TI TOTAL

ÁREA FISCAL E CONTROLE

Professor
Ramon Souza

Tecnologia da Informação

TEORIA

SQL (DDL)

SUMÁRIO

A nossa aula é bem esquematizada, então para facilitar o seu acesso aos **esquemas**, você pode usar o seguinte índice:

GLOSSÁRIO DE TERMOS

Constraint ou restrição: especificação de regras para os dados em uma tabela.

Default: valor padrão.

Functions ou funções: rotinas que retornam valores ou tabelas.

Índice ou index: estruturas de acesso auxiliares associados a tabelas, que são utilizados para agilizar a recuperação de registros em resposta a certas condições de pesquisa.

Replace: substituir algo.

Storage: armazenamento de dados.

Store Procedure ou Procedimento Armazenado: código SQL preparado que você pode salvar, para que o código possa ser reutilizado repetidamente.

Triggers ou gatilhos: programas armazenados que são executados ou disparados automaticamente quando alguns eventos ocorrem.

Visão ou view: tabela virtual derivada de outras tabelas. Maneira alternativa de visualização dos dados. Consulta pré-definida ou armazenada, executada sempre que referenciada.

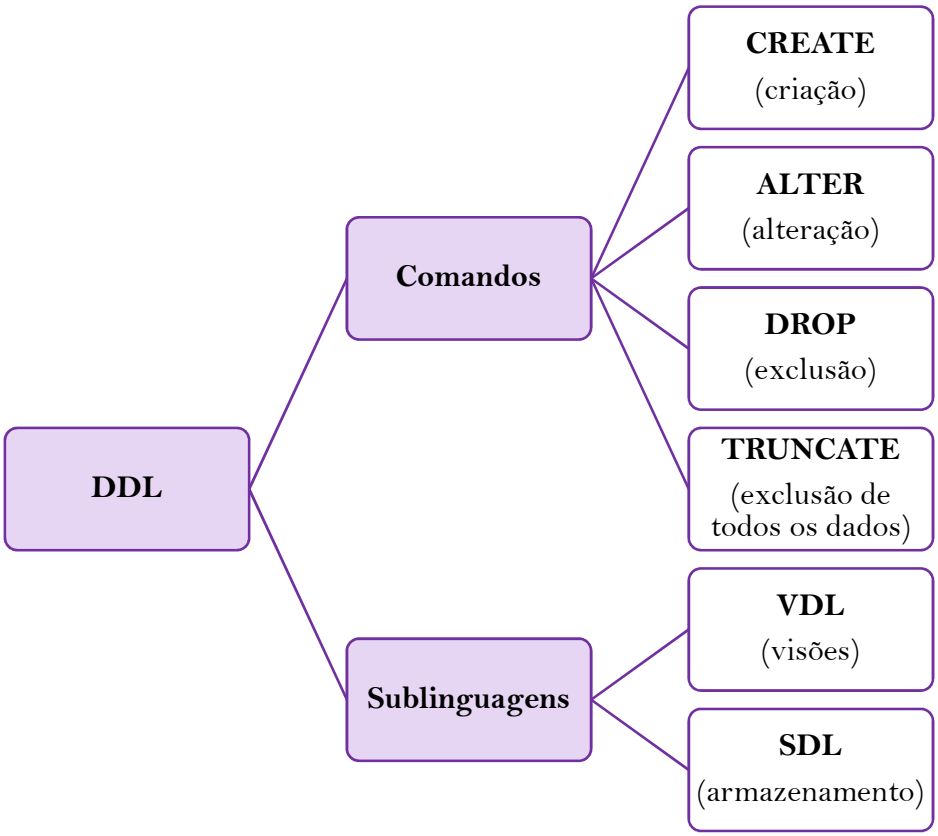
1. SQL (DDL)

1.1 Introdução à DDL

A **DDL (Data Definition Language)** é a sublinguagem do SQL que permite ao utilizador **definir tabelas novas e elementos associados**. Os comandos desta linguagem são **CREATE, ALTER, DROP** e **TRUNCATE**. É importante ressaltar que estas instruções permitem a criação, alteração e exclusão desde o próprio banco de dados até de estruturas como tabelas, visões, procedimentos e triggers.

Vale ressaltar que há autores que tratam falam em algumas linguagens específicas como:

- **VDL (View Definition Language)**: para a **definição de visões**.
- **SDL (Storage Definition Language)**: para a **definição do armazenamento ou especificação do esquema interno**.



Esquema 1 – DDL.

1- (CESPE - 2018 - STJ - Técnico Judiciário - Desenvolvimento de Sistemas)

Julgue o item a seguir, referente à modelagem de dados.

A DDL (data definition language) é usada para a definição da estrutura do banco de dados ou do esquema. São comandos DDL: CREATE, TRUNCATE, GRANT e ROLLBACK.

Resolução:

Os comandos da DDL (Data Definition Language) são CREATE, ALTER e DROP (ou TRUNCATE). GRANT e REVOKE são comandos da DCL (Data Control Language).

Gabarito: **Errado**.

2- (CESPE - 2015 - MEC – Desenvolvedor) Com relação à linguagem de definição de dados (DDL) e à linguagem de manipulação de dados (DML), julgue o próximo item.

A DML utiliza o comando CREATE para inserir um novo registro na tabela de dados.

Resolução:

Para inserir dados em uma tabela, o comando utilizado é o INSERT INTO, que faz parte da DML.

O comando CREATE faz parte da DDL e é usado para criar as estruturas do banco de dados, como as tabelas, visões e outros elementos.

Gabarito: **Errado.**

3- (CESPE - 2014 - ANATEL - Analista Administrativo - Desenvolvimento de Sistemas) Julgue os itens seguintes, a respeito das linguagens de banco de dados.

A DDL (data definition language) é responsável pela especificação da instância do banco de dados e também pode ser usada para especificar propriedades adicionais dos dados, como restrições de consistência.

Resolução:

A DDL não especifica a instância, mas sim o esquema do banco de dados.

Gabarito: **Errado.**

1.2 Trabalhando com banco de dados

Criando um banco de dados

A instrução **CREATE DATABASE** é usada para **criar um banco de dados**.

A sintaxe para criar um banco de dados é:

```
CREATE DATABASE nome_do_banco;
```

Exibindo os bancos de dados

A instrução **SHOW DATABASES** **lista os bancos** de dados existentes.

```
SHOW DATABASES;
```

Excluindo um banco de dados

A instrução **DROP DATABASE** é usada para **deletar um banco** de dados existente.

A sintaxe para deletar um banco de dados é:

```
DROP DATABASE nome_do_banco;
```

ATENÇÃO!!!

Para **criar ou excluir um banco** de dados, **deve-se possuir privilégios de administrador**.

EXEMPLIFICANDO!!!

Para criar um banco de dados chamado estudo, podemos usar o seguinte comando:

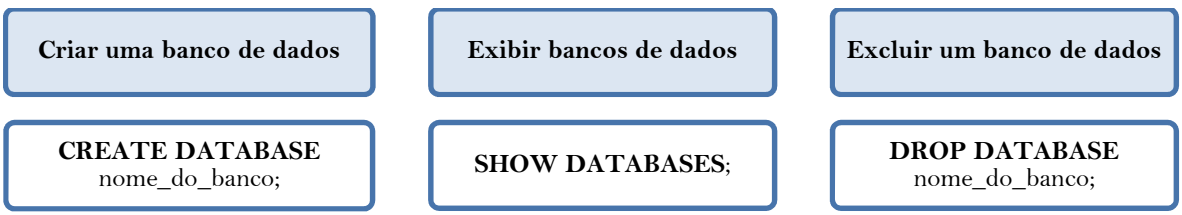
```
CREATE DATABASE estudo;
```

Se desejarmos listar os bancos existentes, podemos usar o comando:

```
SHOW DATABASES;
```

E, se por qualquer motivo, desejarmos deletar o banco estudo, então usaremos o comando DROP:

```
DROP DATABASE estudo;
```



Esquema 2 – Trabalhando com banco de dados.

4- (CESPE / CEBRASPE - 2021 - APEX Brasil - Analista - Tecnologia da Informação e Comunicação)

create database pessoa;

O comando SQL apresentado anteriormente criará

a) um banco de dados denominado pessoa;.

b) uma tabela denominada pessoa;.

c) um tipo de dados denominado pessoa;.

d) um esquema denominado pessoa;.

Resolução:

O comando CREATE DATABASE é usado para criar bancos de dados. Assim, o comando trazido na questão, irá realizar a criação de um banco de dados chamado "pessoa".

Gabarito: **Letra A**.

5- (Quadrix - 2021 - CRBM - 4 - Técnico em Informática) Quanto aos sistemas de bancos de dados e à linguagem de consulta estruturada (SQL), julgue o item.

Em um banco de dados MySQL, para se criar um banco de dados de nome dbEmpresa, é suficiente executar o comando a seguir. CREATE DATABASE dbEmpresa;

Resolução:

O comando CREATE DATABASE é usado para criar bancos de dados. Assim, o comando

CREATE DATABASE dbEmpresa

Irá criar um banco de dados chamado dbEmpresa.

Gabarito: **Certo**.

6- (CESPE / CEBRASPE - 2020 - Ministério da Economia - Tecnologia da Informação - Ciência de Dados) Julgue o item a seguir, a respeito de conceitos de SQL.

O comando CREATE DATABASE TAB é utilizado para criar uma tabela em um banco de dados.

Resolução:

O comando para criar uma tabela é CREATE TABLE e não CREATE DATABASE.

O comando CREATE DATABASE é usado para criar bancos de dados. Assim, o comando trazido na questão, irá realizar a criação de um banco de dados chamado "TAB".

Para criar uma tabela, o comando correto seria:

CREATE TABLE TAB

Gabarito: **Errado**.

1.3 Trabalhando com tabelas

1.3.1 Comandos básicos

Criando uma tabela

A instrução **CREATE TABLE** é usada para **criar uma nova tabela** no banco de dados. A sintaxe básica dessa instrução é:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  ....
);
```

Nesse comando, temos a criação de uma tabela com o nome indicado por nome_da_tabela. As colunas (ou atributos) dessa tabela são determinados pelos elementos coluna1, coluna2 e assim sucessivamente. Além de informar o nome da coluna, o elemento tipo_de_dado serve para informar qual o tipo de dado da coluna (varchar, integer, date, etc.).

ESCLARECENDO!!!

É importante destacar que os tipos de dados possíveis varia de acordo com o SGBD sendo utilizado. Como curiosidade, é possível verificar os tipos de dados possíveis no MySQL Server no seguinte link: https://www.w3schools.com/sql/sql_datatypes.asp.

EXEMPLIFICANDO!!!

Vamos então criar uma tabela para o nosso banco de dados.

