Systems and Database Administration

C19704439



**Scenario: HR**

# Security

## Discussion

A PostgreSQL server hosting HR department data must be secure at all times. Such a database contains sensitive information such as employee identities, bank account information and retail locations which if breached might cause harm to the company and its personnel. Identity theft, financial fraud, compliance concerns, reputational damage, and business disruption could all result from such a breach. Because this HR database could be a high value target for attackers proper security measures must be implemented to secure the data's confidentiality, integrity and availability.

### Major Risks and Challenges

1. Data Breaches: Because the HR database contains sensitive personal information for over a million employees, a data breach is a serious danger. Data breaches can happen for a variety of reasons, including a weak password, insecure data storage, or a cyber-attack. A data breach can result in the loss of sensitive information, identity theft, financial fraud, reputational harm, and legal ramifications.
2. Unauthorised Access: The HR database is accessible by HR staff from HQ and employees from their respective store locations. This makes it vulnerable to unauthorised access by internal employees or external parties. Unauthorised access can lead to data theft, data tampering, or data loss.
3. Compliance: The HR database must adhere to a variety of legal and regulatory obligations, including the General Data Protection Regulation (GDPR). Noncompliance with these standards can result in legal liability, reputational harm, and financial penalties.
4. Data Accuracy: The HR database contains employee records, including their position, social security number, and current salary. It is important that this data is accurate and up to date as it can impact the employee's performance, compensation, and benefits.
5. Roles & Permissions: It is vital to assign the proper responsibilities and permissions to ensure the security of an organization's systems and data. It aids in the prevention of data breaches, the reduction of the danger of insider threats, and the compliance with relevant rules.

### Measures taken to combat risks and challenges

Employees have read-only access to their relevant views, allowing them to view their own records, payments, and performance reports. This ensures that employees can only view their own data and not that of co-workers or other employees. This aids in the preservation of privacy and the prevention of data breaches caused by employees accessing data that they are not allowed to view.

The HR Staff have been granted greater access to the database. They can view all the data, insert, modify, or delete rows as necessary. This ensures that they have the necessary access to manage the performance reviews table and add payments to the payments table.

This is consistent with the idea of least privilege. This means that employees should only be given access to the information they require to accomplish their job tasks. This can help prevent inadvertent or intentional data breaches caused by staff with excessive database access.

PGAdmin has also been installed which provides a variety of features that can be useful in terms of enhancing database security.

