Gerenciando Usuários e Permissões no PostgreSQL

Administrando Usuários e Privilégios no PostgreSQL

Todo agrupamento de bancos de dados possui um conjunto de usuários de banco de dados. Estes usuários são distintos dos usuários gerenciados pelo sistema operacional onde o servidor executa. Eles possuem objetos de banco de dados (por exemplo, tabelas, visões etc.), e podem conceder privilégios nestes objetos para outros usuários controlando, assim, quem pode acessar qual objeto.

A DCL controla os aspectos de autorização de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados. Os nomes dos usuários de banco de dados são globais para todo o agrupamento de bancos de dados (e não apenas próprio de cada banco de dados).

No decorrer deste artigo, veremos conceitos relacionados a usuários, grupos e “roles” (papéis), ao passo que aprenderemos como criá-los e quando utilizá-los em nossas aplicações.

Criação de Usuário de Banco de Dados – CREATE USER

Como foi mencionado, existem usuários que são independentes do sistema operacional, que servem para manipular objetos do SGBD.

No PostgreSQL existe o comando CREATE USER que adiciona um novo usuário no bancos de dados. Este comando já foi muito utilizado, mas mesmo ainda funcionando, hoje é apenas um *alias* para o comando CREATE ROLE que veremos mais adiante. O comando genérico de criação de um usuário está exibido na **Listagem 1**.

Listagem 1. Comando de Criação de um gatilho no PostgreSQL

```
CREATE USER nome [ [ WITH ] opção [ ... ] ]
```

As opções que acompanham este comando estão listadas na **Tabela 1**.

Opção	**Significado**
Nome	O nome do usuário
Id_do_usuário	A cláusula SYSID pode ser utilizada para escolher o identificador de usuário do PostgreSQL do novo usuário.
CREATEDB | NOCREATEDB	Estas cláusulas definem a permissão para o usuário criar banco de dados. Se for especificado CREATEDB, o usuário sendo definido terá permissão para criar seus próprios bancos de dados. Se for especificado NOCREATEDB, nega-

	se ao usuário a permissão para criar banco de dados. Se nenhuma destas cláusulas for especificada, o padrão é NOCREATEDB.
CREATEUSER | NOCREATEUSER	Estas cláusulas determinam se o usuário pode ou não criar novos usuários. CREATEUSER também torna o usuário um super-usuário, o qual pode passar por cima de todas as restrições de acesso. Se nenhuma destas cláusulas for especificada, o padrão é NOCREATEUSER.
Nome_do_grupo	O nome de um grupo existente onde o usuário será incluído como um novo membro. Podem ser especificados nomes de vários grupos.
Senha	Define a senha do usuário. Se não se pretende utilizar autenticação por senha, esta opção pode ser omitida, mas o usuário não poderá mais se conectar se for decidido mudar para autenticação por senha. A senha poderá ser definida ou mudada posteriormente através do comando ALTER USER.
ENCRYPTED | UNENCRYPTED	Estas cláusulas controlam se a senha será armazenada criptografada, ou não, nos catálogos do sistema.
Data_e_hora	A cláusula VALID UNTIL define uma data e hora após a qual a senha do usuário não é mais válida. Se esta cláusula for omitida, a conta será válida para sempre.

Tabela 1. Opções do comando CREATE USER.

Removendo um Usuário no Banco de Dados – DROP USER

Uma vez criado no banco de dados, o comando DROP USER remove o usuário especificado. Porém, esse comando não remove as tabelas, visões ou outros objetos pertencentes ao usuário. Se o usuário possuir algum banco de dados, uma mensagem de erro será gerada.

O comando que remove um usuário do PostgreSQL está mostrado na **Listagem 2**.

Listagem 2. Comando que remove um usuário no PostgreSQL

```
DROP USER nome_usuário
```

Neste caso,*nome_usuário*refere-se ao usuário que está sendo removido. Para remover um usuário que possui um banco de dados, primeiro remove-se o banco de dados ou muda-se o dono do mesmo.

Alterando um Usuário – ALTER USER

Em alguns casos, é conveniente modificar um usuário, seja para alterar sua senha ou validade da mesma ou para incluir a permissão de criação de usuários ou retirá-la, caso exista etc. O comando para modificar um usuário é o ALTER USER e está demonstrado na **Listagem 3**.

Listagem 3. Comando que altera um usuário no PostgreSQL

```
ALTER USER nome [ [ WITH ] opção [ ... ] ]
```

As opções desse comando são exibidas na **Tabela 2**.

Opção	**Significado**
Nome	O nome de usuário cujos atributos estão sendo alterados.
CREATEDB | NOCREATEDB	Estas cláusulas definem a permissão para o usuário criar banco de dados. Se CREATEDB for especificado, o usuário terá permissão para criar seus próprios bancos de dados. Especificando NOCREATEDB nega-se ao usuário a permissão para criar banco de dados (se o usuário for um super-usuário então esta definição não tem efeito prático).
CREATEUSER | NOCREATEUSER	Estas cláusulas determinam se o usuário terá permissão para criar novos usuários. CREATEUSER torna o usuário um super-usuário, não sujeito a restrições de acesso.
Senha	A nova senha a ser utilizada para esta conta.
ENCRYPTED | UNENCRYPTED	Estas palavras chave controlam se a senha é armazenada criptografada, ou não, em pg_shadow (Consulte o comando CREATE USER para obter informações adicionais sobre esta opção).
data_e_hora	A data (e, opcionalmente, a hora) de expiração da senha do usuário. Para fazer com que a senha nunca expire deve ser utilizado 'infinity'.
Novo_nome	O novo nome de usuário.
Parâmetro valor	Define o valor fornecido como sendo o valor padrão para o parâmetro de configuração especificado. Se valor for DEFAULT ou, de forma equivalente, se RESET for utilizado, a definição da variável específica para o usuário é removida, e o valor padrão global do sistema será herdado nas novas sessões do usuário. Use RESET ALL para remover todas as definições específicas do usuário.

Tabela 2. Opções do comando ALTER USER

Uma variação do comando ALTER USER exibido na **Listagem 4**, muda o nome de um usuário. É importante comentar que apenas um super-usuário pode mudar o nome de outro usuário.

