**Task 4: Answers to Database and Authentication Questions**

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

To restrict access to specific columns in a database table, use role-based access control and column-level permissions in PostgreSQL:

* Grant Column-Level Permissions: Use the GRANT command to allow access to specific columns:

REVOKE SELECT ON public.table\_name FROM role\_name;

GRANT SELECT (column1, column2) ON public.table\_name TO role\_name;

This ensures role\_name can only access column1 and column2.

* Verify Privileges: Check column-level grants:

SELECT table\_name, column\_name, privilege\_type, grantee

FROM information\_schema.column\_privileges

WHERE table\_schema = 'public';

* Use Views: Create a view exposing only allowed columns:

CREATE VIEW restricted\_view AS

SELECT column1, column2 FROM public.table\_name;

GRANT SELECT ON restricted\_view TO role\_name;

This approach ensures users only access permitted columns, enhancing security.

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

* User Identification: Determines *who* a user claims to be by providing a unique identifier (username like "luca\_valentin"). It answers, "Who are you?"
* User Authentication: Verifies the user’s identity by checking credentials (password, token) against stored records. It answers, "Can you prove you are luca\_valentin?"
* Key Difference: Identification establishes the claimed identity; authentication confirms its validity. Identification precedes authentication.

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

PostgreSQL supports several secure authentication methods, depending on the environment:

* SCRAM-SHA-256: The most secure password-based method, using a challenge-response mechanism with salted passwords to prevent sniffing and replay attacks:

# In pg\_hba.conf

host all all 0.0.0.0/0 scram-sha-256

* GSSAPI: Suitable for Kerberos-based authentication in enterprise environments: host all all 0.0.0.0/0 gss
* SSPI: Recommended for Windows environments using integrated security:

host all all 0.0.0.0/0 sspi

*SCRAM-SHA-256 is the default recommendation for modern applications due to its robustness.*

**4. What is proxy authentication in PostgreSQL, and why does it simplify role-based access control?**

Proxy Authentication: Allows a user to connect as one role (proxy) and switch to another (target) using SET ROLE without re-authenticating. For example:

*-- Grant proxy access*

GRANT target\_user TO proxy\_user;

*-- Connect as proxy\_user and switch*

SET ROLE target\_user;

*Purpose: simplifies privilege management by allowing a single user account to assume different roles with specific permissions, reducing the need for multiple user accounts.*

Why It simplifies Role-Based Access Control (RBAC):

* Centralized Permissions: privileges are assigned to group roles ( target\_user), and users inherit them via SET ROLE, streamlining administration.
* Fewer Credentials: users need only one set of credentials (for proxy\_user) to access multiple roles.
* Flexibility: users can switch roles dynamically during a session, enabling fine-grained access control.

*An example In the dvd\_rental database:*

An application might use a proxy role app\_user for all connections. The app\_user role is granted access to group roles like rental (with INSERT, UPDATE on rental table) and client\_maria\_stefania (with row-level access to specific customer rows). After connecting as app\_user, the application issues:

SET ROLE rental;

-- Perform rental operations

SET ROLE client\_maria\_stefania;

-- Access customer-specific data

This leverages RBAC by allowing app\_user to inherit the privileges of rental or client\_maria\_stefania dynamically, streamlining access control and reducing the need for multiple login credentials.

Proxy authentication thus simplifies RBAC by enabling efficient, secure, and flexible privilege management through role inheritance and dynamic role switching.