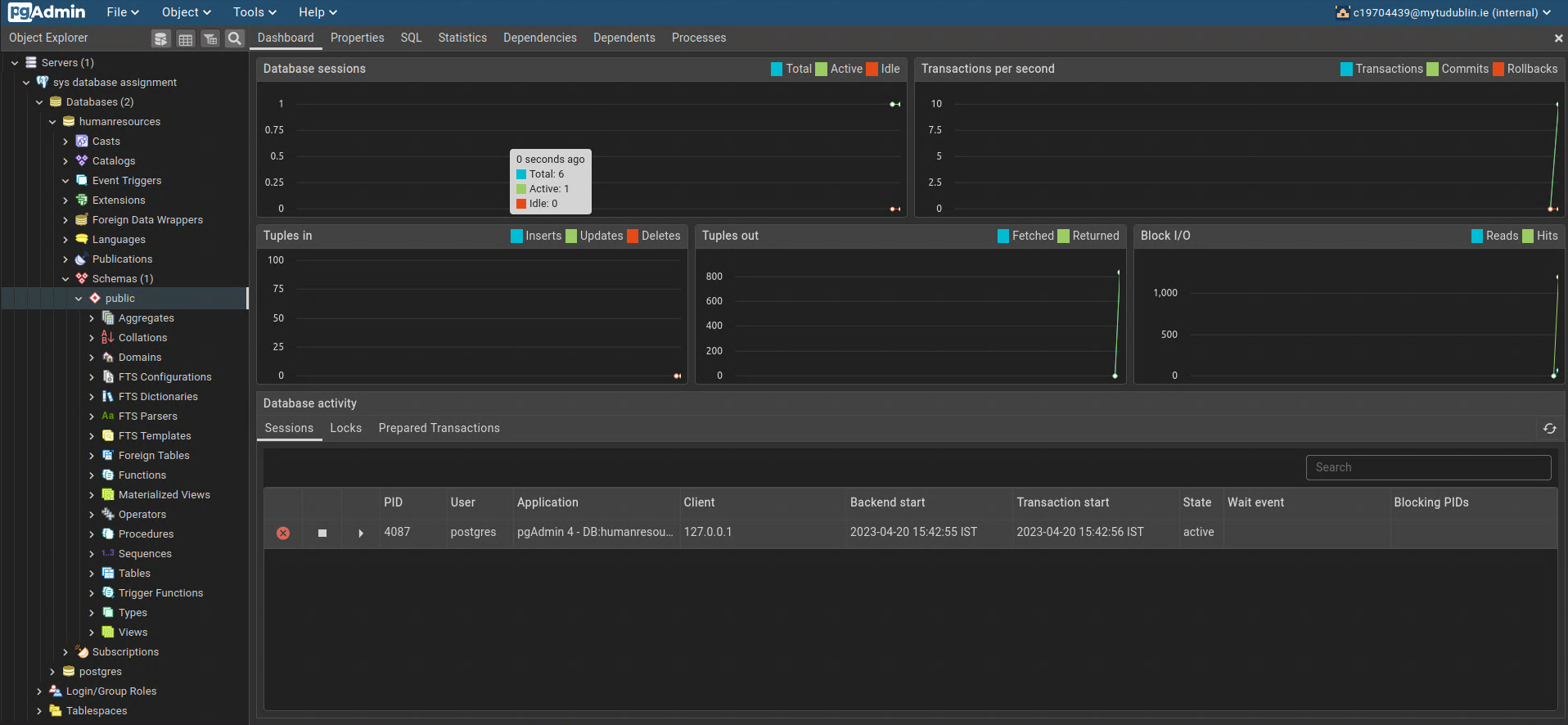


Figure 1 PGAdmin Homepage

Benefits of using PGAdmin include:

1. Ability to manage user accounts, roles, and permissions. Administrators can create new users, revoke user privileges, and set up secure authentication methods such as SSL certificates and encrypted passwords.
2. Supports SSL encryption for database connections, which can help protect sensitive data during transmission.
3. Allows administrators to monitor database activity by enabling logging and setting up auditing policies. This can help identify suspicious activity or security breaches and provide valuable information for forensic analysis.
4. Provides tools for performing backups and restores of databases. Regular backups are an essential part of a comprehensive database security strategy, as they can help recover data in case of accidental or malicious data loss.

### Row Level Security (RLS) vs Views

Both limiting users to specific views and implementing RLS are effective ways of controlling access to data in a database. For my database I have chosen to implement views rather than RLS. I made this decision for several reasons:

1. Simplicity: Limiting users to specific views is a simpler and more straightforward method of controlling access to data. Views are database objects that present data from one or more tables in a specific way. By granting users access to only specific views admins can control what data the users can see without needing to set up any RLS policies.
2. Performance: RLS policies can add additional processing overhead to database queries, which can impact performance as opposed to limiting users to specific views which does not require any additional processing, so queries against those views can be faster and more efficient.
3. Flexibility: In some cases, limiting users to specific views can be more flexible than implementing RLS. Views can be customized to present data in a specific way which can be useful in situations where users only need access to a specific subset of data.

These views are automatically produced in my database for each new user. When new data is inserted into the employees database, a function and trigger are linked that creates a user. As a result, a view is created that captures all of that single employee's data across all tables in the database.

Implementing restrictions this way as opposed to RLS also simplifies the user experience for the employees. Should they wish to view all of their information they need only log in and run a simple command of

SELECT \* FROM <employee\_name>;

A downside to my choice is possible maintenance. Because a new view is being created for each user this could cause extra steps to be added to the process of removing an employee however this could easily be remedied through an addition function that also triggers on the employees table except this time for removal. This would automatically remove the relevant view for the employee based off of the employee\_id.

If you should choose to use RLS as opposed to views you can implement it by following:

Create a policy that allows each employee to view only their own row

CREATE POLICY employees\_policy

ON employees

FOR SELECT

USING (employee\_id = session\_user);

For this implementation you would have to modify the existing function that triggers on insert to the employees table as the RLS policy uses the “session\_user” variable. The employee\_id will need to match the session\_user otherwise users will be prevented from viewing any data. Also this RLS policy only apply to select statements, therefore, users will still be able to insert, update or modify if they have the appropriate permissions.

