Implementação de banco de dados

Aula 2: Linguagem SQL – DML e DDL

Apresentação

Anteriormente, estudamos o Modelo Relacional e a Álgebra Relacional, suas características e visualizamos suas principais operações. Além disso, realizamos vários exercícios de fixação.

Nesta aula, seremos apresentados à linguagem SQL e vamos conhecer as partes que a compõem, estudaremos os comandos de criação de tabelas (DDL) e os comandos que permitem colocar, alterar ou eliminar linhas das tabelas (DML).

Objetivos

- Criar e eliminar tabelas;
- Inserir, alterar e eliminar dados das tabelas.

Fonte: everything possible / Shutterstock.

Linguagem SQL

O padrão mundial de acesso a banco de dados é a Linguagem Estruturada de Consulta (*Structured Query Language*) ou simplesmente SQL, na sigla em inglês.

A linguagem SQL divide-se em partes, cada uma atendendo a uma necessidade específica. Temos, então, a seguinte divisão:

Linguagem de Descrição de Dados (Data Definition Language – DDL) – tem como principais comandos Create, Alter e Drop, destinados, respectivamente, a **criar**, **alterar**

Linguagem de Manipulação de Dados (Data Manipulation Language - DML) – tem como principais comandos Insert, Update, Delete e Select, destinados, respectivamente, a

Linguagem de Controle de Dados (**Data Control Language** – DCL) – tem como principais comandos Grant e Revoke,

destinados a conceder e revogar

e **eliminar** objetos de banco de dados, como usuários, tabelas, índices etc.

inserir linhas nas tabelas, **alterar** linhas, **eliminar** linhas e **consultar** os dados da tabela.

privilégios de acesso respectivamente.

Nosso foco de estudo será nos comandos de criação, alteração e eliminação de tabelas e nos comandos de DML (*Insert, Update, Delete e Select*).

Tabelas

Tabelas são os objetos básicos de armazenamento de dados no modelo relacional. Para criarmos uma tabela, devemos definir o seu **nome**, suas **colunas**, os **tipos de dados das colunas** e suas restrições.

Nome, Coluna, Tipos de dados e Restrições

Clique no botão acima.

Nome

O nome da tabela é normalmente definido durante a modelagem lógica, constituindo às vezes, alguma variação em relação ao nome da entidade. Por exemplo, entidade **Aluno** vira tabela **Aluno** ou **Alunos**.

É importante conhecer as limitações do SGBD na hora de criar a tabela. A maior parte dos SGBD não dão suporte a caracteres em português no nome da tabela. Dessa forma, se temos uma entidade **Aprovação**, teremos que criar a tabela **Aprovacao;** se a entidade tem o nome **Prova Aluno**, teremos que substituir o espaço em banco por: **_ Prova_Aluno**.

Colunas

As colunas das tabelas se originaram dos atributos das entidades conforme vimos na modelagem lógica. Da mesma forma que o nome da tabela, temos que respeitar as limitações do SGBD: não usar espaço em branco, caracteres em português e nos preocupar também com a quantidade máxima de caracteres que o nome da coluna ou tabela pode ter. Muitas vezes, por causa dessa limitação, teremos que abreviar o nome. Por exemplo: o atributo Matrícula do Aluno poderá virar a coluna **Mat_Aluno** na tabela.

Tipos de dados

As colunas possuem um tipo de dado que podem armazenar de forma similar ao conceito de tipo utilizado nas variáveis criadas em programas.

Os SGBD possuem uma variedade muito grande de tipos. Cada SGBD tem o seu conjunto específico, que é, muitas vezes, incompatível com outros SGBD. Nós faremos uso do PostGreSql como SGBD para a realização de exercícios. Nos comandos de criação de tabelas, os tipos básicos são:

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Tipos de Dados	Descrição
VARCHAR(tam)	Caracter de tamanho variável, podendo atingir de 1 até 8.000 bytes, sendo limitado a tam.

