COMPUTAÇÃO UFCG

Banco de Dados 1

SQL/DDL+DML

Neste roteiro, continuaremos trabalhando 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 nos roteiros anteriores).

Envio do roteiro: Você irá enviar um arquivo sql gerado automaticamente após concluir a elaboração do roteiro. Além disso, você vai precisar editar esse arquivo gerado automaticamente para incluir alguns *comandos SQL adicionais*. As instruções para geração deste arquivo estão na última seção deste documento.

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’;

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

Limpando o Banco de Dados

Para fazer o roteiro de hoje, você deve primeiramente redefinir o esquema do seu banco de dados (remover todas as tabelas, constraints, dados, etc).

Se você quiser manter o backup do seu banco, ou de alguma tabela, veja as instruções no final do Roteiro 2 (comando pg_dump). No exemplo do Roteiro 2, note que o parâmetro -t restringe o backup a tabelas específicas. Não use este parâmetro caso pretenda fazer backup do banco inteiro.
Execute o seguinte comando SQL para remover todos os elementos do banco cujo dono (owner) é o usuário atual:

```
DROP OWNED BY CURRENT_USER;
```

Novos tipos de constraints (não usados nos roteiros anteriores)

Nos roteiros anteriores, vimos como especificar constraints com DEFINIÇÃO DE TIPOS, NOT NULL, PRIMARY KEY, FOREIGN KEY e CHECK. Para criação de algumas constraints deste roteiro, recomenda-se consultar o Manual do PostgreSQL sobre a utilização de UNIQUE e EXCLUDE.

UNIQUE:

O comando UNIQUE determina que dois valores de uma certa coluna não podem ser iguais. Por exemplo, se duas pessoas não puderem ter o mesmo nome, fazemos:

```
CREATE TABLE pessoa (
        cpf integer,
        nome varchar(50) UNIQUE,
        data_nasc date
);

CREATE TABLE pessoa (
        cpf integer,
        nome varchar(50),
        data_nasc date,
        UNIQUE (nome)
);

CREATE TABLE example (
```

```
        a integer,
        b integer,
        c integer,
        UNIQUE (a, c)
);
```

Também podemos definir um campo como UNIQUE utilizando a cláusula CONSTRAINT (quando queremos dar um nome à constraint), como fizemos antes para outros tipos de constraints.

Além disso, podemos definir esta constraint utilizando ALTER TABLE, como também fizemos antes para outros tipos de constraints.

Note que este tipo de restrição faz com que toda a tabela seja consultada em busca de duplicidade, reduzindo o desempenho dos comandos INSERT e UPDATE. Para melhorar o desempenho, o SGBD cria automaticamente um índice para este(s) campos (assim como acontece nos campos de chave primária).

EXCLUDE:

Note que o comando UNIQUE impõe uma restrição comparando diferentes linhas da tabela em relação a uma mesma coluna (ou conjunto de colunas).

Porém, há casos em que queremos aplicar restrições a linhas diferentes da tabela (em relação a uma mesma coluna ou conjunto de colunas) onde o UNIQUE não é suficiente. Por exemplo:

- não permitir nomes iguais (em uma relação “pessoa”), exceto se as idades forem diferentes.
- não permitir que existam dois aluguéis para um mesmo veículo (mesmo id de veículo em uma tabela aluguel) se os períodos de locação tiverem interseção entre si.

O comando EXCLUDE segue a seguinte sintaxe (não se preocupe com o trecho “USING gist”, pois será explicado mais abaixo):

```
-- verifica o id_veiculo com o operador “=” e o período de locação (do tipo intervalo de tempo) com o operador de interseção “&&”.
ALTER TABLE locacao
ADD CONSTRAINT locacao_excl
EXCLUDE USING gist (
```

```
    id_veiculo WITH =,
    periodo WITH &&
);
```

Podemos dizer que comando **EXCLUDE é uma generalização do comando UNIQUE**. Assim, podemos representar uma restrição UNIQUE usando EXCLUDE.

Observe que a constraint abaixo significa o **mesmo que uma restrição UNIQUE** na coluna id_veiculo:

```
-- verifica o id_veiculo com o operador "="
ALTER TABLE locacao
ADD CONSTRAINT locacao_veic_excl
EXCLUDE USING gist (
    id_veiculo WITH =);
```

Também podemos criar Restrições de Exclusão contendo a cláusula **WHERE**. Exemplos:

- Não permitir valores repetidos para a coluna C quando o valor de C for "XX". Por exemplo, não permitir mais de uma pessoa com nome "José" na base de dados (permitindo repetições para outros nomes).
- Uma coluna C não pode ter valores repetidos, e menos que o valor de C seja diferente de "XX". Por exemplo, só permitir nomes repetidos se o nome for "Paulo" (não permitir repetição em nenhum outro caso).

Exemplo de EXCLUDE com WHERE:

```
-- verifica duplicações na coluna "nome" apenas nos casos
-- definidos na cláusula WHERE (apenas quando o valor de nome for "XX")
ADD CONSTRAINT unico_se_valor
EXCLUDE USING gist (
     nome with =) WHERE (nome = 'XX');
```

INDEXAÇÃO

A cláusula "**USING gist**" se refere ao tipo de índice utilizado. No SGBD PostgreSQL, para permitir o uso de EXCLUDE com tipos escalares, é necessário instalar a extensão btree_gist para cada banco criado. Esta extensão já foi criada em cada um dos bancos dos alunos que estão sendo utilizados na disciplina.

