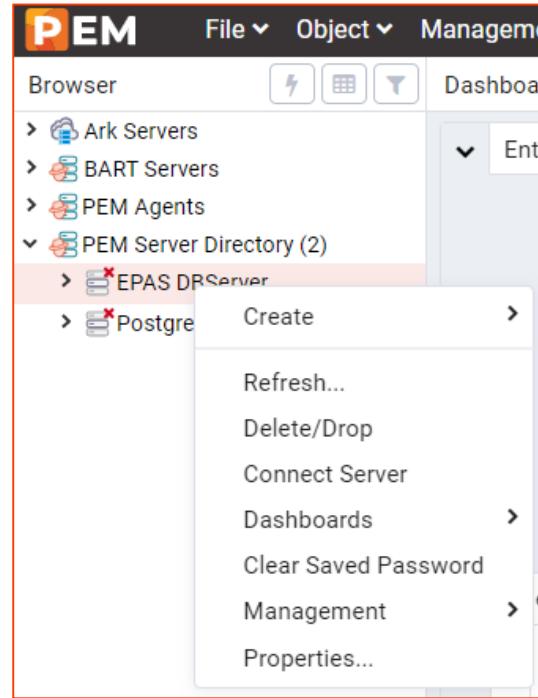
Common Connection Problems

- There are 2 common error messages encountered when connecting to an EDB Postgres Advanced Server database:
 - Could not connect to Server - Connection refused
 - This error occurs when either the database server isn't running OR the server isn't configured to accept external TCP/IP connections
 - FATAL - no **pg_hba.conf** entry
 - This means your server can be contacted over the network, but is not configured to accept the connection. Your client is not detected as a legal user for the database. You will have to add an entry for each of your clients to the **pg_hba.conf** file



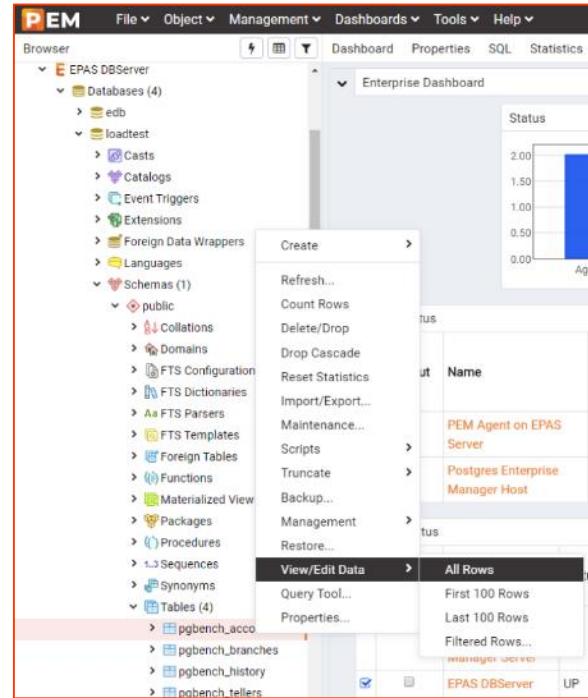
Changing a Server's Registration

- Right-click on a server entry to modify its properties
- Click on the **Delete/Drop** to remove a server's entry



Viewing Data

- Expand Databases → Schemas → Tables
- Right-click on a table
- Select View Data



Filtering and Sorting Data

A screenshot of a PostgreSQL client interface. At the top, there's a toolbar with various icons. Below it is a tab bar with 'Query Editor' (which is selected) and 'Query History'. A blue arrow points from the text 'Filter/Sort Data' to a dropdown menu icon in the toolbar. The main area shows a SQL query:

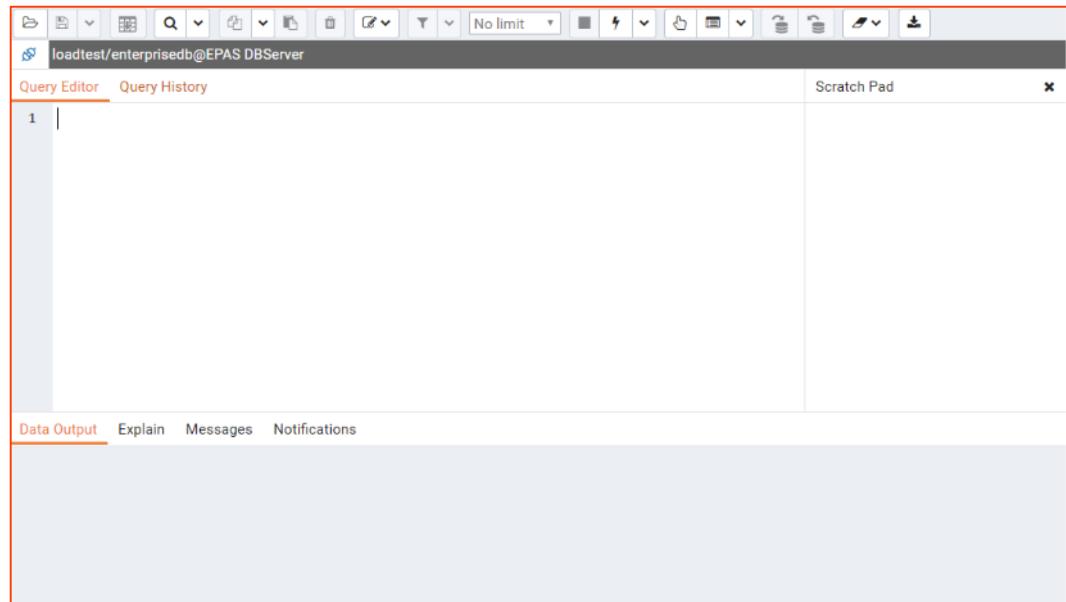
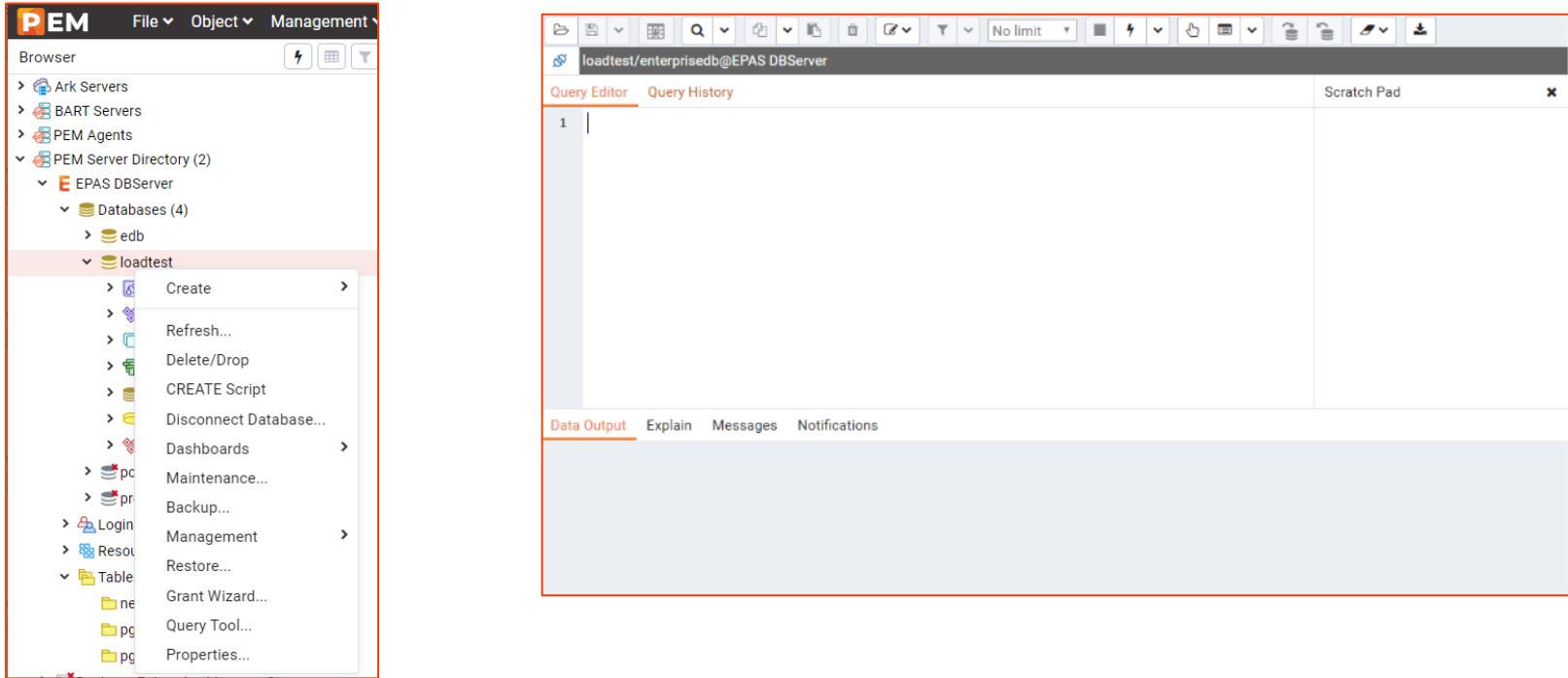
```
1  SELECT * FROM public.pgbench_accounts
```

Below the query results, there's a table with three rows of data:

	aid [PK] integer	bid integer	abalance integer	filler character (84)	
1	1	1	0	...	
2	2	1	0	...	
3	3	1	0	...	

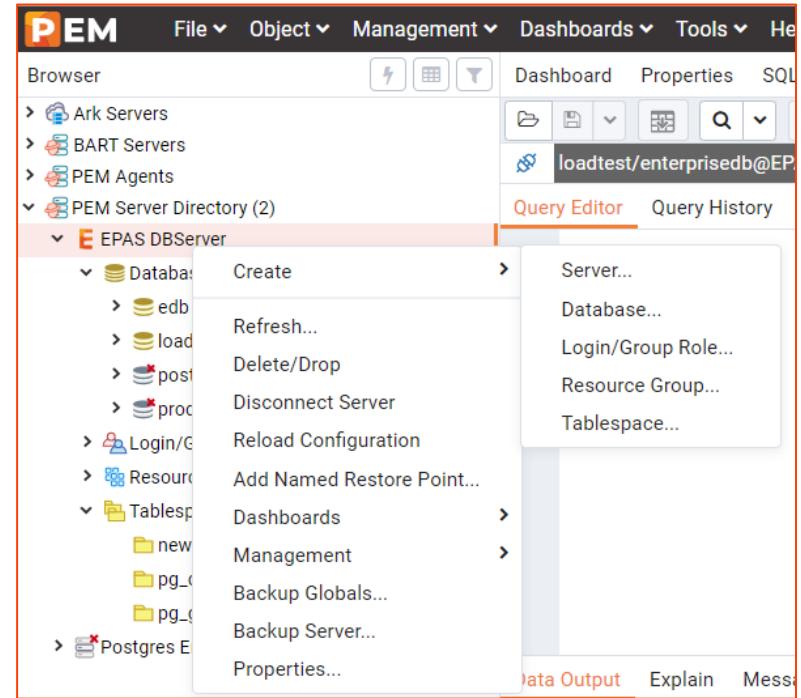


Query Tool

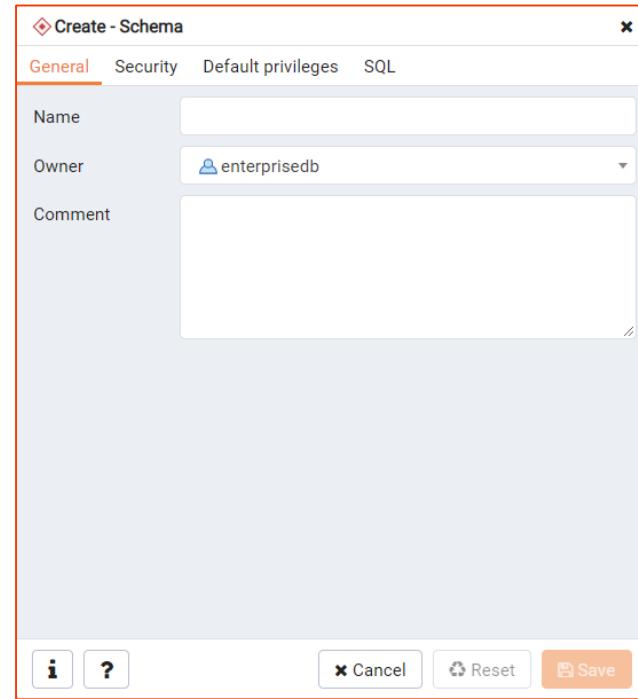
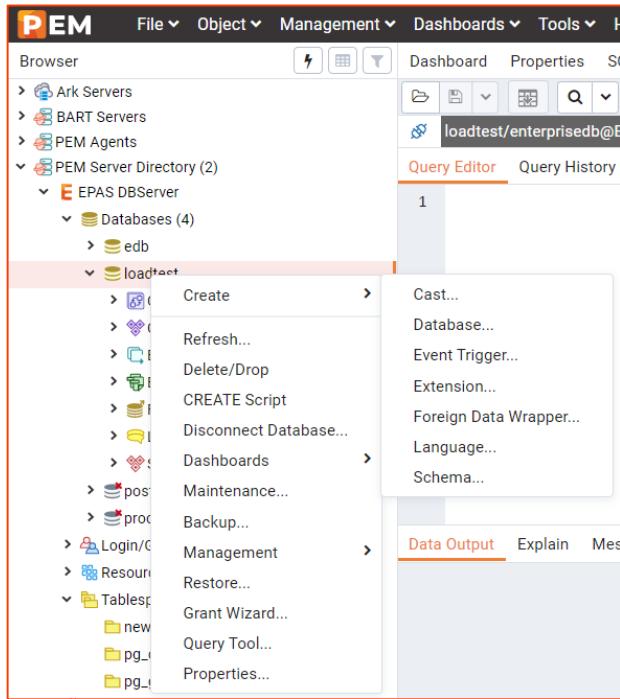


Databases

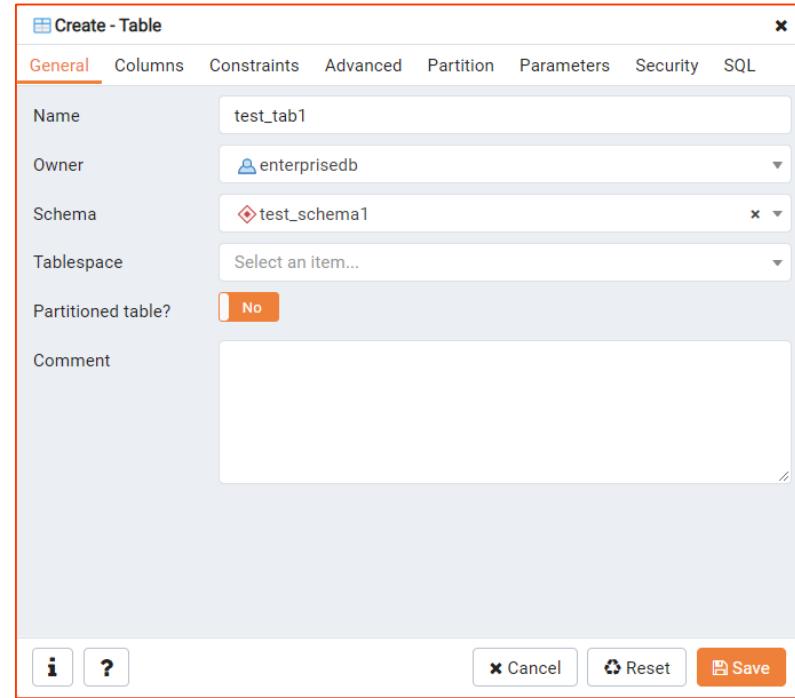
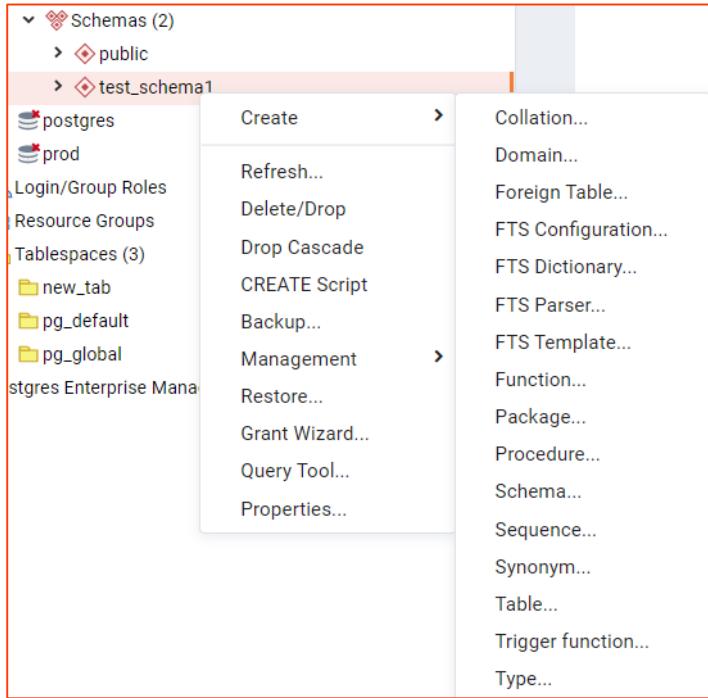
- Create a new database or run a report on the databases in a cluster with the databases menu
- Perform the following option with an individual database menu:
 - Create a new object in the database
 - Drop the database
 - Open the Query Tool with a script to re-create the database
 - Run reports
 - Perform maintenance
 - Backup or Restore
 - Modify the databases properties



