

TECNOLOGIA DA INFORMAÇÃO

Banco de Dados Relacionais – Parte V



Livro Eletrônico



SUMÁRIO

Apresentação	4
Banco de Dados Relacionais – Parte V	5
SQL (Linguagem de Consulta Estruturada)	5
Terminologia Utilizada Nesta Aula.....	7
Regras Básicas para Escrever Comandos SQL.....	8
Cláusula WHERE	23
Cláusula Order By.....	25
Junção de Tabelas	25
INNER JOIN	26
LEFT JOIN	27
RIGTH JOIN	28
FULL OUTER JOIN	29
Funções de Grupo.....	30
Cláusula Group By.....	31
Cláusula Having.....	32
Subqueries (Subconsultas).....	33
Tipos de Domínio na SQL.....	37
Linguagens de Banco de Dados	37
DDL – <i>Data Definition Language</i> ou Linguagem de Definição de Dados.....	37
DML – <i>Data Manipulation Language</i> ou Linguagem de Manipulação de Dados.....	38
DQL – <i>Data Query Language</i> ou Linguagem de Consulta de Dados.....	39
DCL – <i>Data Control Language</i> ou Linguagem de Controle de Dados	39
DTL – <i>Data Transaction Language</i> ou Linguagem de Transação de Dados.....	39
SDL – <i>Storage Definition Language</i> ou Linguagem de Definição de Armazenamento ..	39
VDL – <i>View Definition Language</i> ou Linguagem de Definição de Visões	39
Índices	43
Resumo.....	45

Questões Comentadas em Aula	55
Questões de Concurso	57
Gabarito.....	100
Referências.....	101

APRESENTAÇÃO

Olá, querido(a) amigo(a), tudo bem?

Após os estudos sobre os conceitos fundamentais de bancos de dados, vamos ao aprendizado da **SQL** – uma **linguagem de pesquisa declarativa para Bancos de Dados Relacionais**.

Com o conhecimento de SQL é possível, entre outras coisas, extrair informações dos Bancos de Dados, de forma a torná-los uma ferramenta útil para a realização de auditorias, inspeções etc.

Estarei adicionando inúmeros exercícios para melhor fixação da matéria. Espero que aproveite!

Finalizando, como qualquer questão pode ser decisiva para a sua aprovação, então, **vamos “arregançar as mangas” e partir para mais esta etapa do curso. Boa sorte nos estudos! Estou torcendo pelo seu sucesso ☺!**

Em caso de dúvidas, acesse o fórum do curso ou entre em contato.

Um forte abraço!

Prof^a Patrícia Quintão

Instagram: @coachpatriciaquintao

Telegram: <https://t.me/coachpatriciaquintao>

WhatsApp: (31) 99442.0615

BANCO DE DADOS RELACIONAIS – PARTE V

SQL (LINGUAGEM DE CONSULTA ESTRUTURADA)

O nome da **SQL** é derivado de **Structured Query Language** ou **Linguagem de Consulta Estruturada**. Foi projetada e implementada na IBM Research como uma interface para um sistema experimental de um banco de dados relacional chamado SISTEMA R.

A **SQL** é agora a linguagem-padrão para os SGBDs relacionais comerciais. Isso decorre da sua simplicidade e facilidade de uso.

Ela se diferencia de outras linguagens de consulta a banco de dados pois uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele.

A linguagem SQL possui diversos **comandos** para acessar e manipular dados, bem como realizar a criação, exclusão e alteração das tabelas, índices e demais estruturas de bancos de dados, além de comandos para controlar os dados e transações. Podemos subdividir a linguagem nos seguintes subgrupos: **DML, DDL, DCL e DTL**, estudados nesta aula.

As linhas a seguir sumarizam algumas observações e as relacionam com os termos que utilizaremos em seguida.

