[MAC0313]
Introdução aos Sistemas de Bancos de Dados
Aula 12
Structured Query Language (SQL)
e Comandos para a Criação de Esquemas

Kelly Rosa Braghetto

DCC–IME–USP

19 de setembro de 2014

SQL – *Structured Query Language*

- Foi criada em 1976, na IBM Research
  - Originalmente, era chamada de SEQUEL (*Structured English QUery Language*)
- É a linguagem de consulta/modificação de dados mais usada no mundo
  - É um padrão para SGBDS Relacionais
  - Mesmo os SGBDs Não-Relacionais se "inspiram" na SQL para definir suas linguagens de consulta
- É importante por prover (algum nível de) interoperabilidade para sistemas de bancos de dados

A SQL possui comandos para:

- Consultas ao BD
  - Recursos muito parecidos aos da Álgebra Relacional e do Cálculo Relacional
- Modificação do BD
  - Inserção, remoção, alteração de tuplas em relações
- Definição do esquema do BD
  - Criação, remoção e alteração de tabelas, chaves primárias, chaves estrangeiras, restrições, etc.
- Criação de visões sobre o BD
- Especificação de segurança e autorização
- Controle transacional
- ...

"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 e SQL-2006 – inclusão de funcionalidades relacionadas a XML (entre outras coisas)
- SQL-2008 – inclusão de mais recursos de BDs de objetos

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*

O SGBDR PostgreSQL

- Nesta disciplina, usaremos como SGBDR o PostgreSQL (`http://www.postgresql.org/`)
- O PostgreSQL é software gratuito e de código aberto
- Foi criado em 1995
- Está atualmente na versão 9.3
- É muito bem desenvolvido; possui muitas funcionalidades avançadas
- Possui muito material de apoio disponível
- Bastante usado tanto na academia quanto na indústria

Como usar o PostgreSQL na disciplina

- Foi criado para cada aluno um banco de dados (vazio) em um servidor PostgreSQL que está hospedado em uma máquina do IME
- Você pode acessar o seu banco de dados a partir de qualquer máquina que possua acesso à Internet
- Para acessar o seu banco, é preciso que você tenha instalado em sua máquina um software cliente de conexão para o PostgreSQL

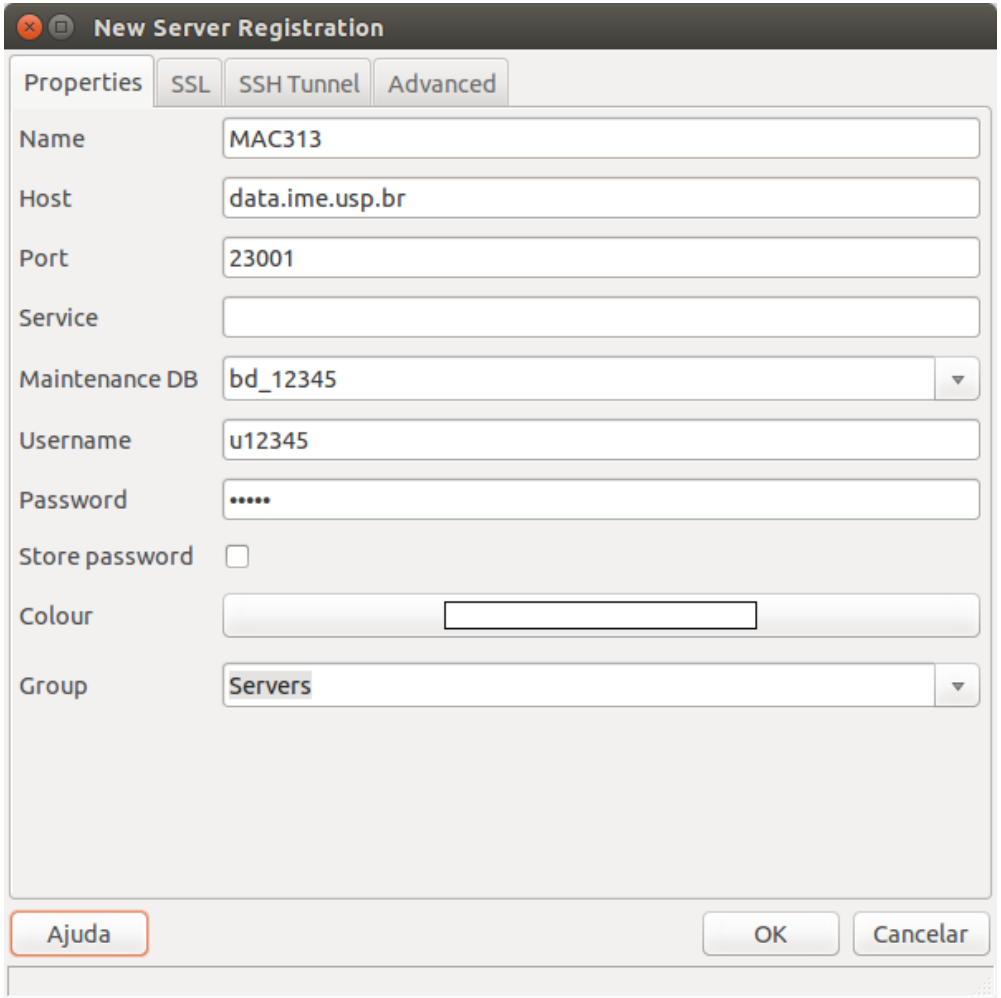
Como usar o PostgreSQL na disciplina

Software cliente que você pode usar:

- **pgAdmin** (`http://www.pgadmin.org`) – ferramenta com interface gráfica
  - Windows: instruções de instalação em `http://www.pgadmin.org/download/windows.php`
  - Linux: instalar pacote chamado `pgadmin3`
  - Mac OSX: instruções de instalação em `http://www.pgadmin.org/download/macosx.php`
- **psql** – ferramenta do tipo *linha de comando*
  - Disponível apenas no Linux: instalar pacote chamado `psql`

Instruções para a conexão com o seu BD via **pgAdmin**

Abra o pgAdmin e entre no menu "File > Add Server..."
Uma janela como a abaixo será aberta:



Instruções para a conexão com o seu BD via **pgAdmin**

Abra o pgAdmin e entre no menu "File > Add Server..."
Atenção para os campos que precisam ser preenchidos na janela que será aberta:

- `Name` – aqui você pode preencher com o nome que quiser; é apenas um apelido para a sua conexão
- `Host` – preencha com "`data.ime.usp.br`"
- `Port` – preencha com "23001"
- `Maintenance DB` – preencha com "`bd_`" + seu número USP
- `Username` – preencha com "`u`" + seu número USP
- `Password` – preencha com seu número USP

Ex.: para um aluno que tem número USP 12345, o `Username` é `u12345`, o `Maintenance DB` é `bd_12345` e a `Password` é `12345`.

Instruções para a conexão com o seu BD via **psql**

Abrir um terminal e executar o programa `psql` como mostrado abaixo:

```
psql -h data.ime.usp.br -p 23001 -U [usuário] -d [BD]
```

sendo que

- `[usuário]` – deve ser substituído por "u" + seu número USP
- `[BD]` – deve ser substituído por "bd_" + seu número USP

Ex.: para um aluno que tem número USP 12345, o comando fica:

```
psql -h data.ime.usp.br -p 23001 -U u12345 -d bd_12345
```

A senha inicial do seu usuário é o seu número USP.

SQL – Declaração de tabela (= relação armazenada)

- É feita por meio do comando `CREATE TABLE`
- Define:
  - O nome da tabela
  - O nome e o domínio das suas colunas (atributos)
  - Quaisquer tipos de restrições sobre colunas
  - Opcionalmente, restrições de chaves primária, secundária e estrangeiras

Relações criadas por meio do comando `CREATE TABLE` são criadas e armazenadas como um arquivo pelo SGBD, ou seja, elas possuem existência física no(s) disco(s)

SQL – Declarações simples de tabelas

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

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Nome             VARCHAR(50),
    Cpf              CHAR(11),
    Sexo             CHAR(1),
    Data_nascimento DATE
);
```

SQL – Tipos de dados de colunas

Números inteiros

- **INTEGER** ou **INT**
  No PostgreSQL, ocupa 4 bytes e representa a seguinte faixa de valores: [-2.147.483.648, +2.147.483.647]
- **SMALLINT** – ocupa geralmente a metade da quantidade de bytes usada por um INTEGER
  No PostgreSQL, ocupa 2 bytes e representa a seguinte faixa de valores: [-32.768, +32767]

SQL – Tipos de dados de colunas

Números reais

- **FLOAT** ou **REAL**
  No PostgreSQL, ocupa 4 bytes e tem precisão de 6 dígitos decimais
- **DOUBLE PRECISION**
  No PostgreSQL, ocupa 8 bytes e tem precisão de 15 dígitos decimais

SQL – Tipos de dados de colunas

Números reais

- **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.

  Exemplo: uma coluna do tipo `DECIMAL(6,4)` poderia armazenar um valor como `0123.45` .

  $\Rightarrow$ O valor padrão para *j* é 0, mas para *i* depende do SGBD
  No PostgreSQL, é possível representar números de até 131.072 dígitos antes da vírgula e até 16.383 dígitos depois da vírgula.

SQL – Tipos de dados de colunas

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.

SQL – Tipos de dados de colunas

Caracteres – considerações importantes:

- Cadeias de caracteres literais devem ser delimitadas por aspas simples (apóstrofos), como em `'MAC0313'`.
- Caracteres em SQL são ***case sensitive***.
  Portanto, `'MAC313'` $\neq$ `'mac313'`.

Obs.: Palavras reservadas da SQL são *case insensitive*, ou seja, podemos usar `create` ou `CREATE` de forma indistinta.

SQL – Tipos de dados de colunas

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 `'MAC0313'`, o valor que será armazenado será o `'MAC0313 '` (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 duas colunas do tipo CHAR(n) são comparadas, ou quando uma coluna desse tipo é convertida para um outro tipo de cadeia de caracteres.
- Já na comparação de colunas do tipo VARCHAR(n), os espaços em branco no final da cadeia são sim considerados.

SQL – Tipos de dados de colunas

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'

SQL – Tipos de dados de colunas

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.

SQL – Tipos de dados de colunas

Booleano

- **BOOLEAN** – admite os valores **TRUE**, **FALSE** ou **UNKNOWN**

Observações:

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

SQL – Tipos de dados de colunas

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.

SQL – Tipos de dados no PostgreSQL

Para mais informações sobre os tipos de dados no PostgreSQL 9.3:
`http://www.postgresql.org/docs/9.3/static/datatype.html`

SQL – Declaração simples de tabela

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Nome              VARCHAR(50),
    Cpf               CHAR(11),
    Casado            BOOLEAN,
    Sexo              CHAR(1),
    Data_nascimento   DATE,
    Salario           DECIMAL(10,2),
);
```

