[MAC0439] Laboratório de Bancos de Dados
Aula 10
Linguagem SQL:
Comandos para a Definição de Esquemas

Kelly Rosa Braghetto

DCC–IME–USP

18 de setembro de 2015

Como se conectar ao seu BD no PostgreSQL da rede Linux

- Via interface *web*:
  - Acesse a página `https://linux.ime.usp.br/phppgadmin/`
  - Para a conexão, use o seu login e senha da rede Linux

- Via linha de comando:
  1. Se conecte via SSH na rede Linux (abra um terminal e digite o comando a seguir):

     ```
     ssh [usuario]@shell.linux.ime.usp.br
     ```

  2. A partir de um máquina da rede Linux, execute o programa `psql` como mostrado abaixo:

     ```
     psql -h dionisio
     ```

Sobre o PgAdmin e o psql

- Esses programas são clientes de conexão com o PostgreSQL
- O psql funciona em modo texto, por linha de comando. Ele aceita comandos da linguagem SQL e alguns comandos que são específicos para PostgreSQL
- O PGAdmin é uma ferramenta gráfica para a administração de BDs mantidos em um servidor PostgreSQL
  - http://www.pgadmin.org/
  - Oferece interface gráfica para as operações feitas por meio da "porção" DDL (*data definition language* da SQL
  - Pode ser instalado localmente em uma máquina
  - Tem uma versão web (o phpPgAdmin), que é a que será usada nas aulas de MAC439 junto com o psql

O programa psql

Um modo mais geral do comando `psql` para estabelecer conexões com BDs é o seguinte:

```
psql -h [HOST] -p [PORTA] -U [NOME USUÁRIO] -d [NOME BD]
```

sendo que

- `[HOST]`: nome ou endereço IP da máquina na qual o BD está hospedado
- `[PORTA]`: porta do host na qual o PostgreSQL está "ouvindo" requisições (porta padrão: 5432)
- `[NOME USUÁRIO]`: nome de um usuário que tenha permissão de acesso ao BD (usuário padrão: login de usuário atualmente conectado à maquina a partir da qual o psql está sendo executado
- `[NOME BD]`: nome do BD ao qual se deseja conectar (nome de BD padrão: o nome de usuário passado para a conexão)

Comandos Úteis do PostgreSQL no `psql`

- Para listar todos os BDs:
  `\l`
- Para listar as tabelas do BD atual:
  `\dt` **ou** `SELECT * FROM pg_catalog.pg_tables`
- Para listar o esquema de uma tabela:
  `\d+ nome_tabela`
- Para ver outras opções de comandos especiais do `psql`:
  `\?`

Obs.: No `psql`, todo comando deve terminar com um ponto-e-vírgula (';')

SQL – *Structured Query Language*

$\Rightarrow$ Foi criada em 1976

Possui comandos para:

- Consultas ao BD (com recursos muito parecidos aos da Álgebra Relacional)
- Modificação do BD (inserção, remoção, alteração)
- Definição do esquema do BD

"Dialetos" da SQL

A SQL tem vários padrões:

- **ANSI** [1] **SQL** (1986)
- SQL-89 – inclusão de restrições de integridade
- **SQL-92** (ou SQL2) – grande atualização da versão anterior
- **SQL-99** (inicialmente chamada de SQL3) – inclusão de expressões regulares, consultas recursivas, gatilhos, comandos para controle de fluxo, funcionalidades relacionadas à orientação a objetos, etc.
- SQL-2003, SQL-2006, SQL-2008 – inclusão de funcionalidades relacionadas a XML (entre outras coisas)

Os SGBDs geralmente implementam a ANSI SQL e partes da SQL-92 e SQL-99, além de suas próprias extensões.

[1] ANSI – *American National Standards Institute*

Criação de *esquemas*

CREATE SCHEMA *nome_esquema*
[**AUTHORIZATION** '*nome_proprietario*'];

- *Esquemas* são criados dentro de um BD
- Um *esquema* funciona como um *namespace*: ele agrupa objetos do BD (tabelas, visões, tipos de dados, funções, etc.) que pertencem a uma mesma aplicação
- É possível ter dois ou mais objetos com um mesmo nome em um BD contanto que cada um esteja em um esquema diferente
- É possível definir/acessar objetos em um esquema qualificando o nome dos objetos. Por exemplo, o comando
  `CREATE TABLE EMPRESA.FUNCIONARIO...`
  cria a tabela `FUNCIONARIO` no esquema `EMPRESA`