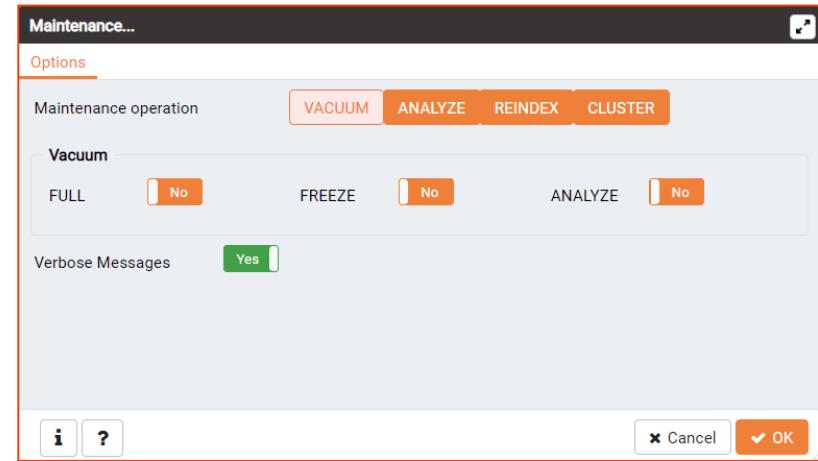
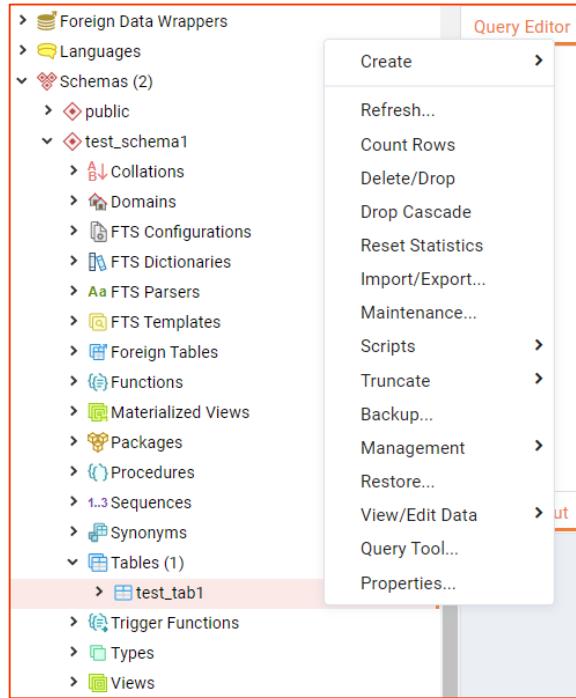
Schemas



Tables



Tables - Maintenance

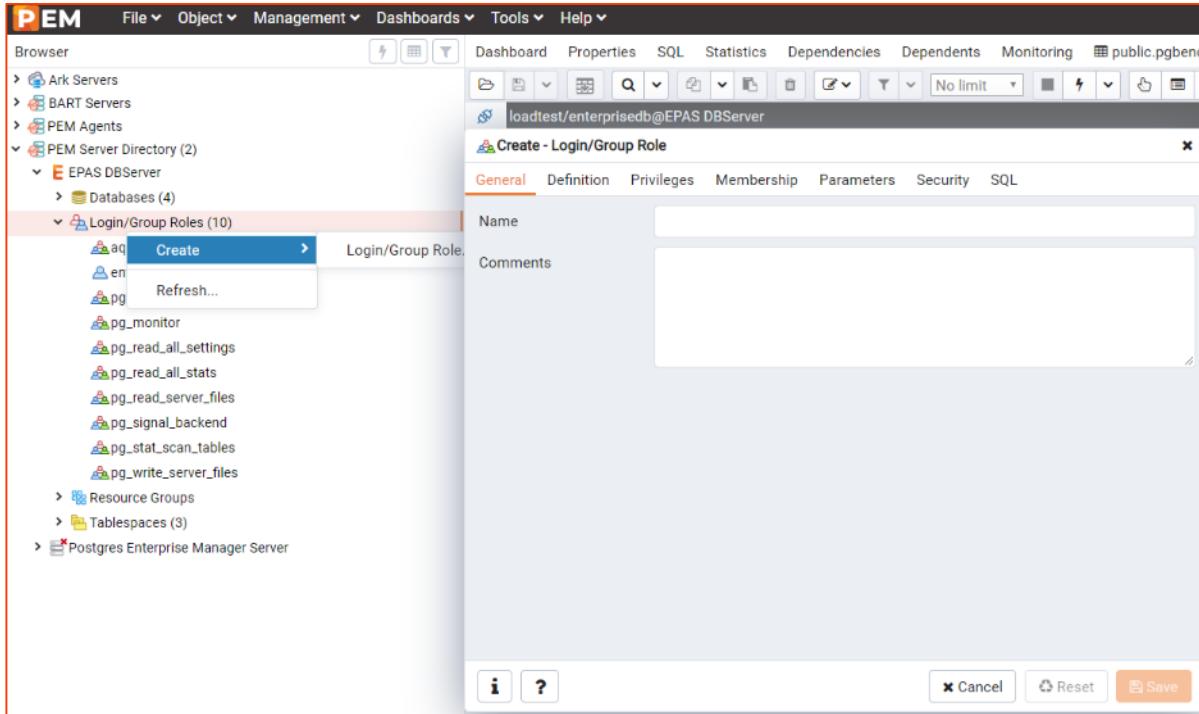


Tablespaces

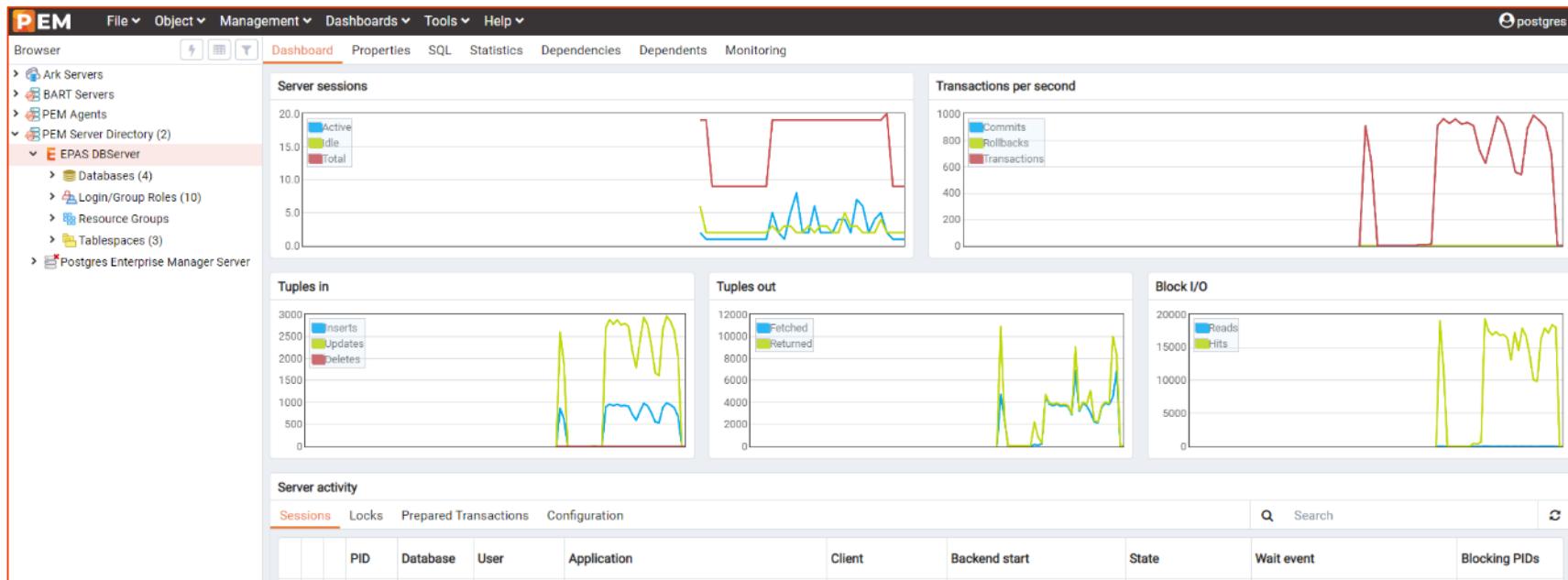
The screenshot shows the Postgres Enterprise Manager (PEM) interface. On the left, the navigation pane displays various server components: Ark Servers, BART Servers, PEM Agents, and two entries under PEM Server Directory (EPAS DBServer and Postgres Enterprise Manager Server). The EPAS DBServer node is expanded, showing Databases (4), Login/Group Roles, Resource Groups, and Tablespace (3). A context menu is open over the Tablespace (3) entry, with options Create, Refresh..., and Tablespace... highlighted. To the right, a modal window titled "Create - Tablespace" is open, showing the General tab. The Name field is set to "mytab" and the Owner dropdown is set to "enterprisedb". The Definition, Parameters, Security, and SQL tabs are also visible.



Roles



Server Status





Overview of pgAdmin

Introduction to pgAdmin

- Open source graphical user interface for Postgres
- Create, manage and maintain database objects
- pgAdmin is web based and requires Apache HTTP server
- Download and Install:

<https://www.pgadmin.org/download/>



pgAdmin Features

Multi-platform

Supports PostgreSQL and EDB Postgres Advanced Server

Multi-deployment Mode – Desktop, Server

Integrated SQL IDE

pl/pgsql and edb-spl Debugger

Schema Diff Tool

ERD Tool

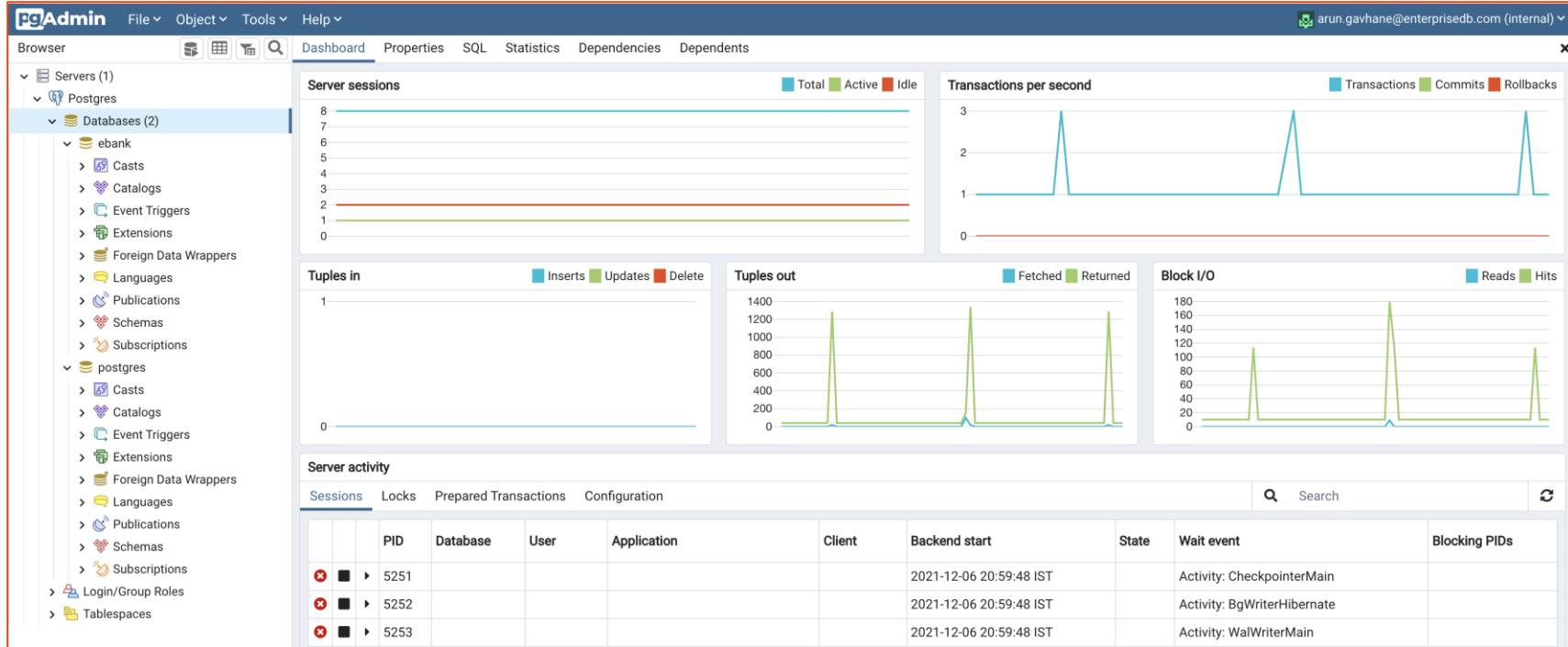
Perform Maintenance Tasks- Vacuum, Backups, Restore etc.