OUTROS MATERIAIS interessantes sobre o assunto:

https://www.tutorialspoint.com/postgresql/postgresql_constraints.htm

https://www.citusdata.com/blog/2018/03/19/postgres-database-constraints/

https://pt.slideshare.net/pgconf/not-just-unique-exclusion-constraints
(slides do Jeff Davis, que foi quem implementou o EXCLUDE no PostgreSQL).

Criando seu banco

Você deve criar um banco de dados para representar minimamente o funcionamento de uma **rede de farmácias**.

O seu banco deve ser capaz de armazenar dados sobre:

- Farmácias;
- Funcionários;
- Medicamentos;
- Vendas;
- Entregas;
- Clientes;

Está **fora** do escopo:

1. Armazenar dados relacionados à gestão de recursos humanos (salário, admissão, demissão, etc).
2. Controlar o estoque dos medicamentos;
3. Não é preciso armazenar detalhes da entrega (data, hora, entregador, etc);

Simplificações do escopo:

1. Em relação ao endereço das farmácias, basta armazenar bairro, cidade, estado;
2. O identificador de farmácias pode ser um número inteiro (não precisa usar cnpj, por exemplo);
3. Outros itens podem ser adicionados aqui se negociados com o “cliente”

O banco de dados deve atender aos **requisitos** (mínimos) abaixo.

1. Uma farmácia pode ser sede ou filial.
2. Funcionários podem ser farmacêuticos, vendedores, entregadores, caixas ou administradores (e não se pretende criar outros tipos de funcionários).

3. Cada funcionário está lotado em uma única farmácia.
4. Cada farmácia tem um (e apenas um) gerente (que é um funcionário);
5. Funcionários podem não estar lotados em nenhuma farmácia;
6. Clientes podem ter mais de um endereço cadastrado.
7. Os endereços do cliente podem ser residência, trabalho ou “outro”.
8. Medicamentos podem ter uma característica: venda exclusiva com receita.
9. Entregas são realizadas apenas para clientes cadastrados (em algum endereço válido do cliente);
10. Outras vendas podem ser realizadas para qualquer cliente (cadastrados ou não).
11. Não deve ser possível excluir um funcionário que esteja vinculado a alguma venda.
12. Não deve ser possível excluir um medicamento que esteja vinculado a alguma venda.
13. Clientes cadastrados devem ser maiores de 18 anos;
    Dica: veja as funções de data/tempo na documentação do PostgreSQL (Table 9-28. Date/Time Functions)
14. Só pode haver uma única farmácia por bairro;
    Tente pensar sozinho em uma solução. Caso não consiga, veja a dica no final do documento.
15. Há apenas uma sede.
    Tente pensar sozinho em uma solução. Caso não consiga, veja a dica no final do documento.
16. Gerentes podem ser apenas administradores ou farmacêuticos.
    Tente pensar sozinho em uma solução. Caso não consiga, veja a dica no final do documento.
17. Medicamentos com venda exclusiva por receita só devem ser vendidos a clientes cadastrados.
    Dica: caso parecido com outra constraint acima.
18. Uma venda deve ser feita por um funcionário vendedor.
    Dica: caso parecido com outra constraint acima.
19. Defina uma coluna na tabela farmácia para representar o estado onde está localizada e adicione um mecanismo para restringir os possíveis valores a serem inseridos nesta coluna: um dos 9 estados do nordeste. Não use CHECK para especificar esta constraint.
    Dica: Ao invés de CHECK, utilize um ENUMERATED TYPE. Veja a Documentação do PostgreSQL sobre o assunto.

Envio do Roteiro

Você deve enviar **um único arquivo (.sql)** através do formulário de submissão de roteiros. Monte seu arquivo da seguinte forma:

1. **Gere um Arquivo SQL** com o comando pg_dump (instruções abaixo). Esse arquivo deverá conter os comandos de criação das tabelas, das constraints e de povoamento do banco.

2. **Concatene (no final do arquivo) comandos adicionais** do tipo INSERT, DELETE ou UPDATE, mostrando que seu banco atende (ou não) aos requisitos acima. Ou

seja, cada comando deve ser executado com sucesso (se não houver impedimentos impostos por constraints) ou deve ser rejeitado / retornar erro (se violar alguma constraint). Neste caso, adicione um comentário acima do comando informando o resultado esperado.

Exemplos (incluindo os comentários)

```
--
-- COMANDOS ADICIONAIS
--

-- deve ser executado com sucesso
-- INSERT INTO farmacia (....) VALUES (...);

-- deve retornar erro por violar a constraint xxxx, uma vez que….
-- INSERT INTO farmacia (....) VALUES (...);
```

ATENÇÃO: estes comandos serão executados logo após a restauração do backup contido no arquivo SQL. Ou seja, assegure que os dados utilizados para povoar o banco são adequados para fazer estas verificações logo em seguida.

SQL:

roteiro3-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 > roteiro3-dump-matricula.sql

Esse comando vai gerar o arquivo roteiro3-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/roteiro3-dump-matricula.sql .

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

Dicas

Só leia as dicas abaixo se realmente tiver pensado/tentado mas não tiver conseguido implementar uma solução.

Só pode haver uma única farmácia por bairro.
Dica: usar constraint do tipo UNIQUE.

Há apenas uma sede.
Dica: usar EXCLUDE (que é uma generalização do UNIQUE).

Gerentes podem ser apenas administradores ou farmacêuticos.
Dica: criar uma coluna redundante na tabela farmacia e especificar a constraint usando o foreign key e check