





Version 1.0   
8 September 2017

Unotech Software Pvt Ltd

**Index**

[**Overview:**](#_hwnfgq7f59a4) **6**

[Document Purpose:](#_ka8gg84kswzy) 6

[Brief Introduction :](#_ew6b75zap4xz) 6

[Architecture:](#_u1orq2xcz4uq) 6

[**Installation Guide:**](#_v932gfk17f5c) **8**

[Installation:](#_1zj1zyquhmps) 8

[Configuration:](#_92urwpg366et) 9

[Network and OS Level:](#_qh0v1a3e2q9x) 10

[Ports:](#_hnqc1wumm990) 10

[OS Level:](#_sq7ys7jai0qg) 10

[Starting Cassandra:](#_carmwfwu3o9a) 11

[Cqlsh & Test:](#_2j9rn2u0dnc2) 12

[Importing Schema:](#_r9csd3xxqmuw) 13

[Export Schema, Data](#_j47ukzdfvwbx) 14

[**Managing the Cluster & Data Center**](#_ugjihg2r9nk4) **16**

[Creating the New Data Center in same Cluster](#_ckpseh62hf94) 16

[Configuration](#_6db151cytbm0) 16

[Adding the new Node to the Datacenter/Cluster](#_1sflnn2co9gk) 16

[Configuration](#_1wr8dsi9fxk4) 16

[Adding the Data Center of different cluster](#_b71shd1bvzp0) 17

[Configuration](#_u21z1te47ykr) 17

[Removing the node from the cluster](#_b8z7o1z0xz5y) 17

[Case 1: nodetool decommission](#_eayy4lx71dtn) 17

[Case 2: nodetool removenode](#_xsoipfcqw6v8) 17

[Case 3: nodetool assassinate](#_aq1sx2dvnz0q) 17

[**Monitoring:**](#_lmmranoylcxz) **19**

[**Maintenance:**](#_4cqfdbtsslwc) **20**

[Good Practices:](#_ke64g6k9rnah) 20

[Maintenance Tools:](#_tojaobbo7l9k) 20

[nodetool rebuild](#_8qn9eyxg5o7d) 20

[nodetool repair](#_wtjvgx1ehiyz) 20

[Best Practice for Nodetool Repair](#_bn311zj8eg3c) 20

[**Security:**](#_9y7xfgc3ljo9) **21**

[Update system\_auth](#_cs8tvbpo1kxb) 21

[Configuration](#_tozcazrcf51p) 22

[Create Username and Password](#_huumxgt40mq0) 22

[Other Commands](#_8eqbjoh94kp8) 22

[**Performance:**](#_jtn8cpj8pkyt) **22**

# 

# 

# Overview:

## 1.1 Document Purpose:

This document is created with the purpose:

* Brief Introduction to the Postgres XL
* Architecture
* Installation of the application
* Configuring
* Commonly faced issues

## 1.2 Brief Introduction :

Postgres-XL is a horizontally scalable open source SQL database cluster, flexible enough to handle varying database workloads:

* OLTP write-intensive workloads
* Business Intelligence requiring MPP parallelism
* Operational data store
* Key-value store
* GIS Geospatial
* Mixed-workload environments
* Multi-tenant provider hosted environments

## 

## 

## 1.3 Product Architecture:

# xl_cluster_architecture1.jpg

# 

## GTM

Keeps track of all the transactions and has responsibility of cluster wide tracking of database primitives like mutexes and transactions. This node is used by both Data Nodes and Coordinator nodes. This is a Single Point Of Failure of failure as it is only a single node. For this reason there is a GTM backup put in place.

## GTM Backup

GTM Failover is a node that handles the switch between the GTM and GTM Backup in case of a failure condition.

## Coordinator Nodes

These nodes are what the users connect to, to run the actual queries. These are essentially postgres instances, so any tool that can connect to a postgres database will also work on these. Users can connect to any coordinator node to run any queries on the postgresxl cluster.

A coordinator node keeps track of what data resides on what machine, makes the query plan and manages the execution of the query and returns the result. A coordinator node does not save any data by itself so even one of the nodes crashes, it does not impact the functioning of the cluster.

## Data Nodes

These nodes save the actual data, these are also postgres instances but these would not be directly connected by anyone to query data, these nodes will be queried by the coordinator nodes.

### DataDistribution-04-v2.jpg

### Distribute By Hash

For tables that do not or in the future are expected to not fit into one machine will be distributed across several machines. Data of one row would go into one machine. To decide what machine the data would need to go into a hash of one of the pre-decided columns is taken. Based on the result of the hash the data is placed into a particular node.