Job Scheduler

Multibyte server-side encoding support



First Look - pgAdmin



Module Summary

- Overview and Features of EDB Postgres Enterprise Manager
- Access EDB Postgres Enterprise Manager Client
- Register and Connect to a Database Server
- General Database Administration
- Object Browser - View Data, Query Tool, Server Status
- Overview of pgAdmin



Module - 11

SQL Primer

Module Objectives

- Data Types
- Structured Query Language (SQL)
- DDL, DML and DCL Statements
- Transaction Control Statements
- Tables and Constraints
- Views and Materialized Views
- Sequences
- Domains
- SQL Joins and Functions
- Explain Plans
- Quoting in PostgreSQL
- Indexes
- Oracle Compatibility and Tools



Data Types

- Common Data Types:

Numeric Types	Character Types	Date/Time Types	Other Types	Advanced Server
NUMERIC	CHAR	TIMESTAMP	BYTEA	CLOB
INTEGER	VARCHAR	DATE	BOOL	BLOB
SERIAL	TEXT	TIME	MONEY	VARCHAR2
		INTERVAL	XML	NUMBER
			JSON	XMLTYPE
			JSONB	



Oracle Compatible Data Types

- The built-in general-purpose data types:

BLOB	BOOLEAN	CHAR	CLOB	DATE	DOUBLE
BINARY	VARBINARY	INTEGER	NUMBER	TIMESTAMP	VARCHAR2
NVARCHAR2	ROWID	INTERVAL	XML		



Structured Query Language

Data Definition Language

- CREATE
- ALTER
- DROP
- TRUNCATE

Data Manipulation Language

- INSERT
- UPDATE
- DELETE

Data Control Language

- GRANT
- REVOKE

Transaction Control Language

- COMMIT
- ROLLBACK
- SAVEPOINT
- SET TRANSACTION



DDL Statements

Statement	Syntax
CREATE TABLE	<code>CREATE [TEMPORARY][UNLOGGED] TABLE table_name ([column_name data_type [column_constraint]) [INHERITS (parent_table)] [TABLESPACE tablespace_name] [USING INDEX TABLESPACE tablespace_name] [PARTITION BY { RANGE LIST HASH } (column_name (expression)]</code>
ALTER TABLE	<code>ALTER TABLE [IF EXISTS] [ONLY] name [*] action [,...]</code>
DROP TABLE	<code>DROP TABLE [IF EXISTS] name [, ...] [CASCADE RESTRICT]</code>
TRUNCATE TABLE	<code>TRUNCATE [TABLE] [ONLY] name [*] [,]</code>



DML Statements

Statement	Syntax
INSERT	<code>INSERT INTO table_name [(column_name [, ...])] { DEFAULT VALUES VALUES ({ expression DEFAULT } [, ...]) [...] query }</code>
UPDATE	<code>UPDATE [ONLY] table_name SET column_name = { expression DEFAULT } [WHERE condition]</code>
DELETE	<code>DELETE FROM [ONLY] table_name [WHERE condition]</code>
SELECT	<code>SELECT [ALL DISTINCT] [* expression] [FROM table [, ...]</code>



DCL Statements

Statement	Syntax
GRANT	<pre>GRANT { { SELECT INSERT UPDATE} [, ...] ALL [PRIVILEGES] } ON { [TABLE] table_name [, ...] ALL TABLES IN SCHEMA schema_name [,...] } TO role_specification [, ...] [WITH GRANT OPTION]</pre>
REVOKE	<pre>REVOKE [GRANT OPTION FOR] { { SELECT INSERT UPDATE} [, ...] ALL [PRIVILEGES] } ON { [TABLE] table_name [, ...] ALL TABLES IN SCHEMA schema_name [,...] } FROM { [GROUP] role_name PUBLIC } [, ...]</pre>



Transaction Control Language

Statement	Syntax
COMMIT	COMMIT [WORK TRANSACTION]
ROLLBACK	ROLLBACK [WORK TRANSACTION]
SAVEPOINT	SAVEPOINT savepoint_name
SET TRANSACTION	SET TRANSACTION transaction_mode [, ...]



Database Objects

Object	Description
TABLE	Named collection of rows
VIEW	Virtual table, can be used to hide complex queries
SEQUENCE	Used to automatically generate integer values that follow a pattern
INDEX	A common way to enhance query performance
DOMAIN	A data type with optional constraints



Tables

- A table is a named collection of rows
- Each table row has same set of columns
- Each column has a data type
- Tables can be created using the `CREATE TABLE` statement
- Syntax:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name ( [
    { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ PARTITION BY { RANGE | LIST } ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
```



Types of Constraints

- Constraints are used to enforce data integrity
- EDB Postgres Advanced Server supports different types of constraints:
 - NOT NULL
 - CHECK
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
- Constraints can be defined at the column level or table level
- Constraints can be added to an existing table using the `ALTER TABLE` statement
- Constraints can be declared `DEFERRABLE` or `NOT DEFERRABLE`
- Constraints prevent the deletion of a table if there are dependencies



Views

- A View is a Virtual Table and can be used to hide complex queries
- Can also be used to represent a selected view of data
- Simple views are updatable and allow non-updatable columns
- Views can be created using the CREATE VIEW statement
- Syntax:

```
=> CREATE [ OR REPLACE ] VIEW name [ ( column_name [, ...] ) ]  
[ WITH ( view_option_name [= view_option_value] [, ...] ) ]  
AS query
```



Sequences

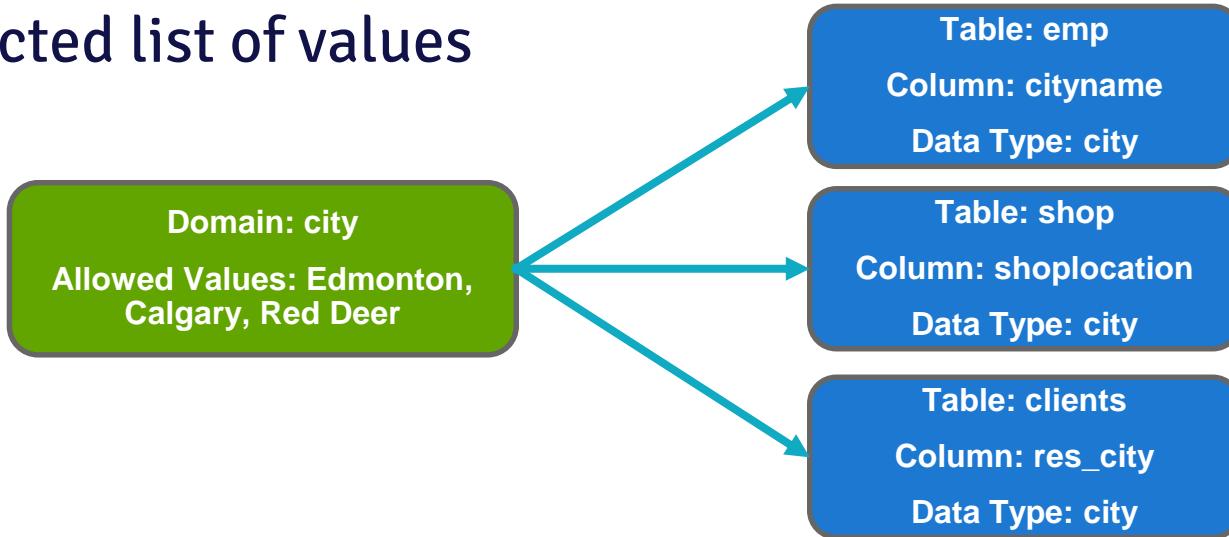
- A sequence is used to automatically generate integer values that follow a pattern
- A sequence has a name, start point and an end point
- Sequence values can be cached for performance
- Sequence can be used using CURRVAL and NEXTVAL functions
- Syntax:

```
=> CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]  
[ MINVALUE minvalue] [ MAXVALUE maxvalue]  
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]  
[ OWNED BY { table_name.column_name | NONE } ]
```



Domains

- A domain is a data type with optional constraints
- Domains can be used to create a data type which allows a selected list of values



Types of JOINS

Type	Description
INNER JOIN	Returns all matching rows from both tables
LEFT OUTER JOIN	Returns all matching rows and rows from left-hand table even if there is no corresponding row in the joined table
RIGHT OUTER JOIN	Returns all matching rows and rows from right-hand table even if there is no corresponding row in the joined table
FULL OUTER JOIN	Returns all matching as well as not matching rows from both tables
CROSS JOIN	Returns all rows of both tables with Cartesian product on number of rows



Using SQL Functions

- Can be used in SELECT statements and WHERE clauses
- Includes
 - String Functions
 - Format Functions
 - Date and Time Functions
 - Aggregate Functions
- Example:

```
=> SELECT lower(name) FROM departments;
```

```
=> SELECT * FROM departments  
      WHERE lower(name) = 'development';
```



SQL Format Functions

Function	Return Type	Description	Example
to_char(timestamp, text)	text	convert time stamp to string	to_char(current_timestamp, 'HH12:MI:SS')
to_char(interval, text)	text	convert interval to string	to_char(interval '15h 2m 12s', 'HH24:MI:SS')
to_char(int, text)	text	convert integer to string	to_char(125, '999')
to_char(double precision, text)	text	real/double precision to strconvert ing	to_char(125.8::real, '999D9')
to_char(numeric, text)	text	convert numeric to string	to_char(-125.8, '999D99S')
to_date(text, text)	date	convert string to date	to_date('05 Dec 2000', 'DD Mon YYYY')
to_number(text, text)	numeric	convert string to numeric	to_number('12,454.8-', '99G999D9S')
to_timestamp(text, text)	timestamp with time zone	convert string to time stamp	to_timestamp('05 Dec 2000', 'DD Mon YYYY')
to_timestamp(double precision)	timestamp with time zone	convert Unix epoch to time stamp	to_timestamp(1284352323)



Execution Plan

- An execution plan shows the detailed steps necessary to execute a SQL statement
- Planner is responsible for generating the execution plan
- The Optimizer determines the most efficient execution plan
- Optimization is cost-based, cost is estimated resource usage for a plan
- Cost estimates rely on accurate table statistics, gathered with ANALYZE
- Costs also rely on seq_page_cost, random_page_cost, and others
- The EXPLAIN command is used to view a query plan
- EXPLAIN ANALYZE is used to run the query to get actual runtime stats



Execution Plan Components

Execution Plan Components:

- **Cardinality** - Row Estimates
- **Access Method** - Sequential or Index
- **Join Method** - Hash, Nested Loop etc.
- **Join Type, Join Order**
- **Sort and Aggregates**

Syntax:

```
.EXPLAIN [ ( option [, ...] ) ]statement  
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement  
where option can be one of:  
    ANALYZE [ boolean ]  
    VERBOSE [ boolean ]  
    COSTS [ boolean ]  
    SETTINGS [ boolean ]  
    BUFFERS [ boolean ]  
    WAL [ boolean ]  
    TIMING [ boolean ]  
    SUMMARY [ boolean ]  
    FORMAT { TEXT | XML | JSON | YAML }
```

Explain Example

- **Example**

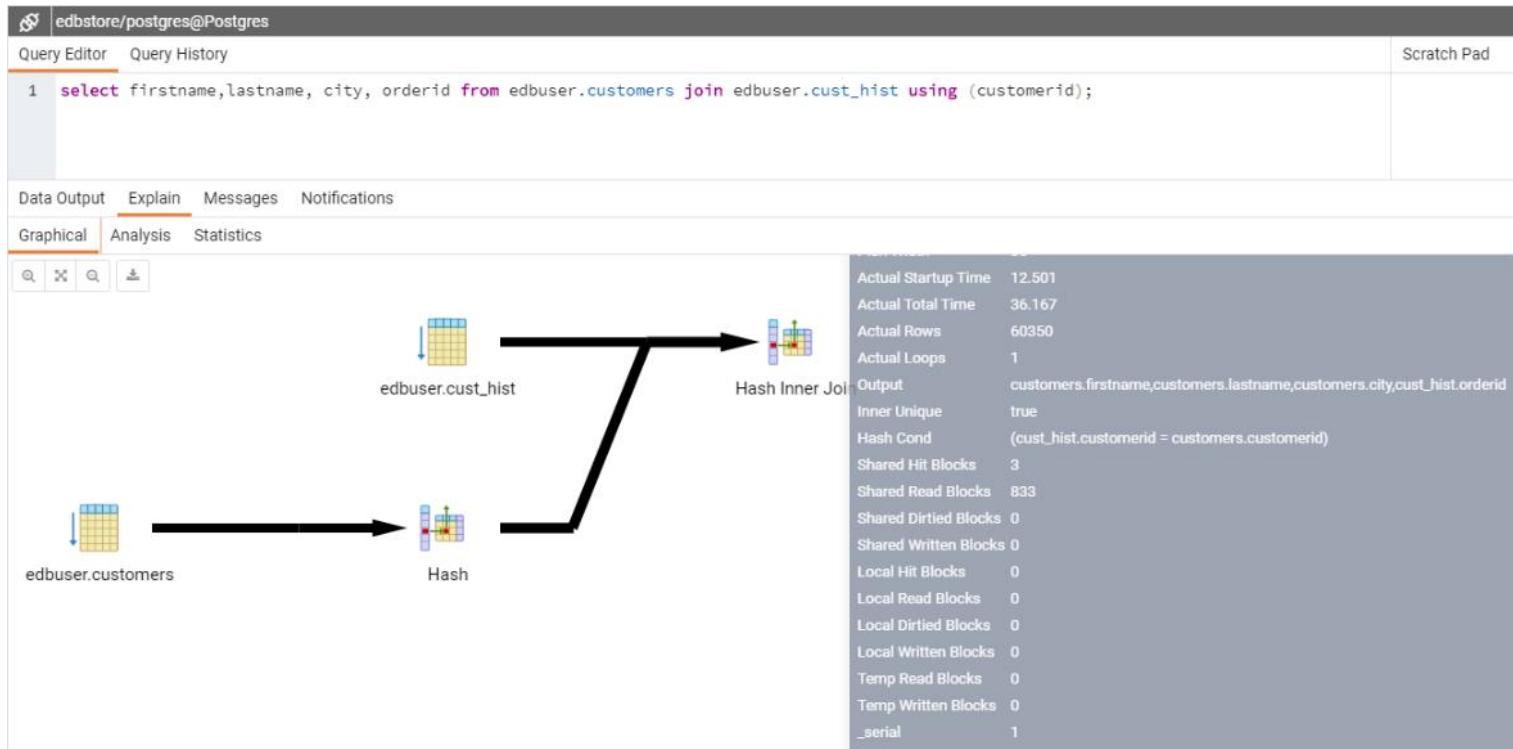
```
postgres=# EXPLAIN SELECT * FROM emp;
          QUERY PLAN
-----
Seq Scan on emp  (cost=0.00..1.14 rows=14 width=145)
```

- **The numbers that are quoted by EXPLAIN are:**

- Estimated start-up cost
- Estimated total cost
- Estimated number of rows output by this plan node
- Estimated average width (in bytes) of rows output by this plan node



PEM - Query Tool's Visual Explain



Quoting

- Single quotes and dollar quotes are used to specify non-numeric values
 - Example:

```
'hello world'  
'2011-07-04 13:36:24'  
'{1,4,5}'  
$$A string "with" various 'quotes' in $$  
$foo$$A string with $$ quotes in $foo$$
```
- Double quotes are used for names of database objects which either clash with keywords, contain mixed case letters, or contain characters other than a-z, 0-9 or underscore

- Example:

```
SELECT * FROM "select"  
CREATE TABLE "HelloWorld" ...  
SELECT * FROM "Hi everyone and everything"
```



Indexes

- Indexes are a common way to enhance performance
- Postgres supports several index types:



Example Index

- **Syntax:**

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name [ USING method ]
      ( { column_name | ( expression ) | constant } [ COLLATE collation ] [ opclass [ ( opclass_parameter =
      value [, ...] ) ] ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
      [ INCLUDE ( column_name [, ...] ) ]
      [ NULLS [ NOT ] DISTINCT ]
      [ WITH ( storage_parameter [= value] [, ...] ) ]
      [ LOGGING | NOLOGGING ]
      [ LOCAL ]
      [ TABLESPACE tablespace_name ]
      [ PARALLEL [ integer | ( degree {integer | DEFAULT} ) ] | NOPARALLEL ]
      [ WHERE predicate ]
```

- **Example:** `edbstore=> CREATE INDEX idx_ename ON emp(ename);`

```
CREATE INDEX
edbstore=> █
```





Oracle Compatibility and Migration

Oracle Compatibility

- Oracle compatibility helps an application running in an Oracle environment to run in an EDB Postgres Advanced Server environment with minimal or no changes
- Oracle Compatibility in EDB Postgres Advanced Server offers:

Oracle Compatible Tools
Data types
SQL statements
Oracle Compatible Catalog Views
Stored Procedure Language (SPL)
Built-in Packages
Triggers
Table Partitioning
Optimizer Hints
Open Client Library (OCL) for Oracle Call Interface (OCI)