Listagem 4. Comando para alterar o nome de um usuário

```
ALTER USER nome RENAME TO novo_nome
```

Conceitos de Grupos de Usuários no PostgreSQL

Assim como nos sistemas operacionais baseados em Unix, os grupos são uma forma lógica de juntar usuários para facilitar o gerenciamento de privilégios. Tais privilégios podem ser concedidos, ou revogados, para o grupo como um todo. Para criar um grupo, deve ser utilizado o comando CREATE GROUP, como mostrado na **Listagem 5**.

Listagem 5. Comando para criar um grupo no PostgreSQL

```
CREATE GROUP nome_do_grupo;
```

Podemos adicionar ou remover usuários em um grupo existente utilizando o comando ALTER GROUP da **Listagem 6**, respectivamente.

Listagem 6. Comandos para adicionar e remover usuários em um grupo

```
ALTER GROUP nome_do_grupo ADD USER nome_do_usuário;
  ALTER GROUP nome_do_grupo DROP USER nome_do_usuário;
```

Uma vez criado um grupo, ainda podemos removê-lo. Tal operação pode ser feita usando o comando DROP GROUP, conforme **Listagem 7**.

Listagem 7. Comando para remover um grupo no PostgreSQL

```
DROP GROUP nome_do_grupo;
```

O comando DROP GROUP remove os grupos, mas não remove os usuários membros do grupo.

Uma coisa importante que deve ser comentada é que não existe o comando CREATE GROUP no padrão SQL. O conceito de "papéis" (roles) que veremos adiante é semelhante ao de grupos e será o foco principal deste artigo.

Alterando Grupos no PostgreSQL – ALTER GROUP

Assim como podemos modificar um usuário criado no banco de dados, podemos, da mesma forma, modificar um grupo. Para fazermos isso, utilizamos o comando ALTER

GROUP. A **Listagem 8** adiciona, remove e altera usuários em um grupo, respectivamente.

Listagem 8. Comandos para alterar um grupo no PostgreSQL

```
ALTER GROUP nome_do_grupo ADD USER nome_do_usuário;
  ALTER GROUP nome_do_grupo DROP USER nome_do_usuário;
  ALTER GROUP nome_do_grupo RENAME TO novo_nome;
```

Criando Papéis (roles) no PostgreSQL – CREATE ROLE

O comando CREATE ROLE adiciona um novo papel (*role*) ao agrupamento de bancos de dados do PostgreSQL. O papel é uma entidade que pode possuir objetos do banco de dados e possuir privilégios do banco de dados. Ele pode ser considerado como sendo um "usuário", um "grupo", ou ambos, dependendo de como é utilizado.

O comando CREATE ROLE é o substituto dos comandos CREATE USER e CREATE GROUP por possuir mais recursos que os mesmos. A **Listagem 9** demonstra o comando de criação de um ROLE, enquanto a **Tabela 3** explica as opções que podem ser utilizadas neste comando.

Listagem 10. Comando para criar um role no PostgreSQL

```
CREATE ROLE nome [ [WITH] opção [...]];
```

Opção	**Significado**
Nome	Nome do papel
SUPERUSER | NOSUPERUSER	Estas cláusulas determinam se o novo papel é um "super-usuário", o qual pode passar por cima de todas as restrições de acesso dos bancos de dados. Se nenhuma destas duas cláusulas for especificada, o padrão é NOSUPERUSER.
CREATEDB | NOCREATEDB	Estas cláusulas definem a permissão para o papel criar bancos de dados. Se nenhuma destas duas cláusulas for especificada, o padrão é NOCREATEDB.
CREATEROLE | NOCREATEROLE	Estas cláusulas determinam se o papel terá permissão para criar novos papéis (ou seja, executar o comando CREATE ROLE).
CREATEUSER | NOCREATEUSER	Estas cláusulas são formas obsoletas, mas ainda aceitas, das cláusulas SUPERUSER e NOSUPERUSER.
INHERIT | NOINHERIT	Estas cláusulas determinam se o papel "herda" os privilégios dos papéis dos quais é membro. Um papel com o atributo INHERIT pode utilizar, automaticamente, todos

	os privilégios de banco de dados que foram concedidos a todos os papéis dos quais é um membro direto ou indireto.
LOGIN NOLOGIN	Estas cláusulas determinam se o papel pode estabelecer uma conexão (log in), ou seja, se o papel pode ser fornecido como nome de autorização inicial da sessão durante a conexão do cliente. Um papel com o atributo LOGIN pode ser considerado como sendo um usuário. Se nenhuma destas duas cláusulas for especificada, o padrão é NOLOGIN, exceto quando CREATE ROLE for chamada através de sua forma alternativa CREATE USER.
CONNECTION LIMIT limite_de_conexões	Se o papel puder estabelecer uma conexão, esta cláusula especifica quantas conexões simultâneas este papel pode estabelecer. -1 (o padrão) significa sem limite.
PASSWORD senha	Define a senha do papel (A senha só é útil para os papéis que possuem o atributo LOGIN, mesmo assim pode ser definida uma senha para os papéis sem este atributo).
ENCRYPTED UNENCRYPTED	Estas cláusulas controlam se a senha será armazenada criptografada, ou não, nos catálogos do sistema;
VALID UNTIL 'carimbo_do_tempo'	A cláusula VALID UNTIL define uma data e hora após a qual o papel não é mais válido. Se esta cláusula for omitida, a senha será válida para sempre.
IN ROLE nome_do_papel	A cláusula IN ROLE relaciona um ou mais papéis existentes, aos quais o novo papel será adicionado imediatamente como sendo um novo membro.
IN GROUP nome_do_papel	A cláusula IN GROUP é uma forma obsoleta de IN ROLE.
ROLE nome_do_papel	A cláusula ROLE relaciona um ou mais papéis existentes a serem automaticamente adicionados como membros do novo papel (Isto tem como efeito tornar o novo papel um "grupo").
ADMIN nome_do_papel	A cláusula ADMIN é como a cláusula ROLE, mas os papéis especificados são adicionados ao novo papel WITH ADMIN OPTION, dando aos mesmos o direito de permitir que outros papéis se tornem membros deste grupo.
USER nome_do_papel	A cláusula USER é uma forma obsoleta da cláusula ROLE.
SYSID id_usuário	A cláusula SYSID é ignorada, mas é aceita para manter a compatibilidade com as versões anteriores.