### Distribute By Replication

This is used for tables that can easily fit into one server and are not expected to grow substantially in the future. The same data is replicated to all the nodes. The benefit of this is that any node can be used to query this data. Certain use cases where the table is small but is updated very frequently is when you may not want to use this method.

## 

## Replicated Slaves

Because there is not built in redundancy like cassandra where 2 - 3 copies of data are maintained here the redundancy is maintained by maintaining replicated nodes for each data node.

# 

# 

## 1.3 Currently Deployed Architecture:

Separate PDF document

# 2. Installation Guide:

## 2.1 Installation:

### Points to remember before install

* Do the proper passwordless ssh between users created for the Postgres-XL
* Configure, make of Postgresxl and make of pgxc\_ctl should be done in user not root
* After make, make install has to be done in root, Installation all from the root
* Data and coordinator directories are created in $HOME/serveradm/postgres(pgxcl), u can change this directory

#### Install the basic utilities and dependencies for Postgresxl

* yum -y update
* yum install -y tar openssh-server openssh-clients openssh wget vim sudo

#### Do Yum for following packages, which install all the dependencies for postgresxl

* yum groupinstall -y "Development tools"
* yum -y install readline-devel zlib-devel

#### Create the user for the Postgres for simplicity we are creating the user as ‘postgres’ or ‘toro’ in our case

* adduser postgres
* passwd postgres
* usermod -aG wheel postgres

#### Creating the folder structure for the installation

* cd /opt/
* mkdir damocles
* cd damocles
* wget<https://nchc.dl.sourceforge.net/project/postgres-xl/Releases/Version_9.5r1/postgres-xl-9.5r1.4.tar.gz>
* tar -zxvf postgres-xl-9.5r1.4.tar.gz

#### 

#### Installation of the postgresxl - Now login as postgres user

* su postgres
* cd postgres-xl-9.5r1.4/
* ./configure
* make

#### Switch as root user

* make install

#### Switch back to postgres user

* cd /opt/damocles/postgres-xl-9.5r1.4/contrib/pgxc\_ctl
* make

#### Switch to root

* make install

### Environmental Variables

Add the environmental variables for the postgresxl in bashrc. Postgresxl is installed in /usr/local/pgsql/ open bashrc and the environmental variables

###### vim ~/.bashrc

###### #Postgres-XL enviromental variables starts

###### export POST\_HOME=/usr/local/pgsql

###### export PATH=$POST\_HOME/bin:$PATH

###### #Postgres-xl enviromental variables ends

###### source ~/.bashrc

### Verify the environment variable

echo $POST\_HOME

*output>>* /usr/local/pgsql

***Please repeat these procedure/installation in all machines***

### Passwordless ssh between the machines

This will guide you to do the passwordless ssh between machine because postgres communicates between nodes by ssh so make sure when you do ssh it should not ask for password

* ssh-keygen -t rsa
* cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys
* chmod 0600 ~/.ssh/authorized\_keys
* service sshd start
* ssh localhost

### Passwordless ssh between machines(this is demonstration between

serveradm@10.138.32.25 **to** serveradm@10.138.32.232**) change the user and ip address for rest of machines**

* ssh serveradm@10.138.32.232 mkdir -p .ssh
* cat .ssh/id\_rsa.pub | ssh serveradm@10.138.32.232 'cat >> .ssh/authorized\_keys'
* ssh serveradm@10.138.32.232 "chmod 700 .ssh; chmod 640 .ssh/authorized\_keys"
* ssh serveradm@10.138.32.232

Once the passwordless ssh is done, lets start the configuration of the postgresxl

## 2.2 PGXC CTL Configuration:

### pgxc\_ctl.conf

**This configuration has to be done in GTM master not in all machine**

* cd /opt/damocles/postgres-xl-9.5r1.4/contrib/pgxc\_ctl
* cp pgxc\_ctl\_conf\_part\_minimal pgxc\_ctl.conf
* vim pgxc\_ctl.conf

Add the below configuration lines

#### Data, Location and Other Configuration

pgxcOwner=postgres #give user\_name created for postgres during the installation

pgxcUser=$pgxcOwner

tmpDir=/tmp

localTmpDir=$tmpDir

configBackup=n

configBackupHost=pgxc-linker

configBackupDir=/opt/damocles/data/pgxc

configBackupFile=$configBackupDir/pgxc\_ctl.bak

#give the path the data directory where data and coordinate nodes to be created.