CHAR(tam)	Caracter de tamanho fixo, podendo tam ser especificado no máximo de 8.000 caracteres.
TEXT	Caracter de tamanho variável sem limite.
INTEGER	Número inteiro de 4 bytes podendo ir de -2147483648 a +2147483647.
NUMERIC(I,d)	Numérico com precisão. Em i, estabelecemos quantos dígitos tem o número e em d quantos há na parte decimal. Por exemplo, para especificar um número com formato 99999.99. Tipo seria NUMERIC(7,2)
DATE	Data entre 01 de janeiro de 4713 AC até 31 de dezembro de 32767 DC.
TIMESTAMP	Armazenamento da data e da hora juntos
BIT	Tipo booleano armazena 0 (falso) ou 1 (verdadeiro)

Esses tipos mostrados na tabela acima são uma pequena parte dos tipos existentes no PostgreSql, mas serão suficientes para nossos exercícios.

NOTA:

Abordaremos também as particularidades do SQL do Oracle e do SQL Server no que diferem do PostgreSql.

Restrições

As restrições, normalmente chamadas *constraint* visam estabelecer regras para orientar o SGBD na forma de manter a integridade do banco.

Aqui vamos tratar das seguintes *constraint*:

Restrição (*Constraint*)	Descrição
NOT NULL (Obrigatório)	Especifica se a coluna é obrigatória para todas as linhas da tabela, não podendo conter nenhum valor NULO.
UNIQUE (Chave alternada)	A coluna ou combinação de diversas colunas tem que ser única para todas as linhas da tabela, não permitindo repetições.
PRIMARY KEY (Chave primária)	É a chave primária de identificação unívoca da tabela. Pode ser uma ou uma combinação de colunas.
FOREIGN KEY (Chave estrangeira)	É uma coluna que garante a integridade de uma relação entre duas tabelas, sendo referenciada por uma chave primária da outra tabela.

Comando de criação de tabela

O comando de Criação de Tabela é o Create Table.

Vejamos sua sintaxe:

```
CREATE TABLE nome_da_tabela(
 (nome_col1 tipo_col1 [restri_col1] [,
 nome_col2 tipo_col2 [restri_col2] [,
```

```
nome_coln tipo_coln [restri_coln] [,
restri_tab1 [restri_tab2]);
```

Onde:

Palavra Chave	Descrição
nome_da_tabela	Nome que será atribuído à tabela.
Nome_coln	Nome atribuído à coluna n.
Tipo_coln	Tipo de dado atribuído à coluna n.
restri_coln	Define uma restrição de integridade automática à qual os campos da coluna devem obedecer.
restri_tabn	Define uma restrição de integridade automática à qual toda a tabela deve obedecer.

Vamos ver um exemplo de criação.

Considere a tabela Departamento:

Tabela	Coluna	Tipo	Tamanho	Observação
Departamento	ID	numérico	7	Chave Primária
	NOME	caracter	40	obrigatório

O comando que a criaria no PostGreSql seria:

```
CREATE TABLE DEPARTAMENTO
(ID NUMERIC(7) PRIMARY KEY,
NOME VARCHAR(40) NOT NULL)
```

Observe no comando a *constraint* PRIMARY KEY, definindo a coluna ID como chave primária e a *constraint* NOT NULL, estabelecendo que NOME é de preenchimento obrigatório.

Sugiro que você crie a tabela no PostGreSql conforme as orientações do vídeo da aula.

Para vermos a tabela criada, a forma mais fácil é consultarmos o conteúdo de toda a tabela. Para isso, podemos dar o comando:

```
SELECT *
FROM DEPARTAMENTO
```

Esse comando é básico e lista todo o conteúdo da tabela.

Note que a tabela está vazia, mas foi criada no banco com duas colunas.

No SQL Server, o comando é o mesmo.

O comando funciona também no Oracle:

Mais *Constraints*

Até agora utilizamos apenas as *constraints* NOT NULL e PRIMARY KEY.

Vejamos agora as *constraints* UNIQUE e FOREIGN KEY.