Tabela 3. Opções do comando CREATE ROLE.

Deve ser tomado cuidado com o privilégio CREATEROLE. Não existe o conceito de herança para os privilégios do papel com CREATEROLE. Isto significa que mesmo que o papel não possua um determinado privilégio, mas tenha permissão para criar outros papéis, poderá facilmente criar um papel com privilégios diferentes do próprio papel (exceto criar papéis com privilégio de super-usuário).

Removendo um Role – DROP ROLE

O comando DROP ROLE remove os papéis especificados. A **Listagem 10** demonstra como o comando DROP ROLE funciona.

Listagem 10. Comando para remover um role no PostgreSQL

```
DROP ROLE [ IF EXISTS ] nome [, ...]
```

O papel não poderá ser removido se ainda estiver sendo referenciado em qualquer banco de dados do agrupamento. Assim será lançado um erro caso tente-se removê-lo. Antes de remover o papel, é necessário remover todos os objetos pertencentes ao mesmo (ou mudar o dono), e revogar todos os privilégios concedidos pelo papel.

Alterando um Role – ALTER ROLE

O comando ALTER ROLE altera os atributos de um papel do PostgreSQL.

A**Listagem 11**demonstra o comando genérico de alteração de um role.

Listagem 11. Comando para alterar um role no PostgreSQL

```
ALTER ROLE nome [ [ WITH ] opção [ ... ] ]
```

O comando ALTER ROLE pode modificar um role, utilizando as mesmas opções mostradas na **Tabela 3.**

Concedendo e Revogando Privilégios no PostgreSQL

Quando um objeto do banco de dados é criado, é atribuído um dono ao mesmo. O dono é o usuário que executou o comando de criação do objeto. Para mudar o dono de uma tabela, índice, seqüência ou visão deve ser utilizado o comando ALTER TABLE. A **Listagem 12** mostra a modificação de um dono de uma tabela de um banco de dados.

Listagem 12. Comando para alterar um dono de uma tabela no PostgreSQL

```
ALTER TABLE nome_da_tabela OWNER TO novo_dono
```

Por padrão, somente o dono (ou um super-usuário) pode fazer qualquer coisa com o objeto. Para permitir o uso por outros usuários, devem ser concedidos privilégios aos mesmos. Em SQL, existe o comando GRANT. Ele possui duas funcionalidades básicas: conceder privilégios para um objeto do banco de dados (tabela, visão, seqüência, banco de dados, função, linguagem procedural, esquema e espaço de tabelas) e conceder o privilégio de ser membro de um papel.

Existem vários privilégios distintos: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE e ALL PRIVILEGES. A **Tabela 4** exibe os privilégios possíveis e os seus significados.

Privilégio	**Significado**
SELECT	Permite consultar qualquer coluna da tabela, visão ou seqüência especificada. Também permite utilizar o comando COPY TO.
INSERT	Permite inserir novas linhas na tabela especificada. Também permite utilizar o comando COPY FROM.
UPDATE	Permite modificar os dados de qualquer coluna da tabela especificada.
DELETE	Permite excluir linhas da tabela especificada.
RULE	Permite criar regras para uma tabela ou para uma visão.
REFERENCES	Para criar uma restrição de chave estrangeira é necessário possuir este privilégio, tanto na tabela que faz referência quanto na tabela que é referenciada.
TRIGGER	Permite criar gatilhos na tabela especificada
CREATE	Para bancos de dados, permite a criação de novos esquemas no banco de dados. Para esquemas, permite a criação de novos objetos no esquema. Para mudar o nome de um objeto existente é necessário ser o dono do objeto*e*possuir este privilégio no esquema que o contém. Para espaços de tabelas (TABLESPACE), permite a criação de tabelas e índices no espaço de tabelas, e permite a criação de bancos de dados possuindo este espaço de tabelas como seu espaço de tabelas padrão (Deve ser observado que revogar este privilégio não altera a colocação dos objetos existentes).
TEMPORARY	Permite a criação de tabelas temporárias ao usar o banco de dados.

TEMP	
EXECUTE	Permite utilizar a função especificada e qualquer operador implementado utilizando a função. Este é o único tipo de privilégio aplicável às funções (Esta sintaxe funciona para as funções de agregação também).
USAGE	Para as linguagens procedurais, permite o uso da linguagem especificada para criar funções nesta linguagem. Este é o único tipo de privilégio aplicável às linguagens procedurais. Para os esquemas, permite acessar os objetos contidos no esquema especificado (assumindo que os privilégios requeridos para os próprios objetos estejam atendidos). Essencialmente, concede a quem recebe o direito de "procurar" por objetos dentro do esquema.
ALL PRIVILEGES	Concede todos os privilégios disponíveis de uma só vez. A palavra chave PRIVILEGES é opcional no PostgreSQL, embora seja requerida pelo SQL estrito.

Tabela 4. Todos os possíveis privilégios de acesso

Em alguns casos também se torna necessário revogar alguns privilégios de acesso a usuários ou grupos de usuários. Para que isso seja feito, utiliza-se o comando REVOKE.

Deve ser observado que os super-usuários do banco de dados podem acessar todos os objetos, independentemente dos privilégios definidos para o objeto. Assim, não é aconselhável operar como um super-usuário a não ser quando for absolutamente necessário.

Se um super-usuário decidir submeter o comando GRANT ou REVOKE, o comando será executado como se tivesse sido submetido pelo dono do objeto afetado. Em particular, os privilégios concedidos através destes comandos aparecerão como se tivessem sido concedidos pelo dono do objeto.