Please create this folder structure before you make this configuration in all the machines

dataDirRoot=/opt/damocles/data/DATA/pgxl/nodes

### 

#### GTM Master Configuration

gtmName=gtm #gtm master name

gtmMasterServer=10.138.32.25

gtmMasterPort=8080

gtmMasterDir=$dataDirRoot/gtm

gtmExtraConfig=none

gtmMasterSpecificExtraConfig=none

#### GTM Slave Configuration

gtmSlave=y

gtmSlaveName=gtmSlave

gtmSlaveServer=10.138.32.27

gtmSlavePort=20002

gtmSlaveDir=$dataDirRoot/gtm\_slv

#### 

#### GTM Slave Configuration

gtmProxyDir=$dataDirRoot/gtm\_pxy

gtmProxy=y

gtmProxyNames=(gtm\_pxy1)

gtmProxyServers=(10.138.32.25)

gtmProxyPorts=(20101)

gtmProxyDirs=($gtmProxyDir.1)

gtmPxyExtraConfig=n

#### 

## Coordinators Configuration

postgresql.conf

coordMasterDir=$dataDirRoot/coord\_master

coordSlaveDir=/opt/damocles/data/coord\_slave

coordArchLogDir=/opt/damocles/data/coord\_archlog

coordNames=(coord1 coord2) #give the name of the coordinators with space separated

coordPosrts=(30001 30002) #give the ports of the coordinators with space separated

poolerPorts=(30011 30012) #give the poolerports of the coordinators with space separated

coordPgHbaEntries=(::1/128)

coordPgHbaEntries=(10.138.32.25/16) #mention the GTM, GTM proxy, GTM Slave IP series

coordMasterServers=(10.138.32.232 10.138.32.233) #give the IP of the coordinators with space separated

coordMasterDirs=($coordMasterDir.1 $coordMasterDir.2) #give the directory of the coordinators with space separated

#### Datanode Configuration

datanodeMasterDir=$dataDirRoot/dn\_master

datanodeSlaveDir=$dataDirRoot/dn\_slave

datanodeArchLogDir=$dataDirRoot/datanode\_archlog

primaryDatanode=datanode\_1

datanodeNames=(datanode\_1 datanode\_2)

datanodePorts=(40001 40002)

datanodePoolerPorts=(40011 40012)

datanodePgHbaEntries=(::1/128)

datanodePgHbaEntries=(10.138.32.25/16)

datanodeMasterServers=(10.138.32.232 10.138.32.233)

datanodeMasterDirs=($datanodeMasterDir.1 $datanodeMasterDir.2)

#### Datanode Slave Configuration

datanodeSlave=y

datanodeSlaveServers=(10.138.32.77 10.138.32.80)

datanodeSlavePorts=(40101 40102)

datanodeSlavePoolerPorts=(40111 40112)

datanodeSlaveSync=y

datanodeSlaveDirs=($datanodeSlaveDir.1 $datanodeSlaveDir.2)

datanodeArchLogDirs=( $datanodeArchLogDir.1 $datanodeArchLogDir.2)

Save and exit the pgxc\_ctl.conf

## 2.3 Network and OS Level:

### Ports:

Communication on the following ports must be enabled so the nodes can coordinate with each other. This will mean setting the correct **Selinux and Iptables permissions**.

|  |  |  |
| --- | --- | --- |
| Port Number | Internal to cassandra / External | Purpose |
| 16633 | External | Coordinator Port used to communicate with the application/client/user |
| 20101, 20102,20107,20108,20109,20110,20111,  20112 | Internal | GTM\_proxy, coordinator and datanodes communication |
| 40001-40007 | External | Datanode master Port used to communicate with the application/client/user |
| 6666 | Internal | This is GTM master port for internal communication of the nodes |
|  | External | Datanode slave Port used to communicate with the application/client/user |

**Check communication of nodes internally and externally**

when the service is started, telnet to the ports between cassandra and client machines to make sure the ports are open between them.

