

Banco de Dados I

07 - Introdução à SQL e Linguagem de Definição de Dados

Marcos Roberto Ribeiro



Instituto Federal Minas Gerais - Campus Bambuí

2018

Introdução

- A *Structured Query Language (SQL)* ou Linguagem de Consulta Estruturada, é uma linguagem de consulta para banco de dados relacionais;
- A SQL originou-se do projeto *System R* desenvolvido pela IBM no início da década de 1970;
- A SQL é a linguagem padrão da grande maioria dos SGBD. Isto se deve principalmente a sua simplicidade e facilidade de uso;
- A grande vantagem da SQL em relação a outras linguagens é sua forma declarativa, ou seja, uma consulta SQL especifica qual o resultado esperado e não o caminho para atingi-lo;
- Apesar de sua origem no projeto *System R*, em pouco tempo surgiram vários *dialetos* desenvolvidos por outras empresas. Por este motivo, foram propostos padrões para a linguagem. Os padrões mais recentes são SQL-92 (1992), SQL-1999 (1999) e SQL-2003 (2003);
- Mesmo com as padronizações realizadas, cada SGBD pode possuir algumas características particulares em relação à SQL.

Grupos de Instruções SQL I

As instruções da SQL podem ser divididas nos seguintes grupos:

Data Definition Language (DDL) ou Linguagem de Definição de Dados

Permite a criação de bancos de dados, de tabelas de banco de dados e outros elementos.

Data Manipulation Language (DML) ou Linguagem de Manipulação de Dados

Permite inserir, alterar, apagar e consultar dados.

Data Transaction Language (DTL) ou Linguagem de Transação de Dados

Permite o controle de transações em banco de dados.

Data Control Language (DCL) ou Linguagem de Controle de Dados

Permite modificar as permissões dos usuários sobre o acesso aos dados.

Observações

- Alguns autores separam as instruções de consulta em *Data Query Language* (DQL);
- As instruções DTL e DCL não serão abordadas nesta disciplina.

Trabalhando com o SGBD PostgreSQL

- Para exercitarmos os comandos SQL desta aula vamos utilizar o SGBD PostgreSQL através do cliente de linha de comando **psql -h server -U user**
 - h **server** O parâmetro **server** é o servidor SGBD (pode ser um endereço de rede ou *localhost* para uma conexão local);
 - U **user** Especifica o usuário (**user**) utilizado para realizar a conexão (utilizaremos o usuário **postgres**)
 - f **arquivo.sql** O parâmetro -f permite a execução de instruções SQL em um arquivo
- Vamos conectar com o comando:
psql -h localhost -U postgres
- Para que a conexão seja efetivada será solicitada a senha do usuário;

Instalação do PostgreSQL no Linux

1. Instale os pacotes¹

```
sudo apt install postgresql postgresql-contrib
```

2. Mude para o usuário *postgres* do Linux

```
sudo su postgres
```

3. Acesse o PostgreSQL usando o *psql*

```
psql
```

4. Altere a senha do usuário *postgres* do PostgreSQL²

```
ALTER USER postgres WITH PASSWORD 'novasenha';  
\q
```

5. Volte para o seu usuário e acesse o sistema normalmente

```
exit  
psql -h localhost -U postgres
```

¹Considerando ambientes Debian-like com *sudo* ativado

²Troque *novasenha* pela senha desejada

Comandos Internos do psql I

- Após a conexão podemos utilizar alguns dos seguintes comandos para manipular os bancos de dados do SGBD:
 - `\l` Lista os bancos de dados;
 - `\c BD` Conecta-se ao banco de dados **BD**;
 - `\d [OBJETO]` Lista os objetos do banco de dados ou mostra detalhes do OBJETO informado;
 - `\dt[+]` Lista todas as tabelas do banco de dados;
 - `\di` Lista todas os índices do banco de dados;
 - `\ds` Lista todas as sequências do banco de dados;
 - `\dv` Lista todas as visões do banco de dados;
 - `\d TABELA` Exibe a descrição da *TABELA* informada;
 - `\e` Abre um editor externo e executa a consulta digitada no mesmo;

Comandos Internos do psql II

- `\s [ARQUIVO]` Mostra o histórico de instruções ou salva em um ARQUIVO;
- `\o ARQUIVO` Salva o resultado da consulta em um ARQUIVO;
 - `\q` Para encerrar a conexão com o SGBD;
 - `\h` Mostra sintaxe dos comandos SQL (* para todos os comandos);
 - `\?` Mostra uma tela de ajuda sobre os comandos do psql.

