Administração de Banco de Dados

Aulas 3 e 4

INF INSTITUTO DE INFORMÁTICA UFG UNIVERSIDADE FEDERAL DE GOIÁS

PODER JUDICIÁRIO
Tribunal de Justiça do Estado de Goiás

Programa de Residência em TI

Plano de Aula

- Introdução
- Controle de Acesso em Camadas
  - Camada 1: Autenticação de Conexão
    - pg_hba.conf e as regras de conexão
  - Camada 2: Roles e Seus Atributos
  - Camada 3: Filiação a Roles
    - Conceito de grupos
  - Camada 4: Permissões em Objetos
    - Comandos GRANT e REVOKE
    - Privilégio USAGE
- O que é um Schema?
- Princípios gerais do controle de acesso
  - Regras de Ouro do DBA
- Laboratório
  - Criação do sistema de controle de acesso no banco dvdrental

Introdução

- Autenticação vs Autorização
  - **Autenticação**: verifica se o usuário é quem ele afirma ser
  - **Autorização**: verifica se um usuário autenticado tem permissão para executar uma ação
- **Principle**: nome usado para se referir a uma entidade a qual se pode garantir privilégios de acesso em um sistema de informação
  - Existe apenas um principle no PostgreSQL: o **Role**
    - Por convenção, um role com privilégio de login é chamado de **usuário** (sem o privilégio, é chamado de **grupo**)
- **Superuser**: um role que escapa de todas as checagens de privilégio e pode acessar e modificar qualquer objeto ou configuração do cluster
  - Análogo ao conceito de root em sistemas Linux
  - Usuário **postgres** por padrão

Introdução

- Conceito de posse (**ownership**)
  - No PostgreSQL, o role que cria um objeto (table, view, function, ...) se torna o “dono” (**owner**) do objeto.
  - A posse pode ser alterada/transferida após a criação.
  - É possível consultar as relações de posse usando o comando \d no psql, ou consultando as tabelas no schema pg_catalog.
  - Há três pontos importantes no conceito de posse:
    - Apenas um *superuser* ou *owner* de um objeto pode executar ALTER/DROP nele.
    - Apenas um *superuser* ou *owner* de um objeto pode alterar o *ownership* do objeto.
    - Apenas o *owner* (ou um *superuser*) pode definir privilégios padrão para os objetos que ele cria.

Camada 1: Autenticação de Conexão

- Qualquer ação dentro do BD depende, primeiro, de uma conexão.
- Quem controla as conexões é o arquivo **pg_hba.conf**
  - Host-Based Authentication
  - Define regras de conexão, lidas uma a uma, de cima para baixo
  - A primeira regra vence:
    - Para na primeira linha que corresponde ao tipo de conexão (local ou rede), banco de dados, nome de usuário e endereço de IP fornecidos. Regras subsequentes são ignoradas. Se nenhuma regra corresponder, a conexão falha.
- A estrutura de uma regra é:
  - **TIPO BANCO USUÁRIO ENDEREÇO MÉTODO**
- Placeholder

Camada 1: Autenticação de Conexão

- **TIPO**: local (sockets Unix), host (conexões de rede TCP/IP), hostssl (conexão de rede com SSL obrigatório).
- **BANCO**: O banco de dados ao qual se deseja conectar (nome do banco).
- **USUÁRIO**: O nome do role/usuário que está tentando se conectar (ex: postgres, all).
- **ENDEREÇO**: Uma faixa de endereços IP em notação CIDR (ex: 192.168.1.0/24).
- **MÉTODO**: A forma de autenticação.
  - **scram-sha-256**: Método preferencial, usa um desafio criptográfico que não expõe a senha na rede (Postgresql tem se movido para torná-lo o padrão).
  - **md5**: Método mais antigo, ainda comum, mas menos seguro.
  - **peer**: Para conexões do tipo local, verifica se o nome do usuário do SO é o mesmo do usuário do BD (útil para ADMs locais).
  - **trust**: PERIGO! Aceita conexões sem senha. Nunca use em produção.
  - **reject**: Nega explicitamente a conexão.
  - Mais detalhes no anexo 1.

