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

It can be done in several ways. For example, restrictions can be implemented using **column-level security**. This is achieved with the GRANT and REVOKE commands. For instance, we can create a role for a specific user—say, **Patricia** from the dvdrental database. Then, we revoke all privileges on the payment table from Patricia and grant her SELECT privilege on only the amount column (if we only want to allow her to see how much she paid for the movies).

CREATE ROLE patricia

REVOKE ALL ON payment FROM patricia;

GRANT SELECT (amount) ON payment TO patricia;

Also, it is possible to restrict access by creating **view** by granting access to certain user to the view but not the whole table or letting to see just certain columns.

Example:

CREATE VIEW public\_customer\_info AS

SELECT

customer\_id,

first\_name,

last\_name,

email

FROM customer;

REVOKE ALL ON customer FROM public;

GRANT SELECT ON public\_customer\_info TO reporting\_user;

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

User identification is the process of naming the user—providing a username by which the user is identified. User authentication is about proving a proof that the user is indeed who he/she/they claim to be.

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

Some recommended authentication methods are:

* **SCRAM-SHA-256** - This one is the most recommended for password security. It uses strong hashing to keep passwords safe and makes it really hard for someone to “sniff” login info. Basically, it’s the most secure built-in option available.
* **Client certificates (SSL)** – super secure and great for high-trust environments. It uses SSL to encrypt everything between your app and the database, which helps prevent anyone from spying on user traffic. PostgreSQL doesn’t create SSL itself, but it works great with SSL tools to keep data safe during transfer.
* **GSSAPI / Kerberos** – Perfect for corporate setups. It lets users log in securely without actually sending their passwords to PostgreSQL at all. Way better for enterprise environments where single sign-on and extra security are a must.
* **LDAP** – Handy if you're managing users in a centralized directory. User can control who gets in based on group membership or location in the directory. When setting up pg\_hba.conf, user or admin can even add filters so only certain users are allowed—like filtering by department or team.
* **Peer authentication** - simple and useful for local setups. It checks if your system’s OS user matches a PostgreSQL role. If they match, you're in. No password needed.

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 involves a user first logging in to the database then impersonating someone else so they can make something happen—without requiring that second person's password when you log in. It's done by SET ROLE or SET SESSION AUTHORIZATION. It's useful if you have a shared connection pool but you still desire per-user protection. Proxy authentication also integrates well with role-based access control because you can utilize a single database connection that takes on different roles depending on the user's identity. This simplifies permission management—just assign users to roles with specific privileges.