```
CREATE TABLE Pessoas (
  IDPessoa int,
  Sobrenome varchar(255),
  Nome varchar(255),
  Endereco varchar(255),
);
```

Com esse comando criamos uma tabela chamada Pessoas com as seguintes colunas: IDPessoa, Sobrenome, Nome e Endereco, sendo IDPessoa do tipo inteiro (int) e os demais atributos cadeias de caracteres (varchar). A tabela criada é representada a seguir:

IDPessoa	Sobrenome	Nome	Endereço

Ao utilizar a DDL em conjunto com a DML é possível criar uma tabela a partir de outra tabela existente. Para isso basta usar o auxílio da cláusula AS conforme sintaxe a seguir:

```
CREATE TABLE nome_da_nova_tabela AS
  SELECT coluna1, coluna2,...
  FROM nome_da_tabela_existente
  WHERE ....;
```

Com essa sintaxe, é possível criar uma nova tabela a partir de uma instrução SELECT. Tanto a estrutura da seleção quando os dados selecionados serão armazenados na nova tabela.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Se quisermos criar uma nova tabela com o nome_cliente e o seu país, podemos usar a seguinte sintaxe:

```
CREATE TABLE teste AS
  SELECT Nome_Cliente, Pais
  FROM Clientes;
```

Para consultar dos dados da nova tabela basta usar o comando a seguir:

```
SELECT * FROM teste;
```

O resultado será:

Nome_Cliente	Pais
Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taquería	Mexico
Blondel père et fils	France

ATENÇÃO!!!

Pessoal, em relação aos comandos da DDL, podemos ter algumas variações dependendo do SGBD. Como o objetivo dessa aula não é tratar especificamente de nenhum SGBD, quando necessário, iremos informar as possibilidades de sintaxe para os principais SGBDs de mercado. Tenha uma noção sobre essas possibilidades, mas não fique preso a decorar todas.

Alterando uma tabela

A instrução **ALTER TABLE** é usada para **adicionar, deletar ou modificar colunas em uma tabela existente**. Essa instrução também pode ser utilizada para adicionar ou deletar restrições a esta tabela.

Para **adicionar uma coluna**, usamos a cláusula **ADD**:

```
ALTER TABLE nome_da_tabela
ADD nome_da_coluna tipo_de_dado;
```

Para **modificar uma coluna**, usamos a cláusula **ALTER COLUMN** (SQL Server/Access) **ou MODIFY COLUMN** (MySQL/Oracle até antes do 10G) **ou MODIFY** (Oracle 10G e superiores):

```
ALTER TABLE nome_da_tabela
ALTER COLUMN nome_da_coluna tipo_de_dado;
OU
ALTER TABLE nome_da_tabela
MODIFY COLUMN nome_da_coluna tipo_de_dado;
OU
ALTER TABLE nome_da_tabela
MODIFY nome_da_coluna tipo_de_dado;
```

Para **deletar uma coluna**, usamos a cláusula **DROP COLUMN**:

```
ALTER TABLE nome_da_tabela
DROP COLUMN nome_da_coluna;
```

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Vamos supor que desejamos inserir uma coluna para o número do telefone dos clientes. Então devemos alterar a estrutura da tabela e inserir um novo atributo:

```
ALTER TABLE Clientes
ADD telefone varchar(255);
```

Agora essa tabela terá a seguinte estrutura:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais	Telefone
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany	
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico	
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico	
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France	

Foi criado o campo telefone, porém você não deseja que sejam inseridos caracteres além de números. Então você pode trocar o tipo de dados do telefone para permitir apenas números. Então, poderá usar:

```
ALTER TABLE Clientes
ALTER COLUMN telefone int;
```

Agora o campo telefone será do tipo inteiro.

Suponha, contudo, que você não vê mais a necessidade de que haja um telefone nessa tabela, então você pode simplesmente excluir esse campo:

```
ALTER TABLE Clientes
DROP telefone;
```

A estrutura da tabela retorna ao estado anterior.

Excluindo uma tabela

A instrução **DROP TABLE** é usada para **deletar uma tabela existente**.

A sintaxe para esse comando é:

DROP TABLE nome_da_tabela;

Essa instrução irá deletar todos os dados da tabela, bem como a própria tabela.

Contudo, você pode desejar **excluir apenas os dados da tabela, sem excluir a estrutura dessa tabela**. Para isso, poderá usar o comando **TRUNCATE**:

TRUNCATE TABLE nome_da_tabela;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Se desejarmos deletar essa tabela por completo, basta usar o comando DROP TABLE:

DROP TABLE Clientes;

Assim, essa tabela deixará de existir. Não teremos mais os dados e nem mesmo a estrutura. Assim, caso seja necessária uma tabela para Clientes, deverá ser criada uma nova tabela a partir de um comando CREATE TABLE.

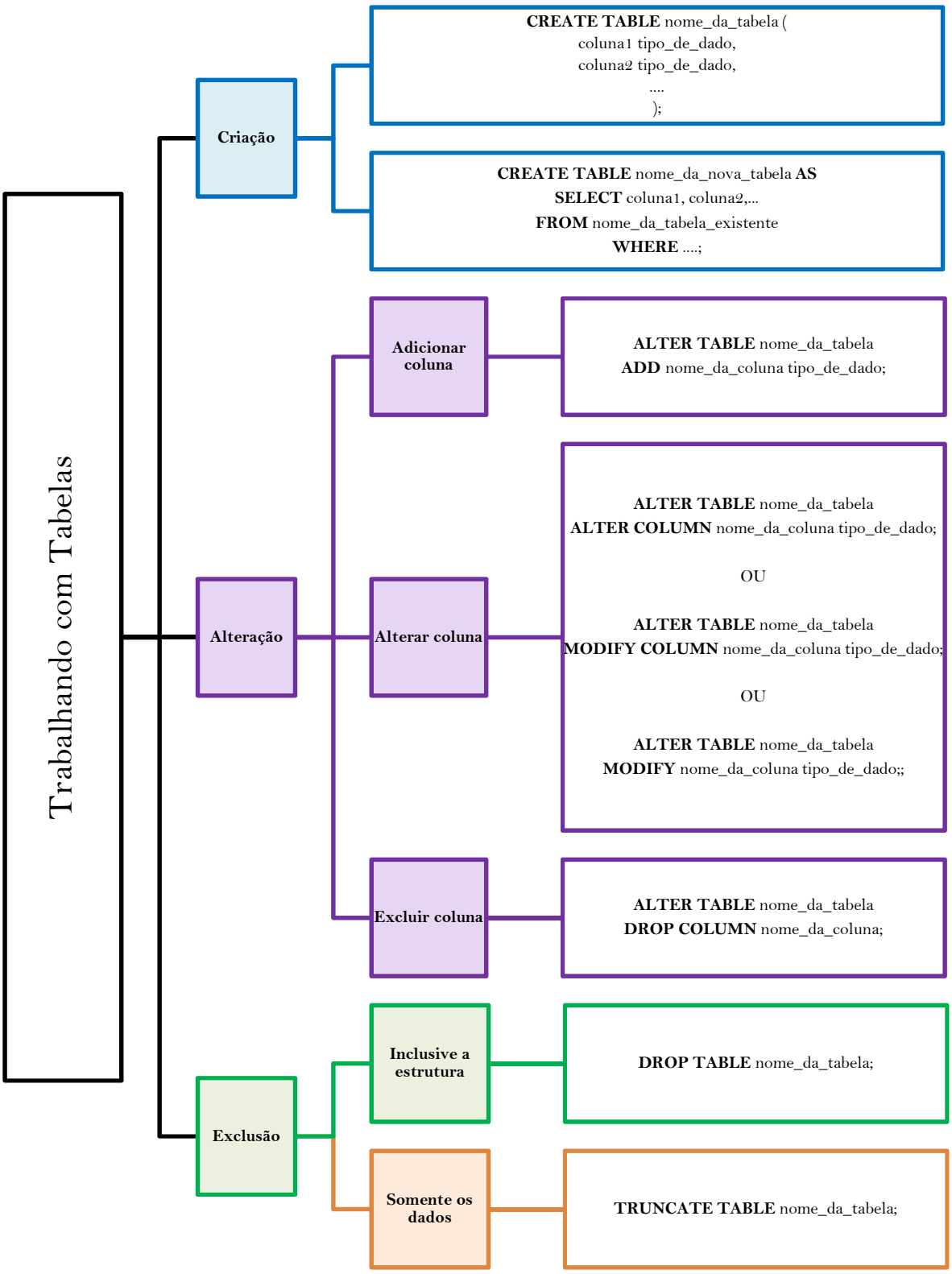
Se, no entanto, quisermos apenas apagar os dados dessa tabela, mas manter a sua estrutura, então usaremos o comando TRUNCATE:

TRUNCATE TABLE Clientes;

A estrutura da tabela será preservada, mas os dados serão apagados:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais

Em esquema temos:



Esquema 3 – Trabalhando com Tabelas.

7- (CESPE / CEBRASPE - 2020 - Ministério da Economia - Tecnologia da Informação - Ciência de Dados) Julgue o item a seguir, a respeito de conceitos de SQL. O comando CREATE DATABASE TAB é utilizado para criar uma tabela em um banco de dados.

Resolução:

O comando para criar uma tabela é **CREATE TABLE** e não CREATE DATABASE.

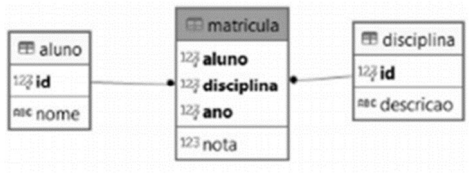
O comando CREATE DATABASE é usado para criar bancos de dados. Assim, o comando trazido na questão, irá realizar a criação de um banco de dados chamado "TAB".

Para criar uma tabela, o comando correto seria:

CREATE TABLE TAB;

Gabarito: Errado.

8- (CESPE / CEBRASPE - 2020 - Ministério da Economia - Tecnologia da Informação - Desenvolvimento de Software)



Tendo como referência o diagrama de entidade relacionamento precedente, julgue o próximo item, a respeito de linguagem de definição de dados e SQL.

A expressão SQL a seguir permite excluir as notas do aluno de nome Fulano.

truncate from matricula where aluno='Fulano'

Resolução:

TRUNCATE é para deletar todos os dados de uma tabela. Caso se deseje excluir somente registros específicos, então deve se usar o comando DELETE.

Ademais, o nome do aluno não está na tabela matrícula, mas sim na tabela aluno, então para deletar é precisa buscar o id desse aluno. O correto seria:

DELETE FROM matricula WHERE aluno = (SELECT id FROM aluno WHERE nome = 'Fulano');

Gabarito: **Errado**.

9- (CESPE - 2018 - FUB - Técnico de Tecnologia da Informação) Julgue o item subsecutivo, a respeito de linguagem de definição e manipulação de dados.

O comando DROP TABLE permite excluir do banco de dados a definição de uma tabela e de todos os seus dados.

Resolução:

A instrução **DROP TABLE** é usada para **deletar uma tabela existente**.

A sintaxe para esse comando é:

DROP TABLE nome_da_tabela;

Gabarito: **Certo**.

10- (CESPE - 2017 - TRE-PE - Analista Judiciário - Análise de Sistemas)

Tabela 3A6AAA

dados da tabela:

ID; nome; idtipo; preco

25; creme; 3; 11,50

31; arroz; 4; 12,50

34; leite; 1; 14,00

42; sabão; 5; 11,00

46; carne; 1; 12,75

48; shampoo; 5; 12,30

58; azeite; 1; 13,25

Assinale a opção que apresenta o comando SQL correto para se incluir um novo campo idcategoria do tipo INT nos dados da tabela 3A6AAA, denominada tbproduto.

a) ALTER TABLE tbproduto INSERT idcategoria INT;

b) ALTER TABLE tbproduto ADD COLUMN idcategoria INT;

c) UPDATE TABLE tbproduto ADD COLUMN idcategoria INT;

d) ADD COLUMN idcategoria INT IN TABLE tbprodut;

e) UPDATE TABLE ADD COLUMN idcategoria INT IN tbproduto;

Resolução:

Como queremos inserir um novo campo em uma tabela, então devemos usar o comando ALTER TABLE. Logo, eliminamos c), d) e e).

Para **adicionar uma coluna**, usamos a cláusula **ADD ou ADD COLUMN**:

ALTER TABLE nome_da_tabela

ADD nome_da_coluna tipo_de_dado;

Assim, a sintaxe correta está na letra b).

Gabarito: **Letra B**.

11- (CESPE - 2016 - TCE-PA - Auditor de Controle Externo - Área Informática - Analista de Suporte) No que concerne à linguagem SQL, julgue o item seguinte.

O comando create table pode ser utilizado para criar tanto uma tabela vazia quanto uma com dados de outra tabela.

Resolução:

Perfeitamente, o comando CREATE TABLE pode definir uma tabela sem dados ou usar alguma outra tabela como base.

A sintaxe básica dessa instrução é:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  ....);
```

