**Task 4. Prepare answers to the following questions**

1. **How can one restrict access to certain columns of a database table?**

Restricting access to specific columns in a database table can be achieved primarily through views and column-level privileges. Using views, you can create a virtual table that exposes only certain columns from the original table. Access is then granted to the view instead of the entire table, effectively hiding the other columns from users. For instance, a view can be designed to display only sensitive customer information such as names and IDs, while excluding financial data.

Additionally, PostgreSQL supports column-level privileges, allowing you to grant specific permissions for individual columns in a table. This method restricts users from accessing columns that aren't explicitly authorized. For example, you could grant SELECT privileges on only a subset of columns in the table, ensuring users cannot query unauthorized fields.

By employing these techniques, sensitive or restricted data can remain secure while users interact with only the necessary parts of a table. Combining these methods with role-based access control ensures granular and efficient access management.

1. **What is the difference between user identification and user authentication?**

User identification is the process of determining who the user claims to be. It involves recognizing the user based on a unique identifier such as a username, email address, or user ID. However, identification alone does not verify the legitimacy of the claim; it only establishes who the user is supposed to be. For example, entering a username into a login form is a part of the identification process.

User authentication, on the other hand, is the process of verifying that the user is indeed who they claim to be. Authentication relies on validating credentials such as a password, biometric data, or a token associated with the user’s identity. This step ensures the identity is genuine and authorized. For instance, when the system matches the entered password with the stored hash, it authenticates the user.

1. **What are the recommended authentication protocols for PostgreSQL?**

PostgreSQL supports various authentication protocols, but the most recommended include SCRAM-SHA-256, SSL/TLS, and certificate-based authentication for secure operations.

SCRAM-SHA-256 is a modern password authentication method that offers robust protection against attacks such as rainbow table attacks. It is considered more secure than older methods like MD5 and is recommended for most scenarios.

SSL/TLS authentication ensures secure data transmission by encrypting communication between the client and the database server. This protocol can be paired with other authentication methods, such as passwords or certificates, to enhance security.

Certificate-based authentication uses SSL client certificates to authenticate users. It provides strong security as only users with valid certificates can access the database, making it ideal for sensitive environments.

Other methods like peer authentication, GSSAPI/Kerberos, and LDAP authentication are also widely used depending on specific needs. For instance, peer authentication is used for local access based on operating system user identity, while LDAP and Kerberos integrate with enterprise authentication systems to enable seamless Single Sign-On (SSO).

1. **What is proxy authentication in PostgreSQL and what is it for? Why does it make the previously discussed role-based access control easier to implement?**

Proxy authentication in PostgreSQL enables one user or application to act on behalf of another user while interacting with the database. This mechanism is particularly useful when an intermediary, such as an application, needs to access the database as if it were the end user, without exposing the user's actual credentials.

Proxy authentication serves several purposes. It simplifies delegated access, where an application authenticates on behalf of a user, and ensures secure database access without requiring users to directly connect. It also facilitates centralized identity management by consolidating authentication and user management at the application layer.

In proxy authentication, the application logs into the database using its own credentials but specifies the end user's identity during the session. PostgreSQL then applies access control policies based on the proxy user’s role and permissions.

This approach simplifies role-based access control (RBAC) by centralizing access management. Instead of assigning individual permissions to each user in the database, you can assign permissions to roles and allow the application to dynamically map users to those roles. This eliminates the need to maintain fine-grained permissions for every user at the database level.

Proxy authentication also enables flexible user mapping, where an application can dynamically assign roles or permissions based on session-specific parameters. Additionally, it ensures consistency, as access policies are centrally enforced, reducing the complexity of managing individual database user accounts.

By leveraging proxy authentication, administrators can streamline RBAC implementations, enhance security, and ensure a more efficient and scalable database access system.