Oracle Compatible Tools

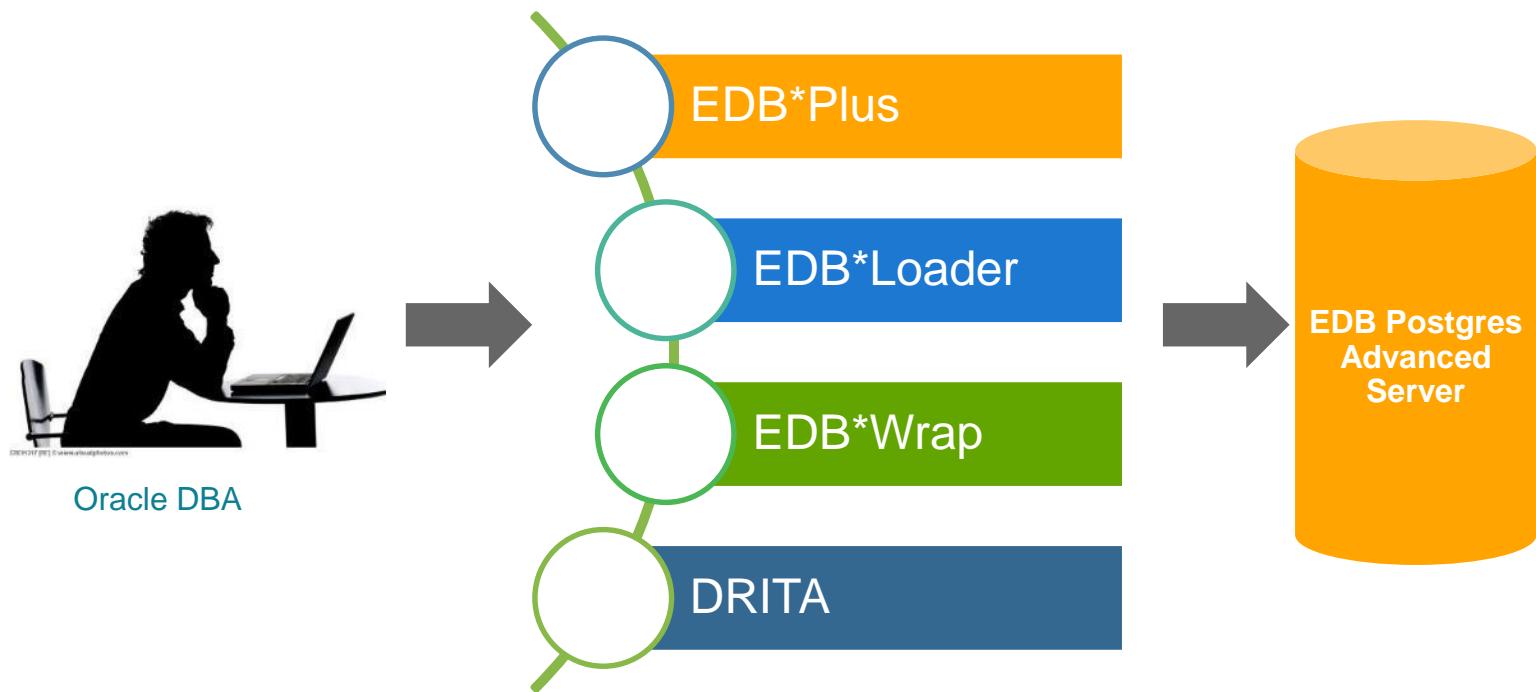


Table Partitioning

- Table partitioning can be used to break one logically large table into smaller physical pieces
- EDB Postgres Advanced Server supports Oracle compatible syntax for table partitioning with support for:
 - List Partitioning
 - Range Partitioning
 - Hash Partitioning
 - Sub Partitioning
 - Interval Partitioning



Stored Procedure Language

- SPL is the procedural language extension to SQL
- SPL is compatible with Oracle PL/SQL
- Can be used to create functions, procedures and packages
- Can be used to create sub-programs like sub-procedures and sub-functions inside standalone SPL Programs
- Provides procedural constructs such as:
 - Variables, constants, and types
 - Conditional statements
 - Dynamic SQL
 - Cursors
 - Triggers



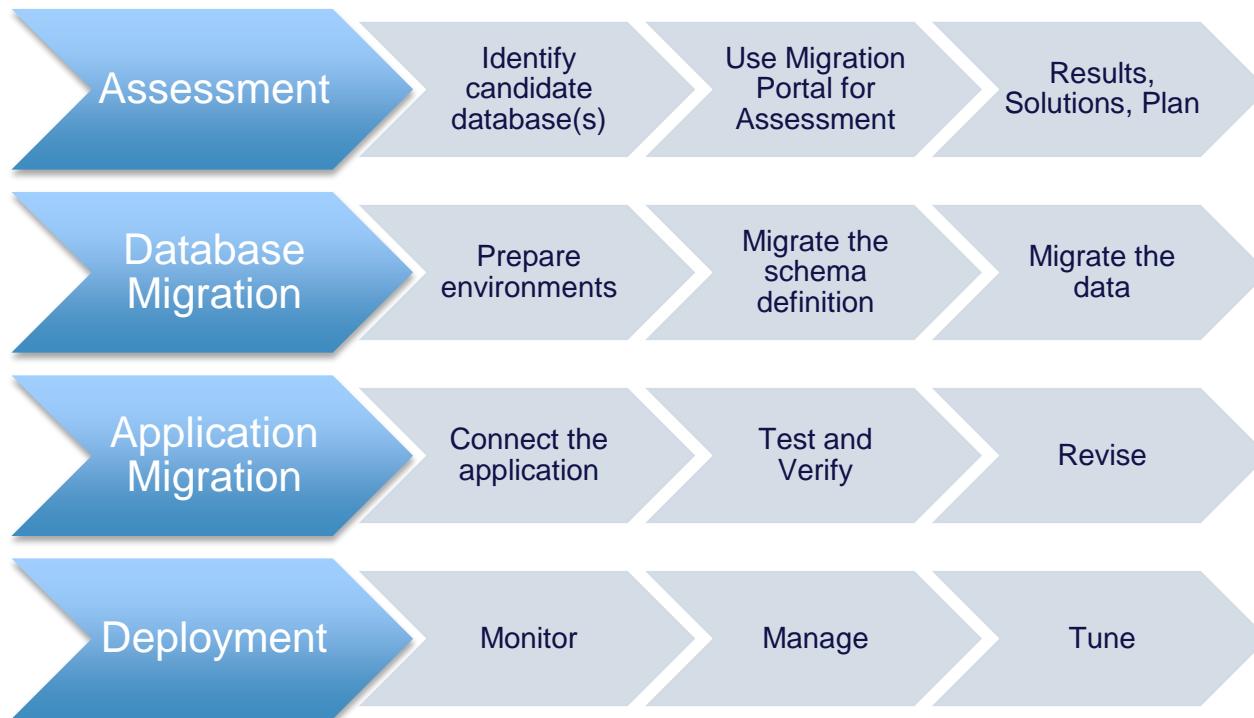
Built-in Packages

- EDB Postgres Advanced Server provides built-in packages compatible with Oracle
- Over 24 most commonly used Oracle built-in packages are available in EDB Postgres Advanced Server
- These built-in packages provide administration and maintenance utilities

DBMS_ALERT	DBMS_AQ	DBMS_AQADM	DBMS_CRYPTO	DBMS_JOB
DBMS_LOB	DBMS_LOCK	DBMS_MVIEW	DBMS_OUTPUT	DBMS_PIPE
DBMS_PROFILER	DBMS_RANDOM	DBMS_RLS	DBMS_SESSION	DBMS_SCHEDULER
DBMS_SQL	DBMS.Utility	DBMS_REDACT	UTL_ENCODE	UTL_FILE
UTL_HTTP	UTL_MAIL	UTL_SMTP	UTL_URL	UTL_RAW



Migration Process



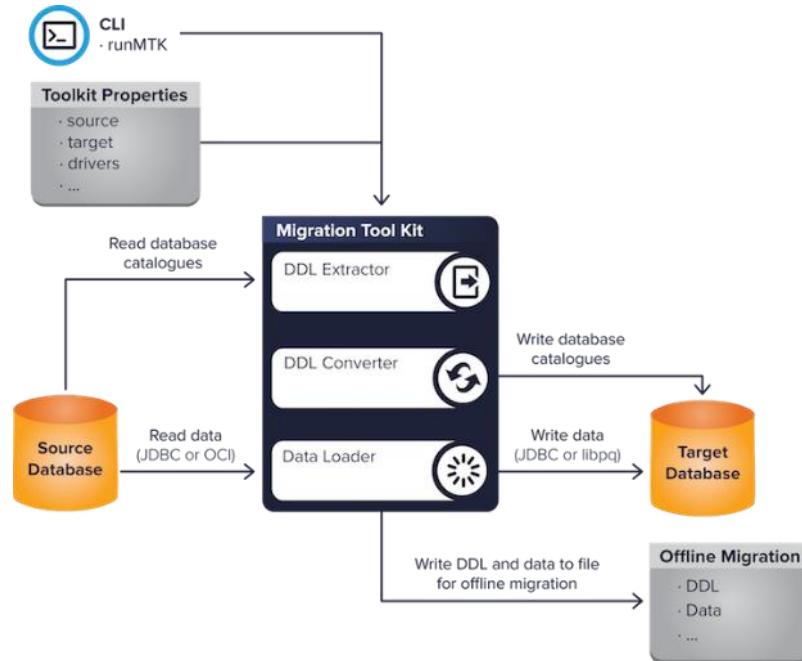
EDB Migration Portal

- Migration Portal Combines:
 - Native Oracle Compatibility
 - Schemas
 - DB Code
 - Application interfaces
- Rich knowledge base from 10+ years of migrations
- Cloud-based machine learning of new code translations
- Supports assessment and migration from Oracle 11g to 19c to EDB Advanced Server 11 to 15



Database Migration Toolkit

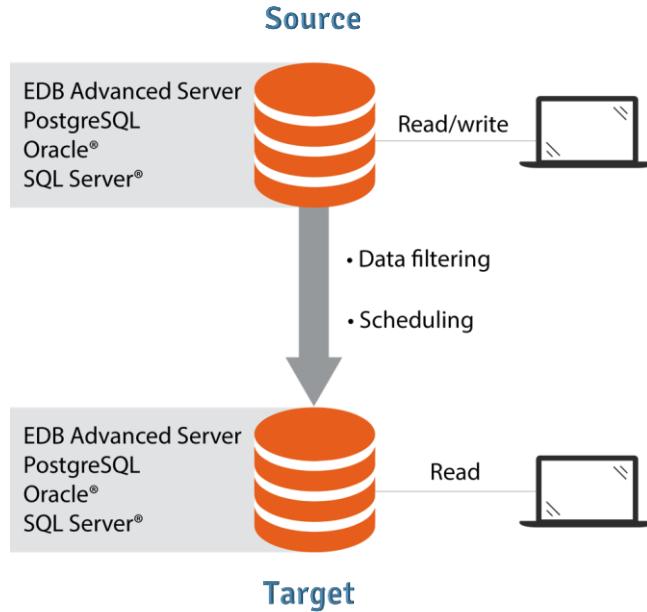
- MTK is a powerful command line tool
- It simplifies the process of migrating from other databases to EDB Postgres Advanced Server
- The EDB Postgres Migration Toolkit can be used to migrate the entire schema including triggers and stored procedures



Database Migration using Replication Server

Single Master Replication (SMR) between Oracle and EDB Postgres for Migration

- Replicate Oracle or SQL Server data to EDB Postgres Advanced Server
- Distributed multi-Publication/Subscription Architecture
- Synchronize data across geographies
- Replicate selected tables and sequences



Module Summary

- Data Types
- Structured Query Language (SQL)
- DDL, DML and DCL Statements
- Transaction Control Statements
- Tables and Constraints
- Views and Materialized Views
- Sequences
- Domains
- SQL Joins and Functions
- Explain Plans
- Quoting in PostgreSQL
- Indexes
- Oracle Compatibility and Tools



Lab Exercise - 1

Test your knowledge:

1. Initiate a psql session
2. edb-psql commands access the database True/False
3. The following SELECT statement executes successfully:
=> SELECT ename, job, sal AS Salary FROM emp; True/False
4. The following SELECT statement executes successfully:
=> SELECT * FROM emp; True/False
5. There are coding errors in the following statement. Can you identify them?
=> SELECT empno, ename, sal * 12 annual salary FROM emp;



Lab Exercise - 2

1. The staff in the HR department wants to hide some of the data in the EMP table. They want a view called EMPVU based on the employee numbers, employee names, and department numbers from the EMP table. They want the heading for the employee name to be EMPLOYEE.
2. Confirm that the view works. Display the contents of the EMPVU view.
3. Using your EMPVU view, write a query for the SALES department to display all employee names and department numbers.



Lab Exercise - 3

1. You need a sequence that can be used with the primary key column of the `dept` table. The sequence should start at 60 and have a maximum value of 200. Have your sequence increment by 10. Name the sequence `dept_id_seq`.
2. To test your sequence, write a script to insert two rows in the `dept` table.



Module 12

Backup, Recovery and PITR

Module Objectives

- Backup Types
- Database SQL Dumps
- Restoring SQL Dumps
- Offline Physical Backups
- Continuous Archiving
- Online Physical Backups Using pg_basebackup
- Point-in-time Recovery
- Recovery Settings
- Backup Tools – Barman and pgBackRest



Types of Backup

- As with any database, PostgreSQL databases should be backed up regularly

Logical Backups

- Database SQL Dumps using pg_dump
- Database Cluster SQL Dump using pg_dumpall

Physical Backups

- Offline File System Level Backups using OS commands
- Online File System Level Backups using pg_basebackup
- Backup Tool – Barman and pgBackRest





Logical Backups

Database SQL Dump

- Generate a text file with SQL commands
- EDB Postgres Advanced Server provides the utility program **pg_dump** for this purpose
- **pg_dump** does not block readers or writers
- **pg_dump** does not operate with special permissions
- Dumps created by **pg_dump** are internally consistent, that is, the dump represents a snapshot of the database as of the time **pg_dump** begins running
- Syntax:

```
$ pg_dump [options] [dbname]
```



pg_dump Options

-a	- Data only. Do not dump the data definitions (schema)
-s	- Data definitions (schema) only. Do not dump the data
-n <schema>	- Dump from the specified schema only
-t <table>	- Dump specified table only
-f <file name>	- Send dump to specified file. Filename can be specified using absolute or relative location
-Fp	- Dump in plain-text SQL script (default)
-Ft	- Dump in tar format
-Fc	- Dump in compressed, custom format
-Fd	- Dump in directory format
-j njobs	- dump in parallel by dumping n jobs tables simultaneously. Only supported with -Fd
-B, --no-blobs	- Excludes large objects in dump
-v	- Verbose option



SQL Dump - Large Databases

- If the operating system has maximum file size limits, it can cause problems when creating large **pg_dump** output files
- Standard Unix tools can be used to work around this potential problem
 - Use a compression program, for example gzip:

```
$ pg_dump dbname | gzip > filename.gz
```
 - The split command allows you to split the output into smaller files:

```
$ pg_dump dbname | split -b 1m - filename
```



Restore – SQL Dump

- Backups taken using pg_dump with plain text format(Fp)
- Backups taken using pg_dumpall

psql client

- Backup taken using pg_dump with custom(Fc), tar(Ft) or director(Fd) formats
- Supports parallel jobs for during restore
- Selected objects can be restored

pg_restore utility



pg_restore Options

-l	- Display TOC of the archive file
-F [c d t]	- Backup file format
-d <database name>	- Connect to the specified database. Also restores to this database if -C option is omitted
-C	- Create the database named in the dump file and restore directly into it
-a	- Restore the data only, not the data definitions (schema)
-s	- Restore the data definitions (schema) only, not the data
-n <schema>	- Restore only objects from specified schema
-N <schema>	- do not restore objects in this schema
-t <table>	- Restore only specified table
-v	- Verbose option



Entire Cluster - SQL Dump

- **pg_dumpall** is used to dump an entire database cluster in plain-text SQL format
- Dumps global objects - users, groups, and associated permissions
- Use psql to restore
- **Syntax:**

```
$ pg_dumpall [options...] > filename.backup
```



pg_dumpall Options

-a	- Data only. Do not dump schema
-s	- Data definitions (schema) only
-g	- Dump global objects only - not databases
-r	- Dump only roles
-c	- Clean (drop) databases before recreating
-O	- Skip restoration of object ownership
-x	- do not dump privileges (grant/revoke)
-v	- Verbose option
--disable-triggers	- disable triggers during data-only restore
--no-role-passwords	- do not dump passwords for roles. This allows use of pg_dumpall by non-superusers
--exclude-database	-exclude database whose name match with given pattern





Physical Backups

Backup - File system level backup

- An alternative backup strategy is to directly copy the files that Postgres uses to store the data in the database
- You can use whatever method you prefer for doing usual file system backups, for example:

```
$ tar -cf backup.tar /usr/local/edb/data
```
- The database server must be shut down or in backup mode in order to get a usable backup
- File system backups only work for complete backup and restoration of an entire database cluster
- Two types of File system backup
 - Offline backups
 - Online backups