- A **SQL** é uma linguagem baseada no inglês, e usa palavras como **SELECT, INSERT** e **DELETE** como parte de seu conjunto de comandos.
- **SQL** é uma **linguagem de pesquisa declarativa para Bancos de Dados Relacionais em oposição a outras linguagens procedurais**. Por ser não procedural, você especifica **QUAL** informação quer, e **não** como trazê-la. Isto reduz o ciclo de aprendizado daqueles que se iniciam na linguagem. Em outras palavras, não é necessário especificar o método de acesso aos dados. O SGBD usa o “otimizador” para interpretar o comando SQL e escolher o melhor caminho para acesso aos dados.
- SQL oferece um **conjunto de comandos** para uma variedade de tarefas diferentes, incluindo:
 - **pesquisar, inserir, alterar e deletar** linhas de uma tabela;
 - **criar, deletar e alterar objetos** de banco de dados;
 - **controlar o acesso aos dados** e objetos;
 - **garantir a consistência dos dados** etc.

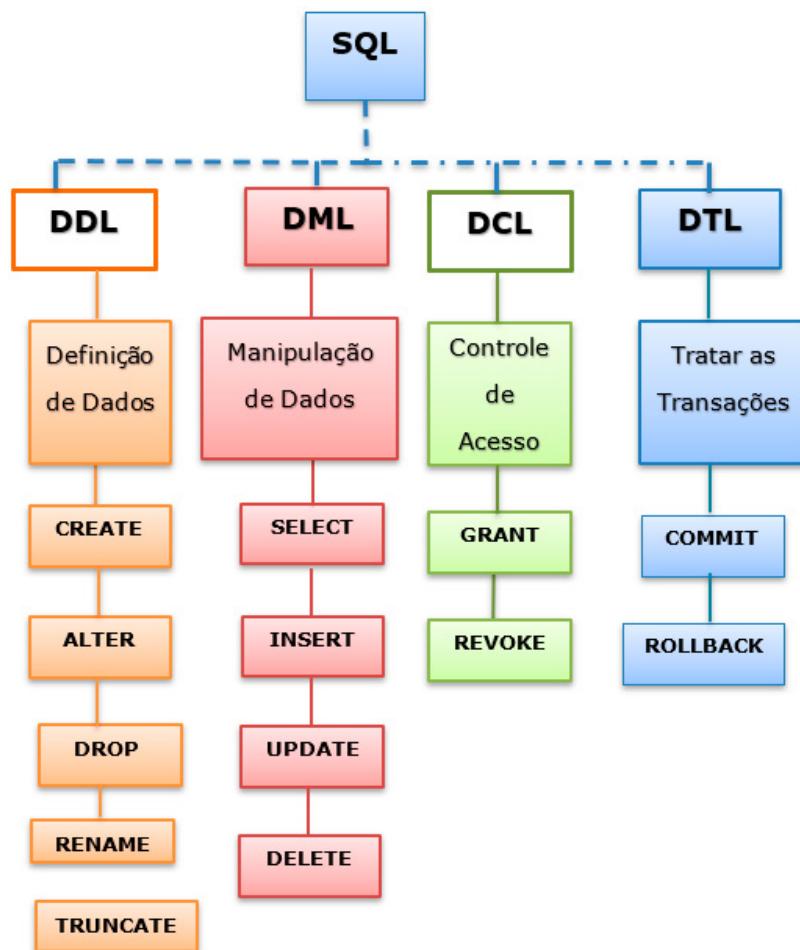


Figura. Comandos da SQL

Mais observações!

- SQL é **utilizada por todos os tipos de usuários** de um banco de dados relacional:
 - administrador de sistemas;
 - administrador de banco de dados;
 - desenvolvedores de sistemas;
 - usuários finais etc.
- SQL não é realmente uma linguagem completa como o Visual Basic, Pascal, Java, pois **não** contém estruturas como IF, GOTO ou FOR não permitindo assim o desenvolvimento de lógicas de programação.
- É composta por um **conjunto de instruções específicas para o gerenciamento de banco de dados**.
- Os comandos SQL podem ser EMBUTIDOS em outra linguagem e utilizados para o acesso ao banco de dados.
- SQL não é por si só um sistema de gerenciamento de banco de dados. **A responsabilidade pelo armazenamento, gerenciamento físico e recuperação dos dados no disco é do SGBD.**

