SYSTEMS AND DATABASE ADMINISTRATION ASSIGNMENT REPORT

C18302166 / Michael Tennson

Human Resources case study

1.Security

When the Human resources database was created. The first thing that had to be done was ensure the security of the database. So, what was first done was the creation of two separate schemas that would be accessed by the employees and the HR staff. Then the employees, stores, payments, and performance reviews were created.

To ensure security on the human resources database tables, specific user roles were created, these roles were the HR staff role and the employee role. Both roles were granted their own unique privileges. The employees were given view - only privileges on their payments, employment information and their performance appraisal record's. The HR staff were given the privileges of being able to insert, update and delete information from the performance appraisal and payments table in the database.

The steps I took to establish security on the huma resources database detail the following

1. When the database has been set up, create the employees' schema and the HR schemas by running the sql query "CREATE SCHEMA employees;" and "CREATE SCHEMA hr"
2. I then created the tables the were needed for the database, which were the employees tables, payments table, stores table and performance appraisals table
3. I then went on to create the employee and HR staff users in the database by running the SQL query "CREATE USER employee" and "CREATE USER hr_staff".
4. What I then done was grant the two users unique privileges.
5. I gave the employee user read only privileges on the performance appraisal, their employees and payments tables. This was done by running the following SQL queries. "GRANT SELECT * FROM payments TO employees" and GRANT SELECT * FROM performance_appraisals TO employees;"

6. I gave the HR_staff user read, write and update privileges on the performance_appraisal and payments tables. This was done by running the query “GRANT INSERT INTO payments TO hr_staff” ,
7. “GRANT INSERT INTO performance_appraisals TO hr_staff”,
8. “GRANT UPDATE payments TO hr_staff”.
9. “GRANT DELETE * FROM performance_appraisals to hr_staff;”

Granting these specific privileges has helped to uphold security by preventing employees having certain privileges that the HR staff are supposed to have on the human resources database. The employees are separate users in the database and so are the HR staff

2.Auditing

When it came to auditing the database. What was first done was finding the PostgreSQL configuration file. This file was then configured so that auditing could be established on the database. I then installed PgAudit to fully establish auditing on postgres. The following steps show how I went out to establish auditing on the database. The reason why auditing is important for the database because auditing helps to create and save logs that detail the status of the database. This is very important for the Human resources database because it helps the database admin of the Human resources database keep track on the status of the database. The following steps detail how I established auditing on the HR DATABASE

1. I first entered the Human resources postgresql database by running the command /usr/local/pgsql/bin/psql humanResources
2. I ran the query “SHOW config_file;” this command will locate the postgresql configuration file which is the directory /usr/local/pgsql/data
3. I then went on to configure the postgresql.conf file by making the following changes to specific parameters so that auditing can be established on the database. **NOTE**: make sure to uncomment each parameter by removing the # symbol at the start of each parameter
4. Log_statement = ‘all’
5. Log_directory = ‘pg_log’
6. Log_filename = ‘postgresql-%y-%m-%d_%=H%M%S.log’
7. Logging_collector = on
8. Log_min_error_statement = error
9. Log_destination = ‘csvlog’
10. The parameters looked like this when they had been edited to all auditing

```
log_destination = 'csvlog'              # Valid values are combinations of
                                        # stderr, csvlog, syslog, and eventlog,
                                        # depending on platform.  csvlog
                                        # requires logging_collector to be on.

# This is used when logging to stderr:
logging_collector = on          # Enable capturing of stderr and csvlog
                                        # into log files. Required to be on for
                                        # csvlogs.
                                        # (change requires restart)

# These are only used if logging_collector is on:
log_directory = 'pg_log'                        # directory where log files are wri
                                        # can be absolute or relative to PGDATA
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log' # log file name pattern,
```

```
log_min_error_statement = error # values in order of decreasing detail:
```

```
log_statement = 'all'                   # none, ddl, mod, all
```

14. I then saved the postgresql.conf file and went on to connect to the human resources database and ran the following query “SELECT pg_reload_conf();” to reload the configuration file.
15. Exit the data directory and return to the home directory.
16. Then, the PGAudit extension was installed. To install PGAudit, enter the following commands
17. git clone https://github.com/pgaudit/pgaudit.git
18. cd pgaudit to enter the directory
19. The contents of the pgAudit directory should look like this

```
c18302166@c18302166:~/pgaudit/pgaudit$ ls
expected  Makefile                 pgaudit--1.5--1.5.1.sql  pgaudit.c     pgaudit.control  sql
LICENSE   pgaudit--1.5.1--1.5.2.sql  pgaudit--1.5.2.sql       pgaudit.conf  README.md        test
```

20.
21. git checkout REL_13_STABLE
22. make install USE_PGXS=1 PG_CONFIG=/usr/pgsql-13/bin/pg_config
23. To enable PGAudit on the database, I ran the query “Create Extension pgaudit;” to start PGAudit on my database

These steps document the process I undertook to establish Auditing on the human resources database

3.Performance optimisation

When it came to performance optimisation, the main factors that I looked at were the memory size of the machine, virtual machine performance, the size of the database and the operating system that the database is on. With these factors in mind, I decided to increase the amount of memory the Human Resources database can use. I first installed tuned to get the commands needed to tune the database. I then looked for the postmaster ID of the database so I could use the PID to check the VM peak memory status.