Principais Tipos de Dados I

- A seguir são descritos os principais tipos de dados do SGBD PostgreSQL. Os tipos de campos podem variar de acordo com o SGBD utilizado, porém a grande maioria está de acordo com o padrão SQL.

Tipos Numéricos Inteiros

Nome	Tamanho	Descrição	Intervalo
smallint, int2	2 bytes	Inteiros menores	-32.768 a $+32.767$
integer, int, int4	4 bytes	Inteiros comuns	$-2,14 \times 10^9$ a $+2,14 \times 10^9$
bigint, int8	8 bytes	Inteiros maiores	$-9,22 \times 10^{18}$ a $+9,22 \times 10^{18}$
smallserial, serial2	2 bytes	small com auto incremento	1 a 32.767
serial, serial 4	4 bytes	integer com auto incremento	1 a $2,14 \times 10^9$
bigserial, serial8	8 bytes	bigint com auto incremento	1 a $+9,22 \times 10^{18}$

Principais Tipos de Dados II

Tipos Numéricos Fracionários

Nome	Tamanho	Descrição	Intervalo
numeric(p,s), decimal(p,s)	variável	Precisão variável	precisão (p) de 131.072 dígitos e escala (s) de 16.383 dígitos
real, float4	4 bytes	Números fracionários	-1×10^{37} a 1×10^{37} com 6 dígitos de precisão
double precision, float8	8 bytes	Números fracionários	-1×10^{307} a 1×10^{308} com 15 dígitos de precisão
money	8 bytes	Números financeiros	$-92.233.720.368.547.758,08$ a $+92.233.720.368.547.758,07$

Tipos Lógicos

Nome	Tamanho	Descrição
boolean	1 byte	Valores lógicos ³

Principais Tipos de Dados III

Tipos Textuais

Nome	Tamanho	Descrição
char(n), character(n)	Variável com o número de caracteres	Tamanho fixo ⁴
varchar(n), character varying(n)	Variável com o número de caracteres	Variável com tamanho máximo
text ⁵	Variável com o número de caracteres	Ilimitado

Tipos Binários

Nome	Tamanho	Descrição
bytea ⁶	Variável	Dados binários como fotos

Principais Tipos de Dados IV

Tipos Temporais

Nome	Tamanho	Descrição	Intervalo
date	4 bytes	Data	4.713 A.C. a 5.874.897 D.C.
interval	12 bytes	Intervalo em anos	-178.000.000 a +178.000.000 D.C.
time	8 bytes	Hora	00:00:00 a 24:00:00
time with time zone	12 bytes	Hora com fuso horário	00:00:00+1459 a 24:00:00-1459
timestamp	8 bytes	Data e hora	4.713 A.C. a 294.276 D.C.
timestamp with time zone, timestampz	8 bytes	Data e hora	4.713 A.C. a 294.276 D.C.

³Verdadeiro (TRUE, 't', '1') ou Falso (FALSE, 'f', '0')

⁴Os caracteres não usados são preenchidos com espaços em branco.

⁵Suportado pelo PostgreSQL, mas não está no padrão SQL.

⁶O padrão SQL define um tipo similar chamado BLOB(*Binary Large Object*).