O conteúdo deste livro eletrônico é licenciado para DANIEL MARTINS FRANCA - 03653309140, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

- SQL é uma “linguagem padrão” que **interage entre o SGBD e os seus componentes ou outros gerenciadores.**
- SQL é uma **linguagem de consulta interativa**: o usuário tecla comandos num editor interativo que recupera e permite uma consulta fácil e rápida ao banco de dados.
- SQL é uma **linguagem de programação de banco de dados**: programadores incluem comandos SQL em seus programas de aplicação para acessar os dados em um banco.
- SQL é uma **linguagem para administração do banco de dados**: o administrador do banco de dados usa SQL para definir a estrutura da base de dados e controlar o acesso aos dados que nela serão armazenados.
- SQL é uma **linguagem cliente/servidor**: programas em computadores pessoais usam SQL para comunicar em uma rede com servidores de bancos de dados.
- SQL é uma **linguagem de bancos de dados distribuídos**: SGBDs distribuídos usam SQL para auxiliar a distribuição dos dados por meio de vários computadores interligados.

DIRETO DO CONCURSO

001. (CESPE/TRE-BA/TÉCNICO JUDICIÁRIO/TELECOMUNICAÇÕES E ELETRICIDADE/2010) Uma linguagem de acesso a bancos de dados relacionais amplamente usada é o SQL.



Trata-se de uma **linguagem padrão para manipulação de bancos de dados relacionais**. Por meio dela, um usuário pode executar comandos para inserir, pesquisar, atualizar ou deletar registros em um banco de dados, criar ou excluir tabelas, conceder ou revogar permissões para acessar o banco de dados, etc.

Certo.

TERMINOLOGIA UTILIZADA NESTA AULA

Uma relação é uma **tabela**.

Um único registro será chamado de **linha (tupla)**.

Um atributo será chamado de **coluna**.

A SQL usa os termos tabela, linha e coluna, em vez dos termos *relação*, *tupla* e *atributo*, respectivamente, para o modelo relacional formal.

Um valor individual existente na interseção de qualquer linha com coluna será chamado de **dado**.

Chave Primária (Primary Key): uma ou mais colunas com valores que são únicos dentro da tabela e por isso podem ser usados para identificar as linhas dessa tabela.

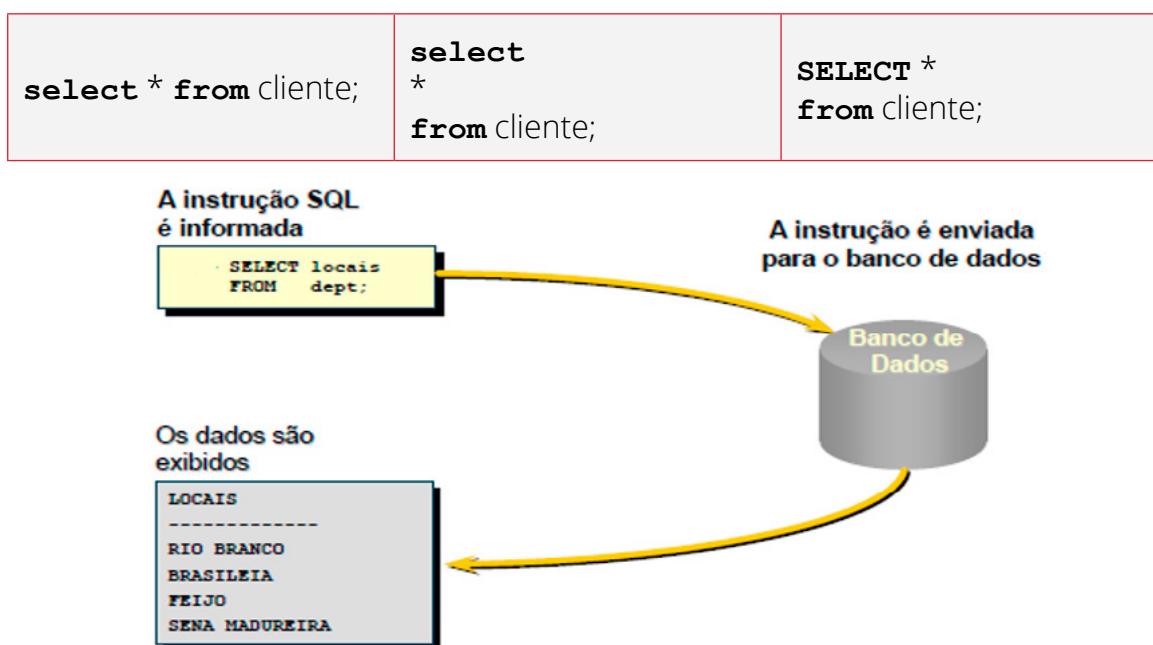
SQL padrão: todo o material a ser apresentado neste curso se refere ao Padrão SQL adotado oficialmente pelos *American National Standards Institute - ANSI* e *International Standard Organization - ISO*.