### Recommended post-deployment policies

1. Regular Access Reviews: Implement a policy to monitor and audit the access controls and permissions assigned to employees on a regular basis. This can assist guarantee that employees only have access to the information they need to accomplish their job tasks.
2. Password Policy: Implement a password policy that requires employees to use strong, complex passwords that are changed regularly. This can help prevent unauthorised access to the database and reduce the risk of data breaches caused by weak passwords.
3. Data Retention Policy: Create a data retention policy that specifies how long data should be kept in the database. This can help to lessen the risk of data breaches caused by outdated or unneeded data by preventing the build up of unnecessary data.
4. Data Backup Policy: Implement a data backup policy that ensures that the database is regularly backed up and that backups are stored securely offsite. This can help prevent data loss in the event of a disaster or other unexpected event.
5. Incident Response Plan: Create an incident response strategy that details what to do in the event of a security incident or data breach. This can make sure issues are handled quickly and effectively, limiting the organization's potential harm.

## Step-by-step

### Enabling SSL

Cd into data directory

Create new server.crt and server.key

openssl req -nodes -new -x509 -keyout server.key -out server.crt -subj  
‘/C=IE/L=Dublin/O=TUDublin/CN=postgres’

Graphical user interface, application

Description automatically generated

Figure 2 Server.crt & Server.key

Edit the postgresql.conf configuration file, which is usually located in the data directory of your PostgreSQL installation.

Look for the following lines in the file:

#ssl = off

#ssl\_cert\_file = 'server.crt'

#ssl\_key\_file = 'server.key'

Uncomment these lines by removing the # symbol at the beginning of each line.

Set the ssl parameter to on and specify the paths to your SSL certificate and key files.

ssl = on

ssl\_cert\_file = '/path/to/server.crt'

ssl\_key\_file = '/path/to/server.key'

Graphical user interface, text

Description automatically generated

Figure 3 SSL postgresql.conf

Save the changes to postgresql.conf and exit the file.

Restart the PostgreSQL server to apply the changes.

# Auditing

Auditing is important in this HR database for several reasons.

1. Auditing helps to ensure that the data in the database is accurate and reliable. By tracking changes to the data, you can quickly identify any discrepancies or errors, and take steps to correct them before they become larger issues.
2. Aids in the detection of potential security breaches or unauthorised database access. You can instantly discover any suspicious activity, such as attempts to access data that the user is not authorised to view or efforts to edit or delete data in the database, by tracking who has accessed the database and when.
3. Ensures that regulatory standards are met. Many industries are required by law to keep accurate and secure records and to report on any changes or access to those documents. You may guarantee that you are satisfying these regulatory standards by inspecting the database and avoiding potential fines or penalties for noncompliance.

PostgreSQL 15 offers several options for auditing, including:

1. Logging: PostgreSQL can be configured to log all database activity to files on disk. This includes login attempts, queries, updates, and more.
2. Audit Extension: PostgreSQL provides a third-party extension called "pgAudit" that can be used to capture and log detailed information about database activity. This extension is highly configurable and can be used to track specific events or user actions.
3. Triggers: PostgreSQL's triggers can be used to capture specific events and actions in the database. For example, a trigger can be set up to capture all updates to a specific table.

## Discussion

I've decided to utilize the PGAudit audit extension. This can be configured to capture a wide range of database events and operations, such as logins, queries, changes, and deletes. You can also customize it to collect certain events depending on a user's role, database, or other criteria.

## Step-by-step

To achieve auditing for the HumanResources database follow these steps:

Note – PostgreSQL v15

git clone <https://github.com/pgaudit/pgaudit.git>

cd pgaudit

git checkout REL\_15\_STABLE

make install USE\_PGXS=1 PG\_CONFIG=/usr/pgsql-15/bin/pg\_config

sudo nano /etc/postgresql/15/main/postgresql.conf

logging\_collector = on

log\_destination = 'csvlog'

shared\_preload\_libraries = 'pgaudit'

pgaudit.log = 'ddl, read, write'

sudo systemctl restart postgresql

log in with postgres user

CREATE SCHEMA pgaudit;

CREATE TABLE pgaudit.audit\_log (

event\_time timestamp with time zone,

user\_name text,

command\_tag text,

object\_schema text,

object\_name text,

audit\_type text,

session\_id text,

server\_address text,

client\_address text,

application\_name text,

backend\_type text,

backend\_pid integer,

statement text,

tags text[]

);

GRANT INSERT ON pgaudit.audit\_log TO postgres;

ALTER DATABASE humanresources SET pgaudit.log='all';

SELECT \* FROM pgaudit.audit\_log;

sudo su postgres

cd /var/lib/postgresql/15/main/

cat current\_logfiles

nano into current logfile

# Performance Optimisation

## Discussion

Bottlenecks and potential performance issues:

1. Triggers: The use of triggers can slow down database performance, especially if they are complex or run frequently. In this database, there are two triggers (insert\_employee\_trigger) and (create\_store\_view\_trigger) that create views and update other tables. If these triggers are executed frequently or on large numbers of records, they could cause performance issues.
2. Views: The creation of views can also slow down performance, especially if they are complex or involve multiple tables. In this database, the create\_store\_employees\_view function creates a view for each store that shows the employees in that store. If there are a large number of stores or employees, this could cause performance issues.
3. Sequences: There are two sequences (employee\_id\_seq) and (store\_id\_seq) in this database. If these sequences are not set up correctly, they could cause performance issues.