### OS Level:

* Make sure all the binary, data and log folders have the correct ownership and permissions for the user who is going to be running cassandra. In this case will most probably be **toro**.
* Make sure this user has the following OS level params are not set very low otherwise cassandra may not start or may perform badly.
* **Make sure all the clocks of the machines are synchronized using NTP because cassandra uses timestamps for conflict resolution of the latest data. If the clocks are different it will have unexpected effects on data.**

in the /etc/sysctl.conf

Make sure the following settings are set:

###### net.core.rmem\_max = 16777216 net.core.wmem\_max = 16777216 net.core.rmem\_default = 16777216 net.core.wmem\_default = 16777216 net.core.optmem\_max = 40960 net.ipv4.tcp\_rmem = 4096 87380 16777216 net.ipv4.tcp\_wmem = 4096 65536 16777216

**Disable CPU frequency scaling if enabled, otherwise this will reduce throughput**

###### for CPUFREQ in /sys/devices/system/cpu/cpu\*/cpufreq/scaling\_governor do [ -f $CPUFREQ ] || continue echo -n performance > $CPUFREQ done

**Correct OS level settings :**

**check using :**

ulimit -a

Make sure they are at least set to the following:

toro - memlock unlimited  
toro - nofile 100000  
toro - nproc 32768  
toro - as unlimited

also make sure /etc/security/limits.conf has these same settings.

Reference: [Recommended Production Settings](https://docs.datastax.com/en/landing_page/doc/landing_page/recommendedSettings.html)

## 2.4 Initialize the Cluster

* pgxc\_ctl
* init all

### Start the nodes

* start all

### Verify the cluster

* monitor all

Running: gtm master

Running: gtm slave

Running: gtm\_proxy gtm\_proxy\_2

Running: gtm\_proxy gtm\_proxy\_1

Running: coordinator master coord1

Running: coordinator master coord2

Running: coordinator master coord3

Running: datanode master datanode\_1

Running: datanode slave datanode\_1

Running: datanode master datanode\_2

Running: datanode slave datanode\_2

Running: datanode master datanode\_3

Running: datanode slave datanode\_3

## 2.5 Verifying the cluster

By creating the database, table, insert and select, in any one of the coordinator access the using ***psql***

***psql –p 30001 postgres***

***postgres=# select \* from pgxc\_node;***

***node\_name | node\_type | node\_port | node\_host | nodeis\_primary | nodeis\_preferred | node\_id***

***------------+-----------+-----------+---------------+----------------+------------------+-------------***

***coord1 | C | 30001 | 10.138.32.212 | f | f | 1885696643***

***coord2 | C | 30002 | 10.138.32.213 | f | f | -1197102633***

***coord3 | C | 30003 | 10.138.32.214 | f | f | 1638403545***

***datanode\_3 | D | 40003 | 10.138.32.214 | f | f | 1787525382***

***datanode\_1 | D | 40001 | 10.138.32.212 | f | f | -675012441***

***datanode\_2 | D | 40002 | 10.138.32.213 | f | t | -1047623914***

***postgres=# create database testcluster;***

CREATE DATABASE  
***postgres=# \c testcluster  
You are now connected to database "testcluster" as user "serveradm".  
testcluster=# create table testdb (id int, name text);***

CREATE TABLE  
***testcluster=# insert into testdb (id, name) select generate\_series(1,5),'foo';  
INSERT 0 5  
testcluster=# select \* from testdb ;  
 id | name  
----+------***

***3 | foo***

***5 | foo***

***1 | foo***

***4 | foo***

***2 | foo***

***(5 rows)***

Here we go Cluster is up and running

## 2.6 Importing Schema:

#### Script

Create the script for Creating the database, table and indexes in normal text file. All schema in the script should be in proper psql syntax

**Example:**

CREATE DATABASE "alltrade"

ENCODING 'UTF8'

LC\_COLLATE = 'en\_US.UTF-8'

LC\_CTYPE = 'en\_US.UTF-8';

\c alltrade;

CREATE TABLE print\_history (

print\_history\_pk bigint NOT NULL,

document\_no\_key0 text NOT NULL,

edition\_data text NOT NULL,

transaction\_type\_key1 text,

createby\_data text,

create\_datetime\_data timestamp with time zone,

updateby\_data text,

update\_datetime\_key2 timestamp with time zone,

int\_created\_date timestamp without time zone,

int\_updated\_date timestamp without time zone,

int\_is\_deleted text

)

DISTRIBUTE BY REPLICATION;

save the script in normal text file

#### Import using Source or File

Start the postgres

psql -p <port> postgres

# After login to the cqlsh session

*\i ‘`*/PATH\_OF\_THE\_SCRIPT/filename*’;*

alternate way is to pass the psql file via the -f or --file switch.

helpful arguments to cqlsh

-e To pass a query to execute directly without going into the shell. Useful when you want to automate queries.

-f To run queries or schema creation scripts from a file directly.