Camada 1: Autenticação de Conexão

- Como o PostgreSQL Utiliza o pg_hba.conf

1. **Tentativa de Conexão**: Um cliente tenta estabelecer uma conexão, informando o host, a porta, o nome do usuário (role) e o banco de dados de destino
2. **Verificação**: O processo Postmaster no servidor reconhece a conexão e cria um processo filho para servi-la.
3. **Leitura**: O processo lê o arquivo pg_hba.conf de cima para baixo, linha por linha.
4. **A Primeira Regra Vence**: O processo para na primeira linha que corresponde com os parâmetros informados na requisição da conexão
5. **Aplicação do Método**: O método de autenticação especificado é executado. Se for bem-sucedido, o acesso é concedido. Se falhar, a conexão é encerrada.
6. **Rejeição Padrão**: Se o processo ler o arquivo inteiro e nenhuma linha corresponder à tentativa de conexão, o acesso é negado.

Camada 1: Autenticação de Conexão

- **Exemplo**:

```
# Permitir que o superusuário 'postgres' se conecte localmente sem senha
local   all         postgres                peer

# Exigir senha SCRAM para qualquer usuário conectando localmente via rede
host    all         all         127.0.0.1/32        scram-sha-256
host    all         all         ::1/128             scram-sha-256

# Permitir que a equipe de análise (rede 192.168.1.0/24) se conecte ao banco
host    dvdrental   all         192.168.1.0/24      scram-sha-256

# Rejeitar todas as outras tentativas de conexão de rede
host    all         all         0.0.0.0/0           reject
```

Camada 2: Roles e Seus Atributos

- No PostgreSQL não há distinção entre “usuários” e “grupos”. Existe apenas um conceito: o **Role**.
  - Uma role pode ser um usuário, um grupo, ou ambos. Pense nele como uma identidade dentro do banco de dados.
- Essa identidade possui atributos inerentes que definem suas propriedades
  - **LOGIN**: Um role com esse atributo é o que chamamos de “usuário” ou “login role”. Ele pode iniciar uma conexão. Um role sem LOGIN é o que chamamos de “grupo”.
  - **PASSWORD**: Para ser usando em roles com o atributo LOGIN.
  - **SUPERUSER**: Pode contornar todas as permissões (arriscado!).
  - **CREATEDB**: Pode criar novos bancos de dados.
  - **CREATEROLE**: Pode criar, alterar e remover outros roles (exceto superusuários).
  - Mais detalhes no anexo 2.
- Placeholder

Camada 2: Roles e Seus Atributos

- **Exemplo**:

```
-- Ana é uma usuária padrão, com permissão para login e uma senha segura.
CREATE ROLE ana_silva WITH
  LOGIN
  PASSWORD 'uma_senha_muito_forte_e_longa';

-- Bruna é uma usuária mais experiente, que pode criar outros roles e bancos.
CREATE ROLE bruna_costa WITH
  LOGIN
  PASSWORD 'outra_senha_segura_e_diferente'
  CREATEDB
  CREATEROLE;
```

Camada 3: Filiação a Roles

- Esta é a forma como o PostgreSQL gerencia “grupos”.
  - Em vez de conceder permissões a cada usuário individualmente (o que é muito difícil de gerenciar), podemos conceder permissões a um “grupo” e depois adicionarmos usuários a ele.
  - **Um grupo é simplesmente um role sem o atributo LOGIN**.
- Essa identidade possui atributos inerentes que definem suas propriedades
  - **LOGIN**: Um role com esse atributo é o que chamamos de “usuário” ou “login role”. Ele pode iniciar uma conexão. Um role sem LOGIN é o que chamamos de “grupo”.
  - **PASSWORD**: Para ser usando em roles com o atributo LOGIN.
  - **SUPERUSER**: Pode contornar todas as permissões (arriscado!).
  - **CREATEDB**: Pode criar novos bancos de dados.
  - **CREATEROLE**: Pode criar, alterar e remover outros roles (exceto superusuários).
  - Mais detalhes no anexo 2.
- Placeholder

Camada 3: Filiação a Roles

- **Exemplo**:

-- Um grupo para analistas que só podem ler dados:

CREATE ROLE analysts_read_only;

GRANT ...