Performance optimisation is important for a production – level database because the database must be performing at its best for the company.

went on to first tune the shared buffers by increasing the size of it. This would help increase the amount of memory that that database can access, increasing its performance. I then connected to the Human resources database and enabled huge pages so that the database can use more memory. The following steps document how the databases performance was optimised.

1. First log in as the postgres user by running the command “sudo su postgres”
2. Run the command “postmaster -D &” to establish postgres server connection
3. Next, enter the directory /usr/local/pgsql/data and run the command “head -n 1 postmaster.pid” to check the postmaster ID. You will need this for the next command
4. Next run the command “grep -I vmpeak /proc/<PID>/status”. This command will show you the VM peak status, which is the amount of memory the VM is running on. This is important to know so you don’t allocate too much memory to the database
5. Next run the command “nano postgresql.conf” and edit the “shared_buffer” parameter. Make sure you increase the memory size. It will be set to the size of 128MB. I, for example,
6. set the shared_buffer size to 4GB. It should look like this
7. 

8. Next exit Postgres and restart it by running “systemctl restart Postgres”
9. Reconnect to postgres, enter the /usr/local/pgsql/data directory and run “head -n 1 postmaster.pid” again to look at the new PID generated
10. Next run the command “grep -I vmpeak /proc/<PID>/status”. Make sure you replace PID with the new PID
11. Next, access the database and run the following Query “ALTER SYSTEM SET huge_pages TO ‘on’;”. Implementing Linux huge pages will allow for the Human resources database to perform better because it’s pages will be 2MB rather than 4kb, giving the database more data, therefore, improving the performance of the database..

4.Backup /Recovery / Availability Policy

Having a backup database for the Human Resources database is very important for the company so that if there were any issues with the main database, the backup database can be brought to use. Having a backup is of utmost importance because employees and HR staff must be able to access that database at all times of the day. What I decided to do was

create a backup database that holds the same tables as the original database, generate a script that holds the details of the backup database. I then went on to create a restore database, create a point-in time recovery database so that there is a Recovery Human resources database. I then went on to turn on archiving mode so that the backup database can have saved records. The following details the steps taken to establish the backup system, recovery system and the availability policy

1. Log in as the postgres user and run the command “mkdir /tmp/humanResources_backup”
2. Run the command “export PATH=$PATH:/usr/local/pgsql/bin” to make pg_ctl a global variable
3. No run the command “pg_ctl -D . initdb” to initialise the database
4. Next I run the command “pg_ctl -D . start” to start up the database
5. Connect to postgresql and create the backup database
6. I then rean the query \connect humanResources_backup to connect to the database
7. I then created all the tables that are on the base Human resources database
8. I then exited postgresql and created a backup sql script
9. I did this by running the command “pg_dump humanResources_backup > humanResources.sql”
10. I then went on to create the restore cluster. I shut down pg_ctl in the backup directory by running the command “pg_ctl -D . stop”
11. I then created the restore directory and ran “pg_ctl -D . initdb” in the directory, I then entered the postgres database and created the humanResources_restore database
12. I then ran the command “pg_dump humanResources_restore < /tmp/humaResources_backup/humanresources.sql” . This commands tells linux to run psql from the SQL file rather than the terminal
13. Next. I established point-in time recovery to recover database information
14. I first created the pitr_backup directory in the /tmp directory
15. I then went on the create the wal_archive directory which would store the archive files.
16. Next I went back to the tmp/pitr_backup directory
17. I then ran the “pg_ctl -D . initdb” command to initialise the database and went on to configure the postgresql.conf file in this directory and made the following changes. Note: I made sure to uncomment all the configurations by removing the ‘#’ at the start of each configuration
18. Archiving_mode = ‘on’
19. archive_command = 'test ! -f /tmp/pitr_backup /%f && cp %p /tmp/pitr_backup /%f'
20. archive_timout = 60
21. I exited and saved the edits made. They should look like the following image below

22.
```
archive_mode = on               # enables archiving; off, on, or always
                                # (change requires restart)
archive_command = 'test ! -f /tmp/wal_archive/%f && cp %p /tmp/wal_archive/%f'
                                # placeholders: %p = path of file to archive
                                #               %f = file name only
                                # e.g. 'test ! -f /mnt/server/archivedir/%f &&
archive_timeout = 60            # force a logfile segment switch after this
                                # number of seconds; 0 disables
```
23. Next I reconnected to the postgresql database and ran the query "SELECT pg_start_backup('test backup', true);" to start the backup and test if the WAL archiving is working correctly
24. I then ran the query SELECT pg_stop_backup(); to shut off the backup
25. If everything ran correctly, the process should look like this
26.
```
postgres=# SELECT pg_start_backup('test backup', true);
 pg_start_backup
-----------------
 0/4000028
(1 row)

postgres=# SELECT pg_stop_backup();
NOTICE:  all required WAL segments have been archived
 pg_stop_backup
----------------
 0/4000100
(1 row)
```

Increasing availability and reducing downtime

To increase availability on the human resources database, The database admin could make sure not to give too many table values the UNIQUE keys. Another means by which the database admin of the human resources database could increase availability would be by creating a primary database server and a secondary database server, with the secondary database server acting as the backup server.

To reduce downtime. A means to do this would be when configuring the backup system. The database admin should not create a file system backup. The file system backup will increase downtime and avoiding it would be very beneficial.