Linguagem de Definição de Dados (DDL)

- A DDL pode ser vista como o esquema físico do banco de dados, ou seja, como o banco de dados é representado no SGBD;
- De certa forma a DDL assemelha-se ao esquema lógico de um banco de dados no que diz respeito a descrição de relações, chaves primárias e chaves estrangeiras;
- As instruções DDL fazem com que o SGBD crie um *dicionário (ou catálogo) de dados* contendo todas as informações necessárias a respeito dos elementos do banco de dados;
- Antes de realizar qualquer operação sobre o banco de dados, o SGBD consulta o dicionário de dados para verificar se a operação realmente pode ser realizada;
- Por exemplo, antes de modificar dados de uma tabela o SGBD testa se os dados a serem modificados correspondem aos tipos corretos de campos, testa se não haverá violações de chaves primárias ou estrangeiras, além de outros tipos de testes para garantir a integridade dos dados.

Instruções DDL I

- As principais instruções DDL são:
 - CREATE** para criação de elementos de banco de dados;
 - ALTER** para alteração de elementos de banco de dados;
 - DROP** para remoção de elementos de banco de dados;
- Podemos usar tais instruções tanto para a criação de bancos de dados quanto para os elementos dentro de um banco de dados;
- Para trabalharmos com bancos de dados acrescentamos a palavra **DATABASE** e o nome do banco de dados após a instrução DDL desejada;
- Por exemplo, para criarmos o banco de dados *bancoteste* em um SGBD devemos usar a instrução:

```
CREATE DATABASE bancoteste;
```

Instruções DDL II

- Já para removermos este banco de dados do SGBD devemos usar a instrução:

```
DROP DATABASE bancoteste;
```

- Podemos notar que as instruções anteriores terminam com ; (ponto e vírgula), da mesma maneira toda instrução SQL também deve terminar com este caractere;
- Experimente executar os seguintes comandos:

```
\h CREATE
```

```
\h DROP
```

```
\h ALTER
```

Banco de Dados Acadêmico I

- Para aprendermos melhor como utilizar as instruções DDL, vamos criar um banco de dados chamado *academico* e trabalhar dentro do mesmo;
- Inicialmente vamos encerrar o **psql** e executá-lo novamente para garantir que não estamos conectados a nenhum banco de dados:

```
\q  
psql -h localhost -U postgres
```

- Agora vamos apagar o banco de dados (se existir), criá-lo e conectar no mesmo:

```
DROP DATABASE academico;  
CREATE DATABASE academico;  
\c academico;
```


- Após a criação do banco de dados podemos acessá-lo diretamente através do **psql** com o parâmetro **-d**:
`psql -h localhost -U postgres -d academico`
- A confirmação de que estamos dentro do banco de dados correto pode ser vista no *prompt* do psql:
`academico=#`

Criação de Tabelas

- A criação de tabelas através de instruções DDL é um pouco mais complexa do que a criação de um banco de dados. Isto porque não é necessário informar apenas o nome da tabela, mas também os campos (com seus respectivos tipos e opções) e também as restrições da tabela (chaves primárias, chaves estrangeiras e outras restrições);
- O formato de uma instrução para criação de tabelas é a seguinte:

```
CREATE TABLE nometabela(  
    campo_1 tipo [opções],  
    ...  
    campo_n tipo [opções],  
    [restrição_1,  
    ...  
    restrição_n]);
```

- Execute o comando `\h CREATE TABLE`.

Opções sobre campos

- Dependendo do SGBD podem ser informadas várias opções sobre cada campo, mas basicamente as opções mais relevantes são:
 - NOT NULL** O campo não pode conter valores nulos, ou seja, o valor do campo de ser informado;
 - DEFAULT *valor_padrão*** Valor padrão para o campo, sempre que for inserido um novo registro o valor inicial para o campo será *valor_padrão*;
 - CHECK *verificação*** Impõe uma verificação sobre os valores do campo, não permitindo valores que não passem pela verificação.