Mais comandos para *esquemas*

- **SHOW** search_path;
  Mostra o esquema atualmente em uso.
- **SET search_path TO** *nome_esquema*;
  Define o esquema atualmente em uso.
- **DROP SCHEMA** *nome_esquema* [**CASCADE**];
  Remove um esquema.
  Com a opção CASCADE, os objetos definidos dentro do esquema também serão removidos.
- **ALTER SCHEMA** nome **RENAME TO** novo_nome;
  ALTER SCHEMA nome **OWNER TO** novo_proprietario;
  Altera as informações de um esquema.
- No PostgreSQL, o esquema padrão é o `public`

Declarações simples de tabelas

CREATE TABLE nome_tabela
[({nome_coluna tipo_dados [{restricao_coluna}]}
[{restricao_tabela }]
)]

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Nome              VARCHAR(50) UNIQUE,
    Cpf               CHAR(11),
    Sexo              CHAR(1) NOT NULL DEFAULT 'F',
    Data_nascimento DATE,
    PRIMARY KEY (Nome)
);
```

Tipos de dados de atributos

Números inteiros

- **INTEGER** ou **INT** (no PostgreSQL, ocupa 4 bytes)
- **SMALLINT** – ocupa geralmente a metade da quantidade de bytes usada por um INTEGER (no PostgreSQL, ocupa 2 bytes)

Números reais

- **FLOAT** ou **REAL** (no PostgreSQL, ocupa 4 bytes)
- **DOUBLE PRECISION** (no PostgreSQL,ocupa 8 bytes)
- **DECIMAL**(i,j) ou **DEC**(i,j) ou **NUMERIC**(i,j) – onde i é a *precisão* e indica o total de dígitos decimais, e j é a *escala* e indica o número de dígitos após o ponto decimal.
  $\Rightarrow$ O valor padrão para j é 0, mas para i depende do SGBD

Tipos de dados de atributos

Caracteres

- **CHARACTER(*n*)** ou **CHAR(*n*)** – onde *n* é o número de caracteres, que define o **tamanho fixo** da cadeia
- **CHAR VARYING** ou **VARCHAR(*n*)** – onde *n* é o número **máximo** de caracteres da cadeia
- **CHARACTER LARGE OBJECT** ou **CLOB** – para grandes cadeias de caracteres de tamanho variável (como documentos). (No PostgreSQL, esse tipo de dado chama-se **TEXT**)

Na SQL padrão, o tamanho padrão para *n* é 1. Mas no PostgreSQL, se não especificarmos o valor de *n* para um atributo do tipo VARCHAR, então ele terá um tamanho ilimitado.

Tipos de dados de atributos

Caracteres – considerações importantes:

- Cadeias de caracteres literais devem ser delimitadas por aspas simples (apóstrofos), como em `'MAC0439'`.
- Caracteres em SQL são ***case sensitive***. Portanto, `'MAC439'` $\neq$ `'mac439'`.
  Mas as palavras reservadas da SQL são *case insensitive*, ou seja, podemos usar `CREATE TABLE` ou `cREAte TAblE` de forma indistinta.

Tipos de dados de atributos

Caracteres – considerações importantes:

- Se um atributo é declarado como CHAR(10), em toda tupla o valor para esse atributo será uma cadeia de 10 caracteres. Portanto, se atribuirmos o literal `'MAC0439'`, o valor que será armazenado será o `'MAC0439 '` (ou seja, serão acrescidos 3 espaços em branco no final da cadeia de caracteres).
- Geralmente, os espaços em branco no final da cadeia são desconsiderados quando dois atributos do tipo CHAR(n) são comparados, ou quando um atributo desse tipo é convertido para um outro tipo de cadeia de caracteres.
- Já na comparação de atributos do tipo VARCHAR(n), os espaços em branco no final da cadeia são sim considerados.

Tipos de dados de atributos

Datas e horários

- **DATE** – exemplo: '2004-10-23' (formato que é sempre válido: YYYY-MM-DD)
- **TIME** – exemplo: '22:45:17' (formato HH:MM:SS)
- **TIMESTAMP** – incluem os campos DATE e TIME e mais posições para frações decimais de segundos. Exemplo: '2014-08-20 15:43:34.827022'

Os tipos TIME e TIMESTAMP podem ter também um qualificador WITH TIME ZONE.
Ex. de valor para um atributo do tipo TIMESTAMP WITH TIME ZONE: '2014-08-20 15:43:34.827022-03'

Tipos de dados de atributos

Datas e horários (continuação)

- **INTERVAL** – especifica um valor usado para incrementar ou decrementar o valor absoluto de uma data, hora ou *timestamp*. Um intervalo é qualificado para ser YEAR-MONTH, DAY-TIME ou uma mistura dos dois. Exemplos:
  - INTERVAL '1-2' – intervalo de 1 ano e 2 meses
  - INTERVAL '3 4:05:06' – intervalo de 3 dias, 4 horas, 5 minutos e 6 segundos
  - INTERVAL '1-2 3 4:05:06' – os dois acima juntos

Podemos considerar que os tipos para data e hora em SQL são essencialmente cadeias de caracteres com um formato especial. No PostgreSQL, a função `now()` fornece a data e hora atual do sistema.

Tipos de dados de atributos

Booleano

- **BOOLEAN** – admite os valores **TRUE**, **FALSE** ou **UNKNOWN** (para o espanto do George Boole!)

Observações:

- O valor UNKNOWN pode resultar de operações de comparação envolvendo o valor NULL; veremos detalhes disso em aulas futuras.

Tipos de dados de atributos

Cadeia de bits

- **BIT(*n*)** – cadeia de bits de tamanho fixo *n*
- **BIT VARYING(*n*)** – cadeia de bits com tamanho máximo *n*
- **BINARY LARGE OBJECT** ou **BLOB** – para grandes cadeias de bits de tamanho variável (como imagens)

No PostgreSQL:

- O único tipo de dados para bits implementado é o **BYTEA**, que equivale ao BLOB do SQL padrão.

Tipos de dados no PostgreSQL

Para mais informações sobre os tipos de dados no PostgreSQL 9.4:
`http://www.postgresql.org/docs/9.4/static/datatype.html`

Criação de domínios

É possível declarar um novo domínio e usar o seu nome como especificação para um atributo.

Exemplo:

```
CREATE DOMAIN TIPO_CPF AS CHAR(11);

CREATE TABLE FUNCIONARIO (
    Nome        VARCHAR(50),
    Cpf         TIPO_CPF,
    Casado      BOOLEAN,
    Salario     DECIMAL(10,2),
    ...
 );
```

Restrições e valores padrão para colunas

Restrição contra valores nulos – cláusula **NOT NULL**

- Define que uma coluna não pode receber o valor NULL
- É especificada de forma ímplicita para colunas que fazem parte da chave primária da tabela

Valor padrão – cláusula **DEFAULT**

- Define um valor que será atribuído à coluna em uma nova tupla sempre que o valor para essa coluna não for fornecido
- Se uma coluna não possuir a restrição de NOT NULL e nenhum valor padrão for definido para ela, então o valor NULL será usado como padrão

Restrições para colunas

Restrição de verificação – cláusula **CHECK**

- Restringe os valores que uma coluna pode assumir

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Nome            VARCHAR(50) NOT NULL,
    Cpf             TIPO_CPF    NOT NULL,
    Salario         DECIMAL(10,2) NOT NULL
                     CHECK (Salario > 650 AND Salario < 50000),
    Idade           INT CHECK (Idade >= 18 AND Idade <= 120),
    Casado          BOOLEAN NOT NULL DEFAULT FALSE,
    Cpf_supervisor  TIPO_CPF DEFAULT '12345678901'
 );
```

Restrições para tabelas

Restrição de verificação – cláusula **CHECK**

- Restringe os valores que um ou mais atributos da tabela podem assumir

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Nome          VARCHAR(50) NOT NULL,
   Cpf           TIPO_CPF    NOT NULL,
   Salario       DECIMAL(10,2) NOT NULL
                  CHECK (Salario > 650 AND Salario < 50000),
   Idade         INT CHECK (Idade >= 18 AND Idade <= 120),
   CHECK (idade < 65 OR salario > 10000)
 );
```

O último `CHECK` implementa a restrição de que todo funcionário com 65 ou mais anos deve receber um salário maior que 10000.

Restrições de chave

Restrição de chave primária – cláusula **PRIMARY KEY**

- Especifica uma ou mais colunas que compõem a chave primárida da tabela
- Se a chave primária tiver uma única coluna, a cláusula pode aparecer como uma *restrição de coluna* na definição da tabela
- Para chaves com uma ou mais colunas, usa-se uma cláusula de *restrição de tabela*

Lembrete: sobre uma chave primária, sempre é imposta uma restrição de NOT NULL.

Restrições de chave

Exemplo:

```
CREATE TABLE DEPARTAMENTO (
   Dnumero     INT PRIMARY KEY,
   Dnome       VARCHAR(30) NOT NULL
 );