SQL – 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,
    Sexo              CHAR(1),
    Data_nascimento   DATE,
    Salario           DECIMAL(10,2),
);
```

SQL – 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 implícita 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

SQL – 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'
 );
```

SQL – Restrições de chave

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

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

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

SQL – Restrições de chave

Exemplo 1: Chave primária como restrição para coluna

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

Exemplo 2: Chave primária como restrição para tabela

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

SQL – 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 para coluna* na definição da tabela
- Para chaves secundárias com uma ou mais colunas, usa-se uma cláusula de *restrição para tabela*
- Assim como ocorre com a chave primária, não pode haver tuplas com valores repetidos para a chave secundária
- Diferentemente do que ocorre com a chave primária, uma coluna da chave secundária pode receber valores NULL

SQL – Restrições de chave

Exemplo 1: Chave secundária como restrição para coluna

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

Exemplo 2: Chave secundária como restrição para tabela

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

SQL – 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**

SQL – Restrições de integridade referencial

Exemplo: Chaves estrangeiras como restrições para colunas

```
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'
                            REFERENCES FUNCIONARIO(Cpf) );

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

SQL – Restrições de integridade referencial

Exemplo: Chaves estrangeiras como restrições para tabelas

```
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) );

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

SQL – Restrições de integridade referencial

Exemplo: Chaves estrangeiras como restrições para colunas e **ações de propagação**

```
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'
                       REFERENCES FUNCIONARIO(Cpf)
                       ON DELETE SET DEFAULT  ON UPDATE CASCADE );

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

SQL – Restrições de integridade referencial

Exemplo: Chaves estrangeiras como restrições para tabela e **ações de propagação**

```
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        );
```

SQL – 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 );
```

SQL – Remoção de tabelas

- A remoção de uma tabela é feita por meio do comando `DROP TABLE` .
- Depois de removida, uma tabela não faz mais parte do BD e não se pode mais acessar nenhuma de suas tuplas.

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

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

Exemplo:

```
DROP TABLE FUNCIONARIO;
```

SQL – *Esquemas*

- *Esquemas* são criados dentro de um BD
- Um *esquema* funciona como um *diretório*: ele pode ser usado para agrupar objetos do BD (tabelas, visões, tipos de dados, funções, etc.) segundo algum critério semântico de organização
  - Por exemplo, podemos pensar em agrupar as tabelas que pertencem a uma mesma aplicação, ou as que se referem a um módulo particular do BD
- É possível ter dois ou mais objetos com um mesmo nome em um BD contanto que cada um esteja em um esquema diferente

SQL – Criação de *esquemas*

- É feita por meio do comando `CREATE SCHEMA`:

  CREATE SCHEMA nome_esquema;

- É possível definir/acessar objetos em um esquema qualificando o nome dos objetos. Por exemplo, os comandos

```
CREATE SCHEMA EMPRESA;
CREATE TABLE EMPRESA.FUNCIONARIO...
```

  cria um esquema chamado `EMPRESA` e depois cria a tabela `FUNCIONARIO` no esquema `EMPRESA`.

SQL – Remoção de esquemas

- **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.

Mais comandos para *esquemas* específicos do PostgreSQL

- **SHOW** search_path;
  Mostra o esquema atualmente em uso.
- **SET search_path TO** *nome_esquema*;
  Define o esquema atualmente em uso.
- No PostgreSQL, o esquema padrão é o `public`

Comandos úteis para quem usa o cliente `psql`

- Para alterar a senha de seu usuário:
  `ALTER ROLE [usuario] WITH ENCRYPTED PASSWORD '[senha]'`
  ou `\password`
- Para listar todos os BDs do servidor ao qual se está conectado:
  `\l`
- Para listar as tabelas do BD atual:
  `\dt` **ou** `SELECT * FROM pg_catalog.pg_tables`
- Para listar o esquema (= estrutura) de uma tabela:
  `\d+ nome_tabela`
- Para ver outras opções de comandos especiais do *psql*:
  `\?`

Referências Bibliográficas

- *Database Systems – The Complete Book*, Garcia-Molina, Ullman e Widom. 2002.
  Capítulo 6
- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010.
  Capítulos 4 e 5