Campos únicos

UNIQUE

Ao estabelecermos a *constraint* UNIQUE para uma coluna, determinamos que ela não pode ter valor repetido. Entretanto, ela não obriga a coluna a ter valor ou não a torna de preenchimento obrigatório e, como NULO não é valor, uma coluna UNIQUE pode possuir várias linhas nulas (sem valor).

Exemplo

CPF char(11) UNIQUE,

Reforçando a Integridade Referencial com Chave Estrangeira *(Foreign Key)*

Foreign Key

Os relacionamentos entre tabelas são criados através da geração de chaves estrangeiras (*foreign key* – FK) nas tabelas FILHO que referenciam colunas chaves nas tabelas PAI.

Para estabelecer essa restrição, acrescentamos REFERENCES na definição da coluna, como exemplo:

ID_DEPTO numeric (7) References Departamento(ID),

Onde:

- Id_depto é o nome da coluna;
- Numeric r(7) o tipo da coluna;
- References identifica a restrição de chave estrangeira;
- Departamento é o nome da tabela para onde aponta a chave estrangeira;
- (ID) é coluna da tabela departamento apontada pela chave estrangeira.

Vejamos um exemplo de criação de tabela com essas restrições.

```
CREATE TABLE EMPREGADO
( ID NUMERIC(7) PRIMARY KEY,
ULT_NOME VARCHAR(20) NOT NULL,
PRIM_NOME VARCHAR(20) NOT NULL,
CARGO VARCHAR(30),
SALARIO NUMERIC(7,2),
DT_ADMISSAO DATE,
CPF CHAR(11) UNIQUE,
ID_DEPTO NUMERIC(7) REFERENCES DEPARTAMENTO(ID))
```

Observe o comando. Através de sua análise, podemos observar que:

- a coluna ID é sua chave primária;
- As colunas ULT_NOME e PRIM_NOME são de preenchimento obrigatório;
- A coluna CPF é única;
- A coluna ID_DEPTO é uma chave estrangeira para a tabela departamento.

Vamos criá-la no PostGgreSql.

Observe a mensagem de sucesso.

Atenção

Importante
Se uma tabela possui uma chave estrangeira para outra, ela tem que ser criada depois da tabela referenciada, senão ocorrerá um erro.

Vamos agora criar uma terceira tabela:

```
CREATE TABLE CLIENTE
( ID NUMERIC(7) PRIMARY KEY,
NOME VARCHAR(40) NOT NULL,
VENDEDOR NUMERIC(7))
```

Acrescentando colunas em tabelas

Podemos acrescentar colunas em tabelas já criadas com o comando Alter Table. Sua sintaxe é:

alter table <nome_tabela> add <nome_coluna> <tipo da coluna> <constraint >

Em que:
<nome_tabela> é o nome da tabela a qual será acrescida à coluna.
<nome coluna> é o nome da coluna que será acrescida.
<tipo da coluna> é o tipo de dado da coluna a ser acrescida.
<constraint> é a restrição, se for o caso, da coluna a ser acrescida.

Exemplo

Na criação da tabela Departamento. Vimos que duas restrições (*constraints*) são estabelecidas como: uma de obrigatoriedade (NOT NULL) e uma Chave Primária.

Vamos assumir, entretanto que nossa tabela, com esse último comando de criação, não foi completamente estabelecida. Está faltando a coluna descrição. Para inseri-la, podemos dar o comando:

ALTER TABLE DEPARTAMENTO ADD descricao VARCHAR(30) NOT NULL;

Vamos executar o comando. Para isso, digite-o no PostgreSql.

Eliminando colunas de tabelas

É possível eliminar colunas de tabelas, inclusive aquelas referenciadas por *constraints* e índices, e até mesmo chaves primárias, únicas e estrangeiras. É verdade que cuidados quanto à aplicação devem ser tomados por parte dos desenvolvedores e DBA, porém, o SGBD implementa essa funcionalidade.

Ao eliminarmos uma coluna, suas restrições, caso existam, também são removidas do dicionário de dados.