Atualmente, o PostgreSQL não suporta conceder ou revogar privilégios para as colunas da tabela individualmente. Uma forma simples de contornar isso seria criar uma visão contendo apenas as colunas desejadas e, então, conceder os privilégios para a visão. Os privilégios especiais do dono da tabela (ou seja, o direito de DROP (remover), GRANT (conceder), REVOKE (revogar), etc.) são sempre implícitos ao fato de ser o dono, não podendo ser concedidos ou revogados.

Estudo de Caso – Hotel

Para demonstrar o controle de permissões de acesso aos objetos de um banco de dados no PostgreSQL, foi criado um banco de dados de um hotel fictício. Este banco de dados possui as tabelas*cliente, reserva, hospedagem, quarto, tipo_quarto, atendimento e serviço.*

No exemplo, foi considerado que cada cliente pode realizar várias reservas e várias hospedagens. Além de, a cada hospedagem, solicitar vários serviços.

Ainda é interessante notar que os quartos podem ser de tipos diferentes (apartamento simples, suíte casal, suíte luxo etc.).

A **Figura 1** apresenta o diagrama Entidade-Relacionamento do banco de dados em estudo.

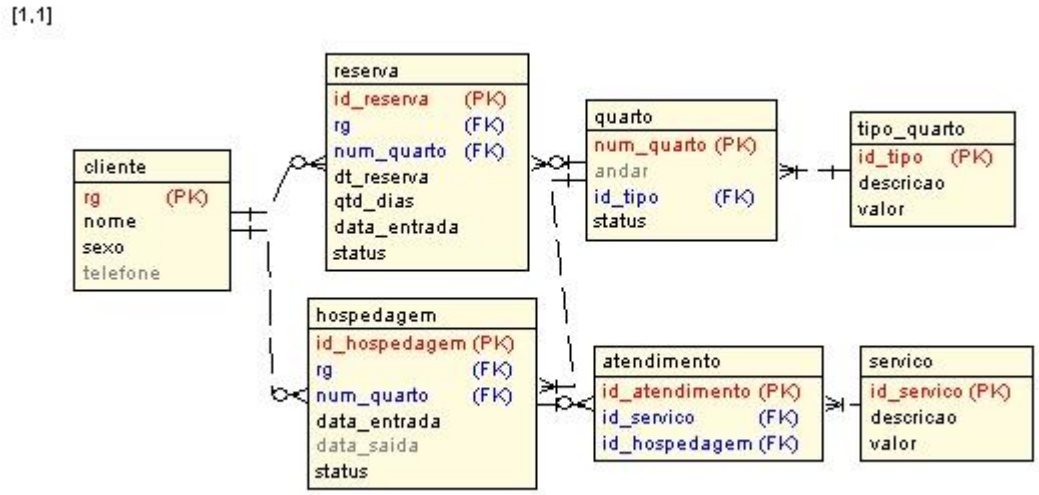

Figura 1. Diagrama Entidade-Relacionamento de um hotel.

Neste banco de dados, além das tabelas foram criadas três funções e uma visão. A primeira função é chamada de*adicionaReserva*e serve para realizar a reserva de um determinado quarto para um cliente. Uma outra função, chamada*adicionaHospedagem,*cria uma hospedagem para o cliente em um quarto. A última função, denominada*realizaPedido,*será responsável por registrar os pedidos feitos pelos clientes, enquanto estiverem hospedados.

Além das funções mencionadas, existe a visão*listaClientes*, que exibe apenas os nomes e os sexos dos clientes, descartando o RG e o telefone dos mesmos. A visão*listaClientes*foi criada para demonstrar a permissão de acessar apenas algumas colunas de uma determinada tabela.

Também foram criados alguns roles e foram dadas permissões diferentes de acesso aos dados para os mesmos. A **Tabela 5** mostra os três roles (neste caso, servem para definir o perfil de alguns grupos de usuários) que têm acesso ao banco e que tipo de permissão cada um deles possui.

Role	Permissão
Gerente	É o grupo de usuário que pode modificar todos os registros de todas as tabelas e utilizar as funções*adicionaReserva,adicionaHospedagem e realizaPedidos,*além de poder dar direitos de acesso aos demais usuários.
Atendente	É o grupo de usuários que pode manipular apenas as funções*adicionaReserva, adicionaHospedagem*e*realizaPedido.*
Estagiário	É o grupo de usuário que só tem acesso a visão*listaClientes.*

Tabela 5. Roles do banco de dados hotel

Implementação do Banco de Dados Hotel no PostgreSQL

O PostgreSQL será utilizado para criar o banco de dados, as funções, a visão e os roles citados na seção anterior. A ferramenta gráfica PgAdmin III foi utilizada para a criação do banco de dados do estudo em caso.

A **Listagem 13** exibe o comando SQL para criação do banco de dados chamado*hotel.*

Listagem 13. Comando SQL para criação do banco*hotel*

```
CREATE DATABASE hotel;
```

As tabelas do banco de dados hotel são criadas pelos comandos da **Listagem 14**.