Ao utilizar a DDL em conjunto com a DML é possível criar uma tabela a partir de outra tabela existente. Para isso basta usar o auxílio da cláusula AS conforme sintaxe a seguir:

```
CREATE TABLE nome_da_nova_tabela AS
  SELECT coluna1, coluna2,...
  FROM nome_da_tabela_existente
  WHERE ....;
```

Gabarito: **Certo**.

12- (FCC - 2019 - TRF - 4ª REGIÃO - Analista Judiciário - Sistemas de Tecnologia da Informação) Uma Analista digitou o comando TRUNCATE TABLE processos; em um banco de dados SQL aberto em condições ideais para

a) excluir os dados da tabela, mas não a tabela em si.

b) excluir a estrutura da tabela e os dados nela contidos.

c) juntar a tabela aberta na memória com a tabela processos.

d) bloquear a tabela processos para uso exclusivo de seu usuário.

e) editar a estrutura da tabela em modo gráfico.

Resolução:

A instrução **DROP TABLE** é usada para **deletar uma tabela existente**.

A sintaxe para esse comando é:

```
DROP TABLE nome_da_tabela;
```

Essa instrução irá deletar todos os dados da tabela, bem como a própria tabela.

Contudo, você pode desejar **excluir apenas os dados da tabela, sem excluir a estrutura dessa tabela**. Para isso, poderá usar o comando **TRUNCATE**:

```
TRUNCATE TABLE nome_da_tabela;
```

Gabarito: **Letra A**.

13- (FCC - 2015 - TRT - 9ª REGIÃO (PR) - Analista Judiciário - Área Apoio Especializado - Tecnologia da Informação)

A tabela relativa a Débitos Trabalhistas a seguir deve ser utilizada para responder à questão.

Considere que a tabela já está criada, os dados iniciais já foram inseridos e o banco de dados a ser utilizado está aberto e funcionando em condições ideais.

Tabela DebTrab

NroProcesso	Principal	Juros	FGTS	Honor Periciais
111/15	25345.00	3801.75	7933.00	4755.00
777/15	125800.00	18870.00	57966.87	7543.00
333/15	8844.50	1326.67	4233.55	1781.00
555/15	327631.00	65526.20	104863.78	11523.00
444/15	5072.00	1014.40	895.14	700.00

Um Analista da área de TI trabalha em uma organização que possui aplicações que utilizam os SGBDs Oracle 11g e SQL Server. Ele identificou que o comando SQL que está correto e pode ser aplicado em ambas as plataformas é

a) ALTER TABLE DebTrab ALTER COLUMN NroProcesso integer;

b) ALTER TABLE DebTrab MODIFY NroProcesso int;

c) ALTER TABLE DebTrab ADD DataPartida data;

d) ALTER TABLE DebTrab ADD IndiceAtualiz float;

e) ALTER TABLE DebTrab DROP COLUMN DataPartida;

Resolução:

Vamos analisar cada um dos itens:

a) **Incorreto**: **ALTER COLUMN** pode ser usado no SQL Server, mas não no Oracle.

b) **Incorreto**: **MODIFY** é usado somente a partir da versão 10g do Oracle, mas não no SQL Server.

c) **Incorreto**: o tipo de dados "data" não existe. O correto seria **"date"**.

d) **Correto**: para **adicionar uma coluna**, usamos a cláusula **ADD**:

```
ALTER TABLE nome_da_tabela
ADD nome_da_coluna tipo_de_dado;
```

e) **Incorreto**: no modelo apresentado, a tabela DebTrab não possui nenhuma coluna DataPartida e, portanto, não é possível deletar algo que não existe.

Gabarito: **Letra D**.

1.3.2 Restrições

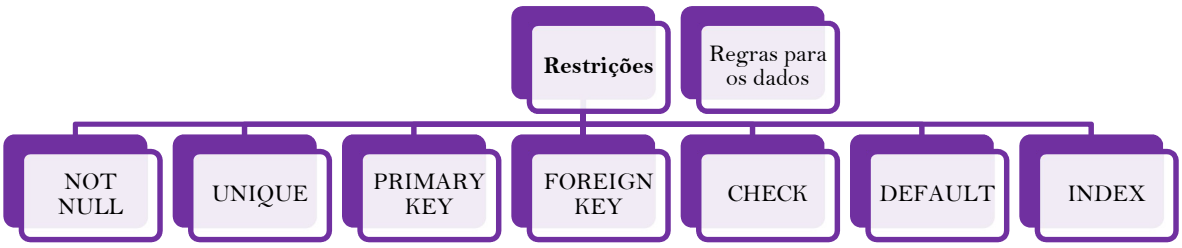
As **restrições SQL (constraints)** são usadas para **especificar regras para os dados em uma tabela**.

As **restrições** são usadas para **limitar o tipo de dados que podem ser colocados em uma tabela**. Isso garante a precisão e a confiabilidade dos dados na tabela. Se houver alguma violação entre a restrição e a ação de dados, a ação será abortada.

As **restrições** podem ser no nível da coluna ou no nível da tabela. As restrições de nível de coluna se aplicam a uma coluna e as restrições de nível de tabela se aplicam à tabela inteira.

As seguintes restrições são comumente usadas no SQL:

- **NOT NULL**: Garante que uma coluna não pode ter um valor NULL.
- **UNIQUE**: Garante que todos os valores em uma coluna sejam diferentes.
- **PRIMARY KEY**: Uma combinação de NOT NULL e UNIQUE. Identifica exclusivamente cada linha em uma tabela.
- **FOREIGN KEY**: Identifica exclusivamente uma linha / registro em outra tabela.
- **CHECK**: Garante que todos os valores em uma coluna satisfaçam uma condição específica.
- **DEFAULT**: Define um valor padrão para uma coluna quando nenhum valor é especificado.
- **INDEX**: Usado para criar e recuperar dados do banco de dados muito rapidamente.



Esquema 4 – Restrições em SQL.

NOT NULL

Por padrão, uma coluna pode conter valores NULL. A restrição **NOT NULL** impõe a uma coluna a regra para NÃO aceitar valores NULL. Isso **obriga um campo a sempre conter um valor**, o que significa que você não pode inserir um novo registro ou atualizar um registro sem adicionar um valor a esse campo.

Para definir uma coluna como NOT NULL basta colocar esta cláusula na definição da coluna durante a criação da tabela ou alterar a coluna informando essa cláusula.

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado NOT NULL,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
  ....
);
OU
ALTER TABLE nome_da_tabela
MODIFY coluna2 tipo_de_dado NOT NULL;
```

EXEMPLIFICANDO!!!

Vamos supor que você deseje que os atributos IDPessoa, Sobrenome e Nome da tabela Pessoas devam sempre ser preenchidos, isto é, não podem ser NULL. Assim, você pode utilizar NOT NULL durante a criação da tabela:

```
CREATE TABLE Pessoas (
  IDPessoa int NOT NULL,
  Sobrenome varchar(255) NOT NULL,
  Nome varchar(255) NOT NULL,
  Endereco varchar(255),
  Cidade varchar(255)
);
```

Assim, ao tentar inserir dados nessa tabela, deverão sempre ser informados valores para IDPessoa, Sobrenome e Nome.

Também é possível atribuir a restrição NOT NULL na cláusula ALTER TABLE:

```
ALTER TABLE Pessoas
MODIFY Cidade varchar(255) NOT NULL;
```

UNIQUE

A restrição **UNIQUE** garante que **todos os valores em uma coluna sejam diferentes**. As restrições UNIQUE e PRIMARY KEY fornecem uma garantia de exclusividade para uma coluna ou conjunto de colunas.

Uma restrição PRIMARY KEY tem automaticamente uma restrição UNIQUE. No entanto, você pode ter muitas restrições UNIQUE por tabela, mas apenas uma restrição PRIMARY KEY por tabela.

A definição de uma coluna como UNIQUE varia de acordo com o SGBD adotado:

- **No SQL Server / Oracle / MS Access**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado UNIQUE,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
);
```

- **No MySQL**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
UNIQUE (coluna1)
);
```

- **No MySQL / SQL Server / Oracle / MS Access para uma ou múltiplas colunas**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
CONSTRAINT nome_da_restricao UNIQUE (coluna1, coluna2));
);
```

Também é possível adicionar uma restrição UNIQUE em uma cláusula ALTER:

- **Para uma coluna**:

```
ALTER TABLE nome_da_tabela
ADD UNIQUE (coluna);
```

- **Para uma ou múltiplas colunas**:

```
ALTER TABLE nome_da_tabela
ADD CONSTRAINT nome_da_restricao UNIQUE (coluna1, coluna2));
```

Para excluir uma restrição, basta utilizar a cláusula DROP:

- **No MySQL**, usa-se DROP INDEX (isso mesmo, nesse caso, utiliza-se DROP INDEX e não DROP UNIQUE):

```
ALTER TABLE nome_da_tabela
DROP INDEX nome_da_restricao;
```

- **No SQL Server / Oracle / MS Access**, usa-se DROP CONSTRAINT:

```
ALTER TABLE nome_da_tabela
DROP CONSTRAINT nome_da_restricao;
```

PRIMARY KEY

A restrição **PRIMARY KEY** **identifica exclusivamente cada registro em uma tabela**. As chaves primárias devem conter valores UNIQUE e não podem conter valores NULL, isto é, uma restrição PRIMARY KEY possui implicitamente uma restrição UNIQUE e também uma restrição NOT NULL.