Options available to optimise performance:

1. Use indexing: Indexes can help to speed up queries by allowing the database to quickly find the data it needs. You could create indexes on the columns used most frequently in WHERE clauses or JOIN statements.
2. Upgrade hardware: This could involve increasing the RAM or CPU resources or moving to a faster storage system such as solid-state drives (SSDs).
3. Optimise queries: Query optimisation can significantly improve the performance of a database by reducing the amount of time it takes to retrieve data. This is especially important for large databases with millions of records or when dealing with complex queries.
4. Use caching: Caching can help to improve performance by storing frequently accessed data in memory, rather than querying the database every time.

I've decided to employ both partitioning and indexing in my database. Without an index, the database would have to scan every row in the table which can take time and resources, especially for large datasets. Partitioning can help to improve database performance by reducing the quantity of data that needs to be scanned when running queries.

## Step-by-step

To create Indexes on employees, payments and performance reviews:

CREATE INDEX employees\_store\_id\_idx ON employees (store\_id);  
CREATE INDEX payments\_employee\_id\_idx ON payments (employee\_id);  
CREATE INDEX performance\_reviews\_employee\_id\_idx ON performance\_reviews (employee\_id);

To partition the performance review tables by the year the review was created at:

CREATE TABLE performance\_reviews (  
 employee\_id INT REFERENCES employees(employee\_id),  
 text TEXT,  
 rating INT,  
 bonus INT,  
 created\_at TIMESTAMP NOT NULL DEFAULT *NOW*()  
)  
PARTITION BY RANGE(*EXTRACT*(YEAR FROM created\_at));  
  
-- Create the child tables for each year  
CREATE TABLE performance\_reviews\_2020 PARTITION OF performance\_reviews  
 FOR VALUES FROM (2020) TO (2021);  
  
CREATE TABLE performance\_reviews\_2021 PARTITION OF performance\_reviews  
 FOR VALUES FROM (2021) TO (2022);  
  
CREATE TABLE performance\_reviews\_2022 PARTITION OF performance\_reviews  
 FOR VALUES FROM (2022) TO (2023);  
  
CREATE TABLE performance\_reviews\_2023 PARTITION OF performance\_reviews  
 FOR VALUES FROM (2023) TO (2024);

# Backup/Recovery/Availability

## Discussion

Options for backup/recovery include:

1. pg\_dump: This is a PostgreSQL application that lets you create a text file containing SQL commands that may be used to reconstruct database objects and data. It is a straightforward backup option, however it may not be appropriate for really large databases.
2. pg\_basebackup: You can use this utility to create a backup of a PostgreSQL database cluster. All database objects, data, and configuration files are included in this backup. It can be used for backup as well as replication.
3. WAL archiving[[1]](#footnote-1): PostgreSQL provides the ability to archive transaction logs for point-in-time recovery. This requires setting up a backup server and archiving the transaction logs to the backup server.
4. Third-party backup solutions: There are also several third-party backup solutions available for PostgreSQL, such as Barman [[2]](#footnote-2)and pgBackRest[[3]](#footnote-3), that offer additional features and functionalities.

## Step-by-step

For my database I have chosen to implement a script that runs pg\_dump on a cron job. To achieve this follow these steps:

1. Create a new file for the script

sudo nano backup\_script.sh

pg\_dump humanresources > /usr/local/backup.sql

1. Apply permissions

chmod +x backup\_script.sh

1. Set up cron task

0 2 \* \* 0 /path/to/backup\_script.sh

Should you wish to restore the Database to any of the previous backups run the following command

pg\_restore -C -d new\_db backup.sql

Additional backup and recovery options are also available through the PGAdmin extension as mentioned before.

Graphical user interface, application

Description automatically generated

Figure 4 PGAdmin backup tool

1. https://www.postgresql.org/docs/current/continuous-archiving.html [↑](#footnote-ref-1)
2. https://pgbarman.org/ [↑](#footnote-ref-2)
3. https://pgbackrest.org/ [↑](#footnote-ref-3)