

Você está em

DevMedia

Artigo

Gerenciando Usuários e Permissões no PostgreSQL

O artigo aborda os conceitos relacionais com gerenciamento de usuários e permissões de acesso em SGBD's, focando o PostgreSQL. Neste artigo, foi implementado um estudo de caso para demonstrar na prática as funções do PostgreSQL no que se refere ao gerenciamento de usuários.



Anotar



Marcado como lido

Artigos



Banco de Dados



Gerenciando Usuários e Permissões no PostgreSQL

Atenção: esse artigo tem uma palestra complementar. [Clique](#) e assista!

De que se trata o artigo:

O artigo aborda os conceitos relacionais com gerenciamento de usuários e permissões de acesso em SGBD's, focando o PostgreSQL. Neste artigo, foi implementado um estudo de caso para demonstrar na prática as funções do PostgreSQL no que se refere ao gerenciamento de usuários.



12



Em que situação o tema é útil:

Em toda aplicação com conexão a banco de dados é importante o controle de acesso diferenciado para cada papel de usuário no sistema. Tendo em vista isto, este artigo torna-se uma boa opção de fonte de pesquisa.

A segurança dos sistemas de informação engloba um número elevado de áreas que poderão estar sob a responsabilidade de uma ou várias pessoas. Entre estas áreas encontram-se a segurança de redes, a segurança física, a segurança de computadores, a segurança das aplicações, a segurança da informação etc.

O responsável pela implementação da segurança dos sistemas de informação em uma organização possui como primeira missão, e mais importante, a garantia da segurança da informação que protege. Esta garantia é conseguida mediante a utilização de vários instrumentos.

Uma política de backup e recuperação de dados adequadamente elaborada e executada protegerá a organização contra a perda de informação devido a falhas de hardware, defeitos de software, erros humanos, intrusos, sabotagem e desastres naturais. Entretanto, esta não é a única maneira existente de proteção das informações. Afinal, precisamos mais do que uma solução relativa ao acontecimento de falhas.

Quando se trata de proteger informações guardadas em banco de dados, nos preocupamos em que dados proteger e como protegê-los. Os SGBD's atuais nos fornecem poderosas ferramentas que auxiliam em tais tarefas. Sendo os bancos de dados sistemas de armazenamento de informação, e sendo esta um elemento de elevado valor, quer seja financeiro, quer seja estratégico, é necessário que haja algum tipo de controle de acesso a essa informação

Você está em

DevMedia

subconjunto da SQL para o controle de permissões dos usuários aos objetos do banco de dados.

Neste artigo, será demonstrado o funcionamento do controle de permissões de acesso aos objetos do banco de dados 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



Você está em

DevMedia

```
1 | 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_usuario	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.



Você está em

DevMedia

	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

```
1 | DROP USER nome_usuario
```

Neste caso, *nome_usuario* 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.

Listagem 3. Comando que altera um usuário no PostgreSQL

```
1 | 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

Você está em

DevMedia

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

```
1 | 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

```
1 | CREATE GROUP nome do grupo
```



Você está em

DevMedia

```
1 | ALTER GROUP nome_do_grupo ADD USER nome_do_usuario;  
2 | ALTER GROUP nome_do_grupo DROP USER nome_do_usuario;
```

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

```
1 | 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

```
1 | ALTER GROUP nome_do_grupo ADD USER nome_do_usuario;  
2 | ALTER GROUP nome_do_grupo DROP USER nome_do_usuario;  
3 | ALTER GROUP nome_do_grupo RENAME TO novo_nome;
```

Criando Papéis (roles) no PostgreSQL – CREATE



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

```
1 | 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	Estas cláusulas são formas obsoletas, mas ainda



Você está em

DevMedia

	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



Você está em

DevMedia

	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_usuario	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

Listagem 10. Comando para remover um role no PostgreSQL