CREATE TABLE LOCALIZACAO_DEP (
   Dnumero     INT NOT NULL,
   Dlocal      VARCHAR(15) NOT NULL,
   PRIMARY KEY(Dnumero,Dlocal),
 );
```

Restrições de chave

Restrição de chave secundária – cláusula **UNIQUE**

- Especifica uma ou mais colunas que compõem uma chave secundária (= alternativa) da tabela
- Se a chave secundária tiver uma única coluna, a cláusula pode aparecer como uma *restrição de coluna* na definição da tabela
- Para chaves secundárias com uma ou mais colunas, usa-se uma cláusula de *restrição de tabela*
- Diferentemente do que ocorre com a chave primária, uma coluna da chave secundária pode receber valores NULL

Restrições de chave

Exemplo:

```
CREATE TABLE DEPARTAMENTO (
    Dnumero     INT PRIMARY KEY,
    Dnome       VARCHAR(30) NOT NULL UNIQUE
 );
```

ou

```
CREATE TABLE DEPARTAMENTO (
    Dnumero     INT,
    Dnome       VARCHAR(30) NOT NULL,
    PRIMARY KEY(DNumero),
    UNIQUE(Dnome)
 );
```

Restrições de integridade referencial

Cláusula **FOREIGN KEY**

- Especifica uma chave estrangeira
- Oferece diferentes opções de ações para o tratamento das violações de integridade referencial causadas por operações de inserção, remoção e alteração:
  - opção **RESTRICT** (padrão) – rejeita a operação de atualização que causará uma violação
  - opção **SET NULL** – atribuirá NULL à chave estrangeira que ficar sem sua "referência"
  - opção **SET DEFAULT** – atribuirá um valor padrão à chave estrangeira que ficar sem sua "referência"
  - opção **CASCADE** – propaga a alteração feita na chave referenciada para as linhas que a referenciam
- As ações acima devem ser qualificadas com as cláusulas **ON DELETE** ou **ON UPDATE**

Restrições de integridade referencial

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Nome          VARCHAR(50) NOT NULL,
   Cpf           CHAR(11) PRIMARY KEY,
   Salario       DECIMAL(10,2)            );

CREATE TABLE DEPARTAMENTO (
   Dnumero         INT PRIMARY KEY,
   Dnome           VARCHAR(30) NOT NULL UNIQUE,
   Cpf_gerente     CHAR(11) NOT NULL DEFAULT '12345678901',
   FOREIGN KEY(Cpf_gerente) REFERENCES FUNCIONARIO(Cpf)
      ON DELETE SET DEFAULT  ON UPDATE CASCADE            );

CREATE TABLE LOCALIZACAO_DEP (
   Dnumero     INT NOT NULL,
   Dlocal      VARCHAR(15) NOT NULL,
   PRIMARY KEY(Dnumero,Dlocal),
   FOREIGN KEY(Dnumero) REFERENCES DEPARTAMENTO(Dnumero)
      ON DELETE CASCADE  ON UPDATE CASCADE                );
```

Nomeando restrições

Com a cláusula **CONSTRAINT**, é possível atribuir nomes às restrições.

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Nome          VARCHAR(50) NOT NULL,
   Cpf           CHAR(11),
   Salario       DECIMAL(10,2),
   CONSTRAINT CHPFUNC
      PRIMARY KEY(Cpf)                    );

CREATE TABLE DEPARTAMENTO (
   Dnumero         INT,
   Dnome           VARCHAR(30) NOT NULL,
   Cpf_gerente     CHAR(11) NOT NULL DEFAULT '12345678901',
   CONSTRAINT CHPDEP
      PRIMARY KEY(Dnumero),
   CONSTRAINT CHSDEP
      UNIQUE(Dnome),
   CONSTRAINT CHEGERDEP
      FOREIGN KEY(Cpf_gerente) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET DEFAULT  ON UPDATE CASCADE          );
```

Remoção de tabelas

DROP TABLE nome_tabela [**CASCADE** | **RESTRICT**];

- Com a opção CASCADE – remove também os objetos que dependem da tabela removida (ex.: views, restrições de chave estrangeira, índices).
- Com a opção RESTRICT (padrão) – impede que a tabela seja removida caso haja no BD objetos que dependam dela

Exemplo:

```
DROP TABLE FUNCIONARIO;
```

Referências Bibliográficas

- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010.
  Capítulo 3
- *A First Course in Database Systems* (1ª edição), Ullman e Widom. Prentice Hall, 1997.
  Capítulo 6
- Manual dos comandos do PostgreSQL (versão 9.4)
  `http://www.postgresql.org/docs/9.4/static/sql-commands.html`

Cenas dos próximos capítulos...

Mais sobre SQL

- Alteração de esquemas
- Recuperação de dados
- Inserção, alteração e remoção de dados