

Class: System Integration

Exercise name: 04b Database Granular Access

Student name: Hero Englund

Index:

[Choice of Database](#)

[Remarks on Real World Application of Access Control](#)

[Database Schema](#)

[Granular Access Control Setup](#)

[User Guide](#)

[Introduction](#)

[Installation Instructions](#)

[Connect to the Database](#)

[Server Address](#)

[Username and Passwords](#)

[Queries](#)

[GUI options](#)

[Query Tool](#)

[Example Queries](#)

[Sue and Pepper](#)

[Superadmin](#)

[Read-Only](#)

[No-access](#)

Choice of Database

Database access should be allocated following the least access principle. Generally speaking, the granularity level of access control depends on the sensitivity of the data, and who has real need to access it. All types of databases offer options for limiting access to database resources from some users on some level.

In relational databases, access rights can typically be defined at least on database, and table level, as read only or read and write privilege. Some also offer access control on schema (collection of related tables) level. Some databases also offer a way to restrict access to a column or a row, or both. The smallest meaningful unit to apply security to in a relational database is a cell at an intersection of a row and a column.

NoSQL databases on the other hand are typically used for storing unstructured data like documents, multimedia or JSON. Since the concepts of rows and columns aren't meaningful when talking about unsorted or semi-sorted data in a document based database, the

mechanisms used to control access vary from those used in relational databases. One such example is flag based access control, where flags are embedded in documents. Based on the flag values and the user's permissions, access is granted or denied.

Role based access control (RBAC), attribute based access control (ABAC) and Flag based access control can be used both in relational and NoSQL databases, but embedding flags into flexible NoSQL documents is easier. Access control can be implemented on all levels in some versions of both Relational and NoSQL databases, but not all.

I chose PostgreSQL for this exercise, because the focus is solely on granularity of database access. PostgreSQL allows controlling permissions down to individual cells. Row-level security makes it possible to define who sees what, based on user attributes and data content.

If the database would contain unstructured data, I would choose a NoSQL database such as MongoDB, because it offers flexible field-level access control and the ability to embed flag-based markers directly within its document structure.

Remarks on Real World Application of Access Control

I would argue that the consideration of granularity level of database access that a given database offers, is a high priority, mostly when security requirements are very high. This is to say that granularity alone will rarely be a deciding factor in a real world scenario.

The choice of a database depends on multiple factors.

- Type of Data
- Type of the System (fx. Distributed system with many microservices)
- Amount of Data
- Ease of Searching Data
- Organisation's mandated policy, licensing, suppliers, contract restraints and pre-existing ecosystem
- Team Familiarity
- Security Requirements set by organisation, clients or legislation
- Type of Queries (mostly read, or many write queries also)
- Scalability
- Associated Cost

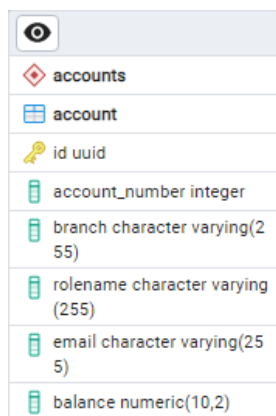
Access control could also be achieved on an architectural level by utilising Data Access Patterns such as CQRS (Command Query Responsibility Segregation) where one database is a read only version of the database that can be modified. These databases could be of different types, optimised for just reading or writing. Queries would be directed to different databases according to the nature of the query ie. read only queries would never touch the database where modifications are done.

Most large companies integrate their database access control with external systems like Active Directory or LDAP (Lightweight Directory Access Protocol). Use of these Directory Services has several benefits. It enables allocation of responsibilities between developers and system administrators, so that each can concentrate on their speciality. Developers can focus on data integrity, performance optimisation and schema design, whereas Sys Admins can take care of user and group management, and access control policies.

Database Schema

This is a PostgreSQL database with a schema “accounts” and within it one table “account”.

The account table has the following fields:



accounts
account
id uuid
account_number integer
branch character varying(255)
rolename character varying(255)
email character varying(255)
balance numeric(10,2)

Granular Access Control Setup

Users are granted different levels of access according to their roles. A superuser has full read and write privileges, while a read only user can view the contents of the table but not write. A no access user can form a connection to the database but not read or write. Additionally, to demonstrate cell level access control, there are two users with the same generic access level. They are only allowed to partially read their own data, and only allowed to write into one cell, to change their email address.

To achieve this different policies are created and applied to users. The stricter the access, the more detailed policies need to be defined. For instance, for the read_own users pepper and sue, the keyword SELECT use is restricted to only certain fields, and UPDATE is only allowed for one cell. Each user must be explicitly given rights to use the schema and the table, underlying any other types of rights they need. In short, this solution allows control on object, field and cell level, and gives us control over which operations users are allowed to perform.

