

[MAC0426] Sistemas de Bancos de Dados  
[IBI5013] Bancos de Dados para Bioinformática  
Aula 13  
Linguagem SQL (Parte 1):  
Comandos para a Definição de Esquemas

Kelly Rosa Braghetto

DCC-IME-USP

04 de maio de 2017

# SQL – *Structured Query Language*

⇒ Foi criada em 1976

Possui comandos para:

- ▶ Consultas ao BD (com recursos muito parecidos aos da Álgebra Relacional – básica + “avançada”)
- ▶ Modificação do BD (inserção, remoção, alteração de dados)
- ▶ Definição do esquema do BD (criação, remoção, alteração da estrutura das relações)

# “Dialeto” da SQL

A SQL tem vários padrões:

- ▶ **ANSI<sup>1</sup> 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.

<sup>1</sup>ANSI – *American National Standards Institute*

## Antes de falar mais sobre a SQL...

... vamos tratar de detalhes operacionais:

- ▶ Para exercitar o uso da SQL, vocês poderão usar a instalação do SGBDR PostgreSQL da Rede Linux.
- ▶ Para cada aluno interessado, um BD de uso exclusivo ao aluno será criado nessa instalação.
- ▶ Como solicitar a criação do seu BD:
  - ▶ Se você já possui conta na rede Linux: escreva para `admin@linux.ime.usp.br` e peça a criação do seu BD, dizendo que está cursando MAC426/IBI5013
  - ▶ Se você não possui conta na rede Linux: passar na sala da administração da rede (125A) portando documento de identificação e solicitar a criação da conta e do BD

# Propaganda “subliminar”

Eu ♥ Rede Linux

# Como se conectar ao seu BD no PostgreSQL da rede Linux

- ▶ Via interface *web*:
  - ▶ Acesse a página <https://phppgadmin.linux.ime.usp.br/>
  - ▶ Para a conexão, use o seu login e senha da rede Linux
- ▶ Via linha de comando (1):
  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 para se conectar ao seu BD:

```
psql -h dionisio
```

## Como se conectar ao seu BD no PostgreSQL da rede Linux

- ▶ Via linha de comando (2) – Se você tem o `psql` instalado na sua máquina, execute o programa como mostrado abaixo para se conectar ao seu BD:

```
psql -h linux.ime.usp.br -U [usuario]
```

# Como se conectar ao seu BD no PostgreSQL da rede Linux

- ▶ Se você tem o pgAdmin3 instalado na sua máquina, para criar uma nova conexão com o SGBD faça como no exemplo abaixo:

The image shows a screenshot of the 'New Server Registration' dialog box in pgAdmin3. The dialog has three tabs: 'Properties', 'SSL', and 'SSH Tunnel'. The 'Properties' tab is active. The fields are as follows:

- Name: Rede Linux
- Host: linux.ime.usp.br
- Port: 5432
- Service: (empty)
- Maintenance DB: [usuario]
- Username: [usuario]
- Password: (masked with asterisks)
- Store password:
- Colour: (empty)
- Group: Servers

At the bottom, there are three buttons: 'Ajuda', 'OK', and 'Cancelar'.



## 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), disponibilizada na rede Linux também

## 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 máquina 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 (;)

## Definição de Esquemas em SQL

## 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 (-2147483648 a +2147483647)
- ▶ **SMALLINT** – ocupa geralmente a metade da quantidade de bytes usada por um **INTEGER**  
no PostgreSQL, ocupa 2 bytes (-32768 a +32767)
- ▶ **BIGINT** – ocupa geralmente o dobro da quantidade de bytes usada por um **INTEGER**  
no PostgreSQL, ocupa 8 bytes (-9223372036854775808 a +9223372036854775807)

# Tipos de dados de atributos

## Números reais

- ▶ **FLOAT** ou **REAL**  
no PostgreSQL, ocupa 4 bytes e tem 6 dígitos de precisão
- ▶ **DOUBLE PRECISION**  
no PostgreSQL, ocupa 8 bytes e tem 15 dígitos de precisão
- ▶ **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.  
⇒ O valor padrão para  $j$  é 0, mas para  $i$  depende do SGBD  
No PostgreSQL, pode ter até 131072 dígitos antes do ponto decimal e até 16383 depois.