### Export Schema, Data

Export the schema from the other cluster using cqlsh as below

**Schema:**

pg\_dump -p 30001 --dbname=<database\_name> --schema-only > 'schema.psql'

pg\_dump -p 30001 --dbname=<database\_name> --table=<table\_name> --schema-only > 'schema.psql'

Copy the schema file into the cluster where you want to import and follow the steps given in ***Import using \i or File***

**Data:**Use pg\_dump to take the backup the data as given below

pg\_dump -p 30001 --dbname=<database\_name> > 'schema.psql'

pg\_dump -p 30001 --dbname=<database\_name> --table=<table\_name> > 'schema.psql'

# 

# 

# 3. Managing the Cluster & Data Center

Use pgxc\_ctl, the simple way to manage the postgres cluster below are the pgxc commands to add the nodes and remove the node

## Add GTM Master

***add gtm master gtm <ip> <port> <dataDir>*** //Syntax  
***PGXC# add gtm master gtm 10.138.32.25 20001 $dataDirRoot/gtm***  //Example

## Add GTM Slave

***add gtm slave < gtm\_slave\_name> <ip> <port> <dataDir>***  //Syntax  
***PGXC# add gtm slave gtm\_slave 10.138.32.25 20001 $dataDirRoot/gtm*** //Example

## Add GTM Proxy

***add gtm\_proxy <gtm\_proxy name> <ip> <port> <dataDir>*** //Syntax

***PGXC# add gtm master gtm 10.138.32.25 20001 $dataDirRoot/gtm*** //Example

## Add Coordinator Master

***add coordinator master <nodename> <nodeip> <nodeport> <nodepoolerport> <nodedatadir> extraconfig extrapghbaconfig*** //Syntax  
***PGXC# add coordinator master coord2 localhost 30002 30012 $dataDirRoot/coord\_master.2 none none*** //Example

## Add Datanode Master

***add datanode master <nodename> <nodeip> <nodeport> <nodepoolerport> <nodedatadir> WALdir extraconfig extrapghbaconfig*** //Syntax  
***PGXC# add datanode master dn3 localhost 40003 40013 $dataDirRoot/dn\_master.3 none none none*** //Example

## Add Datanode Slave

***add datanode slave <node\_name> <nodeip> <node\_port> <nodepoolerport> <nodedatadir> <WALArchdir> <nodearchlogdir>*** //Syntax  
***PGXC# add datanode slave dn1 localhost 40101 40111 $dataDirRoot/dn\_slave.1 none $dataDirRoot/datanode\_archlog.1*** //Example

When Datanode Master is add to the cluster you add the table to the new datanode by simple alter query as below:

***<dbname>=# ALTER <TABLENAME> ADD NODE (dataNode\_master\_name);***The above query will be distribute in the new datanode  
***<dbname>=#SELECT xc\_node\_id, COUNT(\*) FROM <table\_name> GROUP BY xc\_node\_id;***Run the above query to check the distribution of the table in the different data nodes

## Remove GTM Master

***remove gtm master <node\_name> clean***This will delete the folder from node

## Remove GTM Slave

***remove gtm slave <node\_name> clean***This will delete the folder from node

## Remove GTM Proxy

***remove gtm\_proxy <node\_name> clean***This will delete the folder from node

## Remove Coordinator Master

***remove coordinator master <node\_name> clean***This will delete the folder from node

## Remove Datanode Master

Before removing the any datanode master, first delete or remove the data from that particular node by running the below query  
***<dbname>=# ALTER <TABLENAME> DELETE NODE (dataNode\_master\_name);***The above query will redistribute the table into remaining datanode master  
***<dbname>=#SELECT xc\_node\_id, COUNT(\*) FROM <table\_name> GROUP BY xc\_node\_id;***

Run the above query to check the distribution of the table in the different data nodes  
***remove datanode Master <node\_name> clean***This will delete the folder from node

## Remove Datanode Slave

***remove datanode Slave <node\_name> clean***This will delete the folder from node

## 

## 

## Failover

### GTM Master

When GTM Master fails, proxy automatically promotes the GTM Slave as GTM Master and Configuration will be updated at the bottom of the configuration file. After that you can add the GTM slave using pgxc\_ctl.

### GTM Slave

When GTM Slave fails, there is impact on cluster you can fix the issue or you can add new GTM Slave using the pgxc\_ctl.