Alguns SGBDs que utilizam o SQL padrão: Oracle, DB2, SQL Server etc.

REGRAS BÁSICAS PARA ESCRER COMANDOS SQL

- Os comandos podem ser escritos em mais de uma linha;
- **Cláusulas** diferentes são colocadas usualmente em linhas diferentes;
- Podem ser usadas **tabulações**;
- **Comandos** podem ser escritos em letras maiúsculas ou minúsculas.

Alguns exemplos:



Criação de Tabelas

A criação do banco de dados deve começar com a criação das tabelas nas quais os dados serão introduzidos.

Para criar uma tabela em SQL, usa-se o comando **CREATE TABLE**.

O comando DDL para criar uma tabela deve conter os nomes das colunas, os tipos dos seus dados e os tamanhos dos dados a serem introduzidos.

É a seguinte a sintaxe desse comando:

```
CREATE TABLE nome_da_tabela{
    Nome_da_coluna1 tipo_do_dado (tamanho_do_dado),
    Nome_da_coluna2 tipo_do_dado (tamanho_do_dado),
    ...
    Nome_da_colunaN tipo_do_dado (tamanho_do_dado));
```

- Note que toda a descrição da **coluna** é colocada entre parênteses.
- Na criação de tabelas é possível especificar vários tipos de **restrições de integridade (RI)**.

- **Chave Primária:** **PRIMARY KEY**;
- **Chave Estrangeira:** **FOREIGN KEY**;
- **Chave Alternativa (ou alternada):** **UNIQUE**;
- **Restrição de Domínio:** **CHECK**. CHECK é utilizado para restringir os valores de domínio, verificando se eles estão contidos no conjunto de valores especificados.

Veja a seguir um exemplo de Utilização da restrição de domínio (CHECK): Apenas o bar Shizen pode vender cerveja com preço maior que 5,00.

```
CREATE TABLE Vende (
    Nome_bar CHAR(20),
    Nome_cerveja CHAR(20),
    Preco REAL,
    CHECK(Nome_bar='Shizen' OR
          Preco <= 5,00)
);
```

- Pode-se atribuir nomes às restrições de integridade:
 - **CONSTRAINT NOME_RESTRIÇÃO TIPO RESTRIÇÃO.**

Nota: **Constraints** são restrições que você estabelece para uma coluna no banco de dados, ou seja, um método para validar a integridade de todos os dados que entram em sua base.

