



Criação e manipulação de objetos no PostgreSQL

Você vai compreender a instalação do PostgreSQL. Isso proporciona familiaridade com um ambiente computacional típico de banco de dados corporativo. O conhecimento sobre tipos de dados ajuda a escolher opções compatíveis com as informações armazenadas. Para desenvolver sistemas com banco de dados, é preciso dominar os comandos de manipulação de linhas nas tabelas e entender seu controle por meio do conceito de transação.

Prof. Sidney Venturi | Prof.^a Nathielly de Souza Campos

Preparação

Antes de iniciar o estudo deste conteúdo, certifique-se de ter baixado o SGBD PostgreSQL em seu computador.

Objetivos

- Compreender o processo de instalação do PostgreSQL.
- Empregar comandos para a criação e a alteração de tabelas.
- Empregar comandos para a manipulação de linhas nas tabelas.
- Empregar comandos de controle de transação.

Introdução

Ao longo deste conteúdo, vamos analisar as características básicas do sistema gerenciador de banco de dados (SGBD) PostgreSQL, envolvendo as etapas utilizadas para instalar esse SGBD no Linux e no Windows.

O PostgreSQL é um SGBD de código aberto desenvolvido em linguagem C e está disponível para ser utilizado em diversos ambientes de sistemas operacionais, como Linux, Unix, Windows, OS X e Solaris, entre outros.

Vamos explorar vários recursos da linguagem SQL, com foco na aprendizagem de comandos classificados como DDL (Data Definition Language ou Linguagem de Definição de Dados). Aprenderemos também comandos CRUD (Create - Read - Update - Delete), sigla em inglês que faz referência a quatro operações básicas: criação, consulta, atualização e remoção de dados, respectivamente.

Por fim, vamos entender que, internamente, o SGBD trata de diversas operações de maneira atômica, ou seja, um conjunto de comandos deve ser executado como uma unidade lógica de trabalho, ou nenhuma operação deve ser realizada. Trata-se do conceito de transação. Aprenderemos, então, os principais comandos que gerenciam transações.

[Clique aqui](#) para baixar o arquivo com todos os códigos que serão utilizados nas consultas para este estudo.

No vídeo, a seguir, confira a criação e a manipulação de objetos no SGBD PostgreSQL, as características do sistema e os comandos de criação e manipulação de tabelas. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Histórico e arquitetura básica

O PostgreSQL, lançado inicialmente em 1989 como Postgres, é um sistema de gerenciamento de banco de dados relacional de código aberto conhecido por sua robustez e conformidade com os padrões SQL.

Desenvolvido na Universidade da Califórnia, em Berkeley, evoluiu significativamente ao longo das décadas. Sua arquitetura modular e extensível permite uma alta personalização e desempenho, tornando-o uma escolha popular para aplicações empresariais e acadêmicas que demandam confiabilidade e escalabilidade.

No vídeo, a seguir, você vai compreender a arquitetura do sistema gerenciador de banco de dados postgresQL. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Breve histórico

O PostgreSQL surgiu a partir do projeto POSTGRES, assim denominado por ser originário do projeto INGRES (Post INGRES), de responsabilidade da Universidade da Califórnia, em Berkeley.

Acompanhe o histórico desse sistema a seguir.

1986

Implementação

A implementação do POSTGRES foi iniciada, tornando-o operacional em 1987.

1989

Lançamento ao público externo

A primeira versão foi lançada ao público externo. Nos dois anos seguintes, foram lançadas a segunda e terceira versões.

1995

Postgres95

Foi disponibilizado o Postgres95, com revisão no código do projeto e adoção da linguagem SQL (Structured Query Language ou Linguagem de Consulta Estruturada) como interface padrão.

1996

PostgreSQL

O produto foi renomeado para PostgreSQL, começando pela versão 6, considerado continuidade do Postgres95, a versão 5.

Atualmente

PostgreSQL como um dos principais SGBDs de código aberto

O projeto ganhou visibilidade e, atualmente, o PostgreSQL é conhecido como um dos principais SGBDs de código aberto, com versões para Windows, Mac OS e Linux.

Arquitetura do PostgreSQL

Antes de instalarmos o PostgreSQL, é importante entendermos sua arquitetura básica. Ele utiliza o modelo cliente-servidor. Sob esse contexto, confira os seguintes processos que cooperam entre si.

Processo servidor, responsável por funções, como:

- Gerenciar os arquivos do banco de dados.
- Gerenciar as conexões entre os aplicativos e o SGBD.
- Avaliar e executar, no banco de dados, os comandos submetidos pelos clientes.

Aplicativo cliente do usuário, responsável por funções, como:

- Solicitar acesso ao SGBD.
- Enviar comandos para a manipulação de linhas em tabelas. A manipulação pode envolver, por exemplo, inserção, alteração ou mesmo remoção de dados.
- Enviar comandos para consulta em uma ou diversas tabelas, com objetivo de recuperar informações do banco de dados.

Em um ambiente cliente-servidor, tanto o cliente quanto o servidor podem estar localizados em máquinas diferentes, em uma rede local ou mesmo geograficamente distantes.

Em geral, a comunicação entre cliente e servidor ocorre por meio de uma conexão de rede utilizando o protocolo TCP/IP. O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede. Seu nome faz referência a dois protocolos: Transmission Control Protocol / Internet Protocol. Esse protocolo tem por objetivo a padronização das comunicações em rede, em especial as comunicações na Web.

O PostgreSQL suporta várias conexões simultâneas de clientes. Para isso, um processo para cada conexão é iniciado. Em seguida, o cliente e o novo processo realizam comunicação.

Adiante, veremos como instalar o PostgreSQL em seu computador, mas já recomendamos que você acompanhe as versões do PostgreSQL na página oficial do produto para escolher uma versão que lhe interesse, de acordo com seu sistema operacional.

Atividade 1

O PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto amplamente utilizado em diversos cenários, desde pequenas aplicações até grandes ambientes corporativos. Sua arquitetura é projetada para oferecer confiabilidade, flexibilidade e extensibilidade, permitindo aos usuários adaptar e escalar suas implementações conforme necessário.

Qual das alternativas melhor descreve a arquitetura do PostgreSQL?

- A O PostgreSQL é um sistema de banco de dados baseado em um único processo monolítico que gerencia todas as conexões e operações de banco de dados.
- B O PostgreSQL utiliza uma arquitetura cliente-servidor, em que o servidor de banco de dados gerencia as conexões e operações, e os clientes conectam-se ao servidor para executar consultas SQL.
- C O PostgreSQL é um banco de dados distribuído que requer a configuração de múltiplos servidores para operar corretamente.
- D O PostgreSQL não suporta extensões ou módulos adicionais, operando exclusivamente com as funcionalidades básicas incluídas na instalação padrão.
- E O PostgreSQL utiliza uma arquitetura peer-to-peer, em que todos os nós têm as mesmas responsabilidades e não há distinção entre cliente e servidor.



A alternativa B está correta.

A arquitetura do PostgreSQL é baseada no modelo cliente-servidor. Nesse modelo, o servidor (postmaster) gerencia as conexões de clientes, realiza operações no banco de dados e mantém a integridade dos dados. Os clientes conectam-se ao servidor para executar comandos SQL.

Além disso, o PostgreSQL suporta extensões e módulos adicionais, permitindo uma alta personalização e expansão de suas funcionalidades.

Instalação do PostgreSQL no Linux

Agora que já estudamos a arquitetura básica do PostgreSQL, vamos fazer a instalação do SGBD em um sistema Linux, utilizando os pacotes.

Para isso, você deve seguir os seguintes pré-requisitos:

- Acesso root ou permissões sudo na máquina Linux.

- Conexão com a internet.

No vídeo, a seguir, você poderá acompanhar e praticar a instalação do PostgreSQL no Linux. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para instalar, configurar e testar a instalação do PostgreSQL no sistema Linux, siga os passos apresentados a seguir.

Instalar e configurar

1. Atualize os pacotes do sistema.
2. Instale os pacotes necessários.
3. Adicione o repositório do PostgreSQL.
4. Adicione o repositório à lista de fontes.
5. Atualize novamente os pacotes do sistema.
6. Instale o PostgreSQL.
7. Verifique a instalação.
8. Se o serviço não estiver ativo, inicie-o.
9. Configure para iniciar automaticamente com o sistema.
10. Configure o PostgreSQL.
11. No prompt do PostgreSQL, crie uma senha para o usuário postgres.
12. Saia do prompt do PostgreSQL.

13. Saia do usuário postgres.

Atividade 2

O PostgreSQL está entre os sistemas de gerenciamento de banco de dados relacionais mais utilizados e eficientes, atendendo desde pequenas startups até grandes corporações. Em sistemas Linux, sua instalação pode ocorrer por diferentes métodos, cada um com características e impactos próprios. Conhecer essas variações é essencial para administradores de sistemas e desenvolvedores que buscam uma instalação ajustada às suas necessidades.

Explique a diferença entre instalar o PostgreSQL no Linux a partir dos pacotes e a partir do código-fonte. Em sua resposta, mencione as vantagens e as desvantagens de cada método.

Chave de resposta

A instalação a partir de pacotes oferece vantagens, como facilidade e rapidez, pois os pacotes são pré-compilados e configurados para distribuição específica. Além disso, esse método gerencia automaticamente as dependências com gerenciadores como **apt** e **yum**, realiza atualizações automáticas de segurança e funcionalidades e melhora a integração com os scripts de inicialização e os componentes do sistema operacional.

Em contrapartida, as desvantagens incluem a limitação à versão disponível nos repositórios, que pode não ser a mais recente, restringindo o acesso a novas funcionalidades. Há também menor controle sobre a configuração e as opções de compilação, definidas pelo mantenedor do pacote.

A instalação, a partir do código-fonte, oferece vantagens como controle total sobre o processo de compilação e configuração, permitindo ajustes finos no desempenho e a escolha de opções específicas, além de permitir o acesso às versões mais recentes do PostgreSQL, incluindo correções e funcionalidades que ainda não estão disponíveis nos repositórios de pacotes da distribuição.

Quanto às desvantagens da instalação, a partir do código-fonte, incluem maior complexidade e tempo no processo de compilação e instalação, exigindo conhecimentos avançados, necessidade de gerenciamento manual das dependências e bibliotecas, atualizações manuais de segurança e funcionalidade. Isso aumenta a carga de trabalho do administrador e dificulta a integração com os scripts de inicialização e outros componentes do sistema, exigindo ajustes adicionais.

Instalação do PostgreSQL no Windows

É um processo simples que pode ser realizado em poucos passos. Vamos lá!

No vídeo, a seguir, você poderá acompanhar e praticar a instalação do postgresQL no Windows. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Siga os passos apresentados a seguir para instalar e configurar o PostgreSQL no seu sistema Windows.

Fazer o download do instalador

1. Acesse o site oficial do PostgreSQL.

- Vá para PostgreSQL Downloads.

2. Baixe o instalador.

- Clique no link para baixar o instalador do PostgreSQL apropriado para a sua versão do Windows. Você será redirecionado para a página do EDB (EnterpriseDB). Escolha a versão desejada e clique em Download.

Executar o instalador

1. Inicie o instalador: localize o arquivo baixado (postgresql-xx.x.x-windows-x64.exe) e execute-o.

Assistente de instalação:

- Siga as instruções do assistente de instalação.
- Clique em Next na tela de boas-vindas.

Configurar a instalação

1. Escolha do diretório de instalação.

- Selecione o diretório onde o PostgreSQL será instalado (o padrão geralmente é C:\Program Files\PostgreSQL\xx).
- Clique em Next.

2. Seleção de componentes.

- Certifique-se de que as opções desejadas estejam selecionadas (normalmente, todas as opções padrão são adequadas).
- Clique em Next.

3. Escolha do diretório de dados.

- Selecione o diretório onde os dados do PostgreSQL serão armazenados (o padrão geralmente é C:\Program Files\PostgreSQL\xx\data).

- Clique em Next.

4. Configuração do usuário e senha.

- Defina uma senha para o usuário postgres (administrador do banco de dados).
- Memorize essa senha, pois será necessária para acessar o PostgreSQL.
- Clique em Next.

5. Definição da porta de conexão.

- Especifique a porta de conexão (o padrão é 5432).
- Clique em Next.

6. Configuração da localização.

- Selecione a localização (locale) para o banco de dados.
- Clique em Next.

7. Pronto para instalar.

- Revise as configurações e clique em Next para iniciar a instalação.

Concluir a instalação

1. Finalizar a instalação: após a instalação ser concluída, clique em Finish.

Verificar a instalação

1. Acessar o pgAdmin.

- O pgAdmin, uma ferramenta gráfica para gerenciar o PostgreSQL, é instalado junto com o PostgreSQL. Abra o pgAdmin (disponível no menu Iniciar).

2. Conectar ao Servidor PostgreSQL.

- No pgAdmin, crie uma nova conexão ao servidor PostgreSQL usando o usuário postgres e a senha definida durante a instalação.

Testar a conexão

- Verifique se a conexão foi bem-sucedida e explore a interface do pgAdmin para garantir que o PostgreSQL está funcionando corretamente.

Atividade 3

Você foi designado para configurar um ambiente de desenvolvimento em um sistema Windows e precisa instalar e configurar o PostgreSQL como parte desse processo.

Além da instalação do PostgreSQL, a equipe de desenvolvimento está explorando a utilização do Application Stack Builder (ASB) no PostgreSQL.

Explique o propósito e a funcionalidade ASB nesse contexto.

Chave de resposta

O Application Stack Builder (ASB) é uma ferramenta gratuita de código aberto que auxilia a instalação e o gerenciamento de extensões para o PostgreSQL. Ele oferece uma interface gráfica amigável e simplifica o processo de:

1. Instalar extensões: o ASB fornece uma lista completa de extensões disponíveis para o PostgreSQL, incluindo suas funcionalidades e requisitos, permitindo a instalação com um único clique.
2. Gerenciar extensões: o ASB permite visualizar informações detalhadas sobre as extensões instaladas.
3. Configurar extensões: o ASB fornece uma interface para configurar as opções de configuração das extensões instaladas.

Por exemplo, imagine instalar a extensão PostGIS para adicionar funcionalidades geoespaciais ao PostgreSQL. Com o ASB, você pode:

- Abrir o ASB e pesquisar pela extensão PostGIS.

- Selecionar a versão desejada da extensão e clicar em Instalar. O ASB irá baixar e instalar a extensão automaticamente, resolvendo as dependências necessárias.
- Após a instalação, você poderá configurar as opções da extensão PostGIS através da interface do ASB.

pgAdmin 4 e PSQL

São duas ferramentas essenciais para a administração e a interação com o PostgreSQL. Enquanto o pgAdmin oferece uma interface gráfica amigável para a gestão e a visualização dos dados, o PSQL proporciona uma potente interface de linha de comando para a execução de comandos SQL e scripts.

Ambas as ferramentas complementam-se, atendendo às necessidades tanto de administradores de bancos de dados quanto de desenvolvedores, além de ser essenciais para o trabalho eficiente com PostgreSQL.

Você vai conhecer, no vídeo a seguir, as duas principais interfaces para interagir com o PostgreSQL: PGADMIN 4 e PSQL. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Interfaces para interagir com o PostgreSQL

A partir daqui, é importante que você saiba que será necessário o uso de algum tipo de interface que permita conexão ao servidor e, em seguida, o acesso aos objetos de interesse.

Além do pgAdmin 4, a interface gráfica própria que o PostgreSQL disponibiliza o PSQL, uma interface de linha de comando sobre a qual o usuário submete interativamente comandos ao SGBD, via terminal. Além disso, o PostgreSQL possui uma excelente documentação disponível on-line, aplicável tanto para a instalação em Linux quanto para Windows, considerada uma verdadeira enciclopédia de bancos de dados relacionais.

A documentação é válida para o uso dos recursos do PostgreSQL com quaisquer interfaces. Alternativamente, você pode optar por baixar e usar interfaces projetadas por outros desenvolvedores.

```

44 GO
45 SELECT p.Name AS ProductName,
46 NonDiscountSales = (OrderQty * p.UnitPrice) AS NonDiscountSales,
47 Discounts = ((OrderQty * p.UnitPrice) - (OrderQty * p.UnitPrice * d.Discount)) AS Discounts
48 FROM Production.Product p
49 INNER JOIN Sales.SalesOrderHeader soh
50 ON p.ProductID = soh.ProductID
51 ORDER BY ProductName, NonDiscountSales, Discounts
52 GO

```



Exemplo

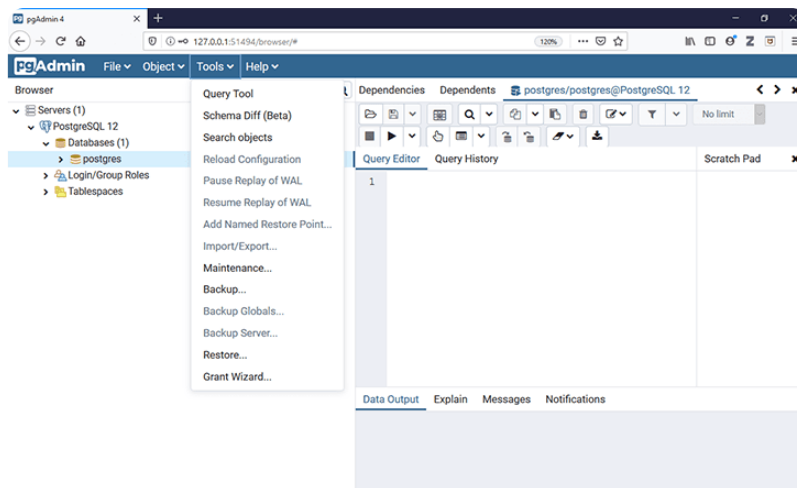
O aplicativo DBeaver possui uma versão livre com excelentes funcionalidades. Trata-se de um aplicativo útil no desenvolvimento de atividades de administração de banco de dados.

Criando database com o pgAdmin 4 e o PSQL

Tendo instalado o PostgreSQL, para nos certificarmos de que o SGBD está funcionando de maneira adequada, realizaremos um teste envolvendo a criação de databases, conforme mostrado a seguir.

- Database BDTESTEPGADMIN, a ser criado usando o pgAdmin 4.
- Database BDTESTEPSQL, a ser criado usando o PSQL.

No Windows, selecione o botão Iniciar, digite “pgAdmin 4” e tecla <enter>. Em seguida, o navegador será aberto, e você terá acesso a um ambiente no qual aparece um único database denominado postgres, criado pelo instalador. Veja!



Tela do pgAdmin 4 com o database postgres criado pelo instalador.

Com o foco no database postgres, clique em Tools e em Query Tool para abrir um editor (Query Editor) no qual você codificará e submeterá (utilizando a tecla F5 ou o botão correspondente com uma seta) comandos SQL ao servidor.

Vamos criar o database BDTESTEPGADMIN a partir do ambiente do pgAdmin 4. Digite o código a seguir no Query Editor e, após, pressione a tecla F5 para executar o comando a seguir.

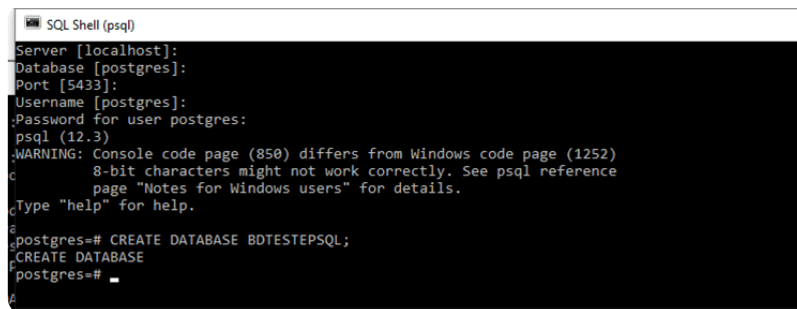
```
sql  
  
CREATE DATABASE BDTESTEPGADMIN;
```

Alternativamente, o database poderia ser criado usando a interface do pgAdmin 4, pressionado o botão direito do mouse sobre a pasta Databases e clicando em Create.

Agora, vamos criar um database usando a interface de terminal PSQL. No Windows, a partir do botão Iniciar, digite "psql" e tecele <enter>. Logo um terminal será aberto e você executará o comando a seguir (teclando <enter> após o comando).

```
sql
CREATE DATABASE BDTESTEPSQL;
```

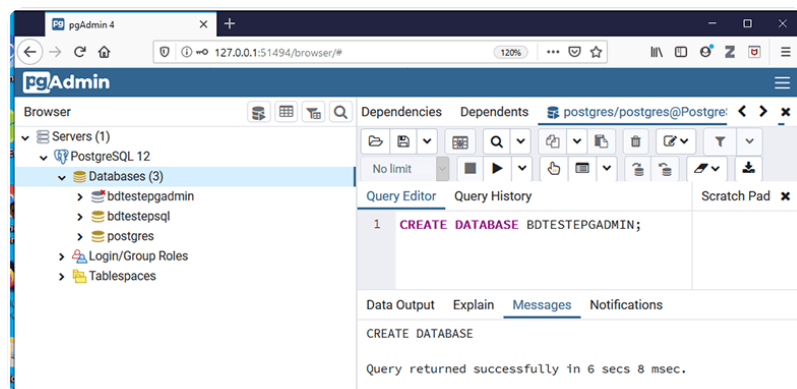
Após o SGBD executar o comando, a tela do PSQL ficará conforme vemos a seguir.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5433]:
Username [postgres]:
Password for user postgres:
psql (12.3)
WARNING: Console code page (850) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.
postgres=# CREATE DATABASE BDTESTEPSQL;
CREATE DATABASE
postgres=#
```

Tela do PSQL após a criação do database BDTESTEPSQL.

Ao final do processo, podemos verificar, no pgAdmin 4, a criação dos dois databases, conforme mostrado na imagem a seguir.



Tela do pgAdmin 4 mostrando os databases recém-criados.

Detalhes sobre a utilização do pgAdmin 4 e do PSQL podem ser pesquisados nas respectivas documentações que acompanham os softwares.

Atividade 4

No ecossistema do PostgreSQL, pgAdmin e PSQL são duas ferramentas que oferecem diferentes abordagens para a administração e a interação com o banco de dados. Entender as características e as funcionalidades de cada uma é muito importante para escolher a ferramenta mais adequada às necessidades específicas de administradores e desenvolvedores.

Qual das seguintes afirmações melhor descreve a diferença entre o pgAdmin e o PSQL no contexto do PostgreSQL?

A O pgAdmin é uma ferramenta de linha de comando, enquanto o PSQL é uma interface gráfica.

B O pgAdmin é utilizado exclusivamente para backup de dados, enquanto o PSQL é utilizado para consultas SQL.

C O pgAdmin oferece uma interface gráfica para gerenciamento de bancos de dados PostgreSQL, enquanto o PSQL é uma interface de linha de comando para execução de comandos SQL.

D O pgAdmin é uma ferramenta proprietária paga, enquanto o PSQL é de código aberto e gratuito.

E O pgAdmin só pode ser utilizado em sistemas operacionais Windows, enquanto o PSQL é compatível apenas com Linux.



A alternativa C está correta.

O pgAdmin é uma ferramenta gráfica de administração para PostgreSQL, fornecendo uma interface visual intuitiva que facilita a gestão de bancos de dados, tabelas, usuários e outras funções administrativas. Ele é adequado para usuários que preferem interações visuais e gestão simplificada dos componentes do banco de dados.

O PSQL é uma ferramenta de linha de comando que permite aos usuários executar comandos SQL diretamente, gerenciar scripts e interagir com o banco de dados em um ambiente de texto. É amplamente utilizado por desenvolvedores e administradores que necessitam de um controle mais granular e automatização de tarefas por meio de scripts.

Prática de criação de banco de dados com PSQL e pgAdmin

Aqui você aprenderá a criar e manipular um banco de dados no PostgreSQL usando tanto o terminal PSQL quanto a interface gráfica pgAdmin.

Praticar com essas ferramentas aumentará sua eficiência na administração e no desenvolvimento de bancos de dados PostgreSQL.

Confira, no vídeo a seguir, a criação de banco de dados com o PGADMIN e pratique o uso dessa ferramenta. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Agora vamos criar e manipular um banco de dados no PostgreSQL usando o terminal PSQL e a interface gráfica pgAdmin. Acompanhe o passo a passo a seguir.

Criando banco de dados utilizando o PSQL

1. Conecte-se ao PostgreSQL com o PSQL.
2. Crie um banco de dados chamado lab_db.
3. Confirme a criação do banco acessando o catálogo do sistema.
4. Conecte-se ao banco de dados recém-criado.
5. Crie uma tabela chamada clientes com os seguintes campos: id, nome, email e idade:

- Comando de criação.

```
sql

CREATE TABLE clientes (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    idade INT
);
```

6. Insira dados na tabela:

```
sql

INSERT INTO clientes (nome, email, idade) VALUES ('João Silva', 'joao.silva@example.com', 30);
INSERT INTO clientes (nome, email, idade) VALUES ('Maria Souza', 'maria.souza@example.com', 25);
```

7. Consulte os dados inseridos na tabela clientes:

```
sql

SELECT * FROM clientes;
```

8. Desconecte do banco de dados.
9. Elimine o banco de dados lab_db.
10. Confirme a eliminação.

Criando banco de dados utilizando o pgAdmin

1. Abra o pgAdmin e conecte-se ao servidor PostgreSQL.
2. Crie um banco de dados chamado lab_db_pgadmin.
3. Expanda o banco de dados lab_db_pgAdmin: clique com o botão direito em Schemas → public → Tables e selecione Create → Table.
4. Na aba General, dê o nome clientes.
5. Na aba Columns, adicione as colunas id e nome:

id: Tipo SERIAL, marque como Primary Key.
nome: Tipo VARCHAR(100).
6. Clique em Save.
7. Insira dados na tabela.
8. Para inserir dados na tabela, clique com o botão direito na tabela clientes e selecione View/Edit Data → All Rows.
9. Insira dois registros diretamente na interface.
10. Para consultar dados na tabela no painel de navegação à esquerda, clique com o botão direito na tabela clientes e selecione Query Tool.

11. Execute a consulta:

```
sql  
SELECT * FROM clientes;
```

12. Para eliminar o banco de dados no painel de navegação à esquerda, clique com o botão direito no banco de dados lab_db_pgAdmin e selecione Delete/Drop.
13. Confirme a exclusão.

Atividade 5

Realize as atividades apresentadas a seguir.

Atividade 1

- Utilizando a linha de comando, crie um database chamado atividade1 e confirme sua criação:

```
sql
Create Database atividade1;
```

- Para confirmar a criação, acesse o catálogo do sistema:

```
sql
Select oid, datname from pg_database.
```

Atividade 2

- Utilizando a linha de comando, elimine o database atividade1 e confirme sua eliminação.
- Para confirmar a eliminação, acesse o catálogo do sistema.

Atividade 3

- Utilizando o pgAdmin, crie o database atividade 3 com codificação win1252 e utilizando como modelo o template0.
- Confirme a criação do banco.

Atividade 4

- Utilizando o pgAdmin, elimine a base de dados atividade3 e confirme sua eliminação.
- Confirme a eliminação do banco.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Comandos SQL para criação de banco de dados e tabelas

A criação de bancos de dados e tabelas no PostgreSQL organiza e facilita a manipulação eficiente dos dados. Utilizando uma sintaxe SQL intuitiva, o PostgreSQL permite definir esquemas, criar tabelas e estabelecer relações entre elas com facilidade.

Suportando tipos de dados avançados e extensões, o PostgreSQL oferece excelentes ferramentas para garantir a integridade, a segurança e a eficiência no gerenciamento dos dados, sendo uma solução versátil para diversas aplicações.

No vídeo, a seguir, você vai entender os comandos SQL para a criação de um banco de dados e de tabelas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Breve histórico da SQL

A SQL foi criada na IBM, na década de 1970, sendo originalmente chamada de SEQUEL (Structured English Query Language), inspirada, principalmente, na aparente facilidade de uso do comando SELECT para consulta a tabelas dos bancos de dados relacionais.



Comentário

Com a evolução dos sistemas gerenciadores de banco de dados (SGBDs), diversas empresas lançaram produtos incorporando funcionalidades à SQL, o que ocasionou problemas de compatibilidade. Buscando uma solução, o ANSI (American National Standards Institute) definiu padrões para a linguagem SQL, a qual passou a ser referenciada ANSI-SQL. O ANSI é uma organização particular norte-americana sem fins lucrativos com o objetivo de facilitar a padronização dos trabalhos de seus membros.

Atualmente, há diversos SGBDs compatíveis com o padrão ANSI SQL, que vai muito além de consultas com o comando SELECT, englobando sublinguagens para a definição de dados (CREATE, ALTER, DROP) e para manipulação de dados (INSERT, UPDATE, DELETE), além de comandos de controle típicos para a administração do banco de dados.

Ao mesmo tempo, vários produtos de SGBDs relacionais apresentam extensões à linguagem, como modo de facilitar o dia a dia dos desenvolvedores.

Acesso ao PostgreSQL

Para criar tabelas em um SGBD PostgreSQL, você deverá seguir os seguintes passos.

Passo 1

Executando o pgAdmin 4, o navegador será aberto, e você terá acesso ao ambiente que permite manipular os objetos do PostgreSQL.

Passo 2

Depois, escolha um database na hierarquia e dê um clique com o botão inverso do mouse. Em seguida, escolha a opção Query Tool.

O pgAdmin é a interface Web padrão do PostgreSQL, mas você pode escolher o utilitário de sua preferência para praticar os comandos que aprenderemos ao longo deste conteúdo.

Criando um banco de dados

Após escolher um utilitário para acessar o servidor PostgreSQL, será necessário criar um database para, em seguida, manipular tabelas. Por exemplo, para criarmos um database denominado bdestudo, é necessário executar o comando a seguir.

```
sql
-- Comando para criar um database.
CREATE DATABASE BDESTUDO;
```

No comando, a linha com “--” corresponde a comentário e seu conteúdo não é processado pelo SGBD. Caso haja necessidade de remover o database bdestudo, basta executar o comando a seguir.

```
sql
-- Comando para remover um database.
DROP DATABASE BDESTUDO;
```

Antes de prosseguirmos com a criação de tabelas, é preciso compreender que todo database criado no PostgreSQL possui um schema padrão denominado public, no qual as tabelas a serem criadas no database serão armazenadas.

Assim, se não especificarmos a qual schema do database pertence uma tabela que estamos criando, esta será armazenada no schema public.

Para especificarmos um schema diferente do public, antes de criar uma tabela, devemos criar o respectivo schema, com o comando CREATE SCHEMA.



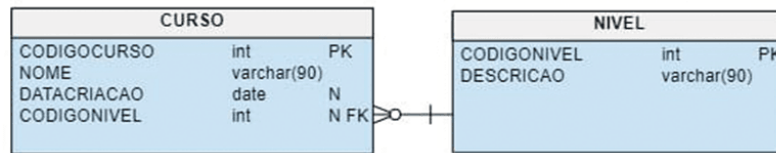
Exemplo

```
CREATE SCHEMA esquema;
```

Qualquer tabela pertencente ao schema esquema, a partir de então, deverá ser especificada pelo seu nome completo: esquema.tabela

Criando tabelas

Já sabemos que um banco de dados, em geral, possui diversas tabelas. As tabelas são criadas com o auxílio do comando CREATE TABLE. Usaremos esse comando para implementar o modelo expresso na imagem a seguir, dentro do database bdestudo anteriormente criado.



Modelo relacional com as tabelas CURSO e NIVEL.

Veja, a seguir, a sintaxe **básica** do comando CREATE TABLE:

```
sql

CREATE TABLE NOMETABELA (
  COLUNA1 - TIPODEDADOS [RESTRIÇÃO],
  COLUNAN - TIPODEDADOS [RESTRIÇÃO],
  PRIMARY KEY (COLUNA),
  FOREIGN KEY (COLUNA) REFERENCES NOMETABELA (COLUNA)
  CONSTRAINT RESTRIÇÃO);
```

Vamos agora analisar o significado de cada item na sintaxe apresentada anteriormente. Veja!

NOMETABELA

Representa o nome da tabela que será criada.

COLUNA1 e COLUNAN

Representam a(s) coluna(s) da tabela.

TIPODEDADOS

Indica tipos de dados ou domínio da coluna.

RESTRIÇÃO

Aponta alguma propriedade associada à coluna em questão. Por exemplo, podemos definir se a coluna é obrigatória ou opcional.

PRIMARY KEY

Indica a coluna, ou conjunto de colunas, representativa da chave primária.

FOREIGN KEY

Sinaliza a coluna, ou conjunto de colunas, com restrição de chave estrangeira.

CONSTRAINT RESTRIÇÃO

Indica alguma restrição que poderá ser declarada.

A sintaxe **completa** a respeito do comando CREATE TABLE no PostgreSQL pode ser encontrada no site oficial do PostgreSQL. Ao final da sintaxe, são descritas as características compatíveis com o padrão SQL.

Tipos de dados

Cada coluna de tabela deve pertencer a um tipo de dado. Veja, a seguir, os tipos mais comuns no PostgreSQL.

bigint

Valores inteiros compreendidos entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807.

char (comprimento)

Útil para sequências de caracteres de tamanho fixo. O parâmetro comprimento determina o valor da sequência. Esse tipo de dado preenche a coluna com espaços em branco até completar o total de caracteres definidos, caso a totalidade do tamanho do campo não esteja preenchida.

date

Data de calendário no formato AAAA-MM-DD.

decimal

Determina a precisão do valor de casas decimais.

double

Precisão do valor de até 15 casas decimais.

int ou integer

Valores inteiros compreendidos entre -2.147.483.648 e 2.147.483.647.

money

Valores monetários compreendidos entre -92.233.720.368.547.758.08 e 92.233.720.368.547.758.07.

numeric

Precisão do valor de casas decimais.

real

Precisão do valor de até seis casas decimais.

serial

Gera valor único inteiro sequencial para um novo registro entre 1 e 2.147.483.647.

smallint

Representa valores compreendidos entre 32.768 e 32.767.

time

Representa horário no intervalo de tempo entre 00:00:00 e 24:00:00.

varchar (comprimento)

Útil para sequência de dados de caracteres com comprimento variável. Não armazena espaços em branco não utilizados para compor string (em branco) em seu lado direito.

Para detalhes sobre os tipos de dados, sugerimos que você acesse a documentação do PostgreSQL, disponível no site oficial do produto.

Exemplo envolvendo criação de tabelas

Veja a seguir o código SQL que permite a criação das tabelas NIVEL e CURSO, respectivamente.

```
1 -- Comando para criar a tabela nivel
2 CREATE TABLE NIVEL (
3     CODIGONIVEL int NOT NULL,
4     DESCRICAO varchar(90) NOT NULL,
5     CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL)
6 );
7 -- Comando para criar a tabela curso
8 CREATE TABLE CURSO (
9     CODIGOCURSO int NOT NULL,
10    NOME varchar(90) NOT NULL UNIQUE,
11    DATACRIACAO date NULL,
12    CODIGONIVEL int NULL,
13    CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO),
14    FOREIGN KEY (CODIGONIVEL) REFERENCES NIVEL (CODIGONIVEL)
15 );
```

A tabela NIVEL está especificada no bloco de comandos entre as linhas 2 e 6. As duas colunas da tabela são obrigatórias.

A tabela CURSO está especificada no bloco de comandos entre as linhas 8 e 15. As colunas DATACRIACAO e CODIGONIVEL são opcionais. Em especial, CODIGONIVEL significa que um curso pode ser criado e, em outro momento, ser associado à informação que caracteriza o nível dele.



Comentário

Note que, como não foi especificado um schema para armazenar essas tabelas, elas pertencerão ao schema padrão public do database bdestudo.

Observe que, na linha 5, a coluna NOME foi declarada com o qualificador UNIQUE. Na prática, o SGBD controlará a propriedade de unicidade na referida coluna, proibindo que haja mais de uma ocorrência da mesma ao longo de todo o ciclo de vida do banco de dados.

Gerenciamento de scripts na prática

Note que o script SQL da imagem possui 15 linhas. Esse é um exemplo didático que possui somente duas tabelas. No entanto, você vai perceber que, no dia a dia, os projetos reais possuem quantidades de tabelas que facilmente podem ultrapassar dois dígitos.

É importante que você conheça comandos DDL, mas o uso prático ocorrerá com o auxílio de ferramentas que automatizam o processo de gestão e administração de dados. Tais ferramentas permitem — manipulando elementos visuais — criar tabelas, relacionamentos e restrições, além de executar outras atividades. As ferramentas permitem gerar código DDL referente ao projeto como um todo, ou parte dele.



Exemplo

Ferramentas DBeaver, Vertabelo, SQL Power Architect, Toad for SQL e Erwin, entre outras.

SGBD PostgreSQL nos bastidores

Aprendemos a criar um database com o uso do comando CREATE DATABASE. No entanto, a submissão de um simples CREATE DATABASE ao PostgreSQL gera uma série de etapas gerenciadas pelo servidor.

Se considerarmos a instalação padrão no Windows, ao criar um data-base, o servidor cria uma pasta identificada por um número (OID), dentro do diretório C:\Program Files\PostgreSQL\12\data\base. Veja!

OS (C:) > Arquivos de Programas > PostgreSQL > 12 > data > base			
	Nome	Data de modificação	Tipo
	1	26/06/2020 17:28	Pasta de arquivos
	13317	04/06/2020 14:34	Pasta de arquivos
	13318	26/06/2020 06:04	Pasta de arquivos
	24600	28/06/2020 13:14	Pasta de arquivos
	41015	30/06/2020 10:42	Pasta de arquivos
	41016	30/06/2020 10:52	Pasta de arquivos

Pastas representativas de databases, antes da criação do database TESTEBANCO.

Após a execução do comando CREATE DATABASE TESTEBANCO, foi criada a pasta 41017, conforme a imagem a seguir.

OS (C:) > Arquivos de Programas > PostgreSQL > 12 > data > base

	Nome	Data de modificação	Tipo
✦	1	26/06/2020 17:28	Pasta de arquivos
✦	13317	04/06/2020 14:34	Pasta de arquivos
✦	13318	26/06/2020 06:04	Pasta de arquivos
✦	24600	28/06/2020 13:14	Pasta de arquivos
	41015	30/06/2020 10:42	Pasta de arquivos
	41016	30/06/2020 10:52	Pasta de arquivos
	41017	30/06/2020 12:32	Pasta de arquivos

Pastas representativas de databases, após da criação de TESTEBANCO.

O PostgreSQL mantém informações sobre todos os databases em um database especial denominado catálogo, cujas tabelas possuem nomes iniciados com o prefixo PG_.



Exemplo

Informações sobre os databases existentes em um servidor são armazenadas na tabela PG_DATABASE.

Caso você queira identificar o nome correspondente ao OID do PostgreSQL, basta executar o comando a seguir.

```
sql
SELECT OID, DATNAME FROM PG_DATABASE;
```

O resultado desse comando pode ser visualizado na seguinte tabela.

	123 oid	ABC datname
1	1	template1
2	13.317	template0
3	13.318	postgres
4	24.600	bdestudo
5	41.015	bdtestepgadmin
6	41.016	bdtestepsql
7	41.017	testebanco

Resultado de consulta à tabela PG_DATABASE.

Em seu computador, o resultado poderá ser diferente do apresentado, tendo em vista as operações que você tenha realizado no SGBD.

Atividade 1

A criação de tabelas é uma das operações fundamentais no PostgreSQL, permitindo a estruturação dos dados em um banco de dados relacional. Com comandos SQL específicos, é possível definir esquemas, especificar tipos de dados e estabelecer chaves primárias e estrangeiras. Para a correta modelagem e organização dos dados, é importante compreender os conceitos e sintaxes envolvidas na criação de tabelas.

Qual das seguintes alternativas descreve corretamente o comando SQL usado para criar uma tabela no PostgreSQL com uma chave primária?

A CREATE DATABASE alunos (id SERIAL PRIMARY KEY, nome VARCHAR(100)).

B CREATE TABLE alunos (id SERIAL, nome VARCHAR(100)).

C CREATE TABLE alunos (id SERIAL PRIMARY KEY, nome VARCHAR(100)).

D CREATE SCHEMA alunos (id SERIAL PRIMARY KEY, nome VARCHAR(100)).

E

```
CREATE INDEX alunos (id SERIAL PRIMARY KEY, nome VARCHAR(100)).
```



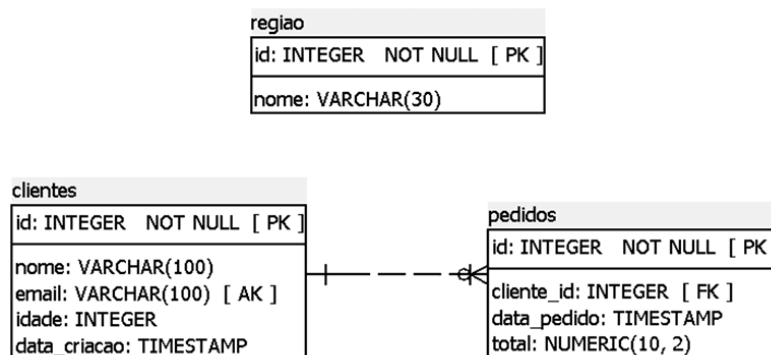
A alternativa C está correta.

A opção correta para criar uma tabela no PostgreSQL com uma chave primária é `CREATE TABLE alunos (id SERIAL PRIMARY KEY, nome VARCHAR(100));`. Esse comando SQL define uma nova tabela chamada `alunos` com duas colunas: `id` e `nome`. A coluna `id` é do tipo `SERIAL`, que é um tipo especial de `integer` autoincrementado, sendo designada como a chave primária da tabela, fazendo com que cada valor seja único e não nulo.

Criação de banco de dados e tabelas na prática

Vamos criar, alterar e eliminar tabelas no PostgreSQL, incluindo o trabalho com tabelas relacionadas. Essas operações são essenciais para o gerenciamento eficiente do esquema de banco de dados em aplicações reais.

Vamos então implementar um banco de dados a partir do modelo lógico a seguir.



Modelo lógico do banco de dados para o estudo de caso.

No vídeo, a seguir, você poderá acompanhar e praticar a criação de banco de dados e tabelas utilizando o pgAdmin. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vamos à prática sobre criação, alteração e eliminação de tabelas no PostgreSQL, incluindo o trabalho com tabelas relacionadas.

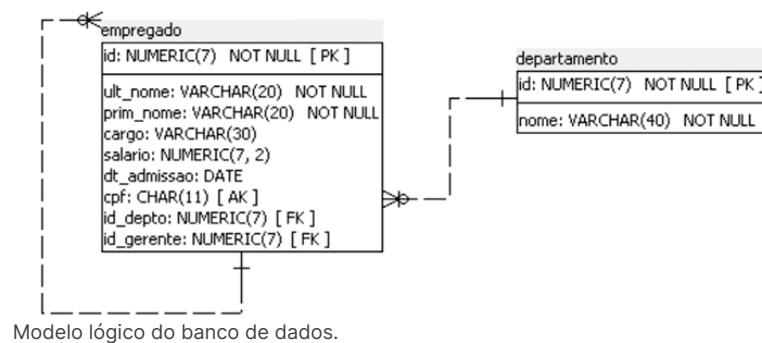
Para **realizar o laboratório**, siga os passos apresentados a seguir.

1. Faça conexão ao SGBD utilizando o pgAdmin4.

2. Crie um banco de dados chamado tabelas_lab.
3. Crie a tabela cliente seguindo o modelo lógico.
4. Crie a tabela pedido seguindo o modelo lógico.
5. Crie a tabela regioao.
6. Insira dados nas tabelas clientes, pedido e região.
7. Consulte os dados das tabelas.

Atividade 2

Execute os passos a seguir para criar o banco de dados da empresa de acordo com este modelo lógico.

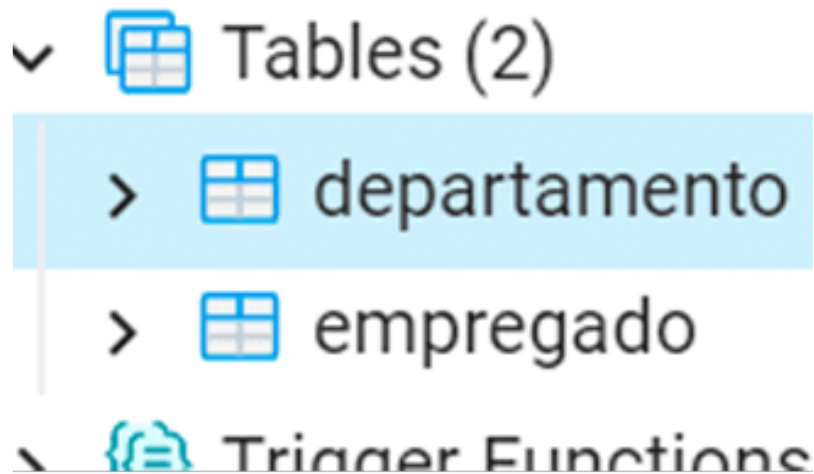


- Acesse o PostgreSQL utilizando o PGAdmin4.
- Crie um banco de dados chamado empresa.
- Abra o query tool conectado ao banco da empresa.
- Crie a tabela Empregado conforme o modelo anterior.

Não deu certo, não é? Você não conseguiu porque a tabela departamento referenciada na chave estrangeira não existe.

- Crie a tabela departamento segundo o modelo anterior.
- Comande novamente a criação de empregado, agora irá funcionar.

E as tabelas irão aparecer na lista de tabelas:



Print das tabelas criadas no pgAdmin.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Comandos SQL para alteração e remoção de tabela

A alteração e a eliminação de tabelas no PostgreSQL são operações que impactam diretamente a manutenção e a evolução de um banco de dados.

Utilizando comandos SQL, é possível modificar a estrutura das tabelas, adicionar ou remover colunas e índices, além de redefinir restrições e relacionamentos. Da mesma forma, a eliminação de tabelas permite a remoção de dados obsoletos ou desnecessários.

Essas operações garantem a flexibilidade e a eficiência do gerenciamento de dados, adaptando o banco de dados às mudanças nas necessidades das aplicações.

No vídeo, a seguir, você vai compreender os comandos SQL para a alteração e a remoção de tabelas. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Alteração de tabela

Você vai se deparar com situações em que será necessário alterar a estrutura de uma tabela já existente.

Vamos levar em consideração a seguinte situação: suponha que surgiu a necessidade de modelar a informação sobre a data de primeiro reconhecimento do curso. Podemos, então, alterar a estrutura da tabela

CURSO, adicionando uma coluna opcional denominada DTRECONH. O comando ALTER TABLE é útil no contexto dessa tarefa.

A seguir, veja a sintaxe básica do comando ALTER TABLE.

```
sql
ALTER TABLE  ADD  ;
```

Em que, na sintaxe apresentada:

- <NOMETABELA>
Representa o nome da tabela na qual haverá a modificação.
- <COLUNA>
Representa o nome da coluna.
- <TIPODEDADOS>
Representa o domínio da coluna.

A sintaxe **completa** a respeito do comando ALTER TABLE pode ser encontrada no site oficial do PostgreSQL.

A seguir, veja o código SQL que permite a alteração da tabela CURSO.

```
sql
-- Comando para alterar a tabela CURSO, adicionando coluna DTRECONH
ALTER TABLE CURSO ADD DTRECONH DATE;
```

Por padrão, o SGBD cria a coluna como opcional. É como se o comando estivesse com o qualificador NULL declarado imediatamente antes do “,”.

E se desejarmos remover uma coluna da tabela? Podemos usar a sintaxe a seguir.

```
sql
ALTER TABLE  DROP  ;
```

Vamos agora remover a coluna DTRECONH da tabela CURSO. A seguir, veja o código SQL que permite a alteração.

```
sql
-- Comando para alterar a tabela CURSO, removendo a coluna DTRECONH
ALTER TABLE CURSO DROP DTRECONH ;
```

Remoção de tabela

Você vai se deparar com situações em que será necessário remover uma tabela do banco de dados. Essa ação é realizada com o auxílio do comando DROP TABLE. A sintaxe desse comando está expressa a seguir.

```
sql  
  
DROP TABLE ;
```

Vamos agora remover a tabela CURSO com o devido código SQL.

```
sql  
  
-- Comando para remover a tabela CURSO  
DROP TABLE CURSO;
```

Mais adiante, vamos conhecer alguns cuidados que devem ser tomados quando formos manipular a estrutura de tabelas relacionadas.

Atividade 3

A alteração e a eliminação de tabelas fazem parte do gerenciamento de bancos de dados no PostgreSQL. Alterar tabelas permite modificar a estrutura existente, como adicionar ou remover colunas, enquanto eliminar tabelas envolve a remoção completa de uma tabela e seus dados. Comandos SQL específicos são utilizados para essas operações, sendo essencial compreender suas sintaxes e implicações para garantir a integridade e a eficiência do banco de dados.

Qual das seguintes alternativas descreve corretamente o comando SQL usado para alterar uma tabela adicionando uma nova coluna e o comando para eliminar uma tabela no PostgreSQL?

☐ A ADD COLUMN alunos idade INT; e REMOVE TABLE alunos;

☒ B ALTER TABLE alunos ADD COLUMN idade INT; e DROP TABLE alunos;

☐ C MODIFY alunos ADD idade INT; e DELETE TABLE alunos;

☐ D UPDATE alunos SET ADD idade INT; e TRUNCATE TABLE alunos;

☐ E ALTER DATABASE alunos ADD COLUMN idade INT; e DELETE FROM alunos;



A alternativa B está correta.

ALTER TABLE alunos ADD COLUMN idade INT; é o comando que altera a tabela alunos, adicionando uma nova coluna chamada idade do tipo inteiro (INT). Já DROP TABLE alunos; é o comando que elimina a tabela alunos completamente do banco de dados, removendo a estrutura da tabela e todos os dados que ela contém.

Tratativas para tabelas relacionadas

No PostgreSQL, tratar tabelas relacionadas por chave estrangeira garante a integridade referencial e mantém os dados organizados em um banco de dados relacional. As chaves estrangeiras estabelecem vínculos entre tabelas, permitindo que os dados em uma tabela sejam referenciados por outra. Saber criar, modificar e eliminar essas relações ajuda a consistência dos dados e facilita operações complexas envolvendo múltiplas tabelas.

No vídeo, a seguir, você vai compreender os cuidados que devemos tomar ao manipular tabelas relacionadas. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Criação e alteração de tabelas relacionadas

No exemplo envolvendo criação de tabelas, o relacionamento entre as tabelas NIVEL e CURSO foi declarado no bloco CREATE TABLE da tabela CURSO (linha 14). No entanto, poderíamos ter optado por criar as tabelas NIVEL e CURSO sem relacionamento, para, em seguida, alterar a tabela CURSO, adicionando a restrição de chave estrangeira.

Na hipótese dessa estratégia, o script SQL ficaria do seguinte modo:

```
1 -- comando para criar a tabela nivel
2 CREATE TABLE NIVEL (
3     CODIGONIVEL int NOT NULL,
4     DESCRICAO varchar(90) NOT NULL,
5     CONSTRAINT CHAVEPNIVEL PRIMARY KEY (CODIGONIVEL));
6 -- comando para criar a tabela curso
7 CREATE TABLE CURSO (
8     CODIGOCURSO int NOT NULL,
9     NOME varchar(90) NOT NULL UNIQUE,
10    DATACRIACAO date NULL,
11    CODIGONIVEL int NULL,
12    CONSTRAINT CHAVEPCURSO PRIMARY KEY (CODIGOCURSO));
13 -- comando para alterar a tabela curso, adicionando chave estrangeira
14 ALTER TABLE CURSO ADD FOREIGN KEY (CODIGONIVEL) REFERENCES NIVEL;
```

Note que, no script anterior, as tabelas foram criadas sem chave estrangeira (linhas 1 a 12). Na linha 14, o comando ALTER TABLE modifica a estrutura da tabela CURSO, implementando a restrição de chave estrangeira que representa o relacionamento entre CURSO e NIVEL.

Cuidados ao manipular tabelas relacionadas

Aprendemos que um banco de dados relacional é composto por diversas tabelas, e cada tabela, em geral, possui várias colunas. Vimos também que o relacionamento entre tabelas é implementado com o uso do mecanismo de chave estrangeira.

Quando definimos que alguma coluna é uma chave estrangeira, na prática, estamos criando uma dependência entre as tabelas envolvidas, pois toda chave estrangeira aponta para o valor de alguma chave primária.

Ao mesmo tempo, todo SGBD precisa manter a integridade dos dados durante todo o ciclo de vida do banco de dados. Com isso, não basta somente conhecermos a estrutura dos comandos para a alteração de tabelas.



Resumindo

Em algumas situações, mesmo que o comando para a alteração ou a exclusão esteja correto do ponto de vista sintático, o SGBD sempre prioriza a integridade dos dados e pode inibir sua execução caso o resultado tenha potencial para gerar inconsistência nos dados.

Vamos estudar um exemplo?

Suponha que temos interesse em remover a tabela NIVEL. Para isso, executaremos o comando SQL a seguir.

```
sql
-- Comando para remover a tabela NIVEL
DROP TABLE NIVEL;
```

O SGBD não removerá a tabela NIVEL e retornará uma mensagem de erro, informando que há um objeto (tabela CURSO) que depende da tabela NIVEL.

Se o SGBD removesse a tabela NIVEL, a tabela CURSO ficaria inconsistente, visto que sua chave estrangeira (CODIGONIVEL) faz referência à chave primária da tabela NIVEL. Assim, antes de remover uma tabela do banco de dados, é necessário avaliar todos os seus relacionamentos.

E se, ainda assim, quiséssemos remover a tabela NIVEL?

Resposta: Antes, precisaríamos remover as dependências, para, em seguida, excluí-la do banco de dados. No entanto, como, no banco de dados, pode haver várias dependências envolvendo a tabela NIVEL — o que tornaria o processo mais demorado — o SGBD dispõe de um recurso que realiza essa tarefa automaticamente. Trata-se de remoção em cascata.

Nesse caso, poderíamos usar o comando SQL a seguir.

```
sql
-- Comando para remover a tabela NIVEL - remoção em cascata
DROP TABLE NIVEL CASCADE;
```

Internamente, o comando altera a estrutura da tabela CURSO, removendo a restrição de chave estrangeira da coluna CODIGONIVEL. Em seguida, a tabela NIVEL é removida do banco de dados.

Atividade 4

Analise o script a seguir e, após, assinale a alternativa correta.

```
1 CREATE TABLE CURSO (  
2     CODIGOCURSO int NOT NULL,  
3     NOME char(90) NOT NULL,  
4     DATACRIACAO date NULL,  
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));  
6  
7 CREATE TABLE DISCIPLINA (  
8     CODIGODISCIPLINA int NOT NULL,  
9     NOME char(90) NOT NULL,  
10    CARGAHORARIA int NOT NULL,  
11    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));  
12  
13 CREATE TABLE CURSODISCIPLINA (  
14     CODIGOCURSO int NOT NULL,  
15     CODIGODISCIPLINA int NOT NULL,  
16     CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA));  
17  
18 ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_CURSO  
19     FOREIGN KEY (CODIGOCURSO)  
20     REFERENCES CURSO (CODIGOCURSO) ;  
21  
22 ALTER TABLE CURSODISCIPLINA ADD CONSTRAINT CURSODISCIPLINA_DISCIPLINA  
23     FOREIGN KEY (CODIGODISCIPLINA)  
24     REFERENCES DISCIPLINA (CODIGODISCIPLINA) ;
```

A A execução completa do script cria três tabelas e todas as colunas são obrigatórias.

B A execução dos comandos entre as linhas 1 e 16 cria três tabelas relacionadas.

C A execução dos comandos entre as linhas 18 e 24 cria três relacionamentos.

D A execução dos comandos entre as linhas 18 e 24 cria dois relacionamentos: o primeiro envolve as tabelas CURSODISCIPLINA e CURSO. O segundo, as tabelas CURSODISCIPLINA e DISCIPLINA.

E A execução dos comandos entre as linhas 18 e 24 cria um relacionamento entre as tabelas CURSO e DISCIPLINA.



A alternativa D está correta.

De fato, há dois blocos de comando entre as linhas 18 e 24.

O primeiro altera a estrutura da tabela CURSODISCIPLINA adicionando à coluna CODIGOCURSO uma restrição de chave estrangeira que implementa o relacionamento entre as tabelas CURSODISCIPLINA e CURSO.

O segundo altera a estrutura da tabela CURSODISCIPLINA, adicionando à coluna CODIGODISCIPLINA uma restrição de chave estrangeira que implementa o relacionamento entre as tabelas CURSODISCIPLINA e DISCIPLINA.

Criação, alteração e remoção de tabelas relacionadas na prática

Vamos à prática de alteração e eliminação de tabelas relacionadas entre si. Veremos, ainda, como executar scripts no PostgreSQL.

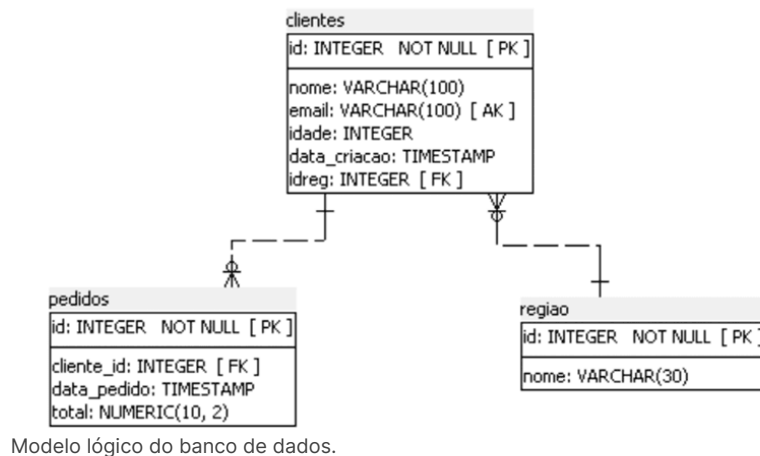
No vídeo, a seguir, você poderá acompanhar e praticar a criação, a alteração e a remoção de tabelas relacionadas. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para executar o laboratório, vamos considerar o modelo lógico apresentado a seguir.



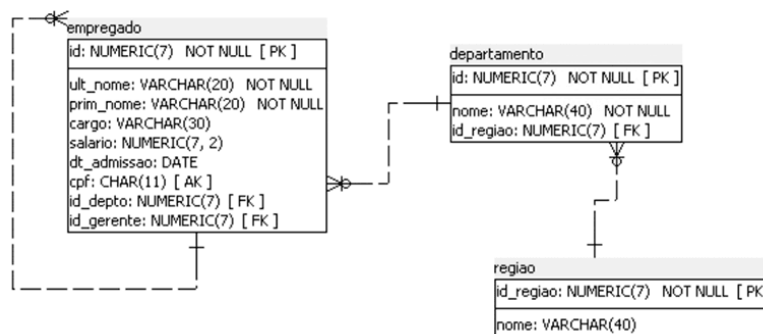
Para realizar o **laboratório**, siga os passos apresentados a seguir.

1. Faça conexão ao SGBD utilizando o pgAdmin4.
2. Acesse o banco tabelas_lab e abra o query tool.
3. Altere a tabela cliente acrescentando a coluna endereço de 200 caracteres.
4. Altere a tabela cliente tornando a coluna idade de preenchimento obrigatório.
5. Altere o nome da coluna endereço de clientes para endereco_residencial.
6. Elimine a coluna endereco_residencial da tabela cliente.
7. Altere a tabela cliente acrescentando uma coluna chamada idreg como chave estrangeira para a tabela regioao.

8. Associar clientes a uma região.
 9. Elimine a tabela pedido.
 10. Elimine a tabela região.
 11. Elimine a tabela cliente.
 12. Rode o script tabelaslab para recriar as tabelas com todas as restrições.
- Clique para conferir o scrip [tabelaslab](#).

Atividade 5

Para realizar a atividade, considere o seguinte modelo lógico:



Modelo lógico do banco de dados.

- Acesse o PostgreSQL utilizando o PGAdmin4.
- Abra o query tool no banco da empresa.
- Crie a tabela região conforme o modelo anterior.
- Acrescente a coluna id_regiao na tabela departamento.
- Defina a coluna id_regiao como chave estrangeira para região.
- Elimine as tabelas região, departamento e empregado.
- Recrie as tabelas utilizando o script da empresa.

Clique para conferir o scrip da [empresa](#).

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Comandos SQL para inserção de linhas em tabela

A inserção de linhas em uma tabela no PostgreSQL permite a população de dados em um banco de dados relacional. Utilizando comandos SQL, é possível adicionar novos registros de forma precisa e eficiente, proporcionando a integridade e a consistência dos dados.

O PostgreSQL oferece diferentes funcionalidades para a inserção de dados, incluindo a possibilidade de inserir múltiplas linhas, retornar valores inseridos e gerenciar conflitos, tornando-o uma excelente ferramenta para o gerenciamento de informações.

No vídeo, a seguir, você vai aprender os comandos SQL para a inserção de linhas em tabelas. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Manipulação de linhas nas tabelas

Quando usamos o termo manipulação, fazemos referência às operações de inserção, atualização ou mesmo eliminação de dados. Em uma linguagem mais comercial, existe o termo **CRUD**, que representa quatro operações básicas: criação, consulta, atualização e remoção de dados, respectivamente.

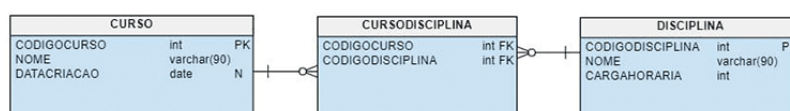
No contexto do SQL, costumamos usar o seguinte mapeamento dos comandos:

- Create: INSERT
- Read: SELECT
- Update: UPDATE
- Delete: DELETE

Os comandos da linguagem SQL que aprenderemos fazem parte da DML (Data Manipulation Language ou Linguagem de Manipulação de Dados) e são usados para inserir, modificar e remover dados.

Modelo para os exemplos

Ao longo de nossa aprendizagem, vamos admitir que o modelo a seguir está criado no banco de dados.



Recomendamos que você crie as tabelas e insira algumas linhas, o que pode ser feito usando o script a seguir, a partir da ferramenta de sua preferência. Para isso, tenha em mente que é necessário estar conectado ao PostgreSQL e ao database bdestudo criado anteriormente.

```
1 CREATE TABLE CURSO (  
2     CODIGOCURSO int NOT NULL,  
3     NOME varchar(90) NOT NULL,  
4     DATACRIACAO date NULL,  
5     CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));  
6 CREATE TABLE DISCIPLINA (  
7     CODIGODISCIPLINA int NOT NULL,  
8     NOME varchar(90) NOT NULL,  
9     CARGAHORARIA int NOT NULL,  
10    CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));  
11 CREATE TABLE CURSODISCIPLINA (  
12     CODIGOCURSO int NOT NULL,  
13     CODIGODISCIPLINA int NOT NULL,  
14     CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),  
15     CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO) ON DELETE CASCADE ,  
16     CONSTRAINT CURSODISCIPLINA_DISCIPLINA FOREIGN KEY (CODIGODISCIPLINA) REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
```

O modelo é útil para gerenciar os dados de cursos, disciplinas e do relacionamento entre esses objetos. Em especial, cada linha da tabela CURSODISCIPLINA representa uma associação entre curso e disciplina.

Inserção de linhas em tabela

A inserção de linhas em tabela é realizada com o auxílio do comando INSERT da SQL. Sua sintaxe **básica** está expressa a seguir.

sql

```
INSERT INTO (COLUNA1, COLUNA2,...,COLUNAn) VALUES (VALOR1, VALOR2,...,VALORn);
```

Na sintaxe, <NOMETABELA> representa a tabela alvo da inserção. Em seguida, são declaradas as colunas que receberão os dados; por último, os dados em si. Note que deve haver uma correspondência entre cada par COLUNA/VALOR, ou seja, o conteúdo de cada coluna deve ser compatível com o tipo de dados ou domínio dela.

A sintaxe **completa** a respeito do comando INSERT pode ser encontrada no site oficial do PostgreSQL.

Vamos estudar um exemplo?

Cadastraremos quatro cursos. Os comandos SQL a seguir podem ser utilizados.

sql

```
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)  
VALUES( 1,'Sistemas de Informação','19/06/1999');  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)  
VALUES( 2,'Medicina','10/05/1990');  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)  
VALUES( 3,'Nutrição','19/02/2012');  
INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO)  
VALUES( 4,'Pedagogia','19/06/1999');
```

Note que, dentro dos parênteses representativos dos conteúdos, o primeiro valor, por ser do tipo inteiro, foi informado sem aspas. Já o segundo valor, por ser do tipo char, foi informado com aspas. Finalmente, o valor referente ao tipo date também foi informado entre aspas. Internamente, o PostgreSQL converte o texto para o formato de data.

Agora, faremos um procedimento semelhante, cadastrando quatro disciplinas. Os comandos SQL a seguir podem ser utilizados.

sql

```
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 1,'Leitura e Produção de Textos',60);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 2,'Redes de Computadores',60);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 3,'Computação Gráfica',40);
INSERT INTO DISCIPLINA (CODIGODISCIPLINA,NOME,CARGAHORARIA)
VALUES( 4,'Anatomia',60);
```

Agora, vamos registrar na tabela CURSODISCIPLINA algumas associações entre cursos e disciplinas. Os comandos SQL a seguir podem ser utilizados.

sql

```
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,2);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,3);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,3);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,1);
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,3);
```

E se submetermos ao SGBD o comando a seguir?

sql

```
INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (3,30);
```

Resposta: O SGBD não realizará a inserção e retornará uma mensagem de erro informando que 30 não é um valor previamente existente na chave primária da tabela DISCIPLINA. Isso acontece porque, quando definimos (linha 16 do script da seção anterior) a chave estrangeira da tabela CURSODISCIPLINA, nós delegamos ao SGBD a tarefa de realizar esse tipo de validação com o objetivo de sempre manter a integridade dos dados do banco de dados. Note que não existe disciplina identificada de código 30 na tabela DISCIPLINA.

Mecanismo de chave primária em ação

Já vimos que o SGBD é responsável por manter a integridade dos dados ao longo de todo o ciclo de vida do banco de dados. A consequência disso pode ser percebida ao tentarmos executar (novamente) o comando a seguir.

sql

```
INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (4,'Anatomia',60);
```

Como já existe um registro com valor de CODIGODISCIPLINA igual a 4, o SGBD exibirá uma mensagem de erro informando que o referido valor já existe no banco de dados. Semelhantemente, devemos lembrar que todo valor de chave primária é obrigatório.

Vamos agora tentar inserir uma disciplina sem valor para CODIGODISCIPLINA, conforme o comando SQL a seguir.

sql

```
INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (NULL, 'Biologia Celular e Molecular', 60);
```

O SGBD exibirá uma mensagem de erro informando que o valor da coluna CODIGODISCIPLINA não pode ser nulo.

Atividade 1

Imagine uma tabela chamada clientes no PostgreSQL com as seguintes colunas: id (chave primária), nome, email e idade. Qual das seguintes opções descreve corretamente o comando SQL usado para inserir um novo cliente na tabela clientes com os valores 1, 'João Silva', 'joao@example.com', e 30?

A INSERT INTO clientes VALUES (1, 'João Silva', 30, 'joao@example.com');

B INSERT INTO clientes (1, 'João Silva', 'joao@example.com', 30);

C INSERT INTO clientes (id, nome, email, idade) VALUES (1, 'João Silva', 'joao@example.com', 30);

D INSERT clientes (id, nome, email, idade) VALUES (1, 'João Silva', 'joao@example.com', 30);

E INSERT INTO clientes SET id = 1, nome = 'João Silva', email = 'joao@example.com', idade = 30;



A alternativa C está correta.

A opção correta para inserir um novo registro na tabela clientes é INSERT INTO clientes (id, nome, email, idade) VALUES (1, 'João Silva', 'joao@example.com', 30);. Esse comando SQL especifica explicitamente as colunas em que os valores serão inseridos, garantindo clareza e precisão na operação.

Prática de inserção de linhas em tabela

Vamos explorar agora várias técnicas de inserção de dados no PostgreSQL, incluindo inserções simples, múltiplas, com retorno de valores, condicionais e a partir de outra tabela. Dominar essas técnicas é fundamental para a manipulação eficiente de dados em aplicações reais.

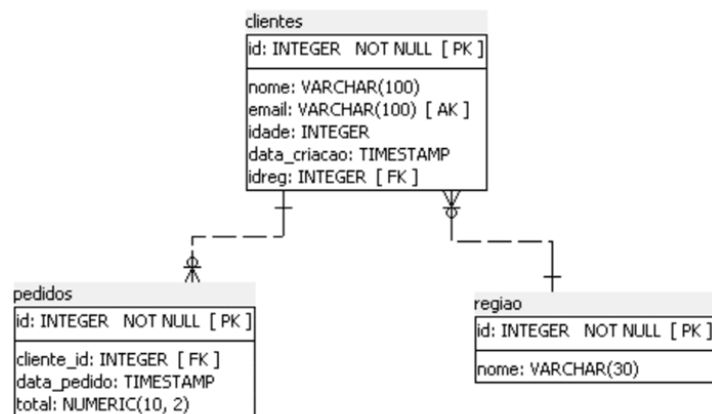
No vídeo, a seguir, aprenda e pratique a inserção de linhas em tabelas utilizando a linguagem SQL. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para executar a prática, vamos considerar o modelo lógico apresentado a seguir.



Modelo lógico do banco de dados.

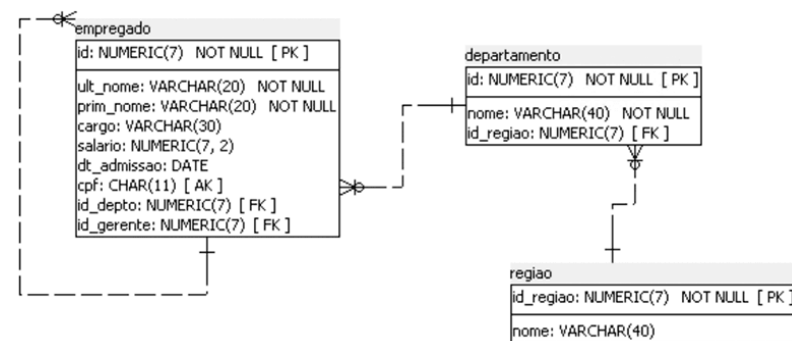
Siga os passos apresentados a seguir.

1. Abra o pgadmin.
2. Abra o query tools no banco tabelas_lab.
3. Realize a inserção Simples de uma linha na tabela clientes.
4. Verifique a linha inserida.
5. Realize a inserção Simples de uma linha na tabela clientes.
6. Realize a inserção múltipla de duas linhas na tabela clientes.
7. Verifique as inserções.
8. Realize a inserção de uma linha com retorno de valores na tabela clientes.
9. Realize a inserção condicional com INSERT ... ON CONFLICT para gerenciar conflitos de chave única (como o email) e tente inserir uma linha com e-mail repetido.
10. Verifique as inserções.
11. Realize a inserção condicional com INSERT ... ON CONFLICT para gerenciar conflitos de chave única (como o email) e tente inserir uma linha com e-mail repetido e, se houver, atualize a idade.

12. Verifique a atualização.
13. Crie uma tabela cópia de clientes chamada clientes_backup sem dados.
14. Insira na tabela clientes_backup os dados dos clientes com mais de 30 anos.
15. Verifique as inserções.

Atividade 2

Para realizar a atividade, considere o modelo lógico apresentado a seguir.



Modelo lógico do banco de dados.

- Acesse o PostgreSQL utilizando o PGAdmin4.
- Abra o query tool no banco da empresa.
- Insira uma linha na tabela de região listando as colunas com ID 1 e nome Norte.
- Insira uma linha sem listar as colunas com ID 2 e nome Sul.
- Valide a inserção.
- Insira na tabela departamento 3 linhas com um único comando com os dados:

Linha 1: 10, 'Administrativo',1;

Linha 2: 20, 'Vendas',1;

Linha 3: 30, 'Compras',2;

- Valide a inserção.
- Execute o script populaempregado.

Clique para conferir o scrip [populaempregado](#).

- Valide a inserção.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Comandos SQL para atualização de linhas em tabela

A operação de atualização (UPDATE) em tabelas do PostgreSQL permite a manutenção e a modificação dos dados existentes no banco de dados. Utilizando comandos SQL, é possível alterar valores de registros de forma seletiva e precisa, garantindo que as informações reflitam as mudanças necessárias nos dados armazenados.

O PostgreSQL oferece funcionalidades avançadas para a execução de atualizações, incluindo a possibilidade de combinar condições complexas e retornar valores modificados, assegurando a eficiência e a integridade dos dados durante o processo.

No vídeo, a seguir, você vai compreender os comandos SQL para a atualização de linhas em tabela. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Atualização de linhas em tabela

A atualização de linhas em tabela é realizada com o auxílio do comando UPDATE da SQL. Sua sintaxe **básica** está expressa a seguir.

```
sql
UPDATE
SET COLUNA1=VALOR1, COLUNA2=VALOR2,...,COLUNAn=VALORn
WHERE ;
```

Na sintaxe, <NOMETABELA> representa a tabela alvo da atualização. Em seguida, é declarada uma lista contendo a coluna e o seu respectivo valor novo. Por último, uma condição lógica, caso seja necessário. Isso

ocorre porque, em geral, estamos interessados em alterar somente um subconjunto de linhas que é obtido a partir do processamento da cláusula WHERE.

A sintaxe **completa** a respeito do comando UPDATE pode ser encontrada no site oficial do PostgreSQL.

A seguir, vamos estudar alguns exemplos!

Alteraremos para 70 a carga horária da disciplina Redes de Computadores. Para isso, basta executar o comando a seguir.

```
sql
UPDATE DISCIPLINA SET CARGAHORARIA=70 WHERE CODIGODISCIPLINA=2;
```

No comando, o SGBD busca, na tabela, a disciplina com o valor da coluna CODIGODISCIPLINA igual a 2. Em seguida, atualiza o valor da coluna CARGAHORARIA para 70.

Note também que poderíamos ter executado o comando a seguir.

```
sql
UPDATE DISCIPLINA SET CARGAHORARIA=70 WHERE NOME='Redes de Computadores';
```

Suponha agora que houve a necessidade de alterar em 20% a carga horária de todas as disciplinas cadastradas no banco de dados. Podemos executar o comando a seguir.

```
sql
UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2;
```

No último comando, note que não foi necessária a cláusula WHERE, visto que o nosso interesse era o de atualizar todas as linhas da tabela DISCIPLINA. Ainda, para obter o **novo** valor, nós utilizamos a expressão CARGAHORARIA*1.2.

Atualização de coluna chave primária

Devemos ter especial cuidado ao planejarmos alterar o valor de coluna com o papel de chave primária em uma tabela.

Vamos supor que seja necessário alterar para 6 o valor de CODIGOCURSO referente ao curso de Pedagogia. Podemos, então, executar o comando a seguir.

```
sql
UPDATE CURSO SET CODIGOCURSO=6 WHERE CODIGOCURSO=4;
```

Perceba que o SGBD processará a alteração, visto que não há vínculo na tabela CURSODISCIPLINA envolvendo esse curso.

No entanto, o que aconteceria se tentássemos alterar para 10 o valor de CODIGOCURSO referente ao curso de Sistemas de Informação?

Seguindo a mesma linha de raciocínio da última alteração, vamos submeter o comando a seguir.

```
sql
UPDATE CURSO SET CODIGOCURSO=10 WHERE CODIGOCURSO=1;
```

O SGBD não realizará a alteração e retornará uma mensagem de erro indicando que o valor 1 está registrado na tabela CURSODISCIPLINA, coluna CODIGOCURSO. Isso significa que, se o SGBD aceitasse a alteração, a tabela CURSODISCIPLINA ficaria com dados inconsistentes, o que não deve ser permitido.

Assim, de modo semelhante ao que aprendemos em Mecanismo de chave primária em ação, deixaremos o SGBD realizar as alterações necessárias para manter a integridade dos dados. Vamos, então, submeter o comando a seguir.

```
sql
ALTER TABLE CURSODISCIPLINA
DROP CONSTRAINTCURSODISCIPLINA_CURSO,
ADD CONSTRAINT CURSODISCIPLINA_CURSO
FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO)
ON UPDATE CASCADE ;
```

O que fizemos?

Usamos o comando ALTER TABLE para alterar a estrutura da tabela CURSODISCIPLINA; removemos a chave estrangeira (comando DROP CONSTRAINT) e, por último, recriamos a chave (ADD CONSTRAINT), especificando a operação de atualização (UPDATE) em cascata.

Assim, após o processamento da alteração anterior, podemos então submeter o comando, conforme a seguir.

```
sql
UPDATE CURSO SET CODIGOCURSO=10 WHERE CODIGOCURSO=1;
```

Atividade 3

Imagine uma tabela chamada funcionários no PostgreSQL com as seguintes colunas: id (chave primária), nome, salário e departamento_id (chave estrangeira referenciando a tabela departamentos). Você precisa atualizar o salário de um funcionário específico cujo ID é 101.

Qual das seguintes alternativas descreve corretamente o comando SQL que você usaria para realizar essa atualização?

A UPDATE funcionarios SET salario = 5500 WHERE id = 101;

B UPDATE funcionarios SET salario = 5500;

C UPDATE funcionarios SET salario = 5500 WHERE nome = '101';

D UPDATE funcionarios SET salario = 5500 WHERE departamento_id = 101;

E UPDATE funcionarios SET salario = 5500 WHERE id = '101';



A alternativa A está correta.

Para atualizar o salário de um funcionário específico com o ID 101 na tabela funcionários, você deve usar o comando que identifica a linha correta com base no ID único (id). Portanto, o comando SQL correto é UPDATE funcionarios SET salario = 5500 WHERE id = 101;.

Comandos SQL para remoção de linhas em tabela

A operação de exclusão (DELETE) de registros em tabelas do PostgreSQL permite a gestão eficiente dos dados em um banco de dados relacional. Utilizando comandos SQL, é possível remover registros específicos de forma seletiva, assegurando que apenas os dados desejados sejam eliminados.

O PostgreSQL oferece uma variedade de opções para a execução de exclusões, incluindo o uso de condições complexas para especificar os registros a serem removidos, garantindo assim a manutenção da integridade e da precisão dos dados armazenados.

Assista ao vídeo, a seguir, para compreender os comandos SQL para a remoção de linhas em tabela.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Remoção de linhas em tabela

A remoção de linhas em tabela é realizada com o auxílio do comando DELETE da SQL. Sua sintaxe **básica** está expressa a seguir.

```
sql
DELETE FROM
WHERE ;
```

Na sintaxe, <NOMETABELA> representa a tabela alvo da operação de deleção de linha(s). Por último, há uma condição lógica, caso seja necessário. Isso ocorre porque, em geral, estamos interessados em remover somente um subconjunto de linhas que é obtido a partir do processamento da cláusula WHERE.

A sintaxe **completa** a respeito do comando DELETE pode ser encontrada no site oficial do PostgreSQL.

Agora, vamos estudar alguns exemplos!

Suponha que temos interesse em apagar do banco de dados a disciplina de Anatomia. Podemos, então, submeter o código a seguir.

```
sql
DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=4;
```

O SGBD localiza, na tabela DISCIPLINA, a linha com o conteúdo da coluna CODIGODISCIPLINA igual a 1. Em seguida, remove do banco de dados a linha em questão.

Agora vamos imaginar que tenha surgido a necessidade de remover do banco de dados a disciplina de Leitura e Produção de Textos. Podemos, então, submeter o código a seguir.

```
sql
DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=1;
```

O SGBD não realizará a remoção e retornará uma mensagem de erro indicando que o valor 1 está registrado na tabela CURSODISCIPLINA, coluna CODIGODISCIPLINA. Se o SGBD aceitasse a remoção, a tabela CURSODISCIPLINA ficaria com dados inconsistentes, o que não deve ser permitido.

Assim, de modo semelhante ao que aprendemos em Mecanismo de chave primária em ação, deixaremos o SGBD realizar as alterações necessárias para manter a integridade dos dados. Vamos, então, submeter o comando a seguir.

```
sql
ALTER TABLE CURSODISCIPLINA
DROP CONSTRAINT CURSODISCIPLINA_DISCIPLINA,
ADD CONSTRAINT CURSODISCIPLINA_DISCIPLINA
FOREIGN KEY (CODIGODISCIPLINA) REFERENCES DISCIPLINA (CODIGODISCIPLINA)
ON DELETE CASCADE ;
```

O que fizemos?

Usamos o comando ALTER TABLE para alterar a estrutura da tabela CURSODISCIPLINA: removemos a chave estrangeira (comando DROP CONSTRAINT) e, por último, recriamos a chave (ADD CONSTRAINT), especificando operação de remoção (DELETE) em cascata.

Assim, após o processamento da alteração anterior, podemos submeter o comando conforme a seguir.

```
sql

DELETE FROM DISCIPLINA WHERE CODIGODISCIPLINA=1;
```

Ao processar o comando, o SGBD verifica se existe alguma linha da tabela CURSODISCIPLINA com valor 1 para a coluna CODIGODISCIPLINA. Caso encontre, cada ocorrência é então removida do banco de dados.

E se quiséssemos eliminar todos os registros de todas as tabelas do banco de dados?

Resposta: Para realizarmos essa operação, precisaremos identificar quais tabelas são mais independentes e quais são as que possuem vínculos de chave estrangeira.

No caso do nosso exemplo, CURSODISCIPLINA possui duas chaves estrangeiras, portanto, é a tabela mais dependente. As demais não possuem chave estrangeira. De posse dessa informação, podemos submeter os comandos a seguir para completar a tarefa.

```
sql

DELETE FROM CURSODISCIPLINA;
DELETE FROM CURSO;
DELETE FROM DISCIPLINA;
```

Perceba que a primeira tabela usada no processo de remoção de linhas foi a CURSODISCIPLINA, pois essa é a responsável por manter as informações sobre o relacionamento entre as tabelas CURSO e DISCIPLINA. Após eliminar os registros de CURSODISCIPLINA, o SGBD removerá com sucesso os registros das tabelas CURSO e DISCIPLINA.

Atividade 4

Analise o script a seguir e, após, assinale a alternativa correta.

```
1 CREATE TABLE CURSO (
2   CODIGOCURSO int NOT NULL,
3   NOME char(90) NOT NULL,
4   DATACRIACAO date NULL,
5   CONSTRAINT CURSO_pk PRIMARY KEY (CODIGOCURSO));
6 CREATE TABLE DISCIPLINA (
7   CODIGODISCIPLINA int NOT NULL,
8   NOME char(90) NOT NULL,
9   CARGAHORARIA int NOT NULL,
10  CONSTRAINT DISCIPLINA_pk PRIMARY KEY (CODIGODISCIPLINA));
11 CREATE TABLE CURSODISCIPLINA (
12   CODIGOCURSO int NOT NULL,
13   CODIGODISCIPLINA int NOT NULL,
14   CONSTRAINT CURSODISCIPLINA_pk PRIMARY KEY (CODIGOCURSO,CODIGODISCIPLINA),
15   CONSTRAINT CURSODISCIPLINA_CURSO FOREIGN KEY (CODIGOCURSO) REFERENCES CURSO (CODIGOCURSO) ON DELETE CASCADE ,
16   CONSTRAINT CURSODISCIPLINA_DISCIPLINA FOREIGN KEY (CODIGODISCIPLINA) REFERENCES DISCIPLINA (CODIGODISCIPLINA) );
17
18 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (1,'Sistemas de Informação',NULL);
19 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (2,'Medicina','10/05/1990');
20
21 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (1,'Leitura e Produção de Textos',60);
22 INSERT INTO DISCIPLINA (CODIGODISCIPLINA, NOME, CARGAHORARIA) VALUES (2,'Redes de Computadores',60);
23
24 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,1);
25 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (1,2);
26 INSERT INTO CURSODISCIPLINA(CODIGOCURSO,CODIGODISCIPLINA) VALUES (2,1);
27
28 DELETE FROM CURSO WHERE CODIGOCURSO=1;
```

Após a execução, com sucesso, do trecho entre as linhas 1 e 26, se executarmos o comando expresso na linha 28, quantas linhas serão removidas do SGBD?

A Nenhuma

B Uma

C Duas

D Três

E Quatro



A alternativa D está correta.

De fato, a chave estrangeira declarada na linha 15 foi criada de maneira a permitir a deleção em cascata. Ao executar o comando da linha 28, o SGBD eliminará tanto os dois registros da tabela CURSODISCIPLINA (com valor de CODIGOCURSO igual a 1) quanto o registro referente ao curso Sistemas de Informação.

Prática de atualização e remoção de linhas em tabela

Vamos explorar agora várias técnicas de atualização e exclusão de dados no PostgreSQL, incluindo atualizações simples, condicionais, exclusões simples e múltiplas. Dominar essas técnicas é essencial para a manutenção e a manipulação eficiente de dados em aplicações reais.

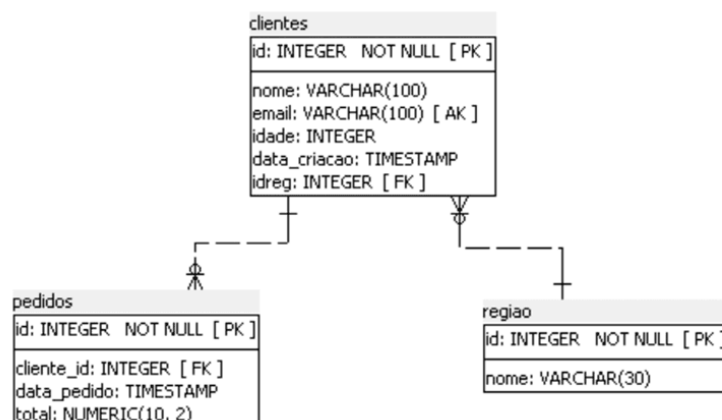
No vídeo, a seguir, você poderá aprender e praticar os comandos SQL de atualização e remoção de linhas em tabela. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para executar o laboratório, vamos considerar o modelo lógico apresentado a seguir.



Modelo lógico do banco de dados.

Para realizar a prática, siga os passos apresentados a seguir.

1. Abra o pgadmin.
2. Abra o query tools no banco tabelas_lab.
3. Atualize a idade do cliente 'João Silva' para 35 anos.
4. Verifique a atualização.
5. Atualize a idade de todos os clientes com idade menor que 30.
6. Verifique as atualizações.
7. Atualize a idade de 'Ana Lima' para 26 anos e retorne a linha atualizada.
8. Atualize id_reg dos clientes 1 e 2 para apontar para a região 1.

Não conseguiu? Vamos lá!

9. Insira na tabela região uma linha com os valores 1 e 'Regiao 1'.
 10. Repita o update do idreg.
 11. Valide a atualização.
 12. Exclua o cliente 'Carlos Pereira'.
 13. Confirme a exclusão.
 14. Exclua todos os clientes com idade maior que 30.
 15. Confirme a exclusão.
 16. Exclua as linhas da tabela região.
- Ainda não conseguiu?**
17. Exclua todas as linhas tabela clientes.
 18. Confirme a exclusão.

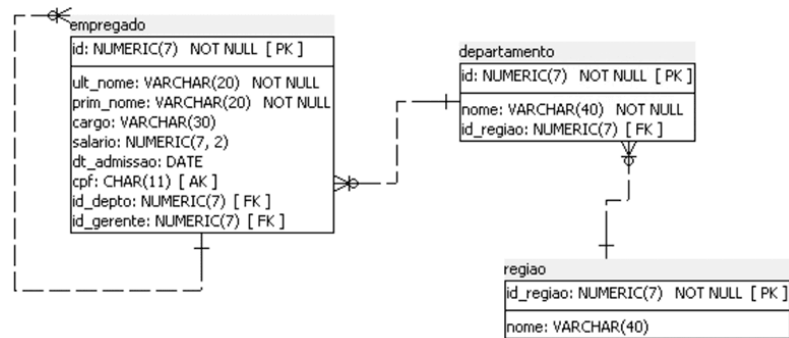
19. Repita a exclusão das linhas de região.

Agora sim! Pois não existem mais linhas em departamento apontando para a linha de região.

20. Confirme a exclusão.

Atividade 5

Para realizar a atividade, considere o modelo lógico apresentado a seguir.



Modelo lógico do banco de dados.

- Acesse o PostgreSQL utilizando o PGAdmin4.
- Abra o query tool no banco da empresa.
- Altere o nome da região de ID para Exemplo1.
- Valide a alteração.
- Altere o salário de todos os empregados para 10000.
- Valide a alteração.
- Elimine todos os empregados.
- Valide a eliminação.
- Elimine a região de ID 1.

Você não conseguiu ainda porque tem departamento apontando para a região 1. Vamos lá!

- Elimine os departamentos da região 1.
- Comande novamente delete na região de ID 1.
- Agora sim! Valide a eliminação.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Controle de transação em banco de dados

Transações em bancos de dados garantem a consistência, a integridade e a atomicidade das operações realizadas. Uma transação é uma sequência de operações SQL que são tratadas como uma unidade indivisível: ou todas as operações são executadas com sucesso e confirmadas (commit), ou todas são desfeitas (rollback) em caso de erro. Esse mecanismo assegura que o banco de dados permaneça em um estado consistente, mesmo diante de interrupções ou falhas durante o processamento das operações.

No vídeo, a seguir, você vai compreender os principais conceitos relacionados com as transações em banco de dados. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Transações em banco de dados

Ao longo do nosso estudo, aprendemos uma série de comandos SQL envolvendo desde a criação de tabelas até operações de manipulação de dados, tais como inserções, atualizações e exclusões. Basicamente, um comando era codificado e submetido ao SGBD PostgreSQL, que, em seguida, executava-o e devolvia algum resultado.

Perceba que, sob o contexto dos exemplos que estudamos, poderíamos concluir que, de certa maneira, havia somente um usuário acessando a totalidade dos recursos do SGBD.



No entanto, em um ambiente de produção, o SGBD gerencia centenas de requisições das aplicações. Com isso, concluímos que há acesso simultâneo a vários recursos que são gerenciados pelo SGBD.

Ao mesmo tempo, o usuário de uma aplicação que faz uso de recursos de um banco de dados, normalmente, está preocupado com o resultado dos processos de negócio que estão automatizados.



Exemplo

Digamos que um sistema acadêmico disponibiliza a inscrição em diversas disciplinas. Determinado estudante, após consultar a oferta e escolher um conjunto de matérias para inscrição, tem expectativa de conseguir inscrever-se em todas as disciplinas alvo da escolha. No entanto, sob o ponto de vista do SGBD, prover a inscrição em todas as disciplinas escolhidas pelo estudante requer a execução, na totalidade, de diversas instruções de inserção de dados em alguma tabela. Além disso, deve existir cuidado especial para, por exemplo, inibir inscrição em disciplina caso não haja mais disponibilidade de vagas. Essa é uma situação típica sobre a qual o SGBD precisa prover uma forma de realizar diversas operações como uma unidade lógica de processamento.

Vamos aprender que a forma de realizar diversas operações como uma unidade lógica de processamento é denominada **transação**. Em sistemas de banco de dados, uma transação corresponde a um programa em execução que forma uma unidade de trabalho.

Os limites de uma transação são especificados por meio dos comandos `begin transaction` (que indica o início de uma transação) e `end transaction` (que indica o término de uma transação) em um programa de aplicação.

Confira, a seguir, um pouco mais sobre a transação.

Leitura

É a transação que não atualiza o banco de dados.

Leitura-gravação

É a transação que atualiza o banco de dados.

Para fins didáticos, um modelo simplificado do banco de dados — coleção de itens nomeados — é usado para estudo dos conceitos de processamento de transações. Cada item de dados pode representar um registro, bloco ou valor de coluna.

Em se tratando de SGBDs multiusuários, várias transações podem ser executadas simultaneamente. No entanto, caso essa execução ocorra de maneira descontrolada, poderão surgir problemas de inconsistências. Observe!

Atualização perdida

Quando duas transações que acessam os mesmos itens de dados têm operações intercaladas de modo a tornar incorretos alguns itens do banco de dados.

Atualização temporária

Quando uma transação atualiza um item do banco de dados e, depois, falha por algum motivo, enquanto, nesse meio tempo, o item é lido por outra transação antes de ser alterado de volta para seu valor original.

Resumo incorreto

Quando uma transação calcula uma função de resumo de agregação em uma série de itens enquanto outras transações atualizam alguns desses itens.

Leitura não repetitiva

Quando uma transação lê o mesmo item duas vezes, e o item é alterado por outra transação entre as duas leituras.

É importante sabermos que, quando o SGBD processa uma transação, todas as operações que a formam devem ser completadas com sucesso para, somente depois, haver a gravação permanente das alterações no banco de dados.

Além disso, caso haja falha em uma ou mais operações de uma transação, as demais operações não devem ser executadas, e a transação deve ser cancelada. Veja o que deve ser feito nesses casos.

Se uma transação forcancelada

Deve ser executado um processo denominado rollback (rolar para trás), o qual força o SGBD a trazer de volta os valores antigos dos registros antes da transação ter iniciado.

Se uma transação forexecutada com sucesso

As atualizações devem ser confirmadas por meio do comando commit (confirmação).

Veja, a seguir, quais são as falhas que podem ocorrer durante o processamento de uma transação.

- Falha do computador.
- Erro de transação ou sistema.
- Condições de exceção detectadas pela transação.
- Falha de disco, problemas físicos e catástrofes.

A seguir, conheceremos as propriedades das transações.

Propriedades de uma transação

Com o objetivo de garantir a integridade dos dados contidos no banco de dados, o SGBD mantém um conjunto de propriedades das transações, denominado ACID (do inglês, Atomicity, Consistency, Isolation and Durability), que representam as características de atomicidade, consistência, isolamento e durabilidade, respectivamente. Observe!

- **Atomicidade:** a transação precisa ser realizada completamente ou não realizada.

- **Consistência:** a transação deve levar o banco de dados de um estado consistente para outro.
- **Isolamento:** a execução de uma transação não deve ser interferida por quaisquer outras transações.
- **Durabilidade:** as mudanças no banco de dados em função da confirmação da transação devem persistir.

Estados de uma transação

Uma transação pode passar pelos seguintes estados ao longo do seu processamento. Veja!

Ativo

Ocorre imediatamente após o início da transação, podendo executar operações de leitura e gravação.

Parcialmente confirmado

Ocorre quando a transação termina.

Confirmado

Ocorre quando, após a verificação bem-sucedida, as mudanças são gravadas permanentemente no banco de dados.

Falha

Ocorre se uma das verificações falhar ou se a transação for abortada durante seu estado ativo.

Confirmado

Ocorre quando a transação sai do sistema.

Para poder recuperar-se de falhas que afetam transações, normalmente, o SGBD mantém um log para registrar todas as operações de transação que afetam os vários itens do banco de dados.

As entradas em um registro de log de uma determinada transação possuem informações de valores antigos e novos do item de dados do banco de dados bem como se a transação foi concluída com sucesso ou se foi abortada.

Atividade 1

No contexto dos bancos de dados relacionais, uma transação permite agrupar uma ou mais operações de banco de dados em uma unidade de trabalho indivisível, assegurando que todas as operações sejam

executadas com sucesso ou revertidas para um estado anterior caso ocorra algum erro. Compreender os princípios básicos das transações é fundamental para o desenvolvimento de aplicações robustas e confiáveis, garantindo que todas as mudanças nos dados sejam realizadas de forma controlada e segura.

Qual das seguintes afirmativas melhor descreve o conceito de transação em bancos de dados relacionais, como no PostgreSQL?

- A Uma transação é um conjunto de operações que não podem ser desfeitas uma vez executadas.
- B Uma transação faz com que todas as operações sejam executadas simultaneamente para garantir a consistência dos dados.
- C Uma transação é um mecanismo que permite agrupar uma ou mais operações de banco de dados em uma unidade de trabalho única e indivisível.
- D Uma transação assegura que apenas uma operação de banco de dados possa ser executada por vez, evitando conflitos de simultaneidade.
- E Uma transação é um procedimento que mantém a segurança física dos dados armazenados no banco de dados.



A alternativa C está correta.

Uma transação em bancos de dados relacionais, como no PostgreSQL, permite agrupar operações de banco de dados em uma unidade de trabalho indivisível. Isso significa que todas as operações em uma transação são tratadas como uma única entidade de execução.

A transação deve ser atômica, ou seja, todas as operações devem ser concluídas com sucesso ou todas devem ser desfeitas se ocorrer um erro, proporcionando a consistência e a integridade dos dados.

Transações no PostgreSQL

Transações em banco de dados no PostgreSQL garantem a consistência e a integridade dos dados durante operações complexas e críticas. Uma transação é um conjunto de operações SQL que são tratadas como uma unidade indivisível, em que todas as operações são executadas com sucesso ou nenhuma delas é aplicada.

Utilizando comandos SQL como BEGIN, COMMIT e ROLLBACK, o PostgreSQL permite iniciar, confirmar ou desfazer transações para que o banco de dados permaneça em um estado consistente mesmo em caso de erros ou falhas durante o processamento.

No vídeo, a seguir, você vai compreender os comandos relacionados com transações no PostgreSQL. Não perca!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

De maneira geral, uma transação no PostgreSQL possui a estrutura apresentada a seguir.

```
sql
BEGIN-- início da transação
-- comandos
COMMIT-- transação confirmada
ROLLBACK-- transação cancelada
END-- mesma função do COMMIT
```

Nos SGBDs, no entanto, a inicialização de uma transação ocorre implicitamente quando executamos alguns comandos.

Vamos estudar um exemplo?

Seja a tabela CURSO, conforme a seguir.

CURSO			
CODIGOCURSO	int		PK
NOME	varchar(90)		
DATAACRIACAO	date		N

Tabela CURSO.

A execução de um INSERT na tabela ocorre no contexto implícito de uma transação.

```
sql
-- BEGIN implícito
INSERT INTO CURSO (CODIGOCURSO,NOME,DATAACRIACAO) VALUES (5,'Engenharia de
Computação',NULL);
-- COMMIT implícito
```

Estudamos que, quando uma transação é desfeita, qualquer operação que faz parte dessa transação deve ser cancelada. Vamos, então, ver como podemos fazer isso no PostgreSQL.

Veja o exemplo a seguir, construído com o objetivo de inserir um registro na tabela CURSO e, em seguida, indicar que a operação de inserção deve ser desfeita pelo SGBD.

```

1 SELECT * FROM CURSO; -- Dados atuais
2 BEGIN;
3 INSERT INTO CURSO (CODIGOCURSO,NOME,DATACRIACAO) VALUES (6,'Psicologia',NULL);
4 SELECT * FROM CURSO; -- Dados após inserção
5 ROLLBACK;
6 END;
7 SELECT * FROM CURSO; -- Dados após desfazer a transação

```

Após o processamento do comando da linha 1, visualizaremos o conteúdo da tabela CURSO da seguinte maneira.

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19

Tabela CURSO após a execução do comando da linha 1.

Após o processamento dos comandos da linha 2 à linha 4, que já representam a inserção de um registro na tabela CURSO sob o contexto de uma transação explícita, teremos o resultado a seguir.

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19
5	6	Psicologia	[NULL]

Tabela CURSO após a execução do comando da linha 4.

Finalmente, após o processamento dos comandos da linha 5 à linha 7, em que a transação é desfeita, teremos o resultado conforme a tabela a seguir.

	codigocurso	nome	datacriacao
1	2	Medicina	1990-05-10
2	1	Sistemas de Informação	1999-06-19
3	3	Nutrição	2012-02-19
4	4	Pedagogia	1999-06-19

Tabela CURSO após a execução do comando da linha 7.

Perceba que o comando da linha 2 (BEGIN) iniciou explicitamente uma transação, a qual foi abortada quando da execução do comando da linha 5 (ROLLBACK).

No exemplo anterior, programamos uma transação que envolveu somente uma operação, a saber, inserção de uma linha na tabela CURSO. No entanto, uma transação pode envolver diversas linhas de uma tabela do banco de dados.

Vamos, então, programar uma transação que consistirá em duas operações de atualização envolvendo a tabela que contém as disciplinas, apresentada conforme a seguir.

DISCIPLINA		
CODIGODISCIPLINA	int	PK
NOME	varchar(90)	
CARGAHORARIA	int	

Tabela DISCIPLINA.

Inicialmente, vamos listar o conteúdo da tabela DISCIPLINA. Podemos, então, usar o comando a seguir.

```
sql
SELECT * FROM DISCIPLINA;
```

Após o processamento do comando, teremos o resultado a seguir.

	codigodisciplina	nome	cargahoraria
1	1	Leitura e Produção de Textos	60
2	2	Redes de Computadores	60
3	3	Computação Gráfica	40
4	4	Anatomia	60

Conteúdo da tabela DISCIPLINA com os dados originais.

Agora, nossa intenção é alterar a carga horária das disciplinas conforme os critérios a seguir.

- Disciplinas que possuem 60 horas: aumento em 20%.
- Disciplinas que possuam 40 horas: aumento em 10%.

O código a seguir expressa uma transação envolvendo operações de atualização de dados na tabela DISCIPLINA.

```
1 BEGIN;
2 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2 WHERE CARGAHORARIA=60;
3 SELECT * FROM DISCIPLINA;
4 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.1 WHERE CARGAHORARIA=40;
5 SELECT * FROM DISCIPLINA;
6 COMMIT;
```

Após a execução do trecho entre as linhas 1 e 3, o conteúdo da tabela disciplina estará conforme a seguir.

	codigodisciplina	nome	cargahoraria
1	3	Computação Gráfica	40
2	1	Leitura e Produção de Textos	72
3	2	Redes de Computadores	72
4	4	Anatomia	72

Conteúdo da tabela DISCIPLINA após a execução da linha 3.

Após a execução da transação, o conteúdo da tabela disciplina estará conforme a seguir.

	codigodisciplina	nome	cargahoraria
1	1	Leitura e Produção de Textos	72
2	2	Redes de Computadores	72
3	4	Anatomia	72
4	3	Computação Gráfica	44

Conteúdo da tabela DISCIPLINA após o término da transação.

Outro ponto interessante no projeto de transações é a utilização de ponto de salvamento (SAVEPOINT). Observe o exemplo a seguir.

```

1 BEGIN;
2 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2 WHERE CARGAHORARIA=60;
3 SELECT * FROM DISCIPLINA;
4 SAVEPOINT CARGA60;
5 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.1 WHERE CARGAHORARIA=40;
6 ROLLBACK TO CARGA60;
7 SELECT * FROM DISCIPLINA;
8 COMMIT;

```

Na linha 4, adicionamos um SAVEPOINT denominado CARGA60. Quando a linha 6 for executada, o SGBD vai desfazer a operação de UPDATE da linha 5.

Atividade 2

Imagine desenvolver uma aplicação web que possibilita aos usuários realizar transferências bancárias entre contas no PostgreSQL.

Qual das seguintes práticas relacionadas ao controle de transações no PostgreSQL é a mais indicada para garantir a consistência dos dados durante uma transferência?

- A Iniciar explicitamente uma transação antes de cada consulta SQL relacionada à transferência e confirmar a transação imediatamente após cada consulta.
- B Usar um controle automático de transações fornecido pelo PostgreSQL, que gerencia automaticamente o início e o término de transações para cada consulta SQL.
- C Não utilizar transações para operações de transferência bancária, pois o PostgreSQL garante a consistência dos dados automaticamente.
- D Utilizar exclusivamente o recurso de rollback automático do PostgreSQL para desfazer automaticamente quaisquer operações de transferência que resultem em erros.
- E Utilizar um mecanismo manual de commit e rollback somente no final de todas as operações de transferência, sem iniciar explicitamente transações.



A alternativa A está correta.

Para garantir a consistência dos dados durante operações críticas como transferências bancárias no PostgreSQL, é recomendável iniciar explicitamente uma transação antes de qualquer operação relacionada à transferência. Isso permite agrupar todas as operações em uma unidade de trabalho indivisível e confirmar (ou rollback, em caso de erro) a transação imediatamente após cada consulta SQL. Assim,

qualquer erro durante a transferência pode ser revertido de forma controlada, mantendo a integridade dos dados.

Prática de transações no PostgreSQL

O controle de transação é um dos principais recursos de um banco de dados relacional como o PostgreSQL, mantendo a integridade dos dados e a consistência do banco de dados mesmo em caso de falhas. Este laboratório prático abordará o uso de transações no PostgreSQL. Vamos lá!

No vídeo, a seguir, você poderá aprender e praticar os comandos SQL para a implementação de transações no PostgreSQL.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para realizar a atividade, siga os passos apresentados a seguir.

1. Abra o pgAdmin.
2. Acesse o banco tabelas_lab.
3. Crie uma tabela chamada tabtrans com duas colunas, uma inteira e outra varchar(10).
4. Abra uma transação.
5. Insira duas linhas na tabela.
6. Consulte a tabela.
7. Crie um ponto de salvamento A.
8. Insira mais 2 linhas.
9. Consulte a tabela.
10. Crie um ponto de salvamento B.
11. Altere duas linhas da tabela.
12. Consulte a tabela.

13. Crie novo ponto de salvamento C.

14. Elimine uma linha.

15. Consulte a tabela.

16. Retorne ao ponto de salvamento B.

17. Consulte a tabela.

18. Retorne ao ponto de salvamento C.

Conseguiu? Não?

19. Retorne ao ponto de salvamento A.

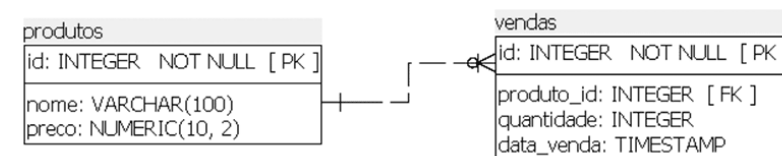
20. Consulte a tabela.

21. Confirme a transação.

22. Consulte a tabela.

Atividade 3

Para realizar a atividade, considere o modelo lógico a seguir.



Modelo lógico do banco de dados.

- Acesse o PostgreSQL utilizando o PGAdmin4.
- Crie um banco de dados chamado bdtransacao.
- Valide a criação do banco.
- Abra o query tool no banco criado.

- Crie as tabelas produtos e vendas conforme definidas no modelo anterior.
- Valide a criação das tabelas.
- Inicie explicitamente uma transação.
- Insira dois produtos na tabela produtos com os seguintes dados:

Linha 1: 'Produto A', 10.00

Linha 2: 'Produto B', 20.00

- Simule uma falha ao tentar inserir uma venda com um produto que não existe:

Linha: 999, 1.

A inserção falhará porque o produto_id 999 não existe na tabela produtos, mas continue! Vamos lá!

- Desfaça a transação.
- Verifique se as inserções anteriores foram revertidas.
- Inicie explicitamente uma nova transação.
- Insira dois produtos na tabela produtos com os seguintes dados:

Linha 1: 'Produto A', 10.00

Linha 2: 'Produto B', 20.00

- Verifique os IDs de produtos existentes.

Obs.: os IDs podem variar, pois sendo o ID serial ao fazer rollback, os valores já utilizados são perdidos.

- Insira uma venda corretamente:

Linha: 6,5

- Confirme a transação.
- Valide as inserções realizadas.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Saiba mais sobre atualização temporária e transação de leitura

Os conceitos de atualização temporária e de transação de leitura são importantes em bancos de dados relacionais, especialmente no contexto de operações complexas e simultâneas.

A atualização temporária refere-se à capacidade de modificar dados de forma transitória, sem persistência imediata no banco de dados principal, enquanto as transações de leitura permitem consultar informações consistentes, mesmo em ambientes de escrita intensiva.

Esses mecanismos são essenciais para garantir a integridade e a precisão dos dados, ao mesmo tempo que permitem flexibilidade na manipulação e consulta das informações armazenadas.

No vídeo, a seguir, você poderá aprofundar seu entendimento sobre a atualização temporária e a transação de leitura. Assista!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Atualização temporária

Sabemos que uma transação não deve atrapalhar o andamento de outra. Pense na execução da nossa última transação, que envolveu dois comandos de atualização na tabela DISCIPLINA.

Vimos que, logo após a execução da linha 3, a única disciplina que não sofreu alteração foi a de Computação Gráfica. Perceba que o SELECT da linha 3 está sendo executado no contexto da transação.

```
1 BEGIN;  
2 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.2 WHERE CARGAHORARIA=60;  
3 SELECT * FROM DISCIPLINA;  
4 SAVEPOINT CARGA60;  
5 UPDATE DISCIPLINA SET CARGAHORARIA=CARGAHORARIA*1.1 WHERE CARGAHORARIA=40;  
6 ROLLBACK TO CARGA60;  
7 SELECT * FROM DISCIPLINA;  
8 COMMIT;
```

O que aconteceria se tivéssemos, em paralelo, outra aplicação que submetesse consulta para acessar os registros da tabela DISCIPLINA no mesmo momento da execução da linha 3 da transação?

Resposta: A consulta em questão enxergaria os dados “originais”, sem quaisquer alterações. Por qual razão? Para não haver o problema da atualização temporária. Queremos dizer que, se a transação fosse desfeita por qualquer motivo, o UPDATE da linha 2 seria também desfeito.

Transação de leitura

Sabemos que uma transação que não modifica dados é denominada transação somente de leitura (READ ONLY). Caso contrário, é denominada leitura-gravação (READ WRITE).

Para especificar o tipo de transação, usaremos o comando SET TRANSACTION <TIPOTRANSAÇÃO>. No PostgreSQL, quando iniciamos uma transação, o padrão é READ WRITE.

Vamos analisar um exemplo envolvendo uma transação READ ONLY:

```
1 BEGIN;  
2 SET TRANSACTION READ ONLY;  
3 UPDATE DISCIPLINA SET CARGAHORARIA=80;  
4 ROLLBACK;
```

Na transação, propositalmente, inserimos um comando de atualização em uma transação que não permite essa categoria de comando. Logo, após a execução da linha 3, o SGBD retornará uma mensagem informando ao usuário que não é possível executar comando de atualização em uma transação do tipo somente leitura.

Atividade 4

No contexto de bancos de dados, transações garantem a execução completa ou a reversão de operações, assegurando consistência e integridade.

Ao projetar uma aplicação que realiza operações de leitura de dados no PostgreSQL, qual das seguintes práticas relacionadas ao uso de transações para leitura é a mais apropriada?

- A Utilizar uma transação explícita para cada consulta de leitura, garantindo que todos os dados sejam bloqueados durante a leitura.
- B Evitar o uso de transações para operações de leitura, pois transações são desnecessárias para consultas simples.
- C Iniciar uma transação automática implícita para cada consulta de leitura, permitindo que o PostgreSQL gerencie automaticamente o controle de transações.
- D Utilizar uma transação explícita para agrupar várias consultas de leitura relacionadas, permitindo a consistência de leitura em todo o conjunto de consultas.

E

Usar o modo de isolamento de leitura "READ UNCOMMITTED" para consultas de leitura, maximizando o desempenho ao permitir leituras sujas.



A alternativa D está correta.

Ao executar operações de leitura em uma aplicação PostgreSQL, o uso de uma transação explícita para agrupar consultas relacionadas ajuda a manter a consistência dos dados. Assim, todas as leituras ocorrem em um estado uniforme ao longo da execução das consultas. Embora transações explícitas possam ter um custo maior em termos de recursos, elas oferecem controle sobre o isolamento e a consistência dos dados consultados.

Prática de atualização temporária e transação de leitura

Veja agora como utilizar o controle de transação no PostgreSQL, incluindo como iniciar, confirmar, reverter transações e lidar com concorrência. O entendimento e o uso correto das transações são fundamentais para manter a integridade e consistência dos dados em aplicações reais.

No vídeo, a seguir, você poderá aprender e praticar a atualização temporária e a transação de leitura.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para realizar a atividade, siga os passos apresentados a seguir.

1. Abra o pgAdmin.
 2. Acesse o banco tabelas_lab com o query tools.
 3. Abra o PSQL.
 4. Faça conexão ao banco tabelas_lab.
- No pgAdmin:**
5. Inicie uma transação.
 6. Consulte a tabela tabtrans.
 7. Insira mais 2 linhas.
 8. Consulte a tabela.

No PSQL:

9. Inicie uma transação.

10. Consulte a tabela tabtrans.

11. Insira mais 2 linhas.

12. Consulte a tabela.

No pgAdmin:

13. Confirme a transação.

14. Consulte a tabela tabtrans.

No PSQL:

15. Consulte a tabela tabtrans.

16. Desfaça a transação.

17. Consulte a tabela.

No pgAdmin:

18. Inicie uma transação.

19. Consulte a tabela tabtrans.

20. Comande update em todas as linhas da tabela tabtrans.

21. Consulte a tabela tabtrans.

No PSQL:

22. Inicie uma transação.

23. Consulte a tabela tabtrans.

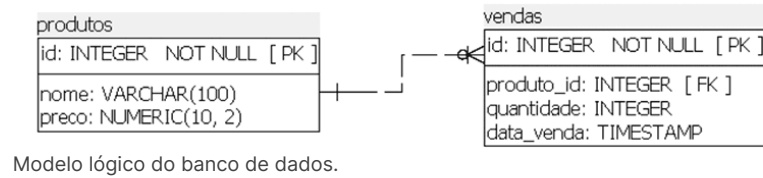
24. Comande update em todas as linhas da tabela tabtrans.

Observe o deadlock.

Atividade 5

O PostgreSQL usa MVCC (Multi-version concurrency control) para gerenciar transações concorrentes. Vejamos como isso funciona!

Para o exercício, vamos nos basear no modelo lógico apresentado a seguir.



Agora, siga os passos apresentados a seguir.

- Acesse o PostgreSQL utilizando o PGAdmin4.
- Abra o query tool no banco brtransacao.
- Acesse o PSQL.
- Faça conexão no bdtransacao no psl.

No pgadmin:

- Inicie uma transação.
- Atualize o preço do produto de nome 'Produto A' para 30.
- Valide a atualização do preço.

No PSQL:

- Consulte a tabela produtos.
- Inicie uma transação.
- Atualize o preço do produto de nome 'Produto B' para 60.
- Consulte a tabela produtos.

No pgAdmin:

- Consulte a tabela produtos.
- Confirme a transação.

- Consulte novamente a tabela produtos.

No PSQL:

- Consulte novamente a tabela produtos.
- Desfaça a alteração do produto B.
- Consulte novamente a tabela produtos.

No pgAdmin:

- Consulte novamente a tabela produtos.

Chave de resposta

Clique para conferir a [solução](#) da atividade.

Considerações finais

O que você aprendeu neste conteúdo?

- Arquitetura do PostgreSQL.
- Instalação do PostgreSQL no Linux e no Windows.
- Utilização de pgAdmin 4 e PSQL para interação com o PostgreSQL.
- Comandos SQL para criação de banco de dados e tabelas.
- Comandos SQL para criação e alteração de tabelas relacionadas.
- Comandos SQL para inserção, atualização e remoção de linhas em uma tabela.
- Comandos SQL para implantação de transações de banco de dados.

Podcast

Ouçá uma revisão do conteúdo, abordando aspectos como o histórico do PostgreSQL, as características e os comandos do sistema. Além disso, também é abordada a compatibilidade do Postgres com o padrão SQL. Não perca!



Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Não pare por aqui! Selecionamos uma indicação de leitura para você:

Early History of SQL, de Donald D. Chamberlin, publicado em IEEE Annals of the History of Computing, em 2012. Como vimos, a linguagem SQL tornou-se um padrão para uso em sistemas gerenciadores de bancos de dados relacionais. O artigo indicado é um interessante material sobre a história da SQL.

Referências

DBEAVER COMMUNITY. Consultado na internet em: 30 maio 2020.

ELMASRI, R.; NAVATHE, S. **Sistemas de banco de dados**. 7. ed. São Paulo: Pearson, 2019.

MANZANO, J. A. M. G. **Microsoft SQL Server 2016 Express Edition Interativo**. São Paulo: Saraiva, 2017.

POSTGRESQL. **Create Table**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Data Type**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Delete**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Download**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Insert**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **PostgreSQL 12.3 Documentation**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Ubuntu**. Consultado na internet em: 30 mai. 2020.

POSTGRESQL. **Update**. Consultado na internet em: 30 mai. 2020.