Listagem 14. Comandos SQL para criação das tabelas do banco*hotel*

```
// Tabela de CLIENTES
  CREATE TABLE cliente
  (
  rg NUMERIC NOT NULL,
       nome VARCHAR(40) NOT NULL,
  sexo CHAR(1) NOT NULL,
  telefone NUMERIC(10,0),
  PRIMARY KEY (rg)
  ) WITHOUT OIDS;

  // Tabela TIPO_QUARTO
  CREATE TABLE tipo_quarto
  (
  id_tipo SERIAL NOT NULL,
  descricao VARCHAR(40) NOT NULL,
```

```
valor NUMERIC(9,2) NOT NULL,
PRIMARY KEY (id_tipo)
) WITHOUT OIDS;

// Tabela QUARTO
CREATE TABLE quarto
(
num_quarto INTEGER NOT NULL,
andar CHAR(10),
id_tipo INTEGER NOT NULL,
status CHAR(01) NOT NULL DEFAULT 'D',
PRIMARY KEY (num_quarto),
FOREIGN KEY (id_tipo) REFERENCES tipo_quarto (id_tipo)
ON UPDATE RESTRICT ON DELETE RESTRICT
) WITHOUT OIDS;

// Tabela SERVIÇO
CREATE TABLE servico
(
id_servico SERIAL NOT NULL,
descricao VARCHAR(60) NOT NULL,
valor NUMERIC(9,2) NOT NULL,
PRIMARY KEY (id_servico)
) WITHOUT OIDS;

// Tabela RESERVA
CREATE TABLE reserva
(
id_reserva SERIAL NOT NULL,
rg NUMERIC NOT NULL,
num_quarto INTEGER NOT NULL,
dt_reserva DATE NOT NULL,
qtd_dias INTEGER NOT NULL,
data_entrada DATE NOT NULL,
     status CHAR(1) NOT NULL DEFAULT 'A',
 PRIMARY KEY (id_reserva),
 FOREIGN KEY (rg) REFERENCES cliente (rg)
 ON UPDATE RESTRICT ON DELETE RESTRICT,
 FOREIGN KEY (num_quarto) REFERENCES quarto (num_quarto)
 ON UPDATE RESTRICT ON DELETE RESTRICT
) WITHOUT OIDS;
```

```
// Tabela HOSPEDAGEM
CREATE TABLE hospedagem
(
     id_hospedagem SERIAL NOT NULL,
     rg NUMERIC NOT NULL,
     num_quarto INTEGER NOT NULL,
     data_entrada DATE NOT NULL,
     data_saida DATE,
     status CHAR(1) NOT NULL,
 PRIMARY KEY (id_hospedagem),
 FOREIGN KEY (rg) REFERENCES cliente (rg)
 ON UPDATE RESTRICT ON DELETE RESTRICT,
 FOREIGN KEY (num_quarto) REFERENCES quarto (num_quarto)
 ON UPDATE RESTRICT ON DELETE RESTRICT
) WITHOUT OIDS;

// Tabela ATENDIMENTO
CREATE TABLE atendimento
(
     id_atendimento SERIAL NOT NULL,
     id_servico INTEGER NOT NULL,
     id_hospedagem INTEGER NOT NULL,
 PRIMARY KEY (id_atendimento),
 FOREIGN KEY (id_servico) REFERENCES servico (id_servico)
 ON UPDATE RESTRICT ON DELETE RESTRICT,
 FOREIGN KEY (id_hospedagem) REFERENCES hospedagem (id_hospedagem)
 ON UPDATE RESTRICT ON DELETE RESTRICT
) WITHOUT OIDS;
```

As funções *adicionaHospedagem*, *adicionaReserva* e *realizaPedido* são criadas com os comandos das **Listagens 15**,**16** e **17**, respectivamente.

Listagem 15. Comando de criação da funçãoa*dicionaHospedagem*

```
CREATE OR REPLACE FUNCTION adicionaHospedagem(rg_cliente numeric, numero_quarto int)
RETURNS void AS
  $
    begin
      perform * from cliente where rg = rg_cliente;
      if found then
        perform * from quarto where upper(status) = 'D' and num_quarto = numero_quarto;
        if found then
```

```
          insert into hospedagem values (default, rg_cliente, numero_quarto,
current_date, null, 'A');
          update quarto set status = 'O' where num_quarto = numero_quarto;
          RAISE NOTICE 'Hospedagem realizada com sucesso!';
        else
          RAISE EXCEPTION 'Quarto indisponivel para hospedagem!';
        end if;
      else
        RAISE EXCEPTION 'Cliente nao consta no cadastro!';
      end if;
    end;
  $
  LANGUAGE plpgsql SECURITY DEFINER;
```

Listagem 16. Comando de criação da função*adicionaReserva*

```
CREATE OR REPLACE FUNCTION adicionaReserva(rg_cliente numeric, numero_quarto int, dias
int, data_entrada date) RETURNS void AS
  $
    begin
      perform * from cliente where rg = rg_cliente;
      if found then
        perform * from quarto where upper(status) = 'D' and num_quarto = numero_quarto;
        if found then
          insert into reserva values (default, rg_cliente, numero_quarto, current_date,
dias, data_entrada, 'A');
          update quarto set status = 'R' where num_quarto = numero_quarto;
          RAISE NOTICE 'Reserva realizada com sucesso!';
        else
          RAISE EXCEPTION 'Quarto indisponivel para reserva!';
        end if;
      else
        RAISE EXCEPTION 'Cliente nao consta no cadastro!';
      end if;
    end;
  $
  LANGUAGE plpgsql SECURITY DEFINER;
```

Listagem 17. Comando de criação da função*realizaPedidos*

```
CREATE OR REPLACE FUNCTION realizaPedido(hosp int, serv int) RETURNS void AS
  $
    begin
```

```
      perform * from hospedagem where upper(status) = 'A' and id_hospedagem = hosp;
      if found then
        perform * from servico where id_servico = serv;
        if found then
          insert into atendimento values (default, serv, hosp);
          RAISE NOTICE 'Pedido realizado com sucesso!';
        else
          RAISE EXCEPTION 'Servico indisponivel!';
        end if;
      else
        RAISE EXCEPTION 'Hospedagem nao consta no cadastro ou ja foi desativada!';
      end if;
    end;
  $
  LANGUAGE plpgsql SECURITY DEFINER;
```

Não é interessante que alguns usuários acessem todas as informações sobre os clientes, como, por exemplo, o RG e o telefone. Por esse motivo, foi criada uma visão que permite apenas visualizar apenas o nome e o sexo dos clientes (veja **Listagem 18**).

Listagem 18. Criação da visão para consultar o nome e o sexo dos clientes

```
CREATE VIEW listaClientes (nome_cliente,sexo) AS
  SELECT nome, sexo FROM cliente
```

Tendo definido os três tipos de usuários que terão acesso aos objetos do banco de dados (veja **Tabela 5**), agora criaremos cada um (veja **Listagem 19**) e, mais adiante, daremos as permissões que ambos têm direito.

Listagem 19. Criação dos papéis (roles) gerente, atendente e estagiário

```
CREATE ROLE gerente;
  CREATE ROLE atendente;
  CREATE ROLE estagiario;
```

Depois de criarmos todos os papéis que irão agrupar todos os usuários com perfis semelhantes, temos que dar as permissões para cada tipo, de acordo com o que foi definido anteriormente.