```
1 | 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

```
1 | 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, sequência 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

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.

	<p>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.</p> <p>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).</p>
TEMPORARY TEMP	Permite a criação de tabelas temporárias ao usar o banco de dados.
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	<p>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.</p> <p>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.</p>

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

Você está em

DevMedia

[1.1]

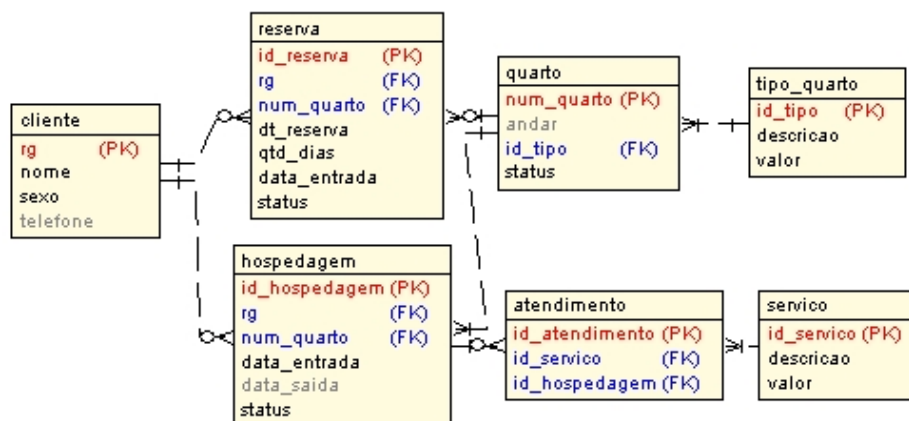


Figura 1.Diagrama Entidade-Relacionamento de um hotel.

Neste banco de dados, além das tabelas foram criadas três funções e uma visão. A primeira função é chamada *deadicionaReservae* 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.

Você está em

DevMedia

Atendente	É o grupo de usuários que pode manipular apenas as funções <i>adicionaReserva</i> , <i>adicionaHospedagem</i> e <i>realizaPedido</i> .
Estagiário	É o grupo de usuário que só tem acesso a visão <i>listaClientes</i> .

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*

```
1 | 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*

```
1 | // Tabela de CLIENTES
2 | CREATE TABLE cliente
3 | (
4 |   rg NUMERIC NOT NULL,
5 |   nome VARCHAR(40) NOT NULL,
```





Você está em

DevMedia

```
13  (
14  id_tipo SERIAL NOT NULL,
15  descricao VARCHAR(40) NOT NULL,
16  valor NUMERIC(9,2) NOT NULL,
17  PRIMARY KEY (id_tipo)
18  ) WITHOUT OIDS;
19
20  // Tabela QUARTO
21  CREATE TABLE quarto
22  (
23  num_quarto INTEGER NOT NULL,
24  andar CHAR(10),
25  id_tipo INTEGER NOT NULL,
26  status CHAR(01) NOT NULL DEFAULT 'D',
27  PRIMARY KEY (num_quarto),
28  FOREIGN KEY (id_tipo) REFERENCES tipo_quarto (id_tipo)
29  ON UPDATE RESTRICT ON DELETE RESTRICT
30  ) WITHOUT OIDS;
31
32  // Tabela SERVIÇO
33  CREATE TABLE servico
34  (
35  id_servico SERIAL NOT NULL,
36  descricao VARCHAR(60) NOT NULL,
37  valor NUMERIC(9,2) NOT NULL,
38  PRIMARY KEY (id_servico)
39  ) WITHOUT OIDS;
40
41  // Tabela RESERVA
42  CREATE TABLE reserva
43  (
44  id_reserva SERIAL NOT NULL,
45  rg NUMERIC NOT NULL,
46  num_quarto INTEGER NOT NULL,
47  dt_reserva DATE NOT NULL,
48  qtd_dias INTEGER NOT NULL,
49  data_entrada DATE NOT NULL,
50  status CHAR(1) NOT NULL DEFAULT 'A',
51  PRIMARY KEY (id_reserva),
52  FOREIGN KEY (rg) REFERENCES cliente (rg)
53  ON UPDATE RESTRICT ON DELETE RESTRICT,
```



12



Você está em

DevMedia

```
62         rg NUMERIC NOT NULL,
63         num_quarto INTEGER NOT NULL,
64         data_entrada DATE NOT NULL,
65         data_saida DATE,
66         status CHAR(1) NOT NULL,
67     PRIMARY KEY (id_hospedagem),
68     FOREIGN KEY (rg) REFERENCES cliente (rg)
69     ON UPDATE RESTRICT ON DELETE RESTRICT,
70     FOREIGN KEY (num_quarto) REFERENCES quarto (num_quarto)
71     ON UPDATE RESTRICT ON DELETE RESTRICT
72 ) WITHOUT OIDS;
73
74 // Tabela ATENDIMENTO
75 CREATE TABLE atendimento
76 (
77     id_atendimento SERIAL NOT NULL,
78     id_servico INTEGER NOT NULL,
79     id_hospedagem INTEGER NOT NULL,
80     PRIMARY KEY (id_atendimento),
81     FOREIGN KEY (id_servico) REFERENCES servico (id_servico)
82     ON UPDATE RESTRICT ON DELETE RESTRICT,
83     FOREIGN KEY (id_hospedagem) REFERENCES hospedagem (id_hospedagem)
84     ON UPDATE RESTRICT ON DELETE RESTRICT
85 ) 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 função *adicionaHospedagem*