### GTM Proxy

If GTM Proxy fails, other gtm proxies available but it will impact the performance, you can add the new GTM proxy or fix the issues with current GTM proxy

### Datanode Master

When Datanode Master slave is down, promote the Datanode Slave of the same master node using the pgxc\_ctl. While promoting there will little downtime as the configuration will be reloaded. Use the below command  
***failover datanode <data\_node\_name>***

# 4. Monitoring:

## Status

For monitoring the status of the node you use the pgxc\_ctl tool, use monitor all to see the status of the nodes. You use monitor on individual nodes for example

monitor all  
monitor gtm  
monitor datanode

## Stats

To check the size of the table run below query

***SELECT***

***relname as "Table",***

***pg\_size\_pretty(pg\_total\_relation\_size(relid)) As "Size",***

***pg\_size\_pretty(pg\_total\_relation\_size(relid) - pg\_relation\_size(relid)) as "External Size"***

***FROM pg\_catalog.pg\_statio\_user\_tables ORDER BY pg\_total\_relation\_size(relid) DESC;***

To check the size of the database run below query

***SELECT***

***pg\_database.datname,***

***pg\_size\_pretty(pg\_database\_size(pg\_database.datname)) AS size***

***FROM pg\_database;***

***To Check the backend process and PID below***

SELECT pg\_stat\_get\_backend\_pid(s.backendid) AS pid,

pg\_stat\_get\_backend\_activity(s.backendid) AS query

FROM (SELECT pg\_stat\_get\_backend\_idset() AS backendid) AS s;

The above are the few examples of Statistics Collector, is a subsystem that supports collection and reporting of information about server activity

Below are the few tables, views, functions which helps for cluster maintenance, reporting

***pg\_stat\_activity***

One row per server process, showing information related to the current activity of that process, such as state and current query

***pg\_stat\_replication***

One row per WAL sender process, showing statistics about replication to that sender's connected standby server. See

***pg\_stat\_ssl***

One row per connection (regular and replication), showing information about SSL used on this connection. See pg\_stat\_ssl for

***pg\_stat\_archiver***

One row only, showing statistics about the WAL archiver process's activity. See pg\_stat\_archiver for details.

***pg\_stat\_bgwriter***

One row only, showing statistics about the background writer process's activity. See pg\_stat\_bgwriter for details.

***pg\_stat\_database***

One row per database, showing database-wide statistics. See pg\_stat\_database for details.

***pg\_stat\_database\_conflicts***

One row per database, showing database-wide statistics about query cancels due to conflict with recovery on standby servers.

***pg\_stat\_all\_tables***

One row for each table in the current database, showing statistics about accesses to that specific table.

***pg\_stat\_sys\_tables***

Same as pg\_stat\_all\_tables, except that only system tables are shown.

***pg\_stat\_user\_tables***

Same as pg\_stat\_all\_tables, except that only user tables are shown.

***pg\_stat\_xact\_all\_tables***

Similar to pg\_stat\_all\_tables, but counts actions taken so far within the current transaction (which are not yet included in pg\_stat\_all\_tables and related views). The columns for numbers of live and dead rows and vacuum and analyze actions are not present in this view.

***pg\_stat\_xact\_sys\_tables***

Same as pg\_stat\_xact\_all\_tables, except that only system tables are shown.

pg\_stat\_xact\_user\_tables

Same as pg\_stat\_xact\_all\_tables, except that only user tables are shown.

pg\_stat\_all\_indexes

One row for each index in the current database, showing statistics about accesses to that specific index.

pg\_stat\_sys\_indexes

Same as pg\_stat\_all\_indexes, except that only indexes on system tables are shown.

pg\_stat\_user\_indexes

Same as pg\_stat\_all\_indexes, except that only indexes on user tables are shown.

Additional functions related to statistics collection are listed

pg\_backend\_pid()

integer Process ID of the server process handling the current session

pg\_stat\_get\_activity(integer)

setof record Returns a record of information about the backend with the specified PID, or one record for each active backend in the system if NULL is specified. The fields returned are a subset of those in the pg\_stat\_activity view.

pg\_stat\_get\_snapshot\_timestamp()

timestamp with time zone Returns the timestamp of the current statistics snapshot

pg\_stat\_clear\_snapshot()

void Discard the current statistics snapshot

pg\_stat\_reset()

void Reset all statistics counters for the current database to zero (requires superuser privileges)

pg\_stat\_reset\_shared(text)

void Reset some cluster-wide statistics counters to zero, depending on the argument (requires superuser privileges).

