Testing Granular Access in a PostgreSQL Database on Azure

This documentation outlines the steps to test granular access for a PostgreSQL database hosted on "Azure Database for PostgreSQL flexible server." The primary objective of this assignment is to demonstrate the implementation of granular access. Follow the steps below to connect and attempt data manipulation for testing granular access:

PostgreSQL is well-suited for implementing granular access control. It provides security features that allow you to define fine-grained permissions at various levels, some of the advantages include:

Role-based access control: Allowing you to define roles with specific sets of permissions and then assign users or other roles to those roles.

Row-Level Security (RLS): PostgreSQL supports row-level security policies, which allow you to control access to individual rows in a table based on specified criteria. RLS is useful for enforcing fine-grained access control based on the data itself.

Column-Level Security: While PostgreSQL does not have native support for column-level security out of the box, you can achieve similar effects using techniques such as views, RLS, or carefully designed database schemas.

Database Overview:

To be able to test, ensure you are working with an appropriate PostgreSQL database client. Enter the following credentials to access the database.

Database credentials:

- Server name/host: granularaccesslasse.postgres.database.azure.com
- Port: 5432
- Database: granular_access_db

User Identification and access level:

The following describes users with different access levels. Try and connect with the different usernames and passwords and execute the relevant queries to see if you get the correct response.

The database contains multiple tables that are only visible to specific users.

READ ONLY:

Username: “readonly”

Password: ”XSW23edc”

READ/WRITE:

Username: ”readwrite”

assword: ”XSW23edc”

NO READ OR WRITE ACCESS:

Username: “noread”

Password: “XSW23edc”

Here are some simple SQL Queries to try. Or come up with your own.

```
SELECT * FROM sensitive_data
```

```
INSERT INTO sensitive_data (data_value) VALUES ('New Data')
```