Antes de dar as permissões sobre as funções é importante explicar a opção SECURITY DEFINER usada na criação das três funções. Esta opção serve para permitir que todos os usuários acessem essas funções com a mesma permissão de quem as criou. Isto é muito importante, pois os usuários dos grupos atendente e estagiário não têm acesso às tabelas do banco de dados hotel, então não poderiam executá-las. Por exemplo, os

grupos atendente e estagiário não executariam a função*adicionaHospedagem,*porque na sua definição existem consultas que buscam valores que estão em algumas tabelas como clientes e quartos e esses grupos de usuários não têm permissão para acessa-las.

Para resolver este problema, foram criadas todas as funções com a opção SECURITY DEFINER, e antes de dar a permissão de execução para os grupos gerente, atendente e estagiário, deve-se revogar o direito de qualquer usuário do banco executar tais funções.

Os comandos que revogam o execução funções*adicionaHospedagem, adicionaReserva e realizaPedido*pelos usuários, estão exibidos na **Listagem 20**.

Listagem 20. Revogando a execução das três funções para todos os usuários

```
REVOKE ALL ON FUNCTION adicionaReserva(numeric,int,int,date) FROM PUBLIC;
  REVOKE ALL ON FUNCTION adicionaHospedagem(numeric,int) FROM PUBLIC;
  REVOKE ALL ON FUNCTION realizaPedido(int,int) FROM PUBLIC;
```

Feito isso, podemos agora dar as permissões de acesso para cada um dos grupos de usuários. O papel gerente poderá modificar todos os registros de todas as tabelas (veja **Listagem 21**), além de acessar as funções *adicionaReserva* (veja **Listagem 22**)*, adicionaHospedagem* (veja **Listagem 23**) *e listaClientes* (veja **Listagem 24**) e a visão *listaClientes* **(**veja **Listagem 25)**.

Listagem 21. Concedendo permissão para o role gerente acessar todas as tabelas e conceder permissões para outros usuários

```
GRANT SELECT, INSERT ON cliente, reserva, hospedagem, quarto, tipo_quarto, atendimento,
servico, listaClientes TO gerente WITH GRANT OPTION;
```

Listagem 22. Concedendo permissão para o role gerente para acessar a função adicionaHospedagem

```
GRANT EXECUTE ON FUNCTION adicionaHospedagem(numeric,int) TO gerente;
```

Listagem 23. Concedendo permissão para o role gerente para acessar a função adicionaReserva

```
GRANT EXECUTE ON FUNCTION adicionaReserva(numeric,int,int,date) TO gerente;
```

Listagem 24. Concedendo permissão para o role gerente para acessar a função realizarPedido

```
GRANT EXECUTE ON FUNCTION realizaPedido(int,int) TO gerente;
```

Listagem 25. Concedendo permissão para o role gerente para acessar a view listaClientes

```
GRANT SELECT ON listaClientes TO gerente
```

O grupo de usuários atendente não pode ter acesso a nenhuma tabela. Ele pode apenas acessar as funções *adicionaHospedagem, adicionaReserva e realizaPedidos.*As permissões para acessar tais funções estão disponíveis nas **Listagens 26, 27 e 28**, respectivamente.

Listagem 26. Concedendo permissão para o role atendente para acessar a função adicionaHospedagem

```
GRANT EXECUTE ON FUNCTION adicionaHospedagem(numeric,int) TO atendente;
```

Listagem 27. Concedendo permissão para o role atendente para acessar a função adicionaReserva

```
GRANT EXECUTE ON FUNCTION adicionaReserva(numeric,int,int,date) TO atendente;
```

Listagem 28. Concedendo permissão para o role atendente para acessar a função realizaPedidos

```
GRANT EXECUTE ON FUNCTION
  realizaPedido(int,int) TO atendente;
```

A **Listagem 29** dá a permissão para o grupo de usuários do tipo estagiário acessar a visão*listaClientes.*

Listagem 29. Concedendo permissão para o role estagiário acessar a visão*listaCliente*

```
GRANT SELECT ON listaClientes TO estagiario;
```

Com o banco de dados, as funções, a visão e os usuários criados, já podemos realizar testes para verificar se o que foi permitido para cada tipo de usuário está de acordo com cada um dos seus perfis.

Simulando Testes para Validar as Permissões

Com todos os papéis criados, iremos realizar alguns testes para validar as permissões que foram concedidas para cada um dos papéis.

Em um primeiro momento, serão criados três usuários pertencentes a cada um dos papéis criados (gerente, atendente e estagiário). A **Listagem 30** exibe a criação do um usuário chamado*tony*e que pertencerá ao grupo de usuários gerente. O parâmetro

“LOGIN” permite que o usuário possa logar no sistema e o parâmetro “PASSWORD” atribui a senha ‘111’ para o mesmo.

Listagem 30. Criação de usuário com papel de gerente

```
CREATE ROLE tony LOGIN PASSWORD '111' IN ROLE gerente;
```

Por sua vez, a **Listagem 31** cria o usuário*maria*com direito de logar no sistema, mas agora pertencendo ao grupo de atendentes.

Listagem 31. Criação de usuário com papel de atendente

```
CREATE ROLE maria LOGIN PASSWORD '222' IN ROLE atendente;
```

O último usuário criado é a*vitoria.*Ela pertencerá ao grupo de estagiários e sua criação está mostrada na **Listagem 32**.

Listagem 32. Criação de usuário com papel de atendente

```
CREATE ROLE vitoria LOGIN PASSWORD '333' IN ROLE estagiario;
```

Com o usuário*tony*logado, pode-se realizar todas as operações com as tabelas, exceto operações próprias dos donos dos objetos, como por exemplo, remover uma tabela (DROP TABLE).

Quando este usuário tenta executar o comando de inclusão de um novo cliente, conforme a **Figura 2**, o sistema executa o comando normalmente.