Calling pg\_stat\_reset\_shared('bgwriter') will zero all the counters shown in the pg\_stat\_bgwriter view. Calling

pg\_stat\_reset\_shared('archiver')

will zero all the counters shown in the pg\_stat\_archiver view.

pg\_stat\_reset\_single\_table\_counters(oid)

void Reset statistics for a single table or index in the current database to zero (requires superuser privileges)

pg\_stat\_reset\_single\_function\_counters(oid)

void Reset statistics for a single function in the current database to zero (requires superuser privileges)

pg\_stat\_get\_activity, the underlying function of the pg\_stat\_activity view, returns a set of records containing all the available information about each backend process. Sometimes it may be more convenient to obtain just a subset of this information. In such cases, an older set of per-backend statistics access functions can be used. These access functions use a backend ID number, which ranges from one to the number of currently active backends. The function pg\_stat\_get\_backend\_idset provides a convenient way to generate one row for each active backend for invoking these functions. For example, to show the PIDs and current queries of all backends:

SELECT pg\_stat\_get\_backend\_pid(s.backendid) AS pid,

pg\_stat\_get\_backend\_activity(s.backendid) AS query

FROM (SELECT pg\_stat\_get\_backend\_idset() AS backendid) AS s;

pg\_stat\_get\_backend\_idset()

set of integer Set of currently active backend ID numbers (from 1 to the number of active backends)

pg\_stat\_get\_backend\_activity(integer)

text Text of this backend's most recent query

pg\_stat\_get\_backend\_activity\_start(integer)

timestamp with time zone Time when the most recent query was started

pg\_stat\_get\_backend\_client\_addr(integer)

inet IP address of the client connected to this backend

pg\_stat\_get\_backend\_client\_port(integer)

integer TCP port number that the client is using for communication

pg\_stat\_get\_backend\_dbid(integer)

oid OID of the database this backend is connected to

pg\_stat\_get\_backend\_pid(integer)

integer Process ID of this backend

pg\_stat\_get\_backend\_start(integer)

timestamp with time zone Time when this process was started

pg\_stat\_get\_backend\_userid(integer)

oid OID of the user logged into this backend

pg\_stat\_get\_backend\_waiting(integer)

boolean True if this backend is currently waiting on a lock

pg\_stat\_get\_backend\_xact\_start(integer)

timestamp with time zone Time when the current transaction was started

# 6. Security:

## 6.2 User Authentication

CREATE

***Syntax:***

***CREATE USER <username> <Attributes>;***

***Atribute:***

***ADMIN role\_name***

***The ADMIN clause is like ROLE, but the named roles are added to the new role WITH ADMIN OPTION, giving them the right to grant membership in this role to others.***

***NOSUPERUSER***

***SUPERUSER***

***These clauses determine whether the new role is a "superuser", who can override all access restrictions within the database. Superuser status is dangerous and should be used only when really needed. You must yourself be a superuser to create a new superuser. If not specified, NOSUPERUSER is the default.***

***CREATEROLE***

***NOCREATEROLE***

***These clauses determine whether a role will be permitted to create new roles (that is, execute CREATE ROLE). A role with CREATEROLE privilege can also alter and drop other roles. If not specified, NOCREATEROLE is the default.***

***REPLICATION***

***NOREPLICATION***

***These clauses determine whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the REPLICATION attribute is a very highly privileged role, and should only be used on roles actually used for replication. If not specified, NOREPLICATION is the default for all roles except superusers.***

***CREATEDB***

***NOCREATEDB***

***These clauses define a role's ability to create databases. If CREATEDB is specified, the role being defined will be allowed to create new databases. Specifying NOCREATEDB will deny a role the ability to create databases. If not specified, NOCREATEDB is the default.***

***LOGIN***

***NOLOGIN***

***These clauses determine whether a role is allowed to log in; that is, whether the role can be given as the initial session authorization name during client connection. A role having the LOGIN attribute can be thought of as a user. Roles without this attribute are useful for managing database privileges, but are not users in the usual sense of the word. If not specified, NOLOGIN is the default, except when CREATE ROLE is invoked through its alternative spelling CREATE USER.***

***CREATEUSER***

***NOCREATEUSER***

***These clauses are an obsolete, but still accepted, spelling of SUPERUSER and NOSUPERUSER. Note that they are not equivalent to CREATEROLE as one might naively expect!***

***ENCRYPTED***

***UNENCRYPTED***

***These key words control whether the password is stored encrypted in the system catalogs. (If neither is specified, the default behavior is determined by the configuration parameter password\_encryption.) If the presented password string is already in MD5-encrypted format, then it is stored encrypted as-is, regardless of whether ENCRYPTED or UNENCRYPTED is specified (since the system cannot decrypt the specified encrypted password string). This allows reloading of encrypted passwords during dump/restore.***