Uma tabela pode ter apenas uma chave primária; e na tabela, essa chave primária pode consistir em colunas únicas ou múltiplas (campos).

A definição de uma coluna como **PRIMARY KEY** varia de acordo com o SGBD adotado:

- **No SQL Server / Oracle / MS Access**:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado PRIMARY KEY,
    coluna2 tipo_de_dado;
);
```

- **No MySQL**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
PRIMARY KEY (coluna1)
);
```

- **No MySQL / SQL Server / Oracle / MS Access para uma ou múltiplas colunas**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
CONSTRAINT nome_da_restricao PRIMARY KEY (coluna1, coluna2));
);
```

Também é possível adicionar uma restrição **PRIMARY KEY** em uma cláusula ALTER:

- **Para uma coluna**:

```
ALTER TABLE nome_da_tabela
ADD PRIMARY KEY(coluna);
```

- **Para uma ou múltiplas colunas**:

```
ALTER TABLE nome_da_tabela
ADD CONSTRAINT nome_da_restricao PRIMARY KEY (coluna1, coluna2));
```

Para excluir uma restrição, basta utilizar a cláusula DROP:

- **No MySQL**:

```
ALTER TABLE nome_da_tabela
DROP PRIMARY KEY;
```

- **No SQL Server / Oracle / MS Access**:

```
ALTER TABLE nome_da_tabela
DROP CONSTRAINT nome_da_restricao;
```

FOREIGN KEY

Uma **FOREIGN KEY** é uma **chave usada para unir duas tabelas**, sendo um campo (ou conjunto de campos) em uma tabela que se refere à PRIMARY KEY em outra tabela.

A tabela que contém a chave estrangeira é chamada de tabela filha e a tabela que contém a chave candidata é chamada de tabela de referência ou pai.

A definição de uma coluna como **FOREIGN KEY** varia de acordo com o SGBD adotado:

- **No SQL Server / Oracle / MS Access**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado PRIMARY KEY,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado FOREIGN KEY REFERENCES tabela_referenciada(chave),
);
```

- **No MySQL**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
PRIMARY KEY (coluna1),
FOREIGN KEY (coluna2) REFERENCES tabela_referenciada (chave)
);
```

- **No MySQL / SQL Server / Oracle / MS Access para uma ou múltiplas colunas**:

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo_de_dado,
  coluna2 tipo_de_dado,
  coluna3 tipo_de_dado,
CONSTRAINT nome_da_restricao FOREIGN KEY (coluna1, coluna2)) REFERENCES tabela_referenciada (chave1, chave2);
);
```

Também é possível adicionar uma restrição **FOREIGN KEY** em uma cláusula ALTER:

- **Para uma coluna**:

ALTER TABLE nome_da_tabela

ADD FOREIGN KEY(coluna) **REFERENCES** tabela_referenciada (chave)**;**

- **Para uma ou múltiplas colunas**:

ALTER TABLE nome_da_tabela

ADD CONSTRAINT nome_da_restricao **FOREIGN KEY**(coluna1, coluna2) **REFERENCES** tabela_referenciada (chave1, chave2);

Para excluir uma restrição, basta utilizar a cláusula DROP:

- **No MySQL**:

ALTER TABLE nome_da_tabela

DROP FOREIGN KEY coluna;

- **No SQL Server / Oracle / MS Access**:

ALTER TABLE nome_da_tabela

DROP CONSTRAINT nome_da_restricao;

ATENÇÃO!!!

As chaves estrangeiras podem ser criadas com a definição de cláusulas de exclusão ou atualização em cascata. Vejamos:

Ao usar a opção **ON DELETE CASCADE**, quando um registro da tabela que possui a chave primária associada a esta chave estrangeira for excluído, então os registros associados também são excluídos. Ex.: ao excluir um determinado País, todos os Estados que estão associados àquele País são também deletados (a associação é verificada pela chave estrangeira).

CONSTRAINT nome_da_restricao **FOREIGN KEY** (coluna1, coluna2)) **REFERENCES** tabela_referenciada (chave1, chave2) **ON DELETE CASCADE**;

Ao usar a opção **ON UPDATE CASCADE**, quando um registro da tabela que possui a chave primária associada a esta chave estrangeira for alterado, então os registros associados também são alterados.

CHECK

A restrição **CHECK** é **usada para limitar o intervalo de valores que pode ser colocado em uma coluna**.

Se você definir uma restrição **CHECK** em uma única coluna, ela permitirá apenas determinados valores para essa coluna.

Se você definir uma restrição **CHECK** em uma tabela, ela poderá limitar os valores em determinadas colunas com base nos valores de outras colunas na linha.

A definição restrição **CHECK** varia de acordo com o SGBD adotado:

- **No SQL Server / Oracle / MS Access**:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado,
    coluna2 tipo_de_dado CHECK (condicao),
    coluna3 tipo_de_dado,
);
```

- **No MySQL**:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado,
    coluna2 tipo_de_dado,
    coluna3 tipo_de_dado,
    CHECK (condicao)
);
```

- **No MySQL / SQL Server / Oracle / MS Access para uma ou múltiplas colunas**:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado,
    coluna2 tipo_de_dado,
    coluna3 tipo_de_dado,
CONSTRAINT nome_da_restricao CHECK (condicao1 AND condicao2));
);
```

Também é possível adicionar uma restrição CHECK em uma cláusula ALTER:

- **Para uma condição**:

```
ALTER TABLE nome_da_tabela
ADD CHECK (condicao);
```

- **Para uma ou múltiplas condições**:

```
ALTER TABLE nome_da_tabela
ADD CONSTRAINT nome_da_restricao CHECK (condicao1 AND condicao2));
```

Para excluir uma restrição, basta utilizar a cláusula DROP:

- **No MySQL**:

```
ALTER TABLE nome_da_tabela
DROP CHECK nome_da_restricao;
```

- **No SQL Server / Oracle / MS Access**:

```
ALTER TABLE nome_da_tabela
DROP CONSTRAINT nome_da_restricao;
```

EXEMPLIFICANDO!!!

Vamos supor que você deseje definir que todos as Pessoas de sua tabela devem ter uma idade superior a 18 anos. Então você pode colocar essa regra em uma cláusula CHECK:

```
CREATE TABLE Pessoas (
    IDPessoa int NOT NULL,
    Sobrenome varchar(255) NOT NULL,
    Nome varchar(255) NOT NULL,
    Endereco varchar(255),
    Cidade varchar(255),
    Idade int CHECK (Idade>18)
);
```

Assim, somente será possível cadastrar pessoas com idade maior que 18 anos.

DEFAULT

A restrição **DEFAULT** é usada para **fornecer um valor padrão para uma coluna**. O valor padrão será adicionado a todos os novos registros SE nenhum outro valor for especificado.

A definição restrição **DEFAULT** pode ser realizada com a seguinte sintaxe:

```
CREATE TABLE nome_da_tabela (
    coluna1 tipo_de_dado,
    coluna2 tipo_de_dado DEFAULT valor,
    coluna3 tipo_de_dado,
);
```

Também é possível adicionar uma restrição DEFAULT em uma cláusula ALTER:

- **No MySQL**:

```
ALTER TABLE nome_da_tabela
ALTER coluna SET DEFAULT valor;
```

- **No SQL Server**:

```
ALTER TABLE nome_da_tabela
ADD CONSTRAINT nome_da_restricao DEFAULT valor;
```

- **No Oracle**:

```
ALTER TABLE nome_da_tabela
MODIFY coluna DEFAULT valor;
```

Para excluir uma restrição, basta utilizar a cláusula DROP:

- **No MySQL**:

```
ALTER TABLE nome_da_tabela
ALTER coluna DROP DEFAULT;
```

- **No SQL Server / Oracle / MS Access**:

```
ALTER TABLE nome_da_tabela
ALTER COLUMN coluna DROP DEFAULT;
```

14- (FCC - 2019 - SEFAZ-BA - Auditor Fiscal - Tecnologia da Informação - Prova II) Em um banco de dados aberto e em condições ideais há uma tabela chamada Contribuinte cuja chave primária é idContribuinte. Há também uma tabela chamada Imposto cuja chave primária é idimposto. Para criar uma tabela de associação chamada Contribuinte_imposto cuja chave primária é composta pelos campos idContribuinte e idImposto, que são chaves estrangeiras resultantes da relação dessa tabela com as tabelas Contribuinte e Imposto, utiliza-se a instrução SQL

a) CREATE TABLE Contribuinte_Imposto(idContribuinte INT, idImposto INT, PRIMARY KEY (idContribuinte), FOREIGN KEY (idContribuinte) REFERENCES Contribuinte (idContribuinte), PRIMARY KEY (idImposto), FOREIGN KEY (idContribuinte) REFERENCES Contribuinte (idContribuinte));

b) CREATE TABLE Contribuinte_Imposto(idContribuinte INT NOT NULL, idImposto INT NOT NULL, PRIMARY KEY (idContribuinte, idImposto), CONSTRAINT fk1 FOREIGN KEY (idContribuinte) REFERENCES Contribuinte (idContribuinte), CONSTRAINT fk2 FOREIGN KEY (idImposto) REFERENCES Imposto (idImposto));

c) CREATE TABLE Contribuinte_Imposto(idContribuinte INT NOT NULL, idImposto INT NOT NULL, PRIMARY KEY (idContribuinte, idImposto), FOREIGN KEY (idContribuinte) SOURCE Contribuinte (idContribuinte), FOREIGN KEY (idImposto) SOURCE Imposto (idImposto));

d) CREATE TABLE Contribuinte_Imposto(idContribuinte INT NOT NULL, idImposto INT NOT NULL, PRIMARY KEY (idContribuinte, idImposto), FOREIGN KEY (idContribuinte, idImposto) REFERENCES (Contribuinte!idContribuinte, Imposto!idImposto));
e) CREATE TABLE Contribuinte_Imposto(idContribuinte INT NOT NULL, idImposto INT NOT NULL, PRIMARY KEY (idContribuinte, idImposto), FOREIGN KEY (idContribuinte, idImposto) REFERENCES all parents);

Resolução:

Vamos analisar cada um dos itens:

a) **Incorreto**: a restrição PRIMARY KEY deve ser única na tabela. Caso a chave fosse simples, poderia ser adotada a definição PRIMARY KEY (atributo).

b) **Correto**: esse comando realiza a criação da tabela conforme solicitado no item. Vamos explicar o comando por partes:

Criação da tabela com os dois atributos solicitados:

CREATE TABLE Contribuinte_Imposto(idContribuinte INT NOT NULL, idImposto INT NOT NULL,

Definição da chave primária formada pelos dois atributos:

PRIMARY KEY (idContribuinte, idImposto),

Definição das chaves estrangeiras:

CONSTRAINT fk1 FOREIGN KEY (idContribuinte) REFERENCES Contribuinte (idContribuinte), CONSTRAINT fk2 FOREIGN KEY (idImposto) REFERENCES Imposto (idImposto));

Nesse caso temos a definição de duas restrições fk1 e fk2, uma para cada parte da chave estrangeira.

c) **Incorreto**: para definir uma chave primária com mais de um atributo, deve-se inserir uma **CONSTRAINT**. Ademais, para relacionar uma chave estrangeira com a tabela referenciada deve-se utilizar a palavra **REFERENCES** e não **SOURCE**.

d) **Incorreto**: para definir uma chave primária com mais de um atributo, deve-se inserir uma **CONSTRAINT**. Ademais, como os atributos relacionados por meio da chave estrangeira são de tabelas diferentes, deve-se utilizar mais de uma cláusula **FOREIGN KEY**.

e) **Incorreto**: mesma justificativa do item d, além de não existir essa referência a all parents, pois devem ser informados as tabelas e atributos sendo referenciados de acordo com a sintaxe **nome_da_tabela (atributo)**.

Gabarito: Letra B.

15- (Quadrix - 2019 - CRESS - SC - Assistente de Comunicação e Tecnologia

Código 1:

```
CREATE TABLE Assistente_Social
(
   ID_Func      NUMERIC(4)    NOT NULL,
   NomeFunc     VARCHAR(30)   NOT NULL,
   Endereco     VARCHAR(50)   NOT NULL,
   DataNasc     DATE          NOT NULL,
   Sexo         CHAR(1)       NOT NULL,
   Salario      NUMERIC(8,2)  NOT NULL,
   ID_Depto     NUMERIC(2)    NOT NULL,
   CONSTRAINT pk_func PRIMARY KEY (ID_Func),
   CONSTRAINT ck_sexo CHECK (Sexo='M' or Sexo='F')
);
```

Código 2:

```
SELECT ID_Depto, AVG(Salario)
FROM Assistente_Social
WHERE AVG(Salario) > 5000
GROUP BY ID_Depto;
```

No que diz respeito aos códigos 1 e 2 da linguagem SQL acima apresentados, julgue o item.

A instrução CREATE TABLE está especificada da forma correta no Código 1.

Resolução:

O comando do Código 1 está totalmente correto. Através desse comando, há a criação da tabela Assistente_Social com os atributos ID_Func, NomeFunc, Endereco, DataNasc, Sexo, Salario e ID_Depto.

Foram definidas também duas restrições:

- CONSTRAINT pk_func PRIMARY KEY (ID_Func) define uma restrição de chave primária, sendo o ID_Func o campo utilizado como chave.
- CONSTRAINT ck_seco CHECK (Sexo='M' or Sexo='F') define uma restrição de checagem, na qual o atributo sexo deve sempre um caractere 'M' ou 'F'.

Gabarito: **Certo**.

16- (CESPE - 2016 - TCE-PA - Auditor de Controle Externo - Área Informática - Analista de Sistema) Julgue o próximo item, relativo à linguagem de definição de dados (DDL).

A expressão DDL abaixo cria a tabela referente ao diagrama de entidade e relacionamento apresentado a seguir.

create table tribunal(

tribunal_codigo integer ,

tribunal_descricao varchar(100),

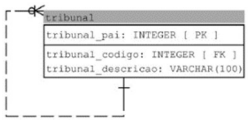
tribunal_pai integer primary key,

constraint fk_tribunal

foreign key (tribunal_codigo)

references tribunal

)



Resolução:

Vamos analisar o comando por partes:

Criação da tabela tribunal:

CREATE TABLE tribunal (

Com os atributos tribunal_codigo, tribunal_descricao e tribunal_pai, sendo este definido como chave primária:

tribunal_codigo integer,
tribunal_descricao varchar(100),
tribunal_pai integer **PRIMARY KEY**,

Com uma restrição de chave estrangeira que faz referência a própria tabela, isto é, declara um autorrelacionamento:

CONSTRAINT fk_tribunal

FOREIGN KEY (tribunal_codigo)

REFERENCES tribunal

)

Gabarito: **Certo**.

17- (CESPE - 2016 - POLÍCIA CIENTÍFICA - PE - Perito Criminal - Ciência da Computação) Em SQL, para alterar a estrutura de uma tabela do banco de dados e incluir nela uma nova foreign key, é correto utilizar o comando

a) convert

b) group by.

c) alter table.

d) update.

e) insert.

Resolução:

A instrução **ALTER TABLE** é usada para **adicionar, deletar ou modificar colunas em uma tabela existente**. Essa instrução também pode ser utilizada para adicionar ou deletar restrições a esta tabela.

Para adicionar uma restrição **FOREIGN KEY** em uma cláusula ALTER:

- **Para uma coluna**:

ALTER TABLE nome_da_tabela

ADD FOREIGN KEY(coluna) **REFERENCES** tabela_referenciada (chave)**;**

- **Para múltiplas colunas**:

ALTER TABLE nome_da_tabela

ADD CONSTRAINT nome_da_restricao **FOREIGN KEY**(coluna1, coluna2) **REFERENCES** tabela_referenciada (chave1, chave2);

Gabarito: **Letra C**.

18- (CESPE - 2016 - POLÍCIA CIENTÍFICA - PE - Perito Criminal - Ciência da Computação) Na linguagem SQL, o comando create table é usado para criar uma tabela no banco de dados; enquanto o relacionamento entre duas tabelas pode ser criado pela declaração

a) null.

b) primary key.

c) constraint.

d) auto_increment.

e) not null.

Resolução:

O relacionamento é realizado através da chave estrangeira ou foreign key. A chave estrangeira pode ser definida em uma cláusula CONSTRAINT da seguinte forma:

CONSTRAINT nome_da_restricao **FOREIGN KEY** (coluna1, coluna2)) **REFERENCES** tabela_referenciada (chave1, chave2);

Gabarito: **Letra C**.

19- (CESPE - 2015 - MEC – Desenvolvedor)

```
CREATE TABLE PESSOA (
ID INTEGER NOT NULL,
NOME CHAR(50) NOT NULL UNIQUE,
CPF DECIMAL (11,0) NULL,
NACIONALIDADE INTEGER NOT NULL,
PRIMARY KEY (ID),
FOREIGN KEY (NACIONALIDADE)
REFERENCES TABELA_NACIONALIDADE(CODIGO_NACIONALIDADE)
);
```

Com base no comando SQL apresentado, julgue o item subsequente.

A cláusula NULL na coluna CPF indica que o conteúdo dessa coluna pode ser zero, já que ela é do tipo DECIMAL (11,0).

Resolução:

A cláusula NULL indica que o conteúdo de CPF pode ser nulo. Nulo é diferente de zero.

Ao comparar qualquer coisa com NULL usando os operadores lógicos comuns, será retornado um resultado desconhecido na comparação (UNKNOWN).

Gabarito: **Errado.**

20- (CESPE - 2015 - MEC – Desenvolvedor) CREATE TABLE PESSOA (

ID INTEGER NOT NULL,

NOME CHAR(50) NOT NULL UNIQUE,

CPF DECIMAL (11,0) NULL,

NACIONALIDADE INTEGER NOT NULL,

PRIMARY KEY (ID),

FOREIGN KEY (NACIONALIDADE)

REFERENCES TABELA_NACIONALIDADE(CODIGO_NACIONALIDADE)

);

Com base no comando SQL apresentado, julgue o item subsequente.

Mais de uma PESSOA pode ter o mesmo NOME e a mesma NACIONALIDADE.

Resolução:

O NOME da PESSOA está definido como UNIQUE, logo não pode ser repetido para mais de um registro.

A restrição **UNIQUE** garante que **todos os valores em uma coluna sejam diferentes**.

Gabarito: **Errado.**

21- (FCC - 2015 - TRT - 9ª REGIÃO (PR) - Analista Judiciário - Área Apoio Especializado - Tecnologia da Informação)

A tabela relativa a Débitos Trabalhistas a seguir deve ser utilizada para responder à questão.

Considere que a tabela já está criada, os dados iniciais já foram inseridos e o banco de dados a ser utilizado está aberto e funcionando em condições ideais.

Tabela DebTrab

NroProcesso	Principal	Juros	FGTS	Honor Periciais
111/15	25345.00	3801.75	7933.00	4755.00
777/15	125800.00	18870.00	57966.87	7543.00
333/15	8844.50	1326.67	4233.55	1781.00
555/15	327631.00	65526.20	104863.78	11523.00
444/15	5072.00	1014.40	895.14	700.00

Um Analista da área de TI trabalha em uma organização que possui aplicações que utilizam os SGBDs Oracle 11g e SQL Server. Ele identificou que o comando SQL que está correto e pode ser aplicado em ambas as plataformas é

a) ALTER TABLE DebTrab ALTER COLUMN NroProcesso integer;

b) ALTER TABLE DebTrab MODIFY NroProcesso int;

c) ALTER TABLE DebTrab ADD DataPartida data;

d) ALTER TABLE DebTrab ADD IndiceAtualiz float;

e) ALTER TABLE DebTrab DROP COLUMN DataPartida;

Resolução:

Vamos analisar cada um dos itens:

a) **Incorreto**: **ALTER COLUMN** pode ser usado no SQL Server, mas não no Oracle.

b) **Incorreto**: **MODIFY** é usado somente a partir da versão 10g do Oracle.

c) **Incorreto**: o tipo de dados "data" não existe. O correto seria **"date"**.

d) **Correto**: para **adicionar uma coluna**, usamos a cláusula **ADD**:

```
ALTER TABLE nome_da_tabela
ADD nome_da_coluna tipo_de_dado;
```

e) **Incorreto**: no modelo apresentado, a tabela DebTrab não possui nenhuma coluna DataPartida e, portanto, não é possível deletar algo que não existe.

Gabarito: **Letra D**.

1.4 Trabalhando com Visões

Criando uma visão

A sintaxe a seguir é utilizada para **criar uma view** em SQL:

```
CREATE VIEW [Nome da View] AS
  SELECT Coluna1, Coluna2,…
  FROM nome_da_tabela
  WHERE...;
```

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Se quisermos criar uma view para os clientes mexicanos, podemos usar a seguinte sintaxe:

```
CREATE VIEW [Mexicanos] AS
  SELECT Nome_Cliente, Cidade
  FROM Clientes
  WHERE Pais=”Mexico”;
```

Para consultar dos dados da visão basta usar o comando a seguir:

```
SELECT * FROM [Mexicanos];
```

O resultado será:

Nome_Cliente	Cidade
Ana Trujillo Emparedados y helados	México D.F.
Antonio Moreno Taquería	México D.F.

Alterando uma visão

Uma visão pode ser **atualizada** com o comando **CREATE OR REPLACE VIEW**:

```
CREATE OR REPLACE VIEW [Nome da View] AS
  SELECT Coluna1, Coluna2,…
  FROM nome_da_tabela
  WHERE...;
```

Essa instrução na verdade poderá executar duas ações:

1. Caso a visão **já exista**, então ela será alterada.
2. Caso a visão ainda **não exista**, então ela será criada.

EXEMPLIFICANDO!!!

Dada a visão [Mexicanos] criada anteriormente, se quisermos alterar a visão para apresentar também CEP dos clientes, basta alterar a visão usando a cláusula **CREATE OR REPLACE VIEW**:

```
CREATE OR REPLACE VIEW [Mexicanos] AS
  SELECT Nome_Cliente, Cidade, CEP
  FROM Clientes
  WHERE Pais="Mexico";
```

Para consultar dos dados da visão basta usar o comando a seguir:

```
SELECT * FROM [Mexicanos];
```

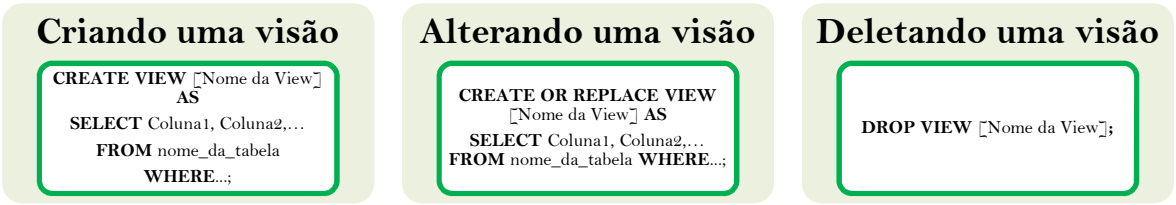
O resultado será:

Nome_Cliente	Cidade	CEP
Ana Trujillo Emparedados y helados	México D.F.	05021
Antonio Moreno Taquería	México D.F.	05023

Excluindo uma visão

Uma visão é **apagada** com o comando **DROP VIEW**:

```
DROP VIEW [Nome da View];
```



Esquema 5 – Trabalhando com visões.

22- (FCC - 2017 - DPE-RS - Analista - Banco de Dados) O comando SQL para criar uma visão V1, a partir de uma tabela T1, obtendo os atributos A1, A2 e A3 e os renomeando para C1, C2 e C3 é:

a) CREATE VIEW V1.C1, V1.C2. V1.C3

SELECT T1.A1, T1.A2, T1.A3;

b) CREATE VIEW V1 (C1, C2, C3)

AS SELECT A1, A2, A3

FROM T1;

c) CREATE VIEW C1, C2, C3 IN V1

FROM A1, A2, A3 OF T1;

d) CREATE VIEW V1

FROM T1

SELECT A1 → C1, A2 → C2, A3 → C3;

e) CREATE VIEW V1 (C1, C2, C3)

AS PART OF T1 (A1, A2, A3);

Resolução:

A sintaxe a seguir é utilizada para **criar uma view** em SQL:

CREATE VIEW [Nome da View] **AS**

SELECT Coluna1, Coluna2,…

FROM nome_da_tabela

WHERE...;

Assim, vamos definir o comando para o solicitado na questão por partes:

Definição do nome da visão:

CREATE VIEW V1 **AS**

Seleção dos atributos A1, A2 e A3 da tabela T1:

SELECT A1, A2, A3 **FROM** T1;

Perceba que para essa seleção não foi definida nenhuma condição e, portanto, não precisamos utilizar a cláusula WHERE.

Contudo, para o comando ficar correto precisamos renomear os atributos na definição da visão. Para isso basta informar os novos nomes entre parênteses após o nome da visão.

CREATE VIEW V1(C1, C2, C3) **AS**

Gabarito: **Letra B**.

23- (VUNESP - 2014 - PRODEST-ES - Analista de Tecnologia da Informação - Desenvolvimento de Sistemas) Considere a tabela T de um banco de dados relacional:

T (ID, Nome, Fone)

Indique a alternativa que contém a consulta SQL correta para criar uma visão V, a partir da tabela T, apenas para os Nomes começando pela letra J.

a) CREATE VIEW V FOR (SELECT T.ID, T.Nome, T. Fone FOR Nome NEXT 'J%')

b) CREATE VIEW V → (SELECT T(ID, Nome, Fone) WHERE Nome NEAR 'J%')

c) CREATE VIEW V (SELECT ID, Nome, Fone FROM T WHERE Nome = 'J%')

d) CREATE VIEW V AS (SELECT ID, Nome, Fone FROM T WHERE Nome LIKE 'J%')

e) CREATE VIEW V FROM (SELECT ID, Nome, Fone OF T WHERE Nome IN 'J%')

Resolução:

A sintaxe a seguir é utilizada para **criar uma view** em SQL:

CREATE VIEW [Nome da View] **AS**

SELECT Coluna1, Coluna2,…

FROM nome_da_tabela

WHERE...;

Logo, podemos rapidamente identificar que somente o item d) está de acordo com essa sintaxe, pois é o único que utiliza a palavra-chave **AS**.

Para selecionar somente os Nomes que começam com a letra J, usa-se o operador LIKE. Em 'J%', temos a condição desejada.

Assim, em **CREATE VIEW** V **AS** (**SELECT** ID, Nome, Fone **FROM** T **WHERE** Nome **LIKE** 'J%') será criada uma visão com base no retorno da consulta prevista no SELECT, que retorna o ID, o Nome e o Fone da tabela T dos registros que possuem Nome começando com J.

Gabarito: **Letra D**.

24- (CESPE - 2016 - TCE-PA - Auditor de Controle Externo - Área Informática - Analista de Suporte) No que concerne à linguagem SQL, julgue o item seguinte.

Ao se criar uma view, não é necessário que os nomes dos atributos da view sejam os mesmos dos atributos da tabela.

Resolução:

Os nomes dos atributos nas visões podem ser definidos com bases nos alias ou apelidos. Sendo assim, não é necessário que o nome dos atributos da visão sejam os mesmos.

Gabarito: **Certo.**

1.5 Trabalhando com Índices

Criando um índice

A sintaxe a seguir é utilizada para **criar um índice** em SQL:

```
CREATE INDEX nome_do_indice
ON nome_da_tabela (coluna1, coluna2, ...);
```

Também é possível criar um índice único, isto é, em que não são permitidos valores duplicados. Para isso, utiliza-se CREATE UNIQUE INDEX:

```
CREATE UNIQUE INDEX nome_do_indice
ON nome_da_tabela (coluna1, coluna2, ...);
```

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Se quisermos criar um índice para a cidade, podemos usar a seguinte sintaxe:

```
CREATE INDEX index_cidade
 ON SELECT Clientes (Cidade);
```

Se quisermos criar um índice ÚNICO para a cidade, podemos usar a seguinte sintaxe:

```
CREATE UNIQUE INDEX index_cidade
 ON SELECT Clientes (Cidade);
```

Alterando um índice

Um índice pode ser **atualizado** com o comando **ALTER INDEX**:

ALTER INDEX nome_do_indice

ON nome_da_tabela (coluna1, coluna2, ...);

ATENÇÃO!!!

A sintaxe completa para a alteração de um índice envolve diversas cláusulas específicas do SGBD. Para nós, isso não importa, basta saber que é possível alterar um índice através de ALTER INDEX.

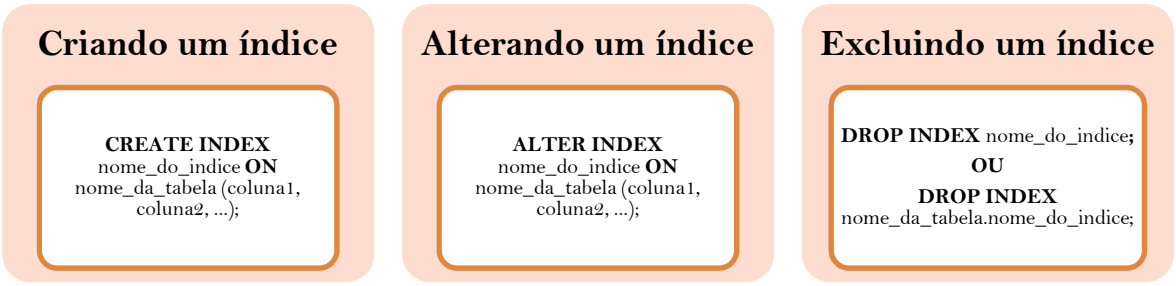
Excluindo um índice

Um índice pode ser **excluído** com o comando **DROP INDEX**:

DROP INDEX nome_do_indice**;**

OU

DROP INDEX nome_da_tabela.nome_do_indice;



Esquema 6 – Trabalhando com índices.

25- (FGV - 2012 - Senado Federal - Analista Legislativo - Análise de Suporte de Sistemas) A DDL da SQL descreve como as tabelas e outros objetos Oracle podem ser definidos, alterados e removidos. De um modo geral, é a parte utilizada pelo DBA. O comando que elimina um índice já criado é

a) REMOVE INDEX

b) DELETE INDEX

c) PURGE INDEX

d) ERASE INDEX

e) DROP INDEX

Resolução:

O comando DROP é o único DDL dentre os itens e é o comando utilizado para excluir estruturas de bancos de dados. Para excluir índices, então usamos DROP INDEX.

Gabarito: **Letra E**.

26- (FUNCAB - 2012 - MPE-RO - Analista de Sistemas) Na criação de uma tabela em um banco de dados MySQL, o parâmetro UNIQUE do comando CREATE INDEX:

a) define a chave estrangeira.

b) define a chave primária.

c) garante a unicidade de um registro.

d) determina a ordem física das linhas correspondentes em uma tabela.

e) determina a direção de classificação de uma determinada coluna.

Resolução:

O parâmetro UNIQUE em um índice garante que não haverá duplicidade nos registros.

Para isso, utiliza-se CREATE UNIQUE INDEX:

```
CREATE UNIQUE INDEX nome_do_indice
ON nome_da_tabela (coluna1, coluna2, ...);
```

Gabarito: **Letra C**.

27- (CESPE - 2008 - STJ - Técnico Judiciário - Informática) Acerca da linguagem SQL, usada para fazer a manipulação e a definição de dados em sistemas gerenciadores de banco de dados, julgue os itens subsequentes.

O comando CREATE INDEX, usado para criar um parâmetro relacionado com uma tabela para buscar dados mais rapidamente, é considerado como DDL.

Resolução:

Sim, o comando CREATE INDEX é comando DDL. Esse comando permite criar um índice e possui a seguinte sintaxe básica.