Restrições em Tabelas

- As restrições de tabelas permitem a criação de chaves primárias, chaves estrangeiras e outras restrições como unicidade;
- Para criarmos uma chave primária utilizamos as instruções⁷:

```
CONSTRAINT nome_chave PRIMARY KEY (c_1, ... c_n)
```

Onde nome_chave é o nome da chave primária e c_1, ... c_n são os campos que formam a chave primária;

⁷Na verdade, existem outras maneira de criarmos chaves primárias e demais restrições, mas não abordaremos por uma questão de simplicidade

Criando uma Tabela

- Com as instruções apresentadas até o momento podemos criar uma tabela simples contendo uma chave primária. Por exemplo:

```
CREATE TABLE aluno(  
    id_aluno INT NOT NULL,  
    nome_aluno VARCHAR(30) NOT NULL,  
    nascimento DATE,  
    media FLOAT CHECK (media >= 0),  
    CONSTRAINT aluno_pk PRIMARY KEY (id_aluno)  
);
```

- Experimente utilizar o comando `\d aluno` para visualizar a descrição da tabela;
- Como exercício escreva o esquema relacional correspondente a esta tabela.

- Para criamos uma chave estrangeira simples utilizamos as instruções:

```
CONSTRAINT nome_chave FOREIGN KEY (c_1, ... c_n)  
↪ REFERENCES tabela_ref(cc_1, ..., cc_n)
```

Onde *nome_chave* é o nome da chave estrangeira, *c_1, ... c_n* são os campos que formam a chave estrangeira, *tabela_ref* é a tabela referenciada pela chave estrangeira e *cc_1, ... cc_n* são os campos que formam a chave primária em *tabela_ref*;

Criação de Tabelas com Chave Estrangeira

Agora já podemos criar um banco de dados físico a partir de qualquer esquema relacional. Como exemplo vamos considerar o seguinte esquema relacional:

```
aluno(id_aluno integer, nome_aluno varchar(50),  
      nascimento date, media real)  
disciplina(id_disciplina integer,  
           nome_disciplina varchar(30), carga_horaria integer)  
matriculado(*id_disciplina integer, *id_aluno integer,  
            nota real)  
*matriculado.id_disciplina : disciplina.id_disciplina  
*matriculado.id_aluno : alunos.id_aluno
```

Como ainda não aprendemos a alterar tabelas, vamos excluir a tabela **aluno** e criá-la novamente⁸.

⁸É importante lembrar que todos os dados da tabela serão perdidos com esta ação

Exemplo com instruções DDL I

```
DROP TABLE aluno;

CREATE TABLE aluno(
    id_aluno INT NOT NULL,
    nome_aluno VARCHAR(30) NOT NULL,
    nascimento DATE,
    media FLOAT,
    CONSTRAINT aluno_pk PRIMARY KEY (id_aluno)
);

CREATE TABLE disciplina(
    id_disciplina INT NOT NULL,
    nome_disciplina VARCHAR(30) NOT NULL,
    carga_horaria INT NOT NULL,
    CONSTRAINT disciplina_pk PRIMARY KEY (id_disciplina)
);
```


Exemplo com instruções DDL II

```
CREATE TABLE matriculado(  
    id_disciplina INT NOT NULL,  
    id_aluno INT NOT NULL,  
    nota FLOAT,  
    CONSTRAINT matriculado_pk PRIMARY KEY (id_disciplina,  
↪ id_aluno),  
    CONSTRAINT matriculado_fk_aluno FOREIGN KEY (id_aluno)  
↪ REFERENCES aluno(id_aluno),  
    CONSTRAINT matriculado_fk_disciplina FOREIGN KEY  
↪ (id_disciplina) REFERENCES disciplina(id_disciplina));
```

Violação de Chave Estrangeira

- Quando uma tabela **A** apresenta uma chave estrangeira referenciando uma tabela **B**, dizemos que **A** é a *tabela dependente* e **B** é a *tabela referenciada*. Além disto, se a tabela **A** possui dados em sua chave estrangeira, dizemos que os mesmos são *dados dependentes* e seus respectivos valores na chave primária de B são os *dados referenciados*;
- Por padrão, se um dado referenciado for excluído ou alterado, ocorrerá um erro de violação de chave estrangeira. Por exemplo, considerando os seguintes dados para as tabelas *disciplina* e *matriculado* anteriores:

id_disciplina	nome_disciplina	carga_horaria
10	Algoritmos	80
20	Banco de dados	80

Tabela Disciplina

id_disciplina	id_aluno	nota
10	100	80,0
10	200	75,0
20	100	83,0
20	200	78,0

Tabela Matriculado

- Os da tabela disciplina não podem ser excluídos ou terem o campo *id_disciplina* alterado pois possuem dados dependentes na tabela matriculado.