```
1 CREATE OR REPLACE FUNCTION adicionaHospedagem(rg_cliente numeric, numero
2     $
3     begin
4         perform * from cliente where rg = rg_cliente;
5         if found then
6             perform * from quarto where upper(status) = 'D' and num_quarto =
7             if found then
8                 insert into hospedagem values (default, rg_cliente, numero_qua
9                 update quarto set status = 'O' where num_quarto = numero_quart
```



12



Você está em

DevMedia

```
18 | $
19 | LANGUAGE plpgsql SECURITY DEFINER;
```

Listagem 16. Comando de criação da função *adicionaReserva*

```
1 | CREATE OR REPLACE FUNCTION adicionaReserva(rg_cliente numeric, numero_qu
2 |   $
3 |   begin
4 |       perform * from cliente where rg = rg_cliente;
5 |       if found then
6 |           perform * from quarto where upper(status) = 'D' and num_quarto =
7 |           if found then
8 |               insert into reserva values (default, rg_cliente, numero_quarto
9 |               update quarto set status = 'R' where num_quarto = numero_quart
10 |               RAISE NOTICE 'Reserva realizada com sucesso!';
11 |           else
12 |               RAISE EXCEPTION 'Quarto indisponivel para reserva!';
13 |           end if;
14 |       else
15 |           RAISE EXCEPTION 'Cliente nao consta no cadastro!';
16 |       end if;
17 |   end;
18 |   $
19 | LANGUAGE plpgsql SECURITY DEFINER;
```

Listagem 17. Comando de criação da função *realizaPedidos*

```
1 | CREATE OR REPLACE FUNCTION realizaPedido(hosp int, serv int) RETURNS voi
2 |   $
3 |   begin
4 |       perform * from hospedagem where upper(status) = 'A' and id_hospeda
5 |       if found then
6 |           perform * from servico where id_servico = serv;
7 |           if found then
8 |               insert into atendimento values (default, serv, hosp);
9 |               RAISE NOTICE 'Pedido realizado com sucesso!';
10 |           else
```



12



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

```
1 | CREATE VIEW listaClientes (nome_cliente, sexo) AS
2 |     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

```
1 | CREATE ROLE gerente;
2 |     CREATE ROLE atendente;
3 |     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

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

```
1 | REVOKE ALL ON FUNCTION adicionaReserva(numeric,int,int,date) FROM PUBLIC
2 | REVOKE ALL ON FUNCTION adicionaHospedagem(numeric,int) FROM PUBLIC;
3 | 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

```
1 | GRANT SELECT, INSERT ON cliente, reserva, hospedagem, quarto, tipo_quart
```

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

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

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

```
1 | GRANT EXECUTE ON FUNCTION adicionaReserva(numeric,int,int,date) TO geren
```

Você está em

DevMedia

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

```
1 | 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`

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

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

```
1 | GRANT EXECUTE ON FUNCTION adicionaReserva(numeric,int,int,date) TO atend
```

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

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

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

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 de um usuário chamado *tonye* 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

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

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

Listagem 31. Criação de usuário com papel de atendente

```
1 | 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

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


Você está em

DevMedia

conforme a **Figura 2**, o sistema executa o comando normalmente.

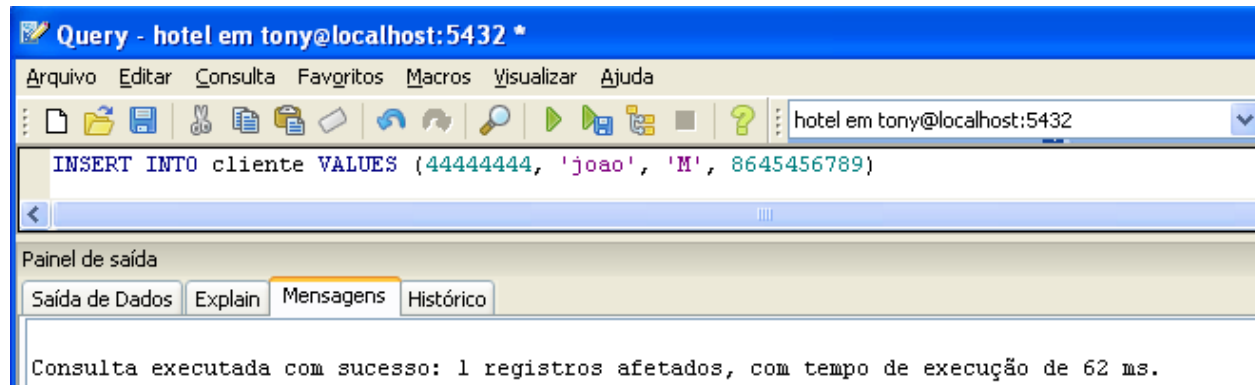


Figura 2. Usuário tony incluindo um novo cliente.

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**).

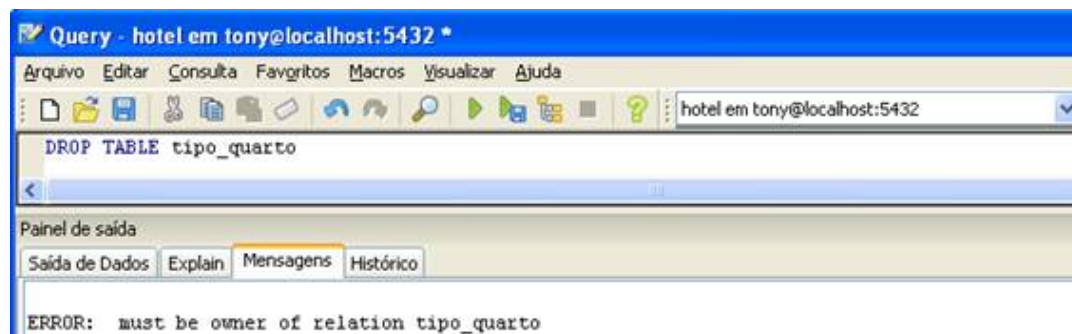


Figura 3. Usuário tony tentando apagar do banco de dados a tabela tipo_quarto

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



Você está em

DevMedia

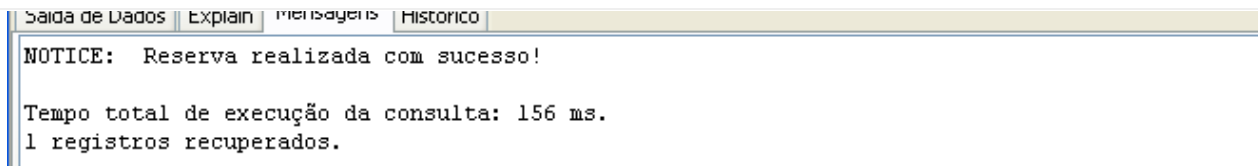


Figura 4. Usuário maria acessando a função *adicionaReserva*

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**).

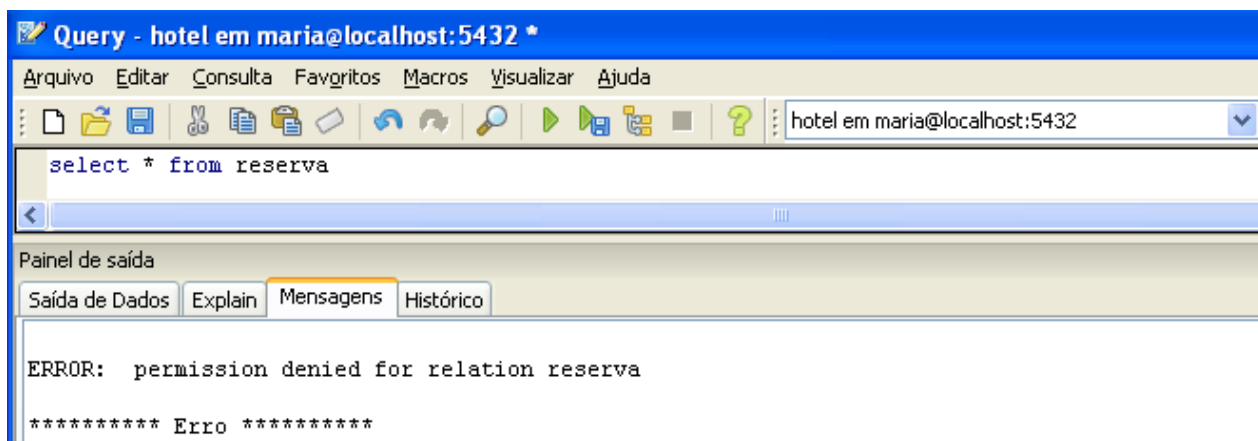


Figura 5. Usuário maria tentando selecionar os registros da tabela reserva

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

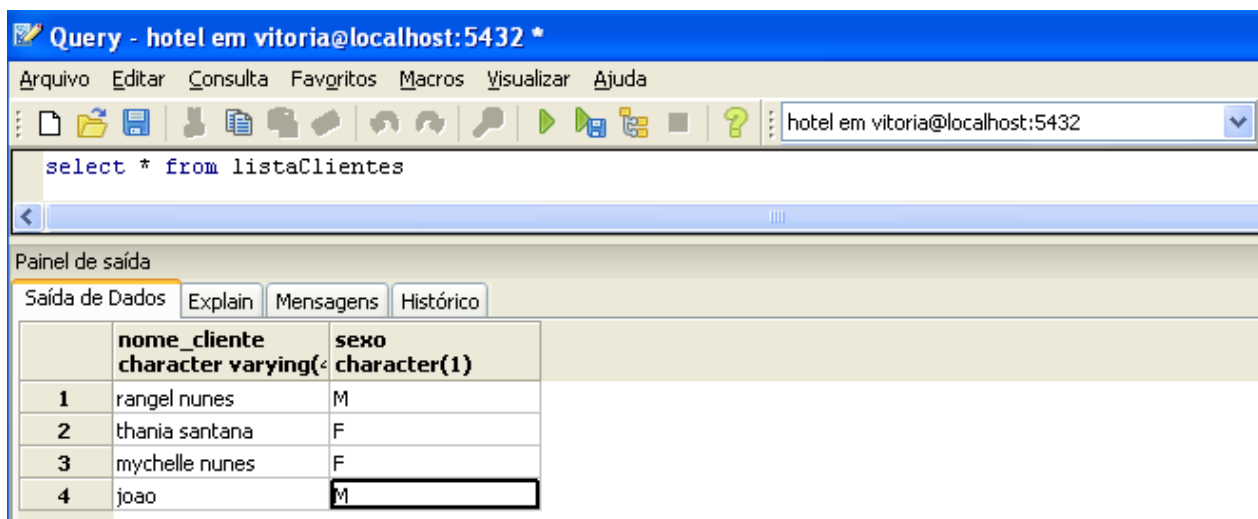


Figura 6. Usuário vitoria acessando a visão *listaClientes*



Você está em

DevMedia

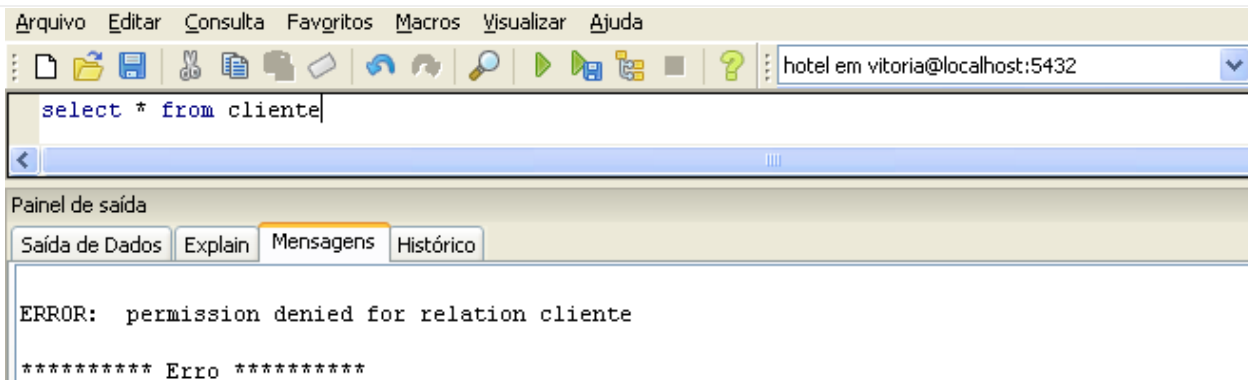


Figura 7. Usuário vitória tentando consultar diretamente a tabela cliente.

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 preservá-los íntegros.

Com o que foi visto neste artigo, percebe-se 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.

Referências

GONZAGA, J. L. Dominando o PostgreSQL. Rio de Janeiro: Editora Ciência Moderna Ltda., 2007. p. 187-204.

ELMASRI, Ramez E.; NAVATHE, Shamkant. Sistemas de Banco de Dados. 4 ed. São Paulo: Pearson Addison Wesley, 2005. p. 544-552.

DAMAS, Luis. SQL – Structured Query Language. 6 ed. São Paulo: LTC, 2007. 396p.

Você está em

DevMedia



Anotar



Marcado como lido



Por **Willamys**

Em 2009

Suporte ao aluno

Minhas dúvidas

Planos de estudo

Fale conosco

Plano para Instituição de ensino

Assinatura para empresas

Assine agora



Hospedagem web por Porta 80 Web Hosting



12