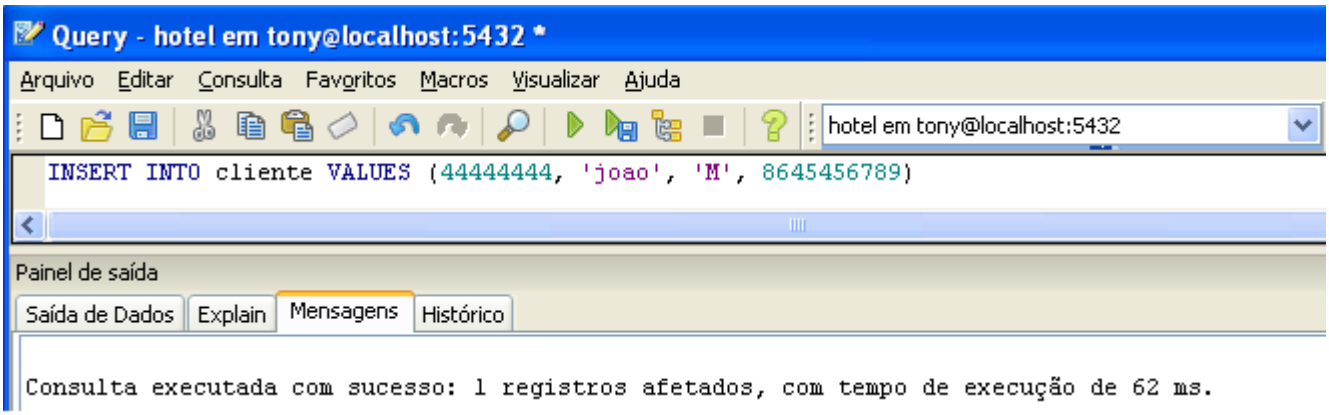

Figura 2. Usuário tony incluindo um novo cliente.

Se usuário também tentar realizar qualquer operação (SELECT, DELETE, UPDATE, INSERT) com qualquer uma das tabelas, o sistema executará o comando normalmente. Porém, o mesmo não ocorrerá, caso ele tente remover uma tabela do banco de dados (veja **Figura 3**).

Figura 3. Usuário tony tentando apagar do banco de dados a tabela tipo_quarto

Agora com o usuário maria logado, tenta-se utilizar a função **adicionaReserva,** conforme **Figura 4**.

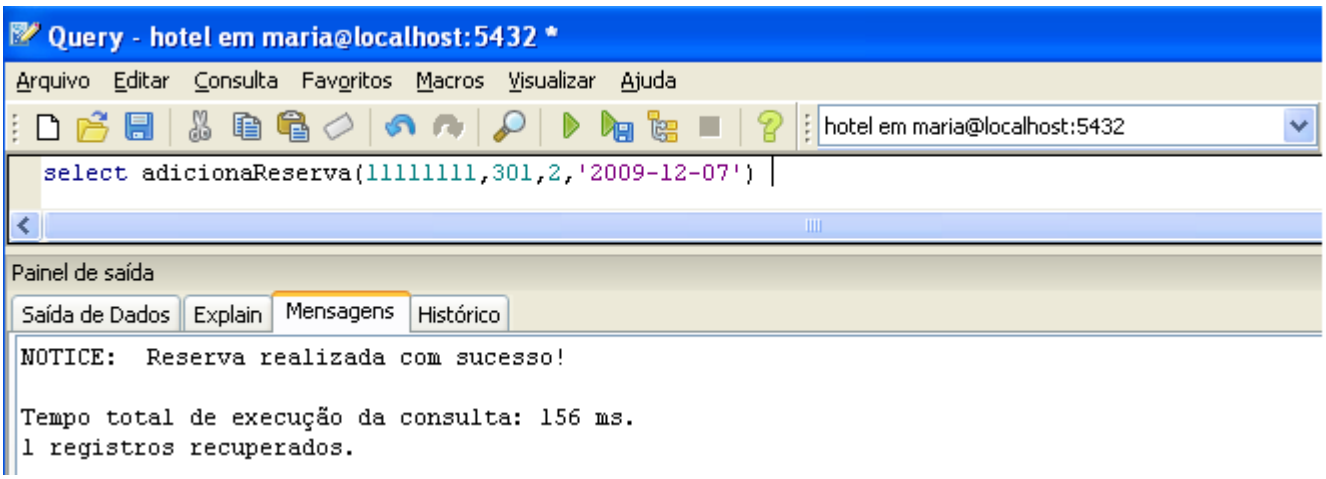

Figura 4. Usuário maria acessando a função*adicionaReserva*

O mesmo não aconteceria se esse usuário tentasse acessar diretamente a tabela reserva, porque o seu perfil não tem permissão para isto (veja **Figura 5**).

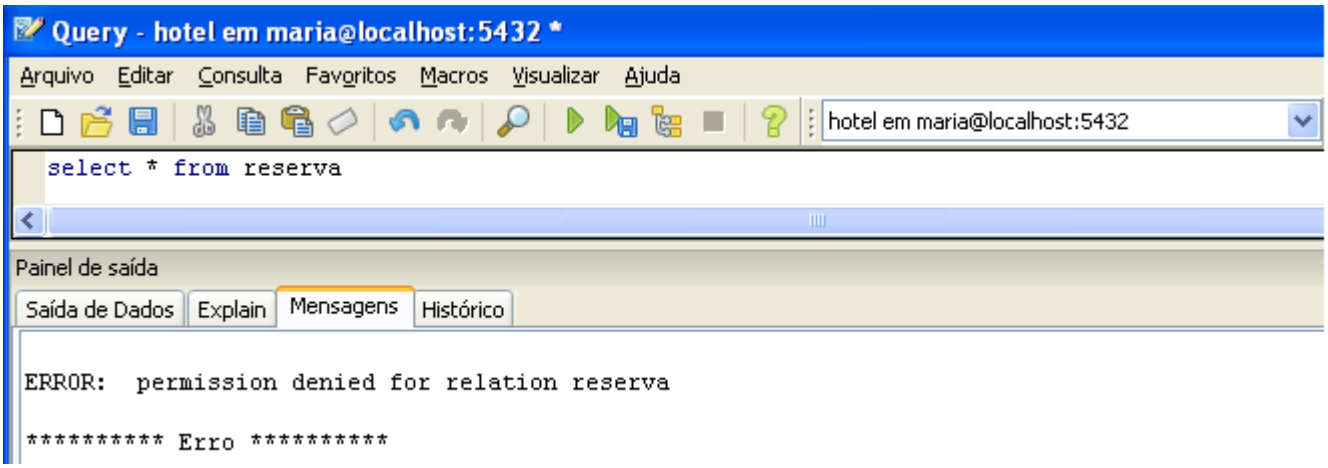

Figura 5. Usuário maria tentando selecionar os registros da tabela reserva

Por último, com o usuário vitoria, tenta-se consultar a visão*listaClientes*(veja **Figura 6**).