-- Um grupo para gerentes que podem editar certas informações:

CREATE ROLE data_managers;

GRANT ...

-- Ana entra no grupo de leitura:

GRANT analysts_read_only TO ana_silva;

-- Bruna fará parte de ambos:

GRANT data_managers TO bruna_costa;

Camada 4: Permissões em Objetos

- Camada mais granular. Usamos os comandos **GRANT** e **REVOKE** para dar e tirar permissões em objetos específicos do banco de dados.
- O comando GRANT tem, basicamente, duas variantes:
  - Garante **privilégios em um objeto** do banco de dados
  - Garante **pertencimento a um role** (grupo)
- **USAGE**: é uma permissão especial que permite a um usuário “olhar para dentro” de um objeto
  - Schemas: permite visualizar os objetos no schema
  - Sequence: permite invocar as funcoes currval e nextval
  - ...

O que é um Schema?

- ***namespace***
  - Coleção nomeada de objetos do banco de dados, incluindo:
    - Tables
    - Views
    - Indexes
    - Types
    - Functions
    - ...

O que é um Schema?

- Schemas nos permitem promover:
  - **Organização**:
    - Num BD complexo podemos ter centenas de tabelas; sem a habilidade de criação de ***namespaces*** podemos acabar com uma sopa de objetos
    - Ex.:
      - Aplicação e-commerce:
        - ***Web***: products, customers, products
        - ***Billing***: invoices, transactions
        - ***Warehousing***: inventory, stock-level tables
        - ***Reporting***: business reports

O que é um Schema?

- Schemas nos permitem promover:
  - **Segurança e Controle de Acesso**:
    - Uso mais crítico; permissões podem ser concedidas a todo um Schema, simplificando o gerenciamento de acesso
    - Ex.:
      - Aplicação e-commerce:
        - ***Web***: acesso CRUD apenas às tabelas acessíveis a um usuário padrão do sistema web
          - Mesmo que um agente malicioso obtenha acesso ao schema web atráves de um usuário comum, o dano fica restrito ao schema de usuário

O que é um Schema?

- Schemas nos permitem promover:
  - **Multi-Tenancy**:
    - Padrão arquitetural segundo o qual um sistema único consegue servir múltiplos clientes distintos (*tenants*)
    - Ex.:
      - Aplicação e-commerce:
        - Tenant A: web user
        - Tenant B: data analyst

O que é um Schema?

- **search_path**
  - Postgresql procura no search_path os schemas aos quais um usuário tem acesso para resolver consultas nas quais o schema não esteja definido
    - Por padrão: $user, public
      - SHOW search_path;
    - Pode ser manipulado:
      - SET search_path = 'web', 'public';

O que é um Schema?

- Comandos para manipulação de schemas:
  - **CREATE SCHEMA my_schema;**: Cria um novo schema chamado my_schema.
  - **DROP SCHEMA my_schema;**: Destrói um schema, mas falha se o mesmo conter objetos.
  - **DROP SCHEMA my_schema CASCADE;**: Destrói um schema e todos os objetos relacionaos (CUIDADO!!!).

Regras de Ouro do DBA

- Use scram-sha-256 e force o uso de SSL no pg_hba.conf
- Jamais use um role SUPERUSER para uma aplicação
- Siga o Princípio do Privilégio Mínimo
- Gerencie permissões através de grupos, não usuários individuais
- Use schemas e views para criar barreiras de segurança
- Não dê acesso direto às tabelas, a menos que seja inevitável
- Audite as permissões regularmente