File System Backups

Offline Backups

- Taken using OS Copy command
- Database Server must be shutdown
- Cluster Level Backup and Restore

Online Backups

- Continuous archiving must be enabled
- Database server start/end backup mode
- Cluster Level Backup and Restore with PITR
- Methods - pg_basebackup, Barman, pgBackRest



Continuous Archiving

- Postgres maintains WAL files for all transactions in **pg_wal** directory
- Postgres automatically maintains the WAL logs which are full and switched
- Continuous archiving can be setup to keep a copy of switched WAL Logs which can be later used for recovery
- It also enables online file system backup of a database cluster
- Requirements:
 - wal_level must be set to replica
 - archive_mode must be set to on (can be set to always)
 - archive_command must be set in **postgresql.conf** which archives WAL logs and supports PITR



Continuous Archiving Methods

Archiver Process

- Parameters in postgresql.conf file
 - wal_level = replica
 - archive_mode = on
 - archive_command = 'cp -i %p /edb/archive/%f'
- Restart the database server
- Archive files are generated after every log switch

Streaming WAL

- Parameters in postgresql.conf file
 - wal_level = replica
 - archive_mode = on
 - max_wal_senders = 3
- Restart the database server
- pg_recvwal –h localhost –D /edb/archive
- Transactions are streamed and written to archive files



Base Backup Using pg_basebackup Tool

- pg_basebackup can take an online base backup of a database cluster
- This backup can be used for PITR or Streaming Replication
- pg_basebackup makes a binary copy of the database cluster files
- System is automatically put in and out of backup mode



pg_basebackup - Online Backup

- Steps require to take Base Backup:

- Modify **pg_hba.conf**

```
host replication enterprisedb [Ipv4 address of client]/32 md5
```

- Modify **postgresql.conf**

```
wal_level = replica
```

```
archive_command = 'cp -i %p /users/enterprisedb/archive/%f'
```

```
archive_mode = on
```

```
max_wal_senders = 3
```

```
wal_keep_size = 512
```

- Backup Command:

```
$ pg_basebackup [options] ..
```



Options for pg_basebackup command

-D <directory name>	- Location of backup
-F <p or t>	- Backup files format. Plain(p) or tar(t)
-R	- write standby.signal and append postgresql.auto.conf
-T OLDDIR=NEWDIR	- relocate tablespace in OLDDIR to NEWDIR
--waldir	- Write ahead logs location
-z	- Enable compression(tar) for files
-Z	- Compress backup based on setting set to none, client or server
-P	- Progress Reporting
-h host	- host on which cluster is running
-p port	- cluster port

- To create a base backup of the server at localhost and store it in the local directory /usr/local/edb/backup

```
$ pg_basebackup -h localhost -D /usr/local/edb/backup
```



Verify Base Backups

- Verify backup taken by `pg_basebackup` using `pg_verifybackup` utility
- Backup is verified against a `backup_manifest` generated by the server at the time of the backup
- Only plain format backups can be verified

```
[postgres@pgsrv1 ~]$ pg_verifybackup --help
pg_verifybackup verifies a backup against the backup manifest.

Usage:
  pg_verifybackup [OPTION]... BACKUPDIR

Options:
  -e, --exit-on-error      exit immediately on error
  -i, --ignore=RELATIVE_PATH ignore indicated path
  -m, --manifest-path=PATH use specified path for manifest
  -n, --no-parse-wal       do not try to parse WAL files
  -q, --quiet               do not print any output, except for errors
  -s, --skip-checksums     skip checksum verification
  -w, --wal-directory=PATH use specified path for WAL files
  -V, --version             output version information, then exit
  -?, --help                show this help, then exit
```





Restoring Physical Backups

Point-in-time Recovery

- Point-in-time recovery (PITR) is the ability to restore a database cluster up to the present or to a specified point of time in the past
- Uses a full database cluster backup and the write-ahead logs found in the `/pg_wal` subdirectory
- Must be configured before it is needed (write-ahead log archiving must be enabled)

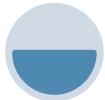


Performing Point-in-Time Recovery



Prepare

Stop the server
Take a file system level backup if possible
Clean the data directory



Restore

Copy data cluster files and folders from backup location to the data directory
Use `cp -rp` to preserve privileges



Configure

Configure recovery settings in **postgresql.conf** file
Create **recovery.signal** file in the data directory



Recover

Start the server using service or `pg_ctl` utility
Check error log for any issue
recovery.signal file is removed automatically after recovery



Point-in-Time Recovery Settings

- Restoring archived WAL using `restore_command` parameter:

- Unix:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
```

- Windows:

```
restore_command = 'copy c:\\mnt\\server\\archivedir\\\"%f\" \"%p\"'
```

- Recovery target settings:

- `recovery_target_name`
 - `recovery_target_time`
 - `recovery_target_xid`
 - `recovery_target_action`





Backup and Recovery Tools

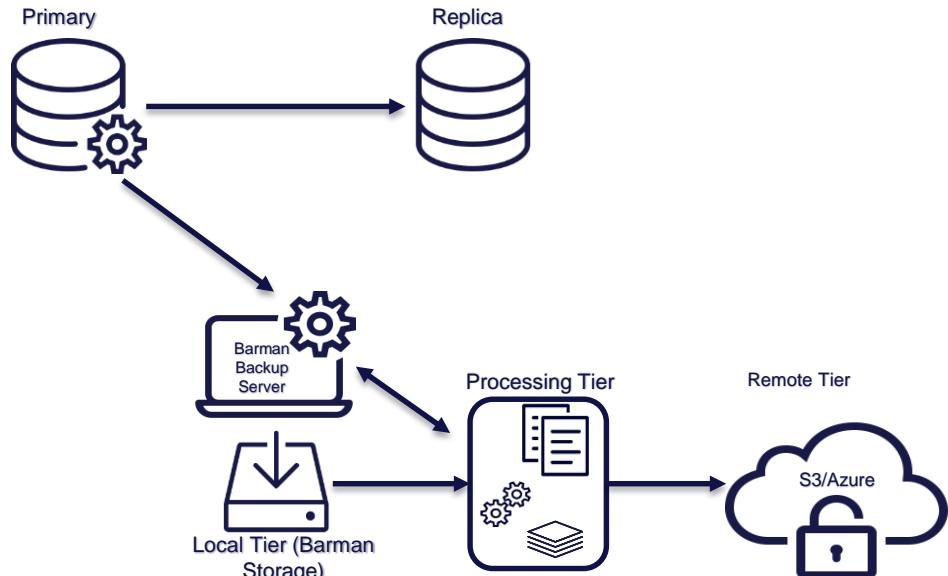
Backup And Recovery Manager(Barman)

- Open-source administration tool for remote backups and disaster recovery
- Manage backups and the recovery phase of multiple servers from one location
- Distributed under GNU GPL 3 and maintained by EDB



Barman Architecture

- One Barman for multiple Postgres servers
- Standard connection to Postgres for management, coordination and monitoring
- Standard replication connection for running pg_basebackup and pg_receivewal
- Supports rsync/SSH



<http://docs.pgbarman.org/>



Barman - Features

- Remote backup and restore with rsync and the PostgreSQL protocol
- Support for file level incremental backups with rsync
- Retention policy support
- WAL Archive Compression with gzip, bzip2, or pigz
- Backup data verification
- Backup with RPO=0 using a synchronous physical streaming replication connection
- Rate limiting

<https://www.pgbarman.org/about/>



Postgres Backup And Restore

pgBackRest



Solves common bottleneck problems with parallel processing for backup, compression, restoring and archiving



Support capabilities like symmetric encryption, and partial restore



Fully supported Open Source backup tool with troubleshooting support



Feature comparison

Capability	Added value	Barman	pgBackRest	Pg_basebackup
SSH protocol support		Yes	Yes	-
PostgreSQL protocol	Works without passwordless ssh.	Yes	-	Yes
Incremental backups		Yes	Yes	-
RPO=0	Restore up to the last commit	Yes	-	-
Rate limiting	Preserve IO for Postgres	Yes	-	Yes
Retention and List backups		Yes	Yes	-
Backup compression	Less backup space required	-	Yes	-
Symmetric encryption	Lower security footprint for the backup data	-	Yes	-
Partial restore (only selected databases)	Restore required data for analysis purposes	-	Yes	-
S3 and Azure Blob Support	Use flexible Cloud Storage for backup storage	Yes	Yes	-
Nagios integration	Monitor your backups with Nagios	Yes	Yes	-



Module Summary

- Backup Types
- Database SQL Dumps
- Restoring SQL Dumps
- Offline Physical Backups
- Continuous Archiving
- Online Physical Backups Using pg_basebackup
- Point-in-time Recovery
- Recovery Settings
- Backup Tools – Barman and pgBackRest



Lab Exercise - 1

1. The `edbstore` website database is all setup and as a DBA you need to plan a proper backup strategy and implement it
 - As the root user, create a folder `/pgbackup` and assign ownership to the Postgres user using the **chown** utility or the Windows security tab in folder properties
 - Take a full database dump of the `edbstore` database with the **pg_dump** utility. The dump should be in plain text format
 - Name the dump file as **`edbstore_full.sql`** and store it in the `/pgbackup` directory



Lab Exercise - 2

1. Take a dump of the `edbuser` schema from the `edbstore` database and name the file as **`edbstore_schema.sql`**
2. Take a data-only dump of the `edbstore` database, disable all triggers for a faster restore, use the `INSERT` command instead of `COPY`, and name the file as **`edbstore_data.sql`**
3. Take a full dump of `customers` table and name the file as **`edbstore_customers.sql`**



Lab Exercise - 3

1. Take a full database dump of `edbstore` in compressed format using the **`pg_dump`** utility, name the file as **`edbstore_full_fc.dmp`**
2. Take a full database cluster dump using **`pg_dumpall`**. Remember **`pg_dumpall`** supports only plain text format; name the file **`edbdata.sql`**



Lab Exercise - 4

In this exercise you will demonstrate your ability to restore a database.

1. Drop database `edbstore`.
2. Create database `edbstore` with owner `edbuser`.
3. Restore the full dump from `edbstore_full.sql` and verify all the objects and their ownership.
4. Drop database `edbstore`.
5. Create database `edbstore` with `edbuser` owner.
6. Restore the full dump from the compressed file `edbstore_full_fc.dmp` and verify all the objects and their ownership.



Lab Exercise - 5

1. Create a directory /opt/arch or c:\arch and give ownership to the Postgres user.
2. Configure your cluster to run in archive mode and set the archive log location to be /opt/arch or c:\arch.
3. Take a full online base backup of your cluster in the /pgbackup directory using the **pg_basebackup** utility.



Module - 13

Routine Maintenance Tasks

Module Objectives

- Updating Optimizer Statistics
- Handling Data Fragmentation using Routine Vacuuming
- Preventing Transaction ID Wraparound Failures
- Automatic Maintenance using Autovacuum
- Re-indexing in Postgres



Database Maintenance

- Data files become fragmented as data is modified and deleted
- Database maintenance helps reconstruct the data files
- If done on time nobody notices but when not done everyone knows
- Must be done before you need it
- Improves performance of the database
- Saves database from transaction ID wraparound failures



Maintenance Tools

- Maintenance thresholds can be configured using the PEM Client
- Postgres maintenance thresholds can be configured in **postgresql.conf**
- Manual scripts can be written watch stat tables like `pg_stat_user_tables`
- Maintenance commands:
 - ANALYZE
 - VACUUM
 - CLUSTER
- Maintenance command `vacuumdb` can be run from OS prompt
- Autovacuum can help in automatic database maintenance



Optimizer Statistics

- Optimizer statistics play a vital role in query planning
- Not updated in real time
- Collects information for relations including size, row counts, average row size and row sampling
- Stored permanently in catalog tables
- The maintenance command `ANALYZE` updates the statistics
- Thresholds can be set using PEM Client to alert you when statistics are not collected on time



Example - Updating Statistics

```
postgres=# CREATE TABLE testanalyze(id integer, name varchar);
CREATE TABLE
postgres=# INSERT INTO testanalyze VALUES(generate_series(1,10000), 'Sample');
INSERT 0 10000
postgres=# SELECT relname, reltuples FROM pg_class WHERE relname='testanalyze';
   relname   |   reltuples
-----+-----
 testanalyze |          0
(1 row)

postgres=# ANALYZE testanalyze;
ANALYZE
postgres=# SELECT relname, reltuples FROM pg_class WHERE relname='testanalyze';
   relname   |   reltuples
-----+-----
 testanalyze |      10000
(1 row)
```



Data Fragmentation and Bloat

- Data is stored in data file pages
- An update or delete of a row does not immediately remove the row from the disk page
- Eventually this row space becomes obsolete and causes fragmentation and bloating
- Set PEM Alert for notifications



Routine Vacuuming

- Obsoleted rows can be removed or reused using vacuuming
- Helps in shrinking data file size when required
- Vacuuming can be automated using autovacuum
- The VACUUM command locks tables in access exclusive mode
- Long running transactions may block vacuuming, thus it should be done during low usage times



Vacuuming Commands

- When executed, the VACUUM command:
 - Can recover or reuse disk space occupied by obsolete rows
 - Updates data statistics
 - Updates the visibility map, which speeds up index-only scans
 - Protects against loss of very old data due to transaction ID wraparound
- The VACUUM command can be run in two modes:
 - VACUUM
 - VACUUM FULL



Vacuum and Vacuum Full

- VACUUM
 - Removes dead rows and marks the space available for future reuse
 - Does not return the space to the operating system
 - Space is reclaimed if obsolete rows are at the end of a table
- VACUUM FULL
 - More aggressive algorithm compared to VACUUM
 - Compacts tables by writing a complete new version of the table file with no dead space
 - Takes more time
 - Requires extra disk space for the new copy of the table, until the operation completes



VACUUM Syntax

- VACUUM [(option [, ...])] [table_and_columns [, ...]]

where option can be one of:

- FULL [boolean]
- FREEZE [boolean]
- VERBOSE [boolean]
- ANALYZE [boolean]
- DISABLE_PAGE_SKIPPING [boolean]
- SKIP_LOCKED [boolean]
- INDEX_CLEANUP [boolean]
- TRUNCATE [boolean]
- PARALLEL integer



Example - Vacuuming

```
edb=# CREATE TABLE testvac (id numeric, name varchar2);
CREATE TABLE
edb=# INSERT INTO testvac VALUES(generate_series(1,10000), 'Sample');
INSERT 0 10000
edb=# SELECT pg_size.pretty(pg_relation_size('testvac'));
 pg_size.pretty
-----
440 kB
(1 row)

edb=# UPDATE testvac SET name='Sample';
UPDATE 10000
edb=# SELECT pg_size.pretty(pg_relation_size('testvac'));
 pg_size.pretty
-----
872 kB
(1 row)

edb=# █
```



Example – Vacuuming (continued)