# 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 (apóstrofes), como em 'MAC0426'.
- ▶ Caracteres em SQL são sensíveis à capitalização. Portanto, 'MAC426'  $\neq$  'mac426'.  
Mas as palavras reservadas da SQL não são, 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 'MAC0426', o valor que será armazenado será o 'MAC0426 ' (ou seja, serão acrescentados 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 de fuso horário – 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, esse tipo de dado chama-se **BYTEA**)

# Tipos de dados no PostgreSQL

Para mais informações sobre os tipos de dados no PostgreSQL 9.6:  
<http://www.postgresql.org/docs/9.6/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 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

## 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 primária 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,  
    Dlocal     VARCHAR(15),  
    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 ChP_FUNC PRIMARY KEY(Cpf)                );
```

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

## Inserção de dados – comando básico

Para testar os comandos de definição de esquemas vistos na aula, insira dados no seu BD:

```
INSERT INTO nome_tabela [(coluna1, coluna2, ...)]  
                        VALUES (valor1,valor2,...);
```

- ▶ Insere uma linha na tabela *nome\_tabela*
- ▶ Quando a ordem das colunas não é especificada no comando, os valores são atribuídos de acordo com a ordem em que as colunas foram criadas na tabela
- ▶ Se os valores passados para o comando não satisfazem as restrições definidas sobre a tabela, a linha não é inserida no BD

## Inserção de dados – comando básico

### Exemplo:

```
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
      VALUES ('Fernando Pessoa', '12345678901', 4532.99);
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
      VALUES ('Clarice Lispector', '12782392989', 1238.23);
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
      VALUES ('Carlos Drummond', '11728237928', 23919.00);

INSERT INTO DEPARTAMENTO(Dnumero, Dnome, Cpf_gerente)
      VALUES (1, 'Contabilidade', '11728237928');
INSERT INTO DEPARTAMENTO(Dnumero, Dnome)
      VALUES (2, 'Recursos Humanos');

INSERT INTO LOCALIZACAO_DEP VALUES (1, 'Rua do Matao');
INSERT INTO LOCALIZACAO_DEP VALUES (1, 'Reitoria');
```

## 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;
```

## Alteração de esquemas – modificando colunas

```
ALTER TABLE nome_tabela ADD [COLUMN] nome_coluna  
                                tipo_dado [restrições];
```

- ▶ Adiciona uma nova coluna em uma tabela

### Exemplo:

```
ALTER TABLE Funcionario  
ADD COLUMN Sexo CHAR(1) DEFAULT 'F';
```

## Alteração de esquemas – modificando colunas

```
ALTER TABLE nome_tabela DROP [COLUMN] nome_coluna  
[RESTRICT | CASCADE];
```

- ▶ Remove uma coluna em uma tabela. Se a opção **CASCADE** for usada, todas as restrições e *views* que referenciam a coluna serão removidas.

### Exemplo:

```
ALTER TABLE Departamento  
DROP COLUMN Cpf_gerente CASCADE;
```

## Alteração de esquemas – modificando a cláusula *default*

```
ALTER TABLE nome_tabela ALTER [COLUMN] nome_coluna  
DROP DEFAULT;
```

- ▶ Remove a cláusula *default* de uma coluna

### Exemplo:

```
ALTER TABLE Funcionario  
ALTER COLUMN Sexo DROP DEFAULT;
```



## Alteração de esquemas – modificando a cláusula *default*

```
ALTER TABLE nome_tabela ALTER [COLUMN] nome_coluna  
SET DEFAULT valor;
```

- ▶ Define uma nova cláusula *default* para uma coluna

### Exemplo:

```
ALTER TABLE Funcionario  
ALTER COLUMN Sexo SET DEFAULT 'M';
```

## Alteração de esquemas – modificando restrições nomeadas

```
ALTER TABLE nome_tabela DROP CONSTRAINT  
nome_restrição;
```

- ▶ Remove uma restrição nomeada

Exemplo:

```
ALTER TABLE Departamento DROP CONSTRAINT ChS_DEP;
```

## Alteração de esquemas – modificando restrições nomeadas

```
ALTER TABLE nome_tabela ADD CONSTRAINT  
[nome_restricao] restrição;
```

- ▶ Define uma nova restrição para a tabela (que pode ser nomeada ou não)

### Exemplo:

```
ALTER TABLE Departamento ADD CONSTRAINT ChS_DEP  
UNIQUE(Dnome);
```

## 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 9.6)  
<http://www.postgresql.org/docs/9.6/static/sql-commands.html>

# Cenas dos próximos capítulos...

## Mais sobre SQL

- ▶ Recuperação de dados
- ▶ Inserção, alteração e remoção de dados