User Guide

Introduction

These instructions describe how to access the database for System Integration Exercise “04b.Database Granular Access” in a Windows environment. The database can be accessed using different tools, but here we are using a pgAdmin4 GUI for visualising the data. We will install it by using the Chocolatey package manager. Pgadmin4 also works on Linux and Mac OS, but this guide doesn’t cover the installation on those platforms. You can find the downloads for different platforms [here](#)

Installation Instructions

If you don’t have Chocolatey package manager, you can install it by pasting this command into Admin Powershell:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object  
System.Net.WebClient).DownloadString("https://community.chocolatey.org/install.ps1"))
```

This is what you will see when you use the script:

```

PS C:\WINDOWS\system32> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex (System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1')
Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)
Getting latest version of the Chocolatey package for download.
Not using proxy.
Getting Chocolatey from https://community.chocolatey.org/api/v2/package/chocolatey/2.2.2.
Downloading https://community.chocolatey.org/api/v2/package/chocolatey/2.2.2 to C:\Users\HELENA\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip
Not using proxy.
Extracting C:\Users\HELENA\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip to C:\Users\HELENA\AppData\Local\Temp\chocolatey\chocoInstall
Installing Chocolatey on the local machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
  Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
  before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
  (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
  and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

Creating Chocolatey folders if they do not already exist.

chocolatey.nupkg file not installed in lib.
Attempting to locate it from bootstrapper.
PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...
WARNING: Not setting tab completion: Profile file does not exist at
'C:\Users\HELENA\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
  first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
PS C:\WINDOWS\system32> choco install pgadmin4

```

Then use choco to install the package by using this command in PowerShell:

```
choco install pgadmin4
```

When asked for verification, select A to accept all to run all scripts, or Y for accepting one script at a time

```

PS C:\WINDOWS\system32> choco install pgadmin4
Chocolatey v2.2.2
Installing the following packages:
pgadmin4
By installing, you accept licenses for the packages.
Progress: Downloading pgadmin4 8.2.0... 100%

pgadmin4 v8.2.0 [Approved]
pgadmin4 package files install completed. Performing other installation steps.
The package pgadmin4 wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): a