```
edb=# VACUUM testvac;
VACUUM
edb=# UPDATE testvac SET name='Sample';
UPDATE 10000
edb=# SELECT pg_size.pretty(pg_relation_size('testvac'));
 pg_size.pretty
-----
 872 kB
(1 row)

edb=# VACUUM FULL testvac;
VACUUM
edb=# SELECT pg_size.pretty(pg_relation_size('testvac'));
 pg_size.pretty
-----
 440 kB
(1 row)

edb=#
```



Preventing Transaction ID Wraparound Failures

- MVCC depends on transaction ID numbers
- Transaction IDs have limited size (32 bits at this writing)
- A cluster that runs for a long time (more than 4 billion transactions) would suffer transaction ID wraparound
- This causes a catastrophic data loss
- To avoid this problem, every table in the database must be vacuumed at least once for every two billion transactions



Vacuum Freeze

- VACUUM FREEZE will mark rows as frozen
- Postgres reserves a special XID, FrozenTransactionId
- FrozenTransactionId is always considered older than every normal XID
- VACUUM FREEZE replaces transaction IDs with FrozenTransactionId, thus rows will appear to be “in the past”
- vacuum_freeze_min_age controls when a row will be frozen
- VACUUM normally skips pages without dead row versions, but some rows may need FREEZE
- vacuum_freeze_table_age controls when a whole table must be scanned



The Visibility Map

- Each heap relation has a Visibility Map which keeps track of which pages contain only tuples
- Stored at `<relnode>_vm`
- Helps vacuum to determine whether pages contain dead rows
- Can also be used by index-only scans to answer queries
- VACUUM command updates the visibility map
- The visibility map is vastly smaller, so can be cached easily



vacuumdb Utility

- The VACUUM command has a command-line executable wrapper called vacuumdb
- vacuumdb can VACUUM all databases using a single command
- Syntax:
 - vacuumdb [OPTION] . . . [DBNAME]
- Available options can be listed using:
 - vacuumdb --help



Autovacuuming

- Highly recommended feature of Postgres
- It automates the execution of VACUUM, FREEZE and ANALYZE commands
- Autovacuum consists of a launcher and many worker processes
- A maximum of autovacuum_max_workers worker processes are allowed
- Launcher will start one worker within each database every autovacuum_naptime seconds
- Workers check for inserts, updates and deletes and execute VACUUM and/or ANALYZE as needed
- track_counts must be set to TRUE as autovacuum depends on statistics
- Temporary tables cannot be accessed by autovacuum



Autovacuuming Parameters

Autovacuum Launcher Process

- `autovacuum`

Autovacuum Worker Processes

- `autovacuum_max_workers`
- `autovacuum_naptime`

Vacuuming Thresholds

- `autovacuum_vacuum_scale_factor`
- `autovacuum_vacuum_threshold`
- `autovacuum_analyze_scale_factor`
- `autovacuum_analyze_threshold`
- `autovacuum_vacuum_insert_scale_threshold`
- `autovacuum_vacuum_insert_threshold`
- `autovacuum_freeze_max_age`



Per-Table Thresholds

- Autovacuum workers are resource intensive
- Table-by-table autovacuum parameters can be configured for large tables
- Configure the following parameters using ALTER TABLE or CREATE TABLE:
 - autovacuum_enabled
 - autovacuum_vacuum_threshold
 - autovacuum_vacuum_scale_factor
 - autovacuum_analyze_threshold
 - autovacuum_analyze_scale_factor
 - autovacuum_vacuum_insert_scale_threshold
 - autovacuum_vacuum_insert_threshold
 - autovacuum_freeze_max_age



Routine Reindexing

- Indexes are used for faster data access
- UPDATE and DELETE on a table modify underlying index entries
- Indexes are stored on data pages and become fragmented over time
- REINDEX rebuilds an index using the data stored in the index's table
- Time required depends on:
 - Number of indexes
 - Size of indexes
 - Load on server when running command



When to Reindex

- There are several reasons to use REINDEX:
 - An index has become "bloated", meaning it contains many empty or nearly-empty pages
 - You have altered a storage parameter (such as fillfactor) for an index
 - An index built with the CONCURRENTLY option failed, leaving an "invalid" index
- Syntax:

```
=> REINDEX [ ( VERBOSE ) ] { INDEX | TABLE | SCHEMA | DATABASE |  
SYSTEM } [ CONCURRENTLY ] name
```



Module Summary

- Updating Optimizer Statistics
- Handling Data Fragmentation using Routine Vacuuming
- Preventing Transaction ID Wraparound Failures
- Automatic Maintenance using Autovacuum
- Re-indexing in Postgres



Lab Exercise - 1

1. While monitoring table statistics on the `edbstore` database, you found that some tables are not automatically maintained by autovacuum. You decided to perform manual maintenance on these tables. Write a SQL script to perform the following maintenance:
 - Reclaim obsolete row space from the `customers` table.
 - Update statistics for `emp` and `dept` tables.
 - Mark all the obsolete rows in the `orders` table for reuse.
2. Execute the newly created maintenance script on `edbstore` database.



Lab Exercise - 2

1. The composite index named `ix_orderlines_orderid` on (`orderid`, `orderlineid`) columns of the `orderlines` table is performing very slowly. Write a statement to reindex this index for better performance.



Module - 14

Moving Data Using COPY Command

Module Objectives

- Loading flat files
- Import and export data using COPY
- Examples of COPY Command
- Using COPY FREEZE for performance
- EDB*Loader
- Data Loading Methods
- Invoking EDB*Loader and Control File
- EDB*Loader Exit Codes



Loading Flat Files into Database Tables

- A "flat file" is a plain text or mixed text file which usually contains one record per line
- EDB Postgres Advanced Server offers two options to load flat files into a database table:
 - EDB*Loader
 - COPY Command





COPY Command

The COPY Command

- COPY moves data between EDB Postgres Advanced Server tables and standard file-system files
- COPY TO copies the contents of a table or a query to a file
- COPY FROM copies data from a file to a table
- The file must be accessible to the server



COPY Command Syntax

Copy From:

- `COPY table_name [(column list)] FROM 'filename' | PROGRAM 'command' | STDIN [options] [WHERE cond.]`

Copy To:

- `COPY table_name[(column list)]|(query) TO 'filename' | PROGRAM 'command' | STDOUT [options]`

Copy Command Options

- `FORMAT format_name`
- `OIDS [boolean]`
- `FREEZE [boolean]`
- `DELIMITER 'delimiter_character'`
- `NULL 'null_string'`
- `HEADER [boolean]`
- `QUOTE 'quote_character'`
- `ESCAPE 'escape_character'`
- `FORCE_QUOTE { (column_name [, ...]) | * }`
- `FORCE_NOT_NULL (column_name [, ...])`
- `FORCE_NULL (column_name [, ...])`
- `ENCODING 'encoding_name'`



Example Export to File

```
=> COPY emp (empno,ename,job,sal,comm,hiredate) TO '/tmp/emp.csv' CSV HEADER;  
COPY  
=> \! cat /tmp/emp.csv  
empno,ename,job,sal,comm,hiredate  
7369,SMITH,CLERK,800.00,,17-DEC-80 00:00:00  
7499,ALLEN,SALESMAN,1600.00,300.00,20-FEB-81 00:00:00  
7521,WARD,SALESMAN,1250.00,500.00,22-FEB-81 00:00:00  
7566,JONES,MANAGER,2975.00,,02-APR-81 00:00:00  
7654,MARTIN,SALESMAN,1250.00,1400.00,28-SEP-81 00:00:00  
7698,BLAKE,MANAGER,2850.00,,01-MAY-81 00:00:00  
7782,CLARK,MANAGER,2450.00,,09-JUN-81 00:00:00  
7788,SCOTT,ANALYST,3000.00,,19-APR-87 00:00:00  
7839,KING,PRESIDENT,5000.00,,17-NOV-81 00:00:00  
7844,TURNER,SALESMAN,1500.00,0.00,08-SEP-81 00:00:00  
7876,ADAMS,CLERK,1100.00,,23-MAY-87 00:00:00  
7900,JAMES,CLERK,950.00,,03-DEC-81 00:00:00  
7902,FORD,ANALYST,3000.00,,03-DEC-81 00:00:00  
7934,MILLER,CLERK,1300.00,,23-JAN-82 00:00:00
```



Example Import from File

```
edb=# CREATE TEMP TABLE empcsv (LIKE emp);
CREATE TABLE
edb=# COPY empcsv (empno, ename, job, sal, comm, hiredate)
edb-# FROM '/tmp/emp.csv' CSV HEADER;
COPY
edb=# SELECT * FROM empcsv;
   empno |    ename    |      job      | mgr |      hiredate      |     sal     |    comm    | deptno
-----+-----+-----+-----+-----+-----+-----+-----+
    7369 | SMITH     | CLERK        |    | 17-DEC-80 00:00:00 | 800.00    |          | 
    7499 | ALLEN     | SALESMAN     |    | 20-FEB-81 00:00:00 | 1600.00   | 300.00   | 
    7521 | WARD      | SALESMAN     |    | 22-FEB-81 00:00:00 | 1250.00   | 500.00   | 
    7566 | JONES     | MANAGER      |    | 02-APR-81 00:00:00 | 2975.00   |          | 
    7654 | MARTIN    | SALESMAN     |    | 28-SEP-81 00:00:00 | 1250.00   | 1400.00  | 
    7698 | BLAKE     | MANAGER      |    | 01-MAY-81 00:00:00 | 2850.00   |          | 
    7782 | CLARK     | MANAGER      |    | 09-JUN-81 00:00:00 | 2450.00   |          | 
    7788 | SCOTT     | ANALYST      |    | 19-APR-87 00:00:00 | 3000.00   |          | 
    7839 | KING      | PRESIDENT    |    | 17-NOV-81 00:00:00 | 5000.00   |          | 
    7844 | TURNER    | SALESMAN     |    | 08-SEP-81 00:00:00 | 1500.00   | 0.00     | 
    7876 | ADAMS     | CLERK        |    | 23-MAY-87 00:00:00 | 1100.00   |          | 
    7900 | JAMES     | CLERK        |    | 03-DEC-81 00:00:00 | 950.00    |          | 
    7902 | FORD      | ANALYST      |    | 03-DEC-81 00:00:00 | 3000.00   |          | 
    7934 | MILLER    | CLERK        |    | 23-JAN-82 00:00:00 | 1300.00   |          | 
(14 rows)
```



Example - COPY Command on Remote Host

- COPY command on remote host using psql

```
$ cat emp.csv | ssh 192.168.192.83 "psql -U edbstore  
edbstore -c 'copy emp from stdin;'"
```



COPY FREEZE

- **FREEZE is a new option in the COPY statement**
 - Add rows to a newly created table and freezes them
 - Table must be created or truncated in current subtransaction
 - Improves performance of initial bulk load
 - Does violate normal rules of MVCC
- **Usage:**
=> COPY tablename FROM filename FREEZE;

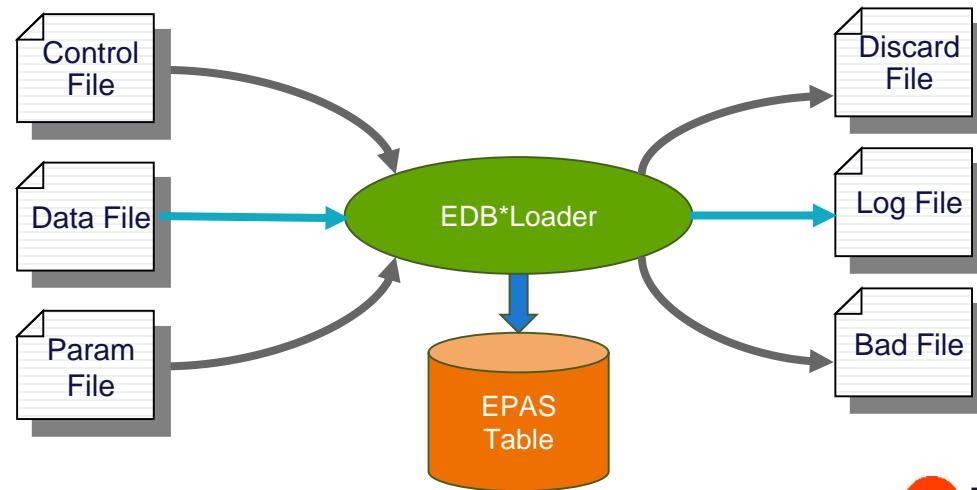




EDB*Loader

EDB*Loader

- EDB*Loader is a high-performance bulk data loader
- Supports Oracle SQL*Loader data loading methods:
 - Conventional
 - Direct
 - Parallel



Data Loading Methods

- Conventional path loading uses basic insert processing and is used to add rows to the table
- Constraints, indexes and triggers are enforced during conventional path data loading
- Direct path loading is faster than conventional path loading, but is non-recoverable
- Direct path loading also requires removal of constraints and triggers from the table
- Conventional path data loading is slower than direct path loading, but is fully recoverable
- A parallel direct path load provides even greater performance



Invoking EDB*Loader

- Use the following command to invoke EDB*Loader from the command line:

```
edbldr [-d DBNAME] [-p PORT] [-c "CONNECTION_STRING"]  
        userid={dbuser[/dbpass]|/} direct={true|false} parallel={true|false}  
        control=control_file_name log=log_file_name  
        errors=num_errors  
        skip_index_maintenance={true|false}  
        skip=num_skips bad=bad_file_name parfile=par_file_name  
        freeze={true|false}  
        handle_conflicts={true|false}
```



The EDB*Loader Control File

- The Oracle SQL*Loader has compatible syntax for control file directives
- The control file includes the instructions that EDB*Loader uses to build the table (or tables) from the input file. It includes information such as:
 - The fully qualified name of the input file
 - The name of the table or tables
 - The name of the columns within the table or tables
 - The delimiters or other selection criteria used to choose the column content
 - The fully qualified names of the bad and discarded files



Control File Syntax

- The syntax for the EDB*Loader control file is as follows:

```
[ OPTIONS (param=value [, param=value] ...) ]  
LOAD DATA  
[ CHARACTERSET charset ]  
[ INFILE '{ data_file | stdin }' ] [ BADFILE 'bad_file' ] [ DISCARDFILE 'discard_file' ]  
[ { DISCARDMAX | DISCARDS } max_discard_recs ]  
[ INSERT | APPEND | REPLACE | TRUNCATE ] [ PRESERVE BLANKS ]  
{ INTO TABLE target_table  
[ WHEN field_condition [ AND field_condition ] ... ] [ FIELDS TERMINATED BY 'termstring'  
[ OPTIONALLY ENCLOSED BY 'enclstring' ] ] [ RECORDS DELIMITED BY 'delimstring' ] [ TRAILING  
NULLCOLS ]  
(field_def [, field_def] ...) } ...
```



EDB*Loader Example

- This example loads data from a file named **/tmp/mydata.csv** into a table named `emp`
- The data within the input file is delimited by a comma
- Create the control file:

```
LOAD DATA INFILE '/tmp/mydata.csv'  
    BADFILE '/tmp/mydata.bad'  
    DISCARDFILE '/tmp/mydata.dsc'  
    INSERT INTO TABLE emp  
    FIELDS TERMINATED BY ","  
    OPTIONALLY ENCLOSED BY '\"' (emno, empname, sal, deptno)
```

- Run the `edbldr` command:

```
$ edbldr -d edb CONTROL=emp.ctl BAD=/tmp/emp.bad LOG=/tmp/emp.log SKIP=1 ERRORS=10
```



EDB*Loader Exit Codes

- EDB*Loader will return one of the following exit codes:

0

- Indicates that all rows loaded successfully

1

- EDB*Loader encountered syntax errors or aborted due to an unrecoverable error

2

- Load completed with some rejected or discarded rows

3

- Indicates EDB*Loaded stopped due to an OS error



Module Summary

- Loading flat files
- Import and export data using COPY
- Examples of COPY Command
- Using COPY FREEZE for performance
- EDB*Loader
- Data Loading Methods
- Invoking EDB*Loader and Control File
- EDB*Loader Exit Codes



Lab Exercise - 1

- In this lab exercise you will demonstrate your ability to copy data:
 1. Unload the `emp` table from the `edbuser` schema to a csv file, with column headers
 2. Create a `copyemp` table with the same structure as the `emp` table
 3. Load the csv file (from step 1) into the `copyemp` table



Module 15

Replication and High Availability Tools

Module Objectives

- Data Replication
- Data Replication in Postgres
- Streaming Replication and Architecture
- Synchronous, Asynchronous and Cascaded Replication
- Setup Streaming Replication
- Logical Replication Architecture
- Failover Manager
- Replication Manager (repmgr)
- Postgres-BDR
- Replication Server (xdb)



Data Replication

- Replication is the process of copying data and changes to a secondary location for data safety and availability
- Data loss can occur due to several reasons
- Replication is aimed towards availability of the data when a primary source goes offline
- Data can be recovered from backup but downtimes are costly
- Replication aims towards lowering downtime
- Failovers can be configured to such a level where application may not notice the primary source is offline



Data Replication in Postgres

- Data replication options:
 - Log-Shipping Standby Servers
 - Streaming Replication
 - Logical Replication
 - Postgres-BDR
 - EDB Replication Server
- Cluster management tools:
 - High Availability Routing for Postgres(HARP)
 - EDB Failover Manager
 - Replication manager(repmgr)

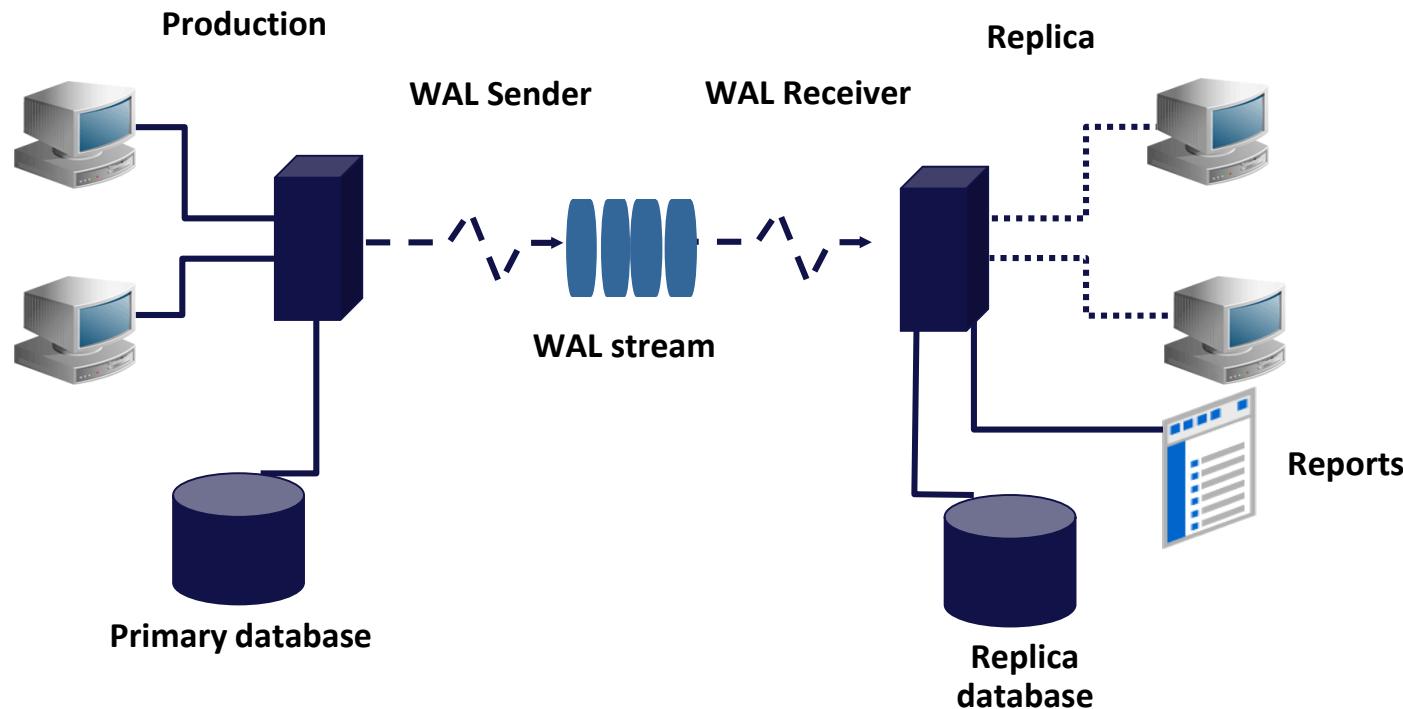


Streaming Replication

- Streaming Replication (Hot Standby) is a major feature of Postgres
- Replica connects to the primary node using REPLICATION protocol
- WAL segments are streamed to replica server
- No log shipping delays, stream WAL content across to replica immediately
- Synchronous/Asynchronous options available
- Supports cascading replication



Hot Streaming Architecture



Asynchronous Replication

- Streaming replication is asynchronous by default but can be configured as synchronous
- Asynchronous
 - Disconnected architecture
 - Transaction is committed on primary and flushed to WAL segment
 - Later transaction is transmitted to replica server(s) using stream
 - Some data loss is possible
 - Replication using WAL Archive method is always asynchronous



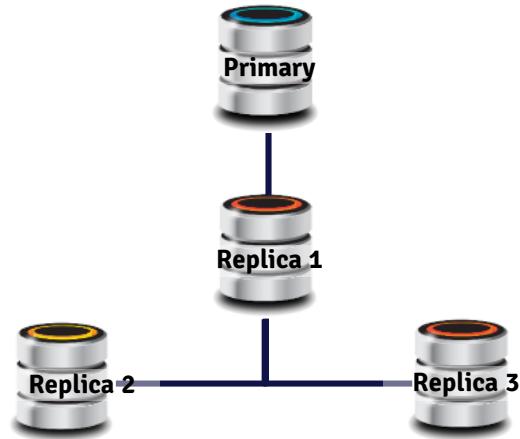
Synchronous Replication

- Synchronous Replication
 - A 2-safe replication method offering zero data loss
 - Transaction must apply changes to primary and synchronously replicated replicas using two-phase commit actions
 - User gets a commit message after confirmation from both primary and replica
 - This will introduce a delay in committing transactions



Cascading Replication

- Streaming replication supports single master node
- Cascade replication can be used to share the replication overhead of primary with other replicas
- Replica can stream changes to other Replicas
- Helps minimize inter-site bandwidth overheads on primary node
- Asynchronous only





Setup Streaming Replication

Primary Server Configuration

- For Physical Streaming Replication:
 - Change WAL Content parameter:
 - wal_level = replica #Default is replica
 - Two options to allow streaming connection:
 - max_wal_senders
 - max_replication_slots
 - Set only the minimum number of segments retained in pg_wal
 - wal_keep_size = 1024



Synchronous Streaming Replication Configuration

- Default level of Streaming Replication is Asynchronous
- Synchronous level can also be configured using additional parameters:
 - synchronous_commit=on
 - synchronous_standby_names
- If the synchronous replica stops responding, then COMMITs will be blocked forever until someone manually intervenes
- Transactions can be configured not to wait for replication by setting the synchronous_commit parameter to local or off



Configure Authentication

- Authentication setting on the primary server must allow replication connections from the replica(s)
- Provide a suitable entry or entries in **pg_hba.conf** with the database field set to `replication`
- Open **pg_hba.conf** of primary server:

```
host    replication    all      192.168.56.2/32    md5
```

Note - You will need to reload the primary server



Take a Full Backup of the Primary Server

- Backup the Primary Server using pg_basebackup:

```
pg_basebackup -h localhost -U postgres -p 5444  
-D /backup/data1 -R
```

-R option

- creates a default copy of **standby.signal** file
- Add primary server connection info to **postgresql.auto.conf**



Replica Configuration

hot_standby	<ul style="list-style-type: none">• Set this parameter to “ON” for read-only replica
primary_conninfo	<ul style="list-style-type: none">• Set connection string to connect with primary or cascaded replica
primary_slot_name	<ul style="list-style-type: none">• Specify replication slot name to be used for connection
promote_trigger_file	<ul style="list-style-type: none">• Set trigger file name for replica promotion
max_standby_streaming_delay	<ul style="list-style-type: none">• Duration for which replica has to wait during query conflicts
wal_receiver_create_temp_slot	<ul style="list-style-type: none">• Authorize WAL receiver process to be able to create a temporary replication slot
recovery_min_apply_delay	<ul style="list-style-type: none">• Parameter used for delayed replication



Replica Recovery Settings

- Replica configuration settings must be set in **postgresql.conf** or **postgresql.auto.conf**
- Create a file name **standby.signal** in the data directory
- **standby.signal** indicates the server should start as a replica
- Last step - start the replica using system services or **pg_ctl**

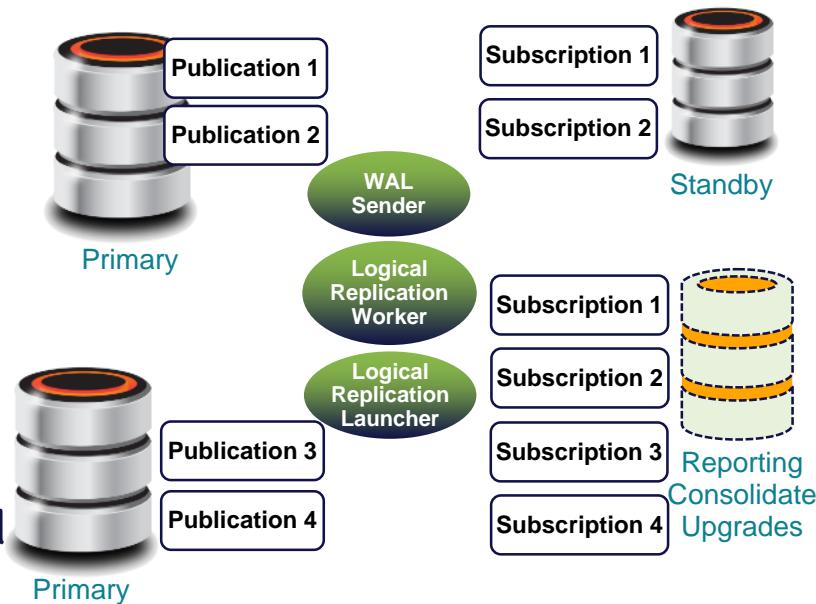




Logical Replication

Logical Replication

- Logical replication is a method of replicating selected data objects and their changes
- Based on publications and subscriptions
- Can be used to consolidate data
- Portable across hardware and software version
- Tables on standby server which are part of a subscription must be treated as read only to avoid conflicts



When to Use Logical Replication

- Sending incremental changes in a single database or a subset of a database to subscribers
- Consolidating multiple databases into a single one
- Replicating between different major versions of Postgres
- Giving access to replicated data to different groups of users
- Sharing a subset of the database between multiple databases



Setting Up Logical Replication

Change `wal_level` to logical in `postgresql.conf`

Add `pg_hba.conf` entry in each server to allow connection

Connect to database in publication instance

Create a publication using `CREATE PUBLICATION` statement

A published table must have a “replica identity” configured in order to be able to replicate `UPDATE` and `DELETE` operations

Connect to database in subscription instance and create a subscription using `CREATE SUBSCRIPTION` statement



Example – Logical Replication Setup

- Initialize sample publication(**primarypub**) and subscription(**primarysub**) instance

```
[postgres@localhost ~]$ initdb --version
```

```
[postgres@localhost ~]$ initdb -D primarypub -U pubdba
```

```
[postgres@localhost ~]$ initdb -D primarysub -U subdba
```

- Edit **postgresql.conf** parameter for both instances

```
[postgres@localhost ~]$ vi primarypub/postgresql.conf
```

- port=5420
- wal_level=logical

```
[postgres@localhost ~]$ vi primarysub/postgresql.conf
```

- port=5421
- wal_level=logical



Example – HBA Entries and Starting Instances

- Add pg_hba.conf entries for connections

```
[enterprisedb@localhost ~]$ vi primarysub/pg_hba.conf
```

```
host all pubdba 192.168.56.101/32 md5
```

```
[enterprisedb@localhost ~]$ vi primarypub/pg_hba.conf
```

```
host all subdba 192.168.56.101/32 md5
```

- Start both instances

```
[enterprisedb@localhost ~]$ pg_ctl -D primarypub/ start
```

```
[enterprisedb@localhost ~]$ pg_ctl -D primarysub/ start
```



Example – Create Tables and Publication

- Connect to default database in publication instance

```
[postgres@localhost ~]$ psql -p 5420 -U pubdba postgres
```

- Create a sample table and publication

```
=# CREATE TABLE pubexample(id INT PRIMARY KEY,  
                           name VARCHAR(30));  
=# INSERT INTO pubexample  
VALUES(generate_series(1,5000),'Test1');  
=# SELECT count(*) FROM pubexample;  
=# CREATE PUBLICATION testpub FOR TABLE pubexample;
```



Example – Create Tables and Subscription

- Connect to default database in subscription instance

```
[postgres@localhost ~]$ psql -p 5421 -U subdba postgres
```

- Create a sample table and subscription

```
=# CREATE TABLE pubexample(id INT PRIMARY KEY,  
                           name VARCHAR(30));  
=# CREATE SUBSCRIPTION testsub CONNECTION  
  'host=localhost port=5420 user=pubdba dbname=postgres'  
  PUBLICATION testpub;  
=# SELECT count(*) FROM pubexample;
```



Example – Test Logical Replication

- Add data to publication

```
[postgres@localhost ~]$ psql -p 5420 -U pubdba postgres
postgres=# INSERT INTO pubexample
VALUES (generate_series(5001,10000),'Test1');
postgres=# \q
```

- Check changes on Subscription

```
[postgres@localhost ~]$ psql -p 5421 -U subdba postgres
postgres=# SELECT count(*) FROM pubexample;
```





Monitoring Basics

Monitoring Replication

- `pg_stat_replication`
 - Show connected replicas and their status on the primary
- `pg_stat_subscription`
 - Shows the status of subscription when using logical replication
- `pg_stat_wal_receiver`
 - Shows the WAL receiver process status on Replica
- **Recovery information functions:**
 - `pg_is_in_recovery()`
 - `pg_current_wal_lsn`
 - `pg_last_wal_receive_lsn`
 - `pg_last_xact_replay_timestamp()`



Example - Monitoring Replication

- Execute:

```
=# SELECT * FROM pg_stat_replication;
```

- Find lag (bytes):

```
=# SELECT pg_wal_lsn_diff(sent_lsn, replay_lsn) FROM pg_stat_replication;
```

- Find lag (seconds):