Sintaxe:

alter table <nome_tabela> drop column <nome_coluna>;

Em que:
<nome_tabela> é o nome da tabela da qual será eliminada a coluna.
<nome coluna> é o nome da coluna que será eliminada.

Por exemplo: se desejarmos eliminar a coluna descrição da tabela Departamento, daremos o seguinte comando:

ALTER TABLE DEPARTAMENTO DROP COLUMN DESCRICAO

Vamos executar o comando. Para isso, digite-o no PostgreSql.

Incluindo uma Foreign Key numa tabela existente

Também podemos incluir a *constraint* de *Foreign Key* após a criação da tabela. Para tal, basta especificar a adição da *constraint* no comando ALTER TABLE.

A tabela Cliente foi criada, mas a coluna Vendedor deveria ser uma chave estrangeira para a tabela Empregado na coluna ID. Podemos dar o seguinte comando de ALTER TABLE:

ALTER TABLE CLIENTE ADD FOREIGN KEY (VENDEDOR) REFERENCES EMPREGADO(ID)

Em que:
CLIENTE – é a tabela a ser alterada
ADD FOREIGN KEY – é a restrição a ser acrescida.
(VENDEDOR) é a coluna que receberá a *constraint*
REFERENCES EMPREGADO(ID) indica a tabela e a coluna referenciadas pela chave estrangeira.

Veja o comando no PostGreSql.

Constraint de colunas e tabelas

As *constraints* podem ser definidas junto com a coluna ou separadamente, no final do comando create table ou com o comando alter table. As *constraints* not null só podem ser definidas junto com a definição da coluna.

As *constraints* de tabela são utilizadas principalmente para criar *constraints* compostas, onde duas ou mais colunas fazem parte da *constraint*. Como, por exemplo, chaves primárias compostas.

Exemplo: A tabela Turmas possui uma chave primary composta pelas colunas CODIGO_TURMA e CODIGO_CURSO. O comando para sua criação é:

```
CREATE TABLE TURMAS
(CODIGO_TURMA NUMBER(6),
CODIGO_CURSO NUMBER(3),
CODIGO_FUNCIONARIO NUMBER(6),
DATA_INICIO DATE,
DATA_FIM DATE ,
SALA NUMBER(2),
PRIMARY KEY (CODIGO_TURMA, CODIGO_CURSO) );
```

E se desejarmos apagar uma tabela?

Para isso temos o Comando Drop, cuja sintaxe é:
DROP TABLE <NOME DA TABELA>

Para eliminarmos a tabela TURMAS, daremos o comando:
DROP TABLE TURMAS;

Execute o comando no PostGreSql.

Observe a mensagem de sucesso.

Agora, se dermos o comando de Select na tabela, teremos uma mensagem de erro.

Veja o que acontece ao eliminar a tabela Departamento.

Não será possível porque a tabela Empregado possui uma Chave Estrangeira para Departamento, e você não poderá eliminar Departamento enquanto essa FK existir.

Atenção

Os comandos de CREATE TABLE, ALTER TABLE e DROP TABLE vistos aqui funcionam de forma exatamente igual no PostGreSql, no Oracle e no Sql Server.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Manipulando dados

Agora que já você já aprendeu a criar, alterar e eliminar tabelas, vai estudar os comandos de manipulação de dados que permitirão que você faça a inclusão, alteração ou eliminação das linhas das tabelas.

Esses comandos são:

Insert – para incluir linhas; Delete – para eliminar linhas; Update – para alterar linhas.

Clique no botão acima.

Inserindo Linhas

O comando *INSERT* insere linhas em uma tabela.

Sua sintaxe básica é:

insert into .<nome_tabela> (coluna1, coluna2, ..., colunan)
values (valor1, valor2, ..., valorn);

Em que:

Cláusula	Descrição
nome_tabela	O nome da tabela a ser atualizada
Coluna n	A coluna que queremos inserir
Valor n	É o novo valor associado à coluna a ser inserida

Vamos inserir uma linha na tabela de departamento.