```
CREATE INDEX nome_do_indice
ON nome_da_tabela (coluna1, coluna2, ...);
```

Gabarito: **Certo**.

1.6 TEMA AVANÇADO: Trabalhando com Procedures

DICA DO PROFESSOR!!!

Pessoal, trago uma rápida discussão sobre a sintaxe básica das procedures, mas informo que esse tema pode ir muito além do que vou expor aqui. Esse é um tema que é pouco cobrado nas questões, mas trago para que pelo menos você tenha uma noção de como se realiza a declaração e execução de uma procedure. Não gaste tempo aprofundando esse tema.

Uma **STORED PROCEDURE** é um **código SQL preparado que você pode salvar, para que o código possa ser reutilizado repetidamente**. Portanto, se você tiver uma consulta SQL que você escreve repetidas vezes, salve-a como um procedimento armazenado e, em seguida, apenas chame-a para executá-la. Você também pode passar parâmetros para um procedimento armazenado, para que o procedimento armazenado possa agir com base nos valores de parâmetro que são passados.

Para criar uma **PROCEDURE**, basta utilizar a seguinte sintaxe:

```
CREATE PROCEDURE nome_da_procedure
AS
declaracoes_SQL
GO;
```

Após criar uma **PROCEDURE**, você pode executá-la simplemesnte executando uma chamada com a cláusula **EXEC**:

```
EXEC nome_da_procedure
```

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Se quisermos criar uma PROCEDURE retornar todos os clientes, podemos usar a seguinte sintaxe:

```
CREATE PROCEDURE TodosOsClientes AS
  SELECT * FROM Clientes;
```

Agora para consultar rapidamente os dados de todos os clientes, basta chamar a Procedure:

```
EXEC TodosOsClientes;
```

Aqui demos um exemplo simples, mas as consultas que estão armazenadas em procedures podem ser bastante complexas e, assim, economizar bastante o tempo de escrita de códigos.

Uma PROCEDURE aceita a definição de parâmetros que podem ser passados quando de sua chamada. Assim,

```
CREATE PROCEDURE nome_da_procedure @parametro1 tipo_de_dado, @parametro2 tipo_de_dado,...
AS
Declarações SQL...
GO;
```

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Se quisermos criar uma PROCEDURE que receba como parâmetros o nome da cidade e o CEP desejado, podemos utilizar a sintaxe seguinte:

```
CREATE PROCEDURE TodosOsClientes @Cidade nvarchar(30), @CEP nvarchar(10)
AS
SELECT * FROM Clientes WHERE Cidade = @Cidade AND CEP = @CEP
GO;
```

Com essa PROCEDURE, agora podemos facilmente consultar clientes de cidades e CEPs específicos. Por exemplo, se desejarmos consultar os clientes de Berlin com CEP 12209, basta executar a PROCEDURE com os devidos parâmetros:

```
EXEC TodosOsClientes City = "Berlin", CEP = "12209";
```

O resultado será:

IDCliente	Nome_Cliente	Nome_Conhecido	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

28- (FCC - 2018 - DPE-AM - Analista em Gestão Especializado de Defensoria - Analista de Banco de Dados) O comando SQL–ANSI para criar um procedimento chamado P1, que selecione os atributos A e B, de uma tabela T é:

a) PROCEDURE P1 IS

SELECT A, B

FROM T;

b) INSERT PROCEDURE P1 INTO DATABASE AS

SELECT A, B

FROM T

c) CREATE PROCEDURE P1( )

SELECT A, B

FROM T;

d) MAKE PROCEDURE P1 (SELECT A, B

FROM T);

e) PROCEDURE P1 AS

SELECT A, B

FROM T

Resolução:

Para criar uma **PROCEDURE**, basta utilizar a seguinte sintaxe:

CREATE PROCEDURE nome_da_procedure

AS

declaracoes_SQL

GO;

Nesse caso, como queremos criar a PROCEDURE P1 com base na seleção dos atributos A e B da tabela T, usamos:

CREATE PROCEDURE P1()

AS

SELECT A, B

FROM T;

GO;

Os () vazios indicam que a PROCEDURE não recebe nenhum parâmetro.

Gabarito: **Letra C**.

1.7 TEMA AVANÇADO: Trabalhando com Triggers

DICA DO PROFESSOR!!!

Trago uma rápida discussão sobre a sintaxe básica das triggers, mas informo que esse tema pode ir muito além do que vou expor aqui. Esse é um tema pouco cobrado nas questões, mas trago para que pelo menos você tenha uma noção de como se realiza a declaração e quais os parâmetros que podem ser usados. Não gaste tempo aprofundando esse tema.

Triggers ou gatilhos são **programas armazenados que são executados ou disparados automaticamente quando alguns eventos ocorrem**.

Os triggers são, de fato, escritos para serem executados em resposta a qualquer um dos seguintes eventos:

- Uma instrução de manipulação de banco de dados (DML) (DELETE, INSERT ou UPDATE)
- Uma instrução de definição de banco de dados (DDL) (CREATE, ALTER ou DROP).
- Uma operação de banco de dados (SERVERERROR, LOGON, LOGOFF, STARTUP ou SHUTDOWN).

A sintaxe para definição de triggers varia de acordo com o SGDB. Vamos ver um exemplo de sintaxe considerando o Oracle: **PROCEDURE**

```
CREATE [OR REPLACE] TRIGGER nome_da_trigger
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF coluna]
ON tabela
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condicao)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Nessa sintaxe podemos destacar os seguintes elementos:

- A opção **[OR REPLACE]** permite a modificação de uma TRIGGER caso ela já exista.
- **{BEFORE | AFTER | INSTEAD OF}**: especifica quando o trigger será executado. BEFORE executa a trigger antes do evento. AFTER executa depois

do evento. A cláusula INSTEAD OF é usada para criar uma trigger em uma VIEW.

- **{INSERT [OR] | UPDATE [OR] | DELETE}**: especifica a operação DML.
- **[OF coluna]**: especifica o nome da coluna que será atualizada.
- **[ON tabela]**: especifica o nome da tabela associada a trigger.
- **[REFERENCING OLD AS o NEW AS n]**: permite que você indique valores novos e antigos para várias instruções DML, como INSERT, UPDATE e DELETE. Neste caso, os valores antigos são referenciados pelo o e os novos por n. Os valores antigos também podem ser acessados por :old.nome_do_campo e os novos com :new.nome_do_campo.
- **[FOR EACH ROW]**: especifica um acionador em nível de linha, ou seja, o acionador será executado para cada linha afetada. Caso contrário, o acionador será executado apenas uma vez quando a instrução SQL for executada, o que é chamado de acionador de nível de tabela.
- **WHEN (condição)**: fornece uma condição para as linhas para as quais o acionador dispararia. Esta cláusula é válida apenas para acionadores de nível de linha.

EXEMPLIFICANDO!!!

O trecho a seguir apresenta um exemplo de Trigger:

```
CREATE OR REPLACE TRIGGER exibir_mudancas_de_salario
BEFORE DELETE OR INSERT OR UPDATE ON clientes
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_dif number;
BEGIN
  sal_dif := :NEW.salario  - :OLD.salario;
  dbms_output.put_line('Salário antigo: ' || :OLD.salario);
  dbms_output.put_line('Salário novo: ' || :NEW.salario);
  dbms_output.put_line('Diferença salarial: ' || sal_dif);
END;
```

Neste exemplo, a Trigger é criada para ser executada sempre antes (BEFORE) de uma instrução DELETE, INSERT ou UPDATE na tabela clientes. Ela será chamada para cada linha afetada pela consulta, pois foi usada a clásula FOR EACH ROW.

29- (VUNESP - 2014 - EMPLASA - Analista Administrativo - Tecnologia da Informação) Considerando o SQL, o formato geral do comando de criação de gatilhos é:

CREATE TRIGGER < nome do trigger>

< tempo de ação do trigger>

< evento para acionar o trigger>

ON < nome da tabela>

< ação>

O parâmetro < tempo de ação do trigger > possui as seguintes opções válidas:

a) BEFORE e AFTER.

b) BEGIN e END.

c) FIRST e LAST

d) SAME e DIFFERENT.

e) START e FINISH.

Resolução:

A sintaxe para definição de triggers varia de acordo com o SGDB. Vamos ver um exemplo de sintaxe considerando o Oracle:

```
CREATE [OR REPLACE] TRIGGER nome_da_trigger
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF coluna]
ON tabela
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condicao)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

O tempo de ação das triggers pode ser especificado por:

{BEFORE | AFTER | INSTEAD OF}: especifica quando o trigger será executado. BEFORE executa a trigger antes do evento. AFTER executa depois do evento. A cláusula INSTEAD OF é usada para criar uma trigger em uma VIEW.

Gabarito: Letra A.

1.8 TEMA AVANÇADO: Trabalhando com Functions

DICA DO PROFESSOR!!!

Trago uma rápida discussão sobre a sintaxe básica das functions, mas informo que esse tema pode ir muito além do que vou expor aqui. Esse é um tema que varia bastante com base no SGBD específico sendo utilizado e não vale a pena de ser aprofundado. Sendo assim, foque em ter uma noção geral, mas não se preocupe em aprofundar esse tema.

Funções ou Functions são **rotinas que retornam valores ou tabelas**. Com elas você poderá construir visões parametrizadas ou ainda construir suas próprias funções.

A sintaxe para definição de functions varia de acordo com o SGDB. Vamos ver um exemplo de sintaxe considerando o Oracle:

```
CREATE [OR REPLACE] FUNCTION nome_da_funcao
[(nome_do_parametro [IN | OUT | IN OUT] tipo [, ...])]
RETURN return_tipo_de_dados
{IS | AS}
BEGIN
  < corpo da função >
END [nome_da_funcao];
```

Nessa sintaxe podemos destacar os seguintes elementos:

- A opção **[OR REPLACE]** permite a modificação de uma FUNCTION caso ela já exista.
- A função deve conter uma instrução de retorno. A cláusula **RETURN** especifica o tipo de dados que você retornará da função.
- Geralmente usa **AS** para criar uma FUNCTION independente e **IS** para as demais.

EXEMPLIFICANDO!!!

O código a seguir apresenta um exemplo de declaração de uma FUNCTION simples:

