**Task 4**

1. In PostgreSQL, column-level access control is implemented using the GRANT and REVOKE statements. You can grant SELECT, INSERT, or UPDATE privileges on specific columns like this:

GRANT SELECT (first\_name, last\_name) ON customer TO rentaluser;

This allows the user to view only the first\_name and last\_name columns of the customer table. You can also:

* Use views to expose only specific columns.
* Combine with row-levelsecurity for finer control.

1. User identification is the process of declaring who the user is — usually by providing a username. User authentication is the process of verifying that the user is actually who they claim to be — typically done by checking a password, certificate, or key.
2. PostgreSQL supports several authentication methods. Recommended secure options include:

**scram-sha-256**:  
The most secure password-based method; recommended over md5. Requires PostgreSQL 10+.

password\_encryption = scram-sha-256

cert (client certificate authentication):  
Used in secure environments where SSL/TLS certificates are managed.

GSSAPI/SSPI/Kerberos:  
Suitable for enterprise single sign-on (SSO) systems.

LDAP and PAM:  
For integration with external authentication systems.

Not recommended:

Trust (no authentication)

Plaintext password (unless used with SSL)

1. **Proxy authentication** allows one role (e.g., a web app user) to assume the identity of another role temporarily using the SET ROLE command.

A web application connects using a general role (e.g., app\_user) and then switches to a specific role:

SET ROLE client\_Mary\_Smith;

Allows fine-grained control per actual user while managing connections through a single technical user.

Simplifies role-based access control (RBAC): different permissions can be managed via roles rather than hard-coded in application logic.

Useful when combined with RLS and personalized policies, as discussed in previous tasks.