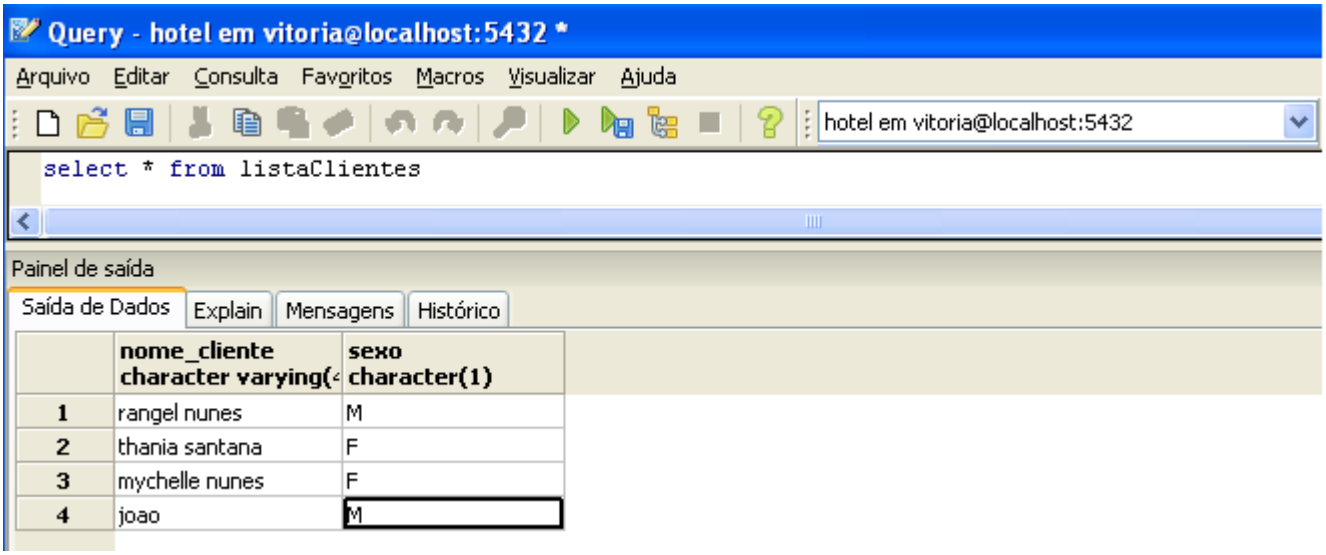

Figura 6. Usuário vitória acessando a visão*listaClientes*

Verifique que vitoria não teve problema porque o seu perfil tem permissão para consultar somente a visão*listaClientes*. O mesmo não ocorreria caso ele tentasse consultar diretamente a tabela cliente (veja **Figura 7**).

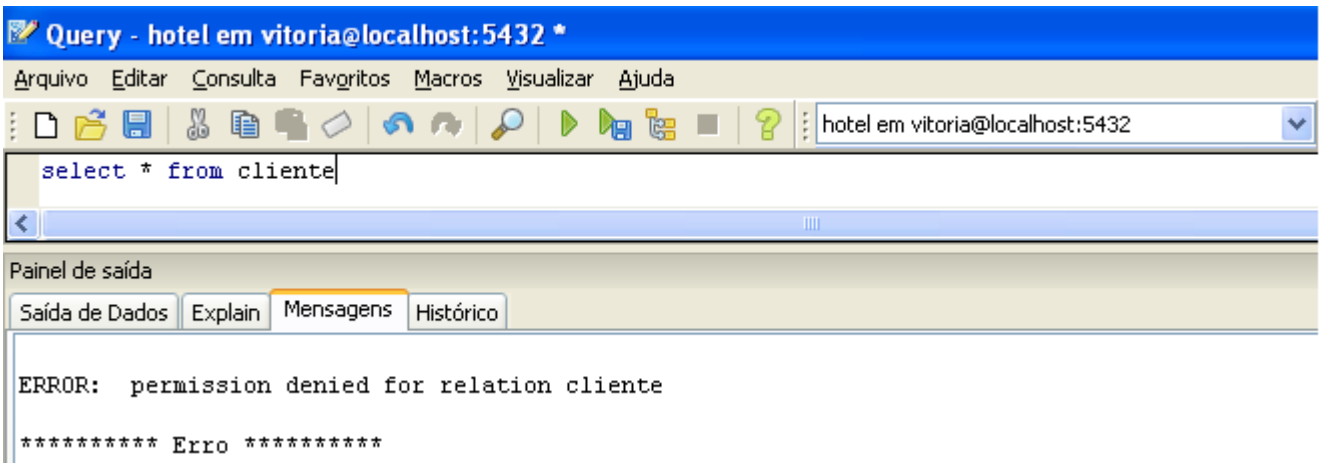

Figura 7. Usuário vitória tentando consultar diretamente a tabela cliente.

Conclusão

Segurança é importante em todos os níveis, principalmente a nível de informação. Hoje o maior patrimônio de uma empresa são seus dados e por isso é tão importante preserva-los íntegros.

Com o que foi visto neste artigo, percebe que podemos controlar de forma bastante eficaz os nossos dados, dando permissão de acesso àqueles que possuem determinadas características e dar acesso diferenciado para outros.

O PostgreSQL é um SGBD bastante utilizado e como foi visto, é uma excelente opção por vários motivos, além de permitir meios eficazes de controle de acesso aos dados.