Downloading pgadmin4 64 bit
  from 'https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v8.2/windows/pgadmin4-8.2-x64.exe'
Progress: 100% - Completed download of C:\Users\HELENA\AppData\Local\Temp\chocolatey\pgadmin4\8.2.0\pg
admin4-8.2-x64.exe (181.83 MB).
Download of pgadmin4-8.2-x64.exe (181.83 MB) completed.
Hashes match.
Installing pgadmin4...
pgadmin4 has been installed.
  pgadmin4 can be automatically uninstalled.
  The install of pgadmin4 was successful.
  Software installed to 'C:\Program Files\pgAdmin 4\'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\WINDOWS\system32>

```

Connect to the Database

We will make a connection as each of the users (each user having their own login and corresponding access rights).

To connect to an Azure Database for PostgreSQL - Flexible Server using pgAdmin 4:

1. **Open pgAdmin 4:** Launch the pgAdmin 4 application on your computer
2. **Register a new server:** In the pgAdmin 4 interface, right-click on "Servers" in the left-side browser tree, and select "Register" -> "Server" (Or just click the Add New Server in the middle of the main view)
3. **Configure server details:** In the "Register - Server" window, you will see multiple tabs. Fill in the following details:
 1. General tab
 1. **Name:** Provide a name for the connection
 2. Connection tab
 1. **Hostname/address:** Enter **postgresql-for-si.postgres.database.azure.com**
 2. **Port:** 5432
 3. **Maintenance database:** Leave the default
 4. **Username:** See usernames and passwords [here](#)
 5. **Password:** See usernames and passwords [here](#) (Toggle "Save password" on if desired)

Register - Server

General Connection Parameters SSH Tunnel Advanced

Host name/address: postgresql-for-si.postgres.database.azure.com

Port: 5432

Maintenance database: postgres

Username: read_only

Kerberos authentication?: ☐

Password:

Save password?: ☐

Role:

Service:

Close Reset Save

Server Address

postgresql-for-si.postgres.database.azure.com

Username and Passwords

Can make a database connection, but not read or write:

Username: no_access

Password: 664c4497-cbc7-442c-a824-008e82dac256

Can only read:

Username: read_only

Password: 2e409266-3300-4702-a06b-8d70fcf9aa95

Can read and write:

Username: superadmin

Password: 7cc4b989-1870-4acb-81e0-76b55b60a23d

Can read part of their own data and modify the email address only:

Username: sue

Password: 73850d24-61ef-45b3-a3b0-6397b154f0cc

Username: pepper

Password: a784cf2b-a25a-4070-af94-1bd213acbbc6

Queries

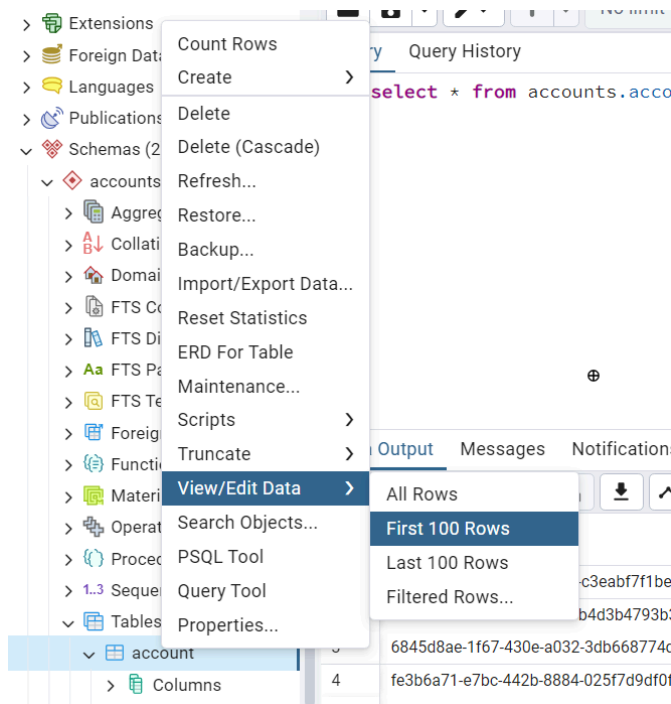
We can use the GUI to navigate in the table and attempt to modify the data, or make PostgreSQL queries in the query window.

GUI options

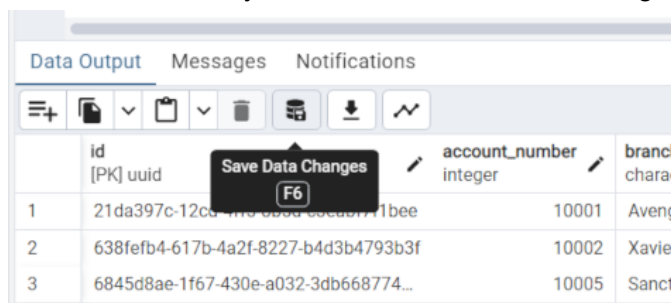
There are lots of elements in the drop downs. For the sake of this exercise, we will only need to concern ourselves with the “accounts” schema that has the “account” table.

To navigate to the table, click on postgres → schemas → accounts → tables → account.

Display data in table via the GUI by r-clicking on the table and selecting “View/Edit Data” and then “All Rows” or “First 100 Rows”:

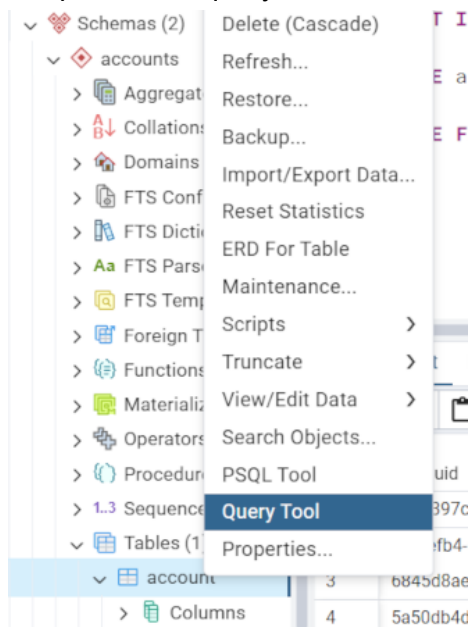


Modify data by selecting the appropriate field and change the value. Change won't be saved in the database until you click the “Save Data Changes” button.



Query Tool

To open a new query window: r-click on the table in question and select “Query Tool”



Type your queries in the Query Tool window and press F5 to execute a query that is marked in the query tool console, or press the play button.

Example Queries

Sue and Pepper

Sue and Pepper are users who only have access to their own account details, and even then two of the fields have been restricted because they are not relevant. The excluded fields are “id” (the account UUID) and “rolename”.

Should work for both of them:

```
SELECT account_number, email, branch, balance FROM accounts.account;
```

Should not work for either:

```
SELECT * FROM accounts.account;
```

```
SELECT FROM accounts.account WHERE rolename='pepper'; (or rolename='sue')
```

```
SELECT rolename, id FROM accounts.account;
```

Should work for both but this example is for Pepper:

```
UPDATE accounts.account SET email='spam@putler.ru' WHERE account_number=10003;
```

```
UPDATE accounts.account SET email='pepper.potts@starkindustries.com' WHERE  
account_number=10003;
```

Superadmin

Superadmin has all access rights in the database.

Should work:

```
SELECT * FROM accounts.account;
```

```
INSERT INTO accounts.account (account_number,branch,rolename,email,balance) VALUES  
( '10006', 'Wakanda','panther', 'black.panther@wakanda.org',5000);
```

```
UPDATE accounts.account SET balance=10 WHERE account_number=10006;
```

```
DELETE FROM accounts.account WHERE branch ='Wakanda';
```

Read-Only

This user can see all accounts with full information but not modify anything.

Should work:

```
SELECT * FROM accounts.account;
```

No-access

All queries from this user should result in “permission denied”.