INSERT INTO DEPARTAMENTO (ID, NOME)
VALUES (100, 'Financeiro');

A relação entre a lista de colunas (ID, NOME) e a lista de valores (100, 'VENDAS) é posicional, portanto:

- A coluna ID receberá o valor 100;
- Nome receberá Vendas.

Execute o comando no PostgreSql e observe a mensagem de linha inserida.

Uma outra forma de dar o comando INSERT é sem referenciar as colunas. Nesse caso, a lista de valores deve estar na ordem das colunas da tabela.

INSERT INTO DEPARTAMENTO VALUES (200, 'Compras')

Desta vez, não foram especificadas as colunas que receberão os valores. Portanto, o comando utilizará todas as colunas da tabela na ordem em que foram criadas.

A nossa tabela possui duas linhas.

Inserindo com valores nulos

Caso alguma coluna deva ficar com NULO em uma inserção, basta omitir o nome da mesma na lista de colunas.
Vejamos um exemplo em que isso ocorre:

No exemplo acima, foram omitidas as colunas CARGO, SALARIO, DT_ADMISSAO e CPF, que ficaram nulas.

Atenção

Observe que NULO não é equivalente a 0 (zero), espaço ou qualquer outro valor. NULO é, justamente, a ausência de qualquer valor na coluna.
Um cuidado que devemos ter é que só é possível fazer isso em colunas que não possuam a *constraint NOT NULL*. Outro detalhe é que nenhuma coluna definida como chave primária poderá conter NULO.

No nosso exemplo, se não fosse especificado um valor para a coluna id, a inserção resultaria em erro. O mesmo erro ocorreria se não tivéssemos valores para as colunas PRIM_NOME e ULT_NOME, já que são de preenchimento obrigatório (*constraint not null*).

Aproveitando ainda esse exemplo de inserção, note que inserimos o PRIM_NOME e depois o ULT_NOME, apesar de na tabela ULT_NOME vir antes de PRIM_NOME. Isso foi possível devido à inserção ser realizada na ordem da lista de colunas.

Uma última observação se refere à Chave Estrangeira ID_DEPTO, cujo valor inserido OBRIGATORIAMENTE tem que existir na tabela DEPARTAMENTO.

Outra forma de inserirmos com valores nulos em uma coluna é utilizando a palavra reservada *null*. Vejamos um exemplo:
Como omitimos a lista de colunas, temos que ter um valor para cada coluna. Então estamos inserindo 20 no ID, Fonseca no Prim_Nome, Antonio no Ult_Nome no NOME , nulo nas colunas CARGO, SALARIO, DT_ADMISSAO e CPF e 200 no Id_depto.

Execute o comando.

Observe a mensagem de linha inserida.

Como omitimos a lista de colunas, temos que ter um valor para cada coluna, na ordem das colunas da tabela.

Você também pode executar vários comandos de inserção juntos; Basta separá-los por:

```
INSERT INTO EMPREGADO
VALUES (2, 'Neves', 'Lauro', 'Diretor de Compras',19500, '07/03/2009','23456789012',200);
INSERT INTO EMPREGADO
VALUES (3, 'Nogueira', 'Mário','Diretor de Vendas', 18000, '07/04/2010','34567890123',100);
INSERT INTO EMPREGADO
VALUES (4, 'Queiroz', 'Mark','Gerente de Compras',8000, '07/11/2010','12345432123',200);
INSERT INTO EMPREGADO
VALUES( 5, 'Rodrigues', 'Alberto', 'Vendedor',4000, '10/1/2008', '87965432123', 100);
INSERT INTO EMPREGADO
VALUES( 6, 'Ugarte', 'Marlene', 'Vendedor', 3500,'23/11/2009', '87654345678',100);
```

Como resultado, temos sete linhas na tabela.

Observe que apesar de termos inserido a data no formato DD/MM/AAAA, o PostgreSql sempre exibe por padrão no formato AAAA/MM/DD. Você pode dar o comando de inserção de uma forma ou de outra, mas, para evitar problemas, use sempre AAAA/MM/DD. É mais seguro.