```
CREATE OR REPLACE FUNCTION total_de_clientes
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM clientes;
  RETURN total;
END;
```

Nesse exemplo, é criada uma FUNCTION total_de_clientes que retorna um número. No corpo da função, é realizada uma contagem da quantidade de clientes da tabela clientes. Portanto, ao chamar essa função, será realizada essa consulta e retornado esse valor.

1.9 TEMA AVANÇADO: Resumo de Procedure, Trigger e Function

O esquema a seguir diferencia PROCEDURE, TRIGGER e FUNCTION:

Esquema 6 – Procedure x Trigger x Function.

30- (FUNCAB - 2014 - PRODAM-AM - Analista de Banco de Dados) A diferença básica dos conceitos de trigger e stored procedure é que, respectivamente:

a) são executadas de acordo com um evento, mas não são inclusas no banco de dados.

b) é executada de acordo com um evento; é chamada para ser executada e são inclusas no banco de dados.

c) são executadas após serem chamadas, porém a primeira não é inclusa no banco de dados.

d) são executadas após serem chamadas, porém a segunda não é inclusa no banco de dados.

e) é chamada para ser executada; é executada de acordo com um evento e não são inclusas no banco de dados.

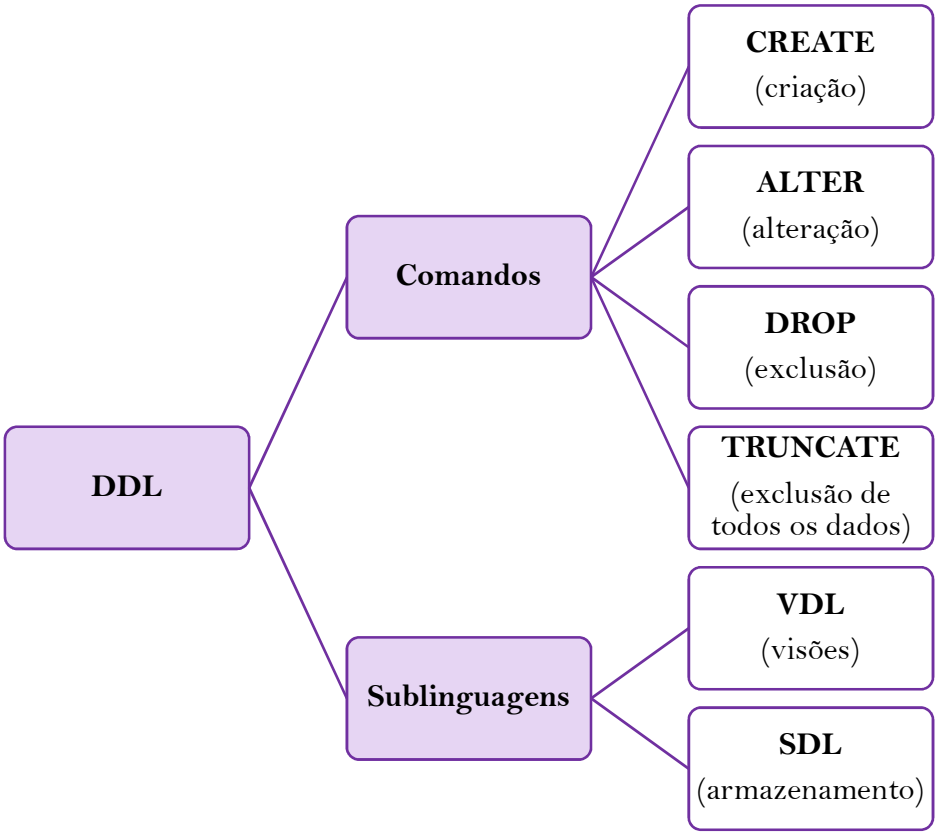
Resolução:

Uma trigger é disparada com base em um evento e a procedure pode ser executada com uma chamada.

Gabarito: **Letra B**.

2. ESQUEMAS DE AULA

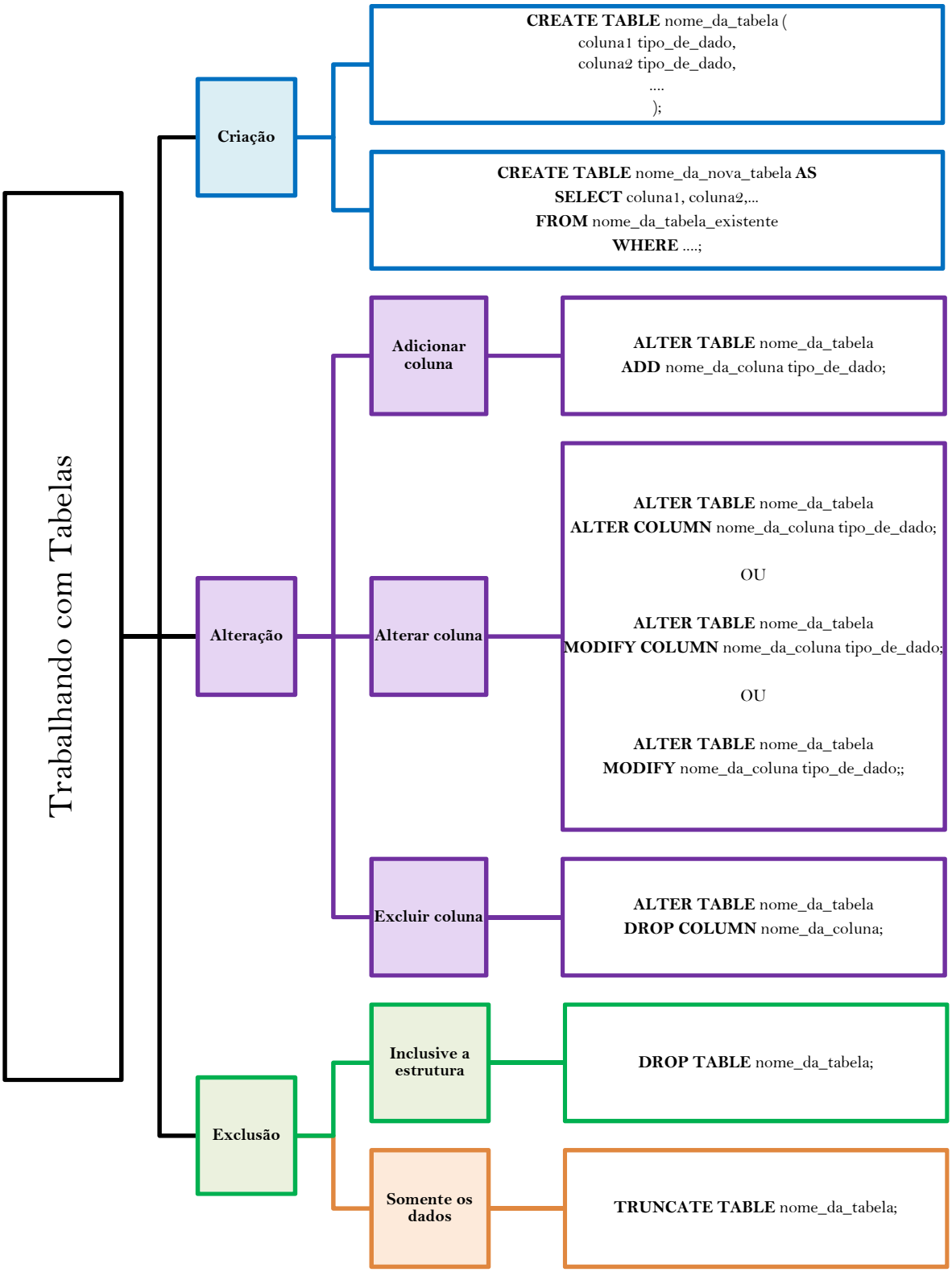
DDL



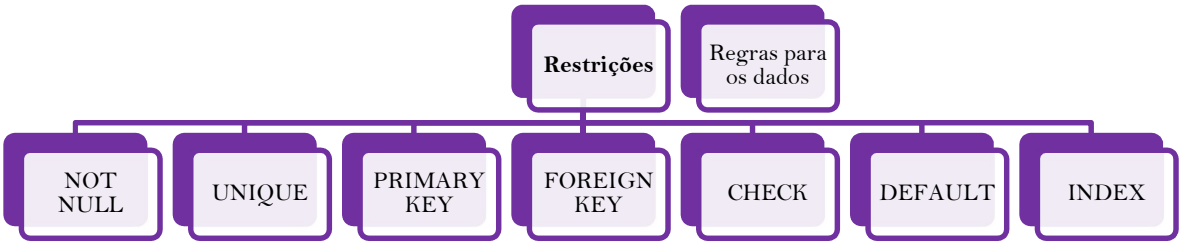
Trabalhando com banco de dados

Criar uma banco de dados	Exibir bancos de dados	Excluir um banco de dados
CREATE DATABASE nome_do_banco;	**SHOW DATABASES**;	**DROP DATABASE** nome_do_banco;

Trabalhando com Tabelas



Restrições em SQL



Trabalhando com visões

Criando uma visão

```
CREATE VIEW [Nome da View]
AS
SELECT Coluna1, Coluna2,...
FROM nome_da_tabela
WHERE...;
```

Alterando uma visão

```
CREATE OR REPLACE VIEW
[Nome da View] AS
SELECT Coluna1, Coluna2,...
FROM nome_da_tabela WHERE...;
```

Deletando uma visão

```
DROP VIEW [Nome da View];
```

Trabalhando com índices

Criando um índice

```
CREATE INDEX
nome_do_indice ON
nome_da_tabela (coluna1,
coluna2, ...);
```

Alterando um índice

```
ALTER INDEX
nome_do_indice ON
nome_da_tabela (coluna1,
coluna2, ...);
```

Excluindo um índice

```
DROP INDEX nome_do_indice;
OU
DROP INDEX
nome_da_tabela.nome_do_indice;
```

Procedures x Trigger x Function

PROCEDURE

Código SQL preparado que você pode salvar, para que o código possa ser reutilizado repetidamente

TRIGGER

Programas armazenados que são executados ou disparados automaticamente quando alguns eventos ocorrem.

FUNCTION

Rotinas que retornam valores ou tabelas.

3. REFERÊNCIAS

TUTORIALSPOINT. **PL/SQL Tutorial**. Disponível em: <https://www.tutorialspoint.com/plsql/plsql_overview.htm>. Acesso em: 14 dez. 2020.

W3SCHOOLS. **SQL Tutorial**. Disponível em: <https://www.w3schools.com/sql/>. Acesso em: 14 dez. 2020.