***Note that older clients might lack support for the MD5 authentication mechanism that is needed to work with passwords that are stored encrypted.***

***ROLE role\_name***

***The ROLE clause lists one or more existing roles which are automatically added as members of the new role. (This in effect makes the new role a "group".)***

***VALID UNTIL 'timestamp'***

***The VALID UNTIL clause sets a date and time after which the role's password is no longer valid. If this clause is omitted the password will be valid for all time.***

***CONNECTION LIMIT connlimit***

***If role can log in, this specifies how many concurrent connections the role can make. -1 (the default) means no limit.***

***INHERIT***

***NOINHERIT***

***These clauses determine whether a role "inherits" the privileges of roles it is a member of. A role with the INHERIT attribute can automatically use whatever database privileges have been granted to all roles it is directly or indirectly a member of. Without INHERIT, membership in another role only grants the ability to SET ROLE to that other role; the privileges of the other role are only available after having done so. If not specified, INHERIT is the default.***

PASSWORD

***Sets the role's password. (A password is only of use for roles having the LOGIN attribute, but you can nonetheless define one for roles without it.) If you do not plan to use password authentication you can omit this option. If no password is specified, the password will be set to null and password authentication will always fail for that user. A null password can optionally be written explicitly as PASSWORD NULL.***

***Example:***

***create user mahantesh SUPERUSER LOGIN CREATEDB PASSWORD 'mahantesh'CREATEROLE ;***

Change Password

***Change password for the user***

***Syntax:***

***\password <username>***

***Example:***

***all\_trade=# \password mahantesh***

***Enter new password:***

***Enter it again:***

Alter

***Syntax:***

***alter role <user\_name> <attribute>;***

***Example:***

***all\_trade=# alter role mahantesh CREATEDB;***

ALTER ROLE

Delete

***Deleting the user***

***Syntax:***

***DROP USER <user\_name>***

***Example:***

***all\_trade=# drop user mahantesh ;***

***DROP ROLE***

## pg\_hba.conf

Here you can give the access to the user or application to access the database. Please add the ip’s only to the coordinatior nodes not to the datanode and add coordinator node ip in datanode pg\_hba.config

***local DATABASE USER METHOD [OPTIONS]***

***host DATABASE USER ADDRESS METHOD [OPTIONS]***

***hostssl DATABASE USER ADDRESS METHOD [OPTIONS]***

***hostnossl DATABASE USER ADDRESS METHOD [OPTIONS]***

(The uppercase items must be replaced by actual values.)

The first field is the connection type: "local" is a Unix-domain

socket, "host" is either a plain or SSL-encrypted TCP/IP socket,

"hostssl" is an SSL-encrypted TCP/IP socket, and "hostnossl" is a

plain TCP/IP socket.

DATABASE can be "all", "sameuser", "samerole", "replication", a

database name, or a comma-separated list thereof. The "all"

keyword does not match "replication". Access to replication

must be enabled in a separate record (see example below).

USER can be "all", a user name, a group name prefixed with "+", or a

comma-separated list thereof. In both the DATABASE and USER fields

you can also write a file name prefixed with "@" to include names

from a separate file.

ADDRESS specifies the set of hosts the record matches. It can be a

host name, or it is made up of an IP address and a CIDR mask that is

an integer (between 0 and 32 (IPv4) or 128 (IPv6) inclusive) that

specifies the number of significant bits in the mask. A host name

that starts with a dot (.) matches a suffix of the actual host name.

Alternatively, you can write an IP address and netmask in separate

columns to specify the set of hosts. Instead of a CIDR-address, you

can write "samehost" to match any of the server's own IP addresses,

or "samenet" to match any address in any subnet that the server is

directly connected to.

METHOD can be "trust", "reject", "md5", "password", "gss", "sspi",

"ident", "peer", "pam", "ldap", "radius" or "cert". Note that

"password" sends passwords in clear text; "md5" is preferred since

it sends encrypted passwords.

OPTIONS are a set of options for the authentication in the format

NAME=VALUE. The available options depend on the different

authentication methods -- refer to the "Client Authentication"

section in the documentation for a list of which options are

available for which authentication methods.

Database and user names containing spaces, commas, quotes and other

special characters must be quoted. Quoting one of the keywords

"all", "sameuser", "samerole" or "replication" makes the name lose

its special character, and just match a database or username with

that name.

This file is read on server startup and when the postmaster receives

a SIGHUP signal. If you edit the file on a running system, you have

to SIGHUP the postmaster for the changes to take effect. You can

use "pg\_ctl reload" to do that.

# 7. Performance: