Banco de Dados 1

SQL/DDL+DML

Abra a Documentação do Postresql 9.5*, pois pode ser útil ao longo do roteiro.

*Há uma versão traduzida da documentação, porém esta é referente à versão 8.0 do PostgreSQL. Caso prefira ler em português, acesse: http://pgdocptbr.sourceforge.net/pg80/index.html

Neste roteiro, começaremos a trabalhar com a linguagem **SQL/DML** (Data Manipulation Language / Linguagem de Manipulação de Dados), que contém um conjunto de instruções para adicionar, modificar, consultar ou remover dados de um banco de dados.

Também continuaremos avançando nosso conhecimento em **SQL/DDL** (já visto no roteiro anterior). Hoje faremos algumas definições de dados mais elaboradas, envolvendo mais restrições de integridade.

Envio do roteiro: Você irá enviar um arquivo sql gerado automaticamente após concluir a elaboração do roteiro. As instruções para geração deste arquivo estão na última seção deste documento. Além disso, você vai precisar enviar um arquivo texto (TXT) com comentários (ambos através do formulário de submissão). Não precisa ir em busca destas informações agora… apenas continue lendo sequencialmente.

Tipos

Para o roteiro de HOJE, vocês devem usar apenas os seguintes tipos:

- Tipos Numéricos
  - Inteiros, Inteiros de Autoincremento, Ponto Flutuante, etc.
  - Veja aqui os tipos numéricos disponíveis no PostgreSQL;
- Tipos Textuais (Character)
  - CHAR(length) - cadeia de caracteres (string) de tamanho fixo. Exemplo: CHAR(5) para comprimento de exatamente 5 caracteres.
  - VARCHAR(length) - cadeia de caracteres (string) de tamanho variável. Exemplo: VARCHAR(20) para comprimento de até 20 caracteres.

    - TEXT - cadeia de caracteres com comprimento ilimitado;
    - Veja aqui os demais tipos textuais disponíveis no PostgreSQL;
- BOOLEAN - true / false;
- DATE - data. Exemplo '2017-12-31';
- TIMESTAMP - data/hora. Exemplo: '2017-12-31 14:05:06';

Inserção de dados (INSERT)

Hoje utilizaremos o comando de inserção (INSERT) para adicionar elementos (tuplas) às nossas tabelas.

Exemplos de uso deste comando:

```
-- Inserção de uma única linha, com valores para todas as colunas (na ordem padrão)

INSERT INTO nomeTabela VALUES
        (valorAtrib1, valorAtrib2, valorAtrib3, valorAtrib4);


-- Inserção de uma única linha, especificando as colunas

INSERT INTO nomeDaTabela (nomeAtrib1, nomeAtrib2, nomeAtrib3, nomeAtrib4)
        VALUES (valorAtrib1, valorAtrib2, valorAtrib3, valorAtrib4);


-- Inserção de múltiplas linhas

INSERT INTO nomeDaTabela (nomeAtrib1, nomeAtrib2, nomeAtrib3, nomeAtrib4)
VALUES
        (valorAtrib1, valorAtrib2, valorAtrib3, valorAtrib4),
        (valorAtrib1, valorAtrib2, valorAtrib3, valorAtrib4);
```

Se a inserção for executada com sucesso, será retornada uma mensagem da seguinte forma:

```
INSERT oid count
```

Onde *count* é a contagem do número de linhas inseridas/atualizadas.

Leitura opcional: se *count* for exatamente 1 (um), e a tabela de destino tiver OIDs, *oid* é o OID atribuído à linha inserida. Esta única linha deve ter sido inserida em vez de atualizada. Caso contrário, *oid* é zero. Os identificadores de objeto (OIDs) são usados internamente pelo PostgreSQL como chaves primárias para várias tabelas do sistema. Os OIDs não são adicionados às tabelas criadas pelo usuário, a menos que WITH OIDS seja especificado quando a tabela for criada ou a variável de configuração default_with_oids esteja ativada.

INSIRA AGORA alguns valores nas suas tabelas já criadas no Roteiro 1.

Consulta aos Dados do BD

Hoje utilizaremos os comandos de projeção e seleção (SELECT/FROM/WHERE) para verificarmos os elementos inseridos em nossas tabelas.

Exemplos de uso:

```
SELECT nomeAtrib1, nomeAtrib2 FROM nomeTabela WHERE condicaoDeSelecao;

SELECT nome, endereco FROM empregado WHERE salario > 1500;
```

CONSULTE AGORA alguns valores inseridos nas suas tabelas.

Remoção de Dados do BD

O comando DELETE é utilizado para remover dados de uma tabela do BD.

Exemplos de uso:

```
-- Remove todas as tuplas de uma tabela
DELETE FROM nomeTabela;

-- Remove tuplas que satisfazem alguma condição de seleção
DELETE FROM tasks WHERE status = 'DONE';
```

Se o comando for executado com sucesso, o sistema retornará uma mensagem no seguinte formato:

```
DELETE count
```

Onde *count* é a quantidade de linhas removidas da tabela.

Se você desejar os detalhes das linhas removidas, execute o comando DELETE da seguinte forma:

```
DELETE FROM tasks WHERE status = 'DONE' RETURNING *;
```

NÃO é preciso remover agora nenhum valor das suas tabelas do Roteiro 1. Utilizaremos este comando daqui a pouco.

Atualizando Dados do BD

O comando UPDATE é utilizado para **atualizar** dados de uma tabela do BD.

Exemplos de uso:

```
UPDATE nomeTabela SET nomeAtrib = valorAtrib WHERE condicao;

UPDATE funcionario SET telefone = '8888-9999' WHERE cpf = '12345678911';

UPDATE funcionario SET salario = salario+200, gratificacao = gratificacao+100  WHERE cidade_lotacao = 'Campina Grande' AND ultima_avaliacao = 'OTIMA';
```

Se o comando for executado com sucesso, o sistema retornará uma mensagem no seguinte formato:

```
UPDATE count
```

Onde *count* é a quantidade de linhas da tabela que foram acessadas para atualização. Ou seja, esta contagem inclui as linhas que satisfizeram a condição de seleção mas onde os valores não precisaram ser alterados (isto é, o valor atual já era o novo valor a ser registrado). Portanto, *count* é maior ou igual a quantidade de linhas efetivamente alteradas.

NÃO é preciso atualizar agora nenhum valor das suas tabelas do Roteiro 1. Utilizaremos este comando daqui a pouco.

ATENÇÃO: se o filtro (WHERE) não for aplicado corretamente no comando UPDATE, tuplas indesejadas serão modificadas. Aplique o filtro de forma que apenas a(s) tupla(s) alvo seja(m) modificada. Uma dica é antes executar um SELECT com o mesmo filtro e conferir que tupla(s) são selecionada(s).

Restrições de Checagem/Verificação (CHECK Constraint)

Utilizamos CHECKS para impor restrições relacionadas aos possíveis valores de nossas colunas.

Exemplos de uso:

```
CREATE TABLE produto (
        id_produto integer,
        preco numeric,
        CHECK (preco > 0)
);

CREATE TABLE produto (
        id_produto integer,
        preco numeric,
        CONSTRAINT preco_valido_chk CHECK (preco > 0)
);

ALTER TABLE produto ADD CONSTRAINT produto_chk_preco_valido CHECK (preco > 0);
```

ATENÇÃO! Deste ponto em diante, é possível que você encontre alguns erros ao executar seus comandos e precise pensar em soluções. Caso isso aconteça, anote os erros que julgar mais relevantes e as soluções que encontrou. Colecione as anotações em um arquivo de texto e salve-as para não perdê-las (você irá enviar esse arquivo texto através do formulário de submissão. Utilize os identificadores de questões (Questão 1, Questão 2, etc) e os identificadores de comandos contidos neste relatório (Comandos 1, Comandos 2, etc) para referenciá-los em seu arquivo e facilitar o entendimento do que foi escrito.

No final deste roteiro, há uma explicação de como você vai gerar automaticamente o código SQL que representa o estado do seu banco de dados (DUMP). Após gerado, você irá enviá-lo utilizando o formulário de submissão.

Criando o BD e Trabalhando com Restrições de Integridade

Agora criaremos o BD para manipularmos no roteiro de hoje.

ATENÇÃO! Só inicie uma questão quando concluir a anterior!!! A ordem dos comandos executados é muito relevante!!

QUESTÃO 1

A seguir estão alguns comandos INSERT que seu banco deve ser capaz de executar com sucesso e outros que a execução deve ser rejeitada. A partir dos comandos INSERT, tente inferir como devem ser estruturadas as tabelas do seu banco. Execute os comandos na sequência abaixo. Só passe para o comando seguinte quando o anterior for executado com sucesso!

```
-- As inserções abaixo devem ser executadas com sucesso

INSERT INTO tarefas VALUES (2147483646, 'limpar chão do corredor central',
'98765432111', 0, 'F');

INSERT INTO tarefas VALUES (2147483647, 'limpar janelas da sala 203', '98765432122',
1, 'F');

INSERT INTO tarefas VALUES (null, null, null, null, null);

-- As inserções abaixo NÃO devem ter a execução autorizada. Se alguma dessas linhas
for inserida, lembre-se de executar um comando DELETE apropriado para remover
exatamente esta linha.

INSERT INTO tarefas VALUES (2147483644, 'limpar chão do corredor superior',
'987654323211', 0, 'F');

INSERT INTO tarefas VALUES (2147483643, 'limpar chão do corredor superior',
'98765432321', 0, 'FF');
```

Comandos 1

Neste momento, você deve ter criado a primeira tabela do BD!

QUESTÃO 2

Agora tente inserir mais um elemento (executando o código abaixo) e, se encontrar um erro, tente ajustar sua tabela para que o erro seja eliminado e você **consiga** realizar a inserção normalmente. Procure uma solução em que não seja preciso fazer DROPs nem DELETEs.

```
INSERT INTO tarefas VALUES (2147483648, 'limpar portas do térreo', '32323232955', 4,
'A');
```

Comandos 2

QUESTÃO 3

Altere a penúltima coluna da sua tabela de forma a **não permitir** as inserções abaixo.

```
INSERT INTO tarefas VALUES (2147483649, 'limpar portas da entrada
principal','32322525199', 32768, 'A');
INSERT INTO tarefas VALUES (2147483650, 'limpar janelas da entrada principal',
'32333233288', 32769, 'A');
```

Comandos 3

Porém, ainda **deve ser possível** executar os INSERTs abaixo (execute-os).

```
INSERT INTO tarefas VALUES (2147483651, 'limpar portas do 1o andar','32323232911',
32767, 'A');
INSERT INTO tarefas VALUES (2147483652, 'limpar portas do 2o andar', '32323232911',
32766, 'A');
```

Comandos 4

QUESTÃO 4

Execute comandos ALTER TABLE / ALTER COLUMN de forma a **não permitir** valores NULL em nenhuma das colunas. Aproveite para ajustar o nome dos atributos para (id, descricao, func_resp_cpf, prioridade, status). Muito provavelmente os nomes destes atributos devem estar diferentes!

Você deve ter encontrado um ERRO ao adicionar as constraints acima, visto que há tuplas com valores NULL. Formule um comando SQL que remova **apenas esta(s)** tupla(s) e depois tente redefinir as constraints novamente.

QUESTÃO 5

Adicione uma restrição de integridade de forma a permitir a execução do primeiro INSERT mas não permitir a execução do segundo. Se o segundo for inserido, utilize o comando DELETE apropriado para remover a linha **exata** !!

```
-- inserção deve funcionar normalmente
INSERT INTO tarefas VALUES (2147483653, 'limpar portas do 1o andar','32323232911',
2, 'A');

-- inserção não deve funcionar após o insert anterior ter sido executado
INSERT INTO tarefas VALUES (2147483653, 'aparar a grama da área frontal',
'32323232911', 3, 'A');
```

Comandos 5

QUESTÃO 6

6.A)

Na tabela tarefas, a coluna func_resp_cpf deve ter exatamente 11 caracteres. Adicione uma constraint para assegurar isso. Use insert com mais e menos caracteres para testar sua constraint.

6.B)

A última coluna desta tabela representa o status da tarefa. Os possíveis status são:

- Aguardando - 'A'
- Realizando - 'R'
- Finalizada - 'F'

Porém, foi decidido que agora os possíveis status serão:

- Planejada - 'P'
- Executando - 'E'
- Concluída - 'C'

Para evitar erros durante as inserções, adicione uma restrição de integridade usando o comando CHECK.

Note que será preciso executar alguns comandos UPDATE antes de adicionar esta restrição.

ATENÇÃO! REPETINDO uma observação anterior: se o filtro (WHERE) não for aplicado corretamente no comando UPDATE, tuplas indesejadas serão modificadas. Aplique o filtro de forma que apenas a(s) tupla(s) alvo seja(m) modificada. Uma **dica** é antes executar um SELECT com o mesmo filtro e conferir que tupla(s) são selecionada(s).

QUESTÃO 7

Adicione outra constraint que restrinja os possíveis valores da coluna prioridade para 0,1,2,3,4,5.

AJUSTES: para os INSERTS que foram executados anteriormente (e corretamente), mas que contêm valores de prioridade maiores do que 5, você deve ajustar o valor deste atributo para 5 nas tuplas correspondentes (use o comando UPDATE).

QUESTÃO 8

Crie a relação:

funcionario (cpf, data_nasc, nome, funcao, nivel, superior_cpf), onde:

- superior_cpf representa o cpf do funcionário supervisor (deve ser chave estrangeira);
- cpf é a chave primária;
- funcao: 'LIMPEZA' ou 'SUP_LIMPEZA' (se for 'LIMPEZA' tem que ter um superior)
- demais atributos são todos NOT NULL;
- nivel: 'J', 'P' ou 'S' (Júnior, Pleno, Senior)

Adicione constraints para assegurar as regras acima.

Verifique as inserções a seguir:

```
-- devem funcionar:
INSERT INTO funcionario (cpf, data_nasc, nome, funcao, nivel, superior_cpf) VALUES
('12345678911', '1980-05-07', 'Pedro da Silva', 'SUP_LIMPEZA', 'S', null);
INSERT INTO funcionario(cpf, data_nasc, nome, funcao, nivel, superior_cpf) VALUES
('12345678912', '1980-03-08', 'Jose da Silva', 'LIMPEZA', 'J', '12345678911');

-- não deve funcionar
INSERT INTO funcionario(cpf, data_nasc, nome, funcao, nivel, superior_cpf) VALUES
('12345678913', '1980-04-09', 'joao da Silva', 'LIMPEZA', 'J', null);
```

Comandos 6

QUESTÃO 9

- Crie 10 exemplos de inserções diferentes que **são** executadas com **sucesso** na relação funcionario.
    - ATENÇÃO: defina valores de id maiores do que os já existentes!
- Crie 10 exemplos de inserções que **não** são executadas na relação funcionário, por serem **bloqueadas** pelas constraints impostas na questão anterior.

Para os comandos SQL acima, procure elaborar casos de teste diferentes para verificar se as constraints estão verificando corretamente as regras que foram definidas.

Os INSERTs que executam com sucesso são incluídos no DUMP gerado automaticamente no final do roteiro. Porém, como os INSERTs que falham não são incluídos, é importante incluir ao menos eles no arquivo com os comentários.

QUESTÃO 10

Observe que precisamos criar uma chave estrangeira entre as relações.

Experimente criar a chave estrangeira usando as **OPÇÕES** indicadas abaixo (nesta ordem).

O que fazer SE ENCONTRAR ERROS ao criar a chave estrangeira?
Se necessário, adicione novos elementos na tabela funcionario para evitar erros. **Não remova** tuplas da tabela tarefas !!

OPÇÃO 1: ON DELETE CASCADE
Após adicionar a constraint, execute um comando DELETE na tabela funcionario que remova o funcionário com “menor” cpf que possui alguma tarefa (como há poucos dados na tabela, você pode identificar este cpf manualmente).

OPÇÃO 2: ON DELETE RESTRICT
Após adicionar a constraint, execute um comando DELETE que seja bloqueado pela constraint. Qual erro você encontrou?

QUESTÃO 11

Vamos agora adicionar nossa última constraint:

Defina a seguinte constraint na tabela tarefas:

> func_resp_cpf integer pode ser NULL, mas somente quando o status for 'P'

Ou seja, uma tarefa pode não ter um responsável quando ela só está planejada, o que parece fazer sentido no mundo real. Mas quando está em execução ou concluída, deve haver o registro do responsável.

Agora altere a constraint criada na Questão 10 para funcionar com a opção:

> ON DELETE SET NULL

Se o atributo func_resp_cpf tiver sido declarado inicialmente como NOT NULL, você deve remover esta constraint.

Para testar, tente remover um funcionário que está associado a tarefas com os 3 diferentes status e verifique se sua constraint funciona de forma adequada. Que mensagem o SGBD retorna quando o status é 'C' ou 'E' ?
Observe que, se você conseguir remover um funcionário indevidamente durante a execução dos testes, você precisará inseri-lo novamente antes de testar novamente.

Envio do Roteiro

O que será enviado:

- Código SQL gerado automaticamente (upload de arquivo SQL);
    - Instruções mais abaixo.
- Comentários (upload de arquivo TXT).

Seu arquivo (DUMP) ficará com com esse aspecto:

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 9.5.19
-- Dumped by pg_dump version 9.5.19

SET statement_timeout = 0;
SET ...

...

ALTER TABLE ...

--
-- Name: funcionario; Type: TABLE; Schema: public; Owner: ...
--

CREATE TABLE public.funcionario (
...
);

...

INSERT INTO (...);

...
```

Aspecto do arquivo .sql para submissão

roteiro2-dump-matricula.sql

Para gerar esse arquivo, siga os passos a seguir:

- Acesse a VM com seu usuário. Se estiver usando psql basta executar \q para voltar ao prompt.
- Execute o comando abaixo, substituindo os itens grifados:

$ pg_dump seuUsuario_db -c --column-inserts -t funcionario -t tarefas > roteiro2-dump-matricula.sql

Esse comando vai gerar o arquivo roteiro2-dump-matricula.sql no diretório atual. Provavelmente será /home/seuUsuario. Use o comando pwd para conferir.

- Desconecte da VM com o comando exit.
- Acesse o diretório na sua máquina local onde deseja salvar o arquivo (usando o comando cd).
- Copie o arquivo gerado na VM para sua máquina:

scp -P 45600 seuUsuario@150.165.15.11:/home/seuUsuario/roteiro2-dump-matricula.sql .

O ponto “.” no final faz com que o arquivo seja copiado no diretório atual.