Ações Adicionais em Chaves Estrangeiras

- Em determinadas situações podemos especificar ações adicionais a serem executadas sobre os dados dependentes caso hajam alterações ou exclusões sobre os dados referenciados;
- Estas ações são impostas pelas instruções **ON DELETE** (para exclusões) e **ON UPDATE** (para alterações) inseridas juntamente com a chave estrangeira;
- As instruções **ON DELETE** e **ON UPDATE** devem ser seguidas pelas ações a serem executadas. As possíveis ações são:
 - SET NULL** Quando o dado referenciado for excluído ou apagado, o dado dependente assume valor nulo.
 - SET DEFAULT** Quando o dado referenciado for excluído ou apagado, o dado dependente assume seu valor padrão.
 - CASCADE** Quando o dado referenciado for excluído ou apagado, o dado dependente a exclusão ou alteração será propagada para o dado dependente.

Exemplo com propagação para dados dependentes I

O exemplo anterior pode ser modificado para que quando uma disciplina seja alterada ou excluída, esta ação seja refletida sobre a tabela *matriculado*;

```
CREATE TABLE aluno(  
  id_aluno INT NOT NULL,  
  nome_aluno VARCHAR(30) NOT NULL,  
  nascimento DATE,  
  media FLOAT,  
  CONSTRAINT aluno_pk PRIMARY KEY (id_aluno));  
  
CREATE TABLE disciplina(  
  id_disciplina INT NOT NULL,  
  nome_disciplina VARCHAR(30) NOT NULL,  
  carga_horaria INT NOT NULL,  
  CONSTRAINT disciplina_pk PRIMARY KEY (id_disciplina));
```

Exemplo com propagação para dados dependentes II

```
CREATE TABLE matriculado(  
    id_disciplina INT NOT NULL,  
    id_aluno INT NOT NULL,  
    nota FLOAT,  
    CONSTRAINT matriculado_pk PRIMARY KEY (id_disciplina, id_aluno),  
    CONSTRAINT matriculado_fk_aluno FOREIGN KEY (id_aluno)  
↪ REFERENCES aluno(id_aluno),  
    CONSTRAINT matriculado_fk_disciplina FOREIGN KEY (id_disciplina)  
↪ REFERENCES disciplina(id_disciplina) ON DELETE CASCADE ON  
↪ UPDATE CASCADE);
```

Restrição de Unicidade

- Em alguns casos temos campos que não pertencem a chave primária, mas que não podem conter valores repetidos, ou seja, cada valor para este campo deve ser único;
- Para criamos uma restrição de unicidade utilizamos as instruções:

```
CONSTRAINT nome_chave UNIQUE (c_1, ... c_n)
```

- Um exemplo de unicidade seria se tivéssemos o campo *cpf* na tabela *aluno*, cada aluno deve possuir um único CPF. Considerando esta situação teríamos a seguinte instrução DDL para criar a tabela *aluno*:

```
CREATE TABLE aluno(  
    id_aluno INT NOT NULL,  
    nome_aluno VARCHAR(30) NOT NULL,  
    cpf CHAR(11) NOT NULL,  
    nascimento DATE,  
    media FLOAT,  
    CONSTRAINT aluno_pk PRIMARY KEY (id_aluno),  
    CONSTRAINT aluno_cpf_key UNIQUE (cpf));
```

Alteração de Tabelas I

- Depois de que uma tabela é criada podemos alterar sua estrutura através da instrução **ALTER TABLE**;
- A instrução **ALTER TABLE** permite adicionar (**ADD**) novos campos ou restrições, modificar (**ALTER**) campos, excluir (**DROP**) campos ou restrições e renomear (**RENAME**) campos ou a tabela;
- Vamos demonstrar cada uma das possibilidades de utilização da instrução **ALTER TABLE** através de exemplos. Para isto vamos considerar a seguinte relação inicial:

Alteração de Tabelas II

```
CREATE TABLE aluno(  
  id_aluno INT NOT NULL,  
  nome_aluno VARCHAR(30) NOT NULL,  
  nascimento DATE,  
  media FLOAT,  
  CONSTRAINT aluno_pk PRIMARY KEY (id_aluno));
```


Alteração de Tabelas III

- Para acrescentarmos um campo com o CPF do aluno, podemos usar a instrução:

```
ALTER TABLE aluno ADD numero_cpf CHAR(11);
```

- Para renomearmos o campos, podemos usar a instrução:

```
ALTER TABLE aluno RENAME COLUMN numero_cpf TO cpf;
```

- Para não permitirmos que o CPF receba valores nulos, podemos usar a instrução:

```
ALTER TABLE aluno ALTER cpf SET NOT NULL;
```

- Para permitirmos novamente que o CPF receba valores nulos, podemos usar a instrução:

Alteração de Tabelas IV

```
ALTER TABLE aluno ALTER cpf DROP NOT NULL;
```

- Para modificarmos o valor padrão do campo *media*, podemos usar a instrução:

```
ALTER TABLE aluno ALTER media SET DEFAULT 0.0;
```

- Para acrescentarmos uma restrição de unicidade sobre o campo *cpf_aluno*, podemos usar a instrução:

```
ALTER TABLE aluno ADD CONSTRAINT aluno_cpf_key UNIQUE  
↪ (cpf);
```

- E por fim, se desejarmos remover a restrição e o campo criados podemos usar a instrução:

Alteração de Tabelas V

```
ALTER TABLE aluno DROP CONSTRAINT aluno_cpf_key;  
ALTER TABLE aluno DROP cpf;
```

Sequências

- Sequências são objetos capazes de gerar números em série;
- As sequências são especialmente úteis para definir campos auto-incremento;
- Para criarmos sequências usamos a instrução:

```
CREATE SEQUENCE NomeSequência  
INCREMENT ValorIncremento  
MINVALUE ValorMínimo  
MAXVALUE ValorMáximo  
START ValorInicial  
CYCLE;
```

- Para uma definição mais simples de sequência pode ser usada apenas a instrução:

```
CREATE SEQUENCE NomeSequência;
```

Ligando sequências a campos

- Para uma utilização mais funcional devemos ligar uma sequência a um campo de uma tabela;
- Esta ligação é feita alterando o valor padrão do campo. Como exemplo vamos utilizar a tabela `aluno`:

```
CREATE SEQUENCE aluno_id_aluno_seq;  
ALTER TABLE aluno ALTER id_aluno SET DEFAULT  
↪ nextval('aluno_id_aluno_seq');
```

- Agora quando forem inseridos dados na tabela **aluno** automaticamente o campo **id_aluno** é preenchido com 1, 2, 3 e assim por diante;

Criando sequências implicitamente

- A criação de sequência pode ser feita de forma implícita utilizando o tipo **SERIAL**. Por exemplo:

```
CREATE TABLE aluno(  
    id_aluno SERIAL NOT NULL,  
    nome_aluno VARCHAR(30) NOT NULL,  
    nascimento DATE,  
    media FLOAT,  
    CONSTRAINT aluno_pk PRIMARY KEY (id_aluno)  
);
```

- Deste modo o PostgreSQL criará automaticamente uma sequência ligada ao campo **id_aluno**.

Comentários de Objetos

- No PostgreSQL é possível inserir comentários em objetos para facilitar a identificação dos mesmos através da instrução **COMMENT ON**;
- Alguns exemplos:


```
COMMENT ON TABLE aluno IS 'Alunos da instituição';
```


```
\dt+
```

```
COMMENT ON COLUMN aluno.id_aluno IS 'Identificador do  
↪ aluno';
```

```
\d+ aluno
```

 (2012).
Postgresql documentation.

 Elmasri, R. and Navathe, S. B. (2011).
Sistemas de banco de dados.
Pearson Addison Wesley, São Paulo, 6 edition.

 Ramakrishnan, R. and Gehrke, J. (2008).
Sistemas de gerenciamento de banco de dados.
McGrawHill, São Paulo, 3 edition.