```
=# SELECT CASE WHEN pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn()
    THEN 0 ELSE
    EXTRACT (EPOCH FROM now() -pg_last_xact_replay_timestamp())
END AS stream_delay;
```



Recovery Control Functions

Name	Return Type	Description
<code>pg_is_wal_replay_paused()</code>	<code>bool</code>	True if recovery is paused.
<code>pg_wal_replay_pause()</code>	<code>void</code>	Pauses recovery immediately.
<code>pg_wal_replay_resume()</code>	<code>void</code>	Restarts recovery if it was paused.





EDB Postgres Distributed - Overview

EDB Postgres Distributed

The most advanced replication solution for Postgres



Maintain extreme high availability

Postgres clusters deployed with EDB Postgres Distributed keep top tier enterprise applications running



Upgrade with near zero downtime

Rolling upgrades of application and database software eliminate the largest source of downtime



Choose the level of consistency

Robust capabilities provide flexibility to meet application data loss requirements



Always ON

Top-tier enterprise applications are critical to an organization's success in all regions where business is conducted, whether a single region or globally



The application represents a promise to its customers



The availability of the application directly ties to revenue generation



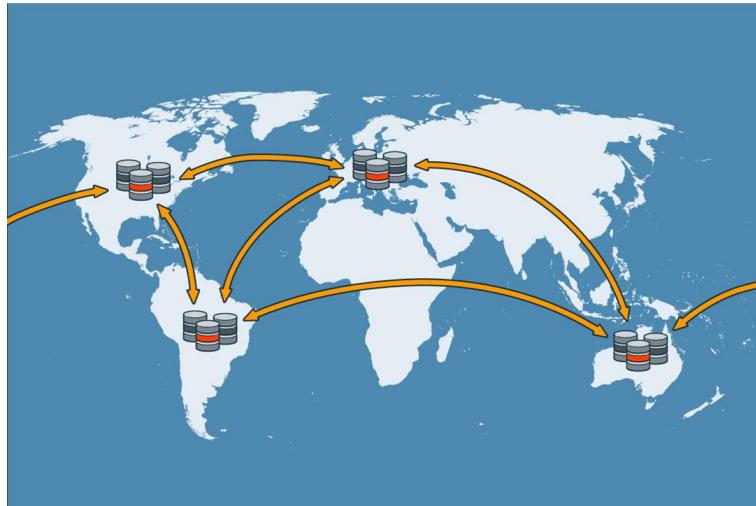
The application must perform well for a good user experience



The application data must always be current and available or the user loses trust

BDR is more than bi-directional replication

Multi-master replication enabling highly available and geographically distributed Postgres clusters



- Logical replication of data and schema enabled via standard Postgres extension
- Data consistency options that span from immediate to eventual consistency
- Robust tooling to manage conflicts, monitor performance, and validate consistency
- Deploy natively to cloud, virtual, or bare metal environments
- Geo-fencing, allowing selectively replicate data for security compliance and jurisdiction control.



Postgres-BDR Features

Multi-Master Synchronous or Asynchronous Replication for Postgres

Flexible Always-ON Deployment Architectures

DDL Replication

Row Level Consistency

DDL and Row Filters

Parallel Apply

Database Rolling Upgrades

Auto Partitioning

Configurable Column-level Conflict Resolution

Conflict-free Replicated Data Types (CRDTs)

Transactions State Tracking Across Failovers

Subscriber-only Nodes

PGD CLI

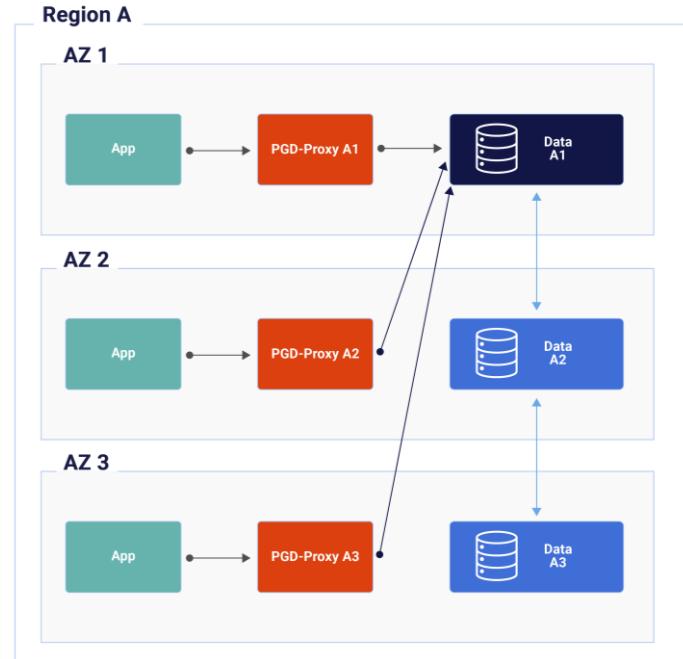
Next Generation Connection Routing using PGD-Proxy

Open Telemetry Integration



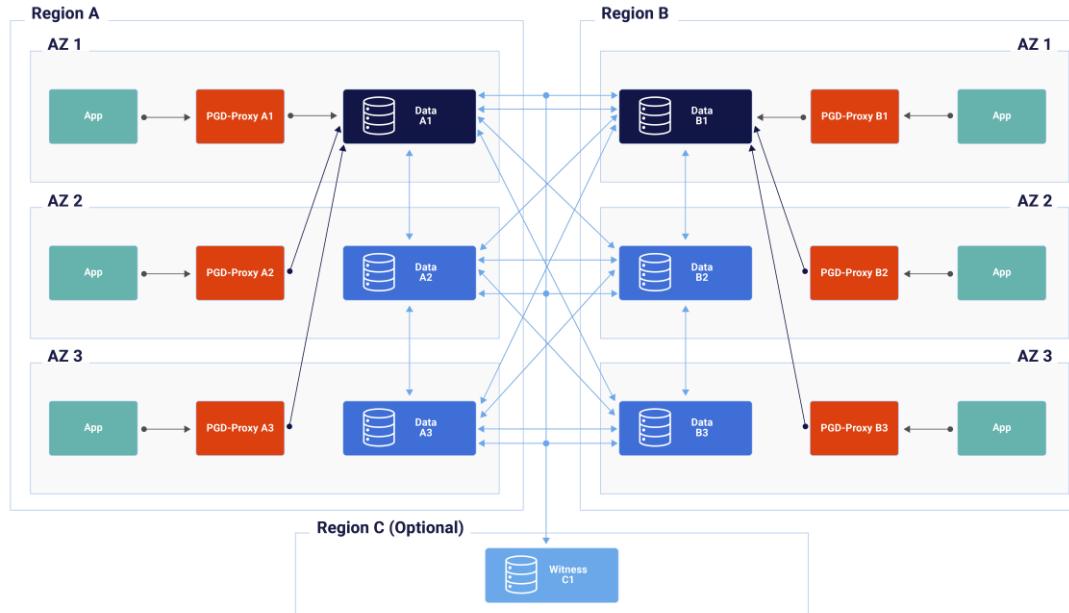
Deployment - Single Location

- Locations = 1, local redundancy = 3, nodes = 3, active locations = 1
- Global group with single data group of A1, A2 and A3
- Lead Primary A1 receiving all writes but changes can also be received by A2 and A2
- Shadow Primary A2, A3 receiving writes
 - Can be 3 data nodes (recommended)
 - Can be 2 data nodes and 1 witness that doesn't hold data (not depicted)



Example Deployment - Multiple Location

Locations = 2, local redundancy = 3, nodes = 3, active locations = 1

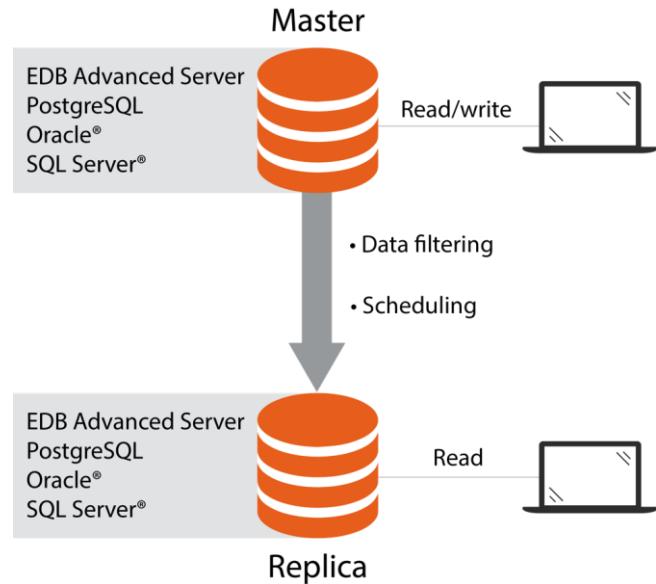




Replication Server Overview

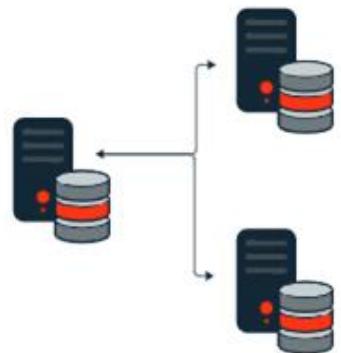
EDB Postgres Replication Server

Single Master Replication (SMR)
for Reporting or Migration



Replication Server

Replicates between Postgres and non-Postgres databases



- Integrate with Oracle or SQL Server databases to offload reporting or to feed data to legacy applications
- Flexibility to replicate a subset of data from the source database
- Graphical user interface provides easy configuration and management
- Includes utility to validate data consistency between the source and target databases

Replication from SQL
Server or Oracle to Postgres



EDB Replication Server Features

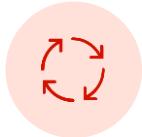
- Replicate Oracle or SQL Server data to EDB Postgres Advanced Server
- Distributed multi-Publication/Subscription Architecture
- Synchronize data across geographies
- Replicate tables and sequences
- Controlled switchover and failover
- Supports cascading replication
- Trigger and Log-based replication
- Snapshot and continuous modes
- Define and apply row filters
- Flexible replication scheduler
- Replication History Viewer
- Graphical Replication Console and CLI





Failover Manager Overview

Why Failover Manager



Ensure business continuity

Monitor health databases and identify failures quickly



Maintain high availability

Meet your SLAs by switching over to the most recent standby

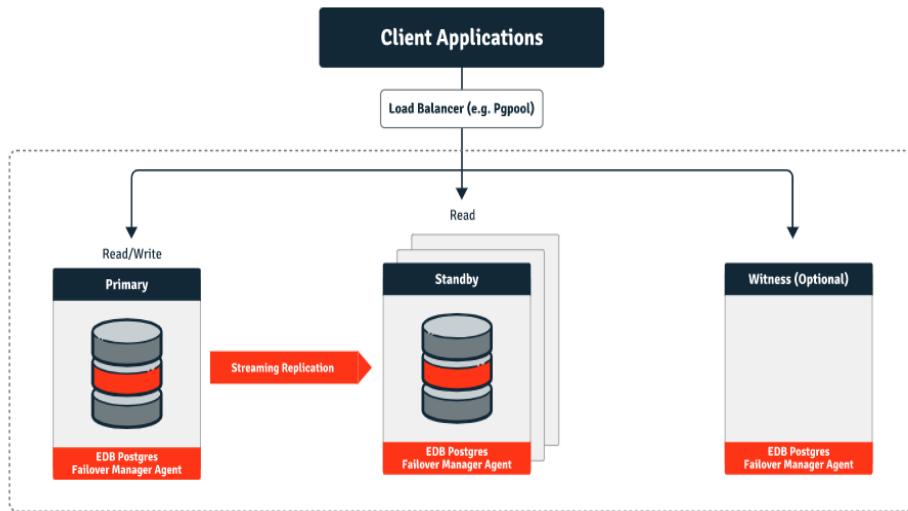


Upgrade with minimal downtime

Switchover on demand to move the primary to standby for maintenance

EDB Postgres Failover Manager

Automatically detect failures



- Monitors database health - detects failures and takes action
- Automatically fails over to the most current standby, reconfigures others
- Reconfigures load balancers on failover - integrates with pgPool and others
- Avoids “split brain” scenarios - Prevents two nodes from thinking that each is primary



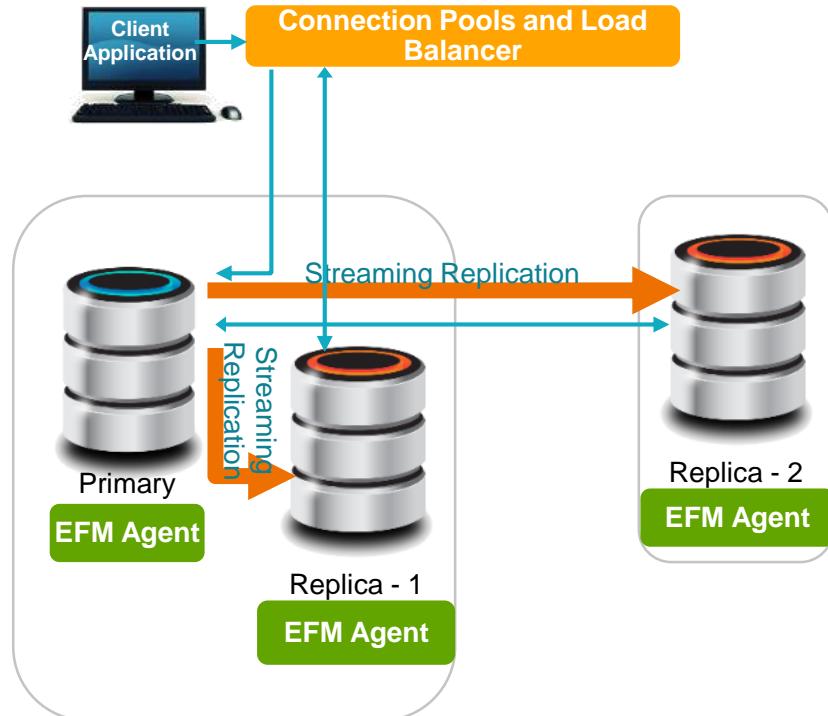
EFM Features

- Multiple health checks for Primary & Replica nodes
- Automatic Failover from Primary to Replica node
- Controlled switchovers for planned events on primary
- Configurable fencing operations
- User configurable failure detection wait times
- Witness node protects against ‘split brain’ scenarios
- Support for multiple streaming replicas
- Replica promotion based on WAL location and node priority
- Real-time notifications to chat rooms, SNMP and SMTP for all cluster status changes



Setup an EFM Cluster

1. Set up Streaming replication between the two servers
2. Install EFM
3. Configure the `efm.properties` file
4. Start EFM
5. Add nodes to EFM cluster
6. Monitor the EFM and database servers





Replication Manager Overview

Replication Manager(repmgr)

Cluster Management tool for Postgres



Maintain high availability

Automatic Failover to Replica in a Streaming Replication Environment



Perform upgrades using switchovers

Add/remove replicas and switchover of primary instance



Open Source

Open Source from EnterpriseDB and licensed under GPL

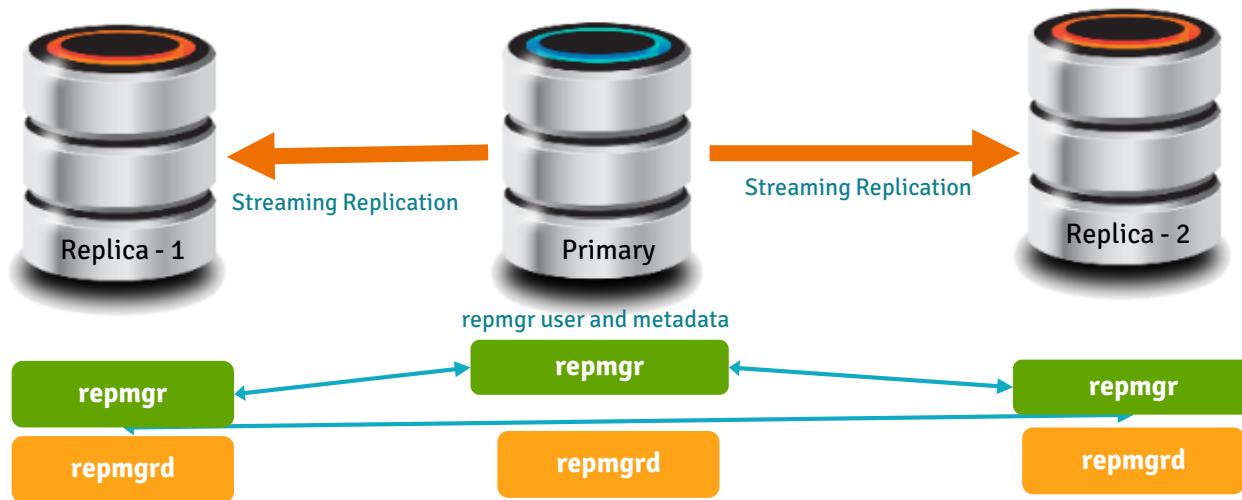


repmgr Features

- Open source tool for managing replication and failover
- Supports Postgres Streaming Replication
- **repmgr** tool for setup:
 - Add/remove replicas
 - Perform switchovers
 - Promote a replica
- **repmgrd** tool:
 - Monitor replication
 - Automatic failover detection with witness protection
 - Email notification



repmgr Architecture



Module Summary

- Data Replication
- Data Replication in Postgres
- Streaming Replication and Architecture
- Synchronous, Asynchronous and Cascaded Replication
- Setup Streaming Replication
- Logical Replication Architecture
- Failover Manager
- Replication Manager (repmgr)
- Postgres-BDR
- Replication Server (xdb)



Course Summary

- Introduction and Architectural Overview
- System Architecture
- EDB Postgres Advanced Server Installation
- User Tools - Command Line Interfaces
- Database Clusters
- Database Configuration
- Data Dictionary
- Creating and Managing Database Objects
- Database Security
- Monitoring and Admin Tools Overview
- SQL Primer
- Backup and Recovery
- Routine Maintenance Tasks
- Data Loading
- Data Replication and High Availability



Next Steps

- Certify your Postgres skills with [EDB Certifications for Postgres](#)
- Continue your skills development with the following classes:
 - Advanced Database Administration
 - Monitoring and Alerting with Postgres Enterprise Manager
 - Tuning and Maintenance
 - See the Training Portal for the full library of Postgres training classes
- Get familiar with the EDB Tools available as part of the EDB Postgres Platform
- For any questions related to EDB Postgres Trainings and Certifications, or for additional information, write to:
 - trainingcoordinator@enterprisedb.com



THANK YOU

trainingcoordinator@enterprisedb.com
www.enterprisedb.com