Atualizando linhas

O comando UPDATE permite que atualizemos dados já existentes nas tabelas.

Sua sintaxe é:

```
UPDATE nome_tabela
SET coluna = expressão
WHERE condição
```

Onde:

Cláusula	Descrição
Nome_tabela	O nome da tabela a ser atualizada.
coluna	A coluna que queremos alterar.
expressão	O novo valor associado à coluna a ser alterada.
condição	A condição que deverá satisfazer as colunas que serão alteradas.

Vejamos um exemplo:

Vamos aumentar o salário de todos empregados em R$ 1000.

```
UPDATE EMPREGADO
SET SALARIO = SALARIO + 1000;
```

Como resultado teríamos:

Repare que:

1. Como o comando foi dado sem cláusula *where*, ele afetou todas as linhas da tabela;

2. Os empregados de id 10 e 20 continuaram com salário nulo. Isso ocorre porque Nulo não é valor, e qualquer valor operado com Nulo tem como resultado Nulo.

Mas vamos agora atribuir um salário a esses dois empregados. Podemos então dar um comando de UPDATE com WHERE para alterar apenas essas duas linhas.

```
UPDATE EMPREGADO
SET SALARIO = 3000
WHERE ID = 10 OR ID = 20;
```

Vejamos como ficou:

Repare que os empregados 10 e 20 possuem salário.

Eliminando linhas

O comando *DELETE* é utilizado para excluir linhas em uma tabela e tem como sintaxe:

```
DELETE FROM nome_tabela
WHERE condição
```

Em que:

Cláusula	Descrição
Nome_tabela	O nome da tabela a ser deletada
condição	A condição que deverá satisfazer as colunas que serão deletadas

Vamos apagar da Tabela Empregado o funcionário de ID 10.

```
DELETE FROM EMPREGADO
WHERE ID = 10;
```

Dessa forma, o resultado seria não ter mais a linha do empregado 10 na tabela.

Atenção

1. Se você desse o comando de delete na Tabela Empregado sem a Cláusula Where, TODAS AS LINHAS da tabela seriam apagadas.
2. Você não consegue apagar linhas na tabela Departamento enquanto existirem linhas em Empregado que as referenciem na Chave Estrangeira.

Vamos, agora, reconstituir nosso ambiente, eliminando as tabelas Departamento e Empregado na ordem correta:

- DROP TABLE CLIENTE;
- DROP TABLE EMPREGADO;
- DROP TABLE DEPARTAMENTO.

Você deve dropar as tabelas nessa ordem devido às FK.

Scripts

Um SCRIPT nada mais é que um conjunto de comandos SQL salvos em um arquivo com a extensão .sql, que é carregado no SGBD. Em seguida, é lido e tem seus comandos executados como um todo.

No PGADMIN, podemos carregar um Script clicando em Abrir Arquivo.

Navegando até o local do arquivo e o selecionando.

Feito isso, o conteúdo do Script é carregado e basta mandar executá-lo.

As tabelas são criadas e os dados inseridos.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1. A partir do Modelo Lógico a seguir, crie as tabelas no PostGreSql.

2. Insira os dados nas tabelas criadas na atividade 1 de forma que as tabelas fiquem conforme as figuras a seguir:

3. Baixe o Script Escola disponível aqui <./galeria/aula2/apoio/Aula_2_ESCOLA.sql> e o execute no PostGreSql.

Notas

Título modal [1]

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Título modal [1]

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Referências

DATE, C. J. **Introdução a sistemas de banco de dados**. 7. ed. Rio de Janeiro: Campus, 2000.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados.** 7. ed. São Paulo: Pearson Addison Wesley, 2015.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de banco de dados.** 5. ed. Rio de Janeiro: Campus, 2006.

Próxima aula

- Estudo do comando Select;
- Cláusulas iniciais (Select e From);
- Cláusula Distinct.

Explore mais

- PostgreSQL