Esquematizando

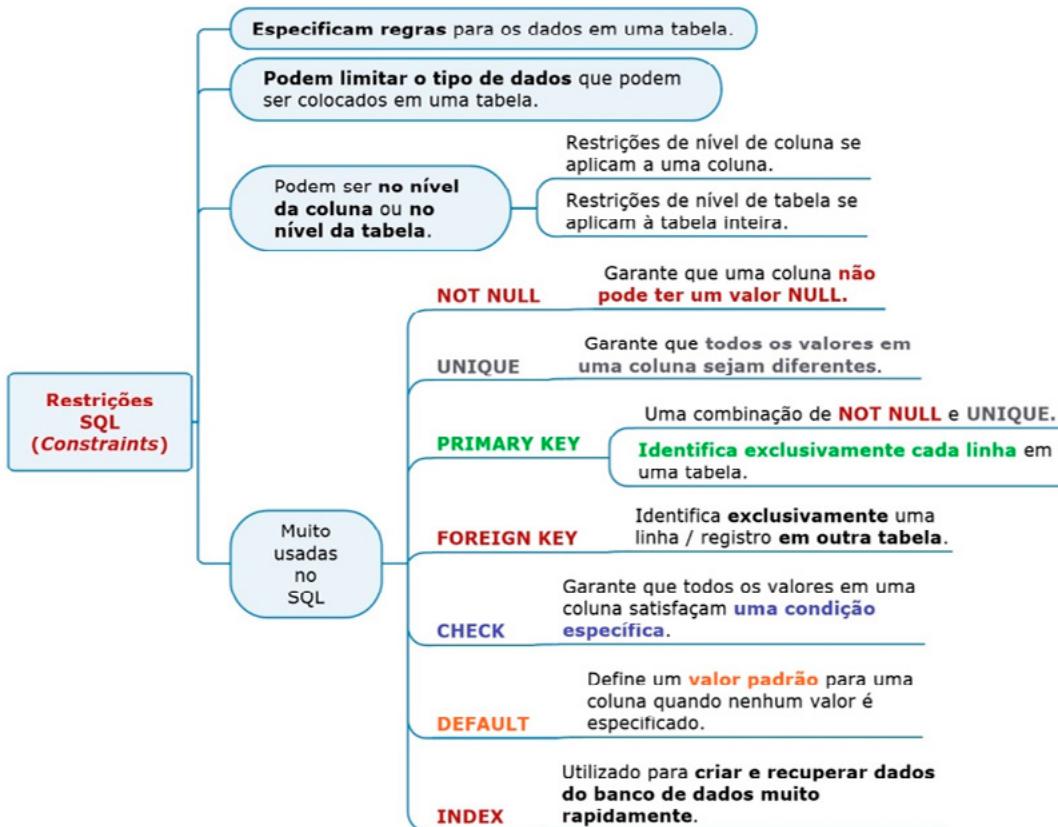


Figura. Restrições (QUINTÃO, 2021)

DIRETO DO CONCURSO

002. (FCC/TCE-SE/ANALISTA DE CONTROLE EXTERNO/ COORDENADORIA DE INFORMÁTICA/2011) Durante a criação de uma tabela - Create Table, em SQL, deseja-se especificar que uma coluna só possa incluir, por exemplo, valores maiores que zero. Uma constraint utilizada para isso é

- a) Verify.
- b) Check.
- c) Max.
- d) Avg.
- e) Having.



Conforme visto, cabe destacar que o **Check** é a **constraint que pode ser usada para restringir os valores de domínio**, verificando se eles estão contidos no conjunto de valores especificados. Como ficaria essa definição de campo em uma tabela qualquer? Veja:
nomeColuna INTEGER NOT NULL CHECK (nomeColuna > 0).

Letra b.

003. (ESAF/ANA/ANALISTA ADMINISTRATIVO/TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO/2009) Em SQL, a cláusula check aplicada a uma declaração de domínio

- a) permite especificar um predicado que deve ser satisfeito por qualquer valor atribuído a uma variável de determinado domínio.

- b) especifica um predicado que deve ser satisfeito por uma tupla em uma relação.
- c) proíbe a inserção de um valor nulo para as variáveis do domínio.
- d) verifica se os atributos considerados formam uma chave candidata.
- e) não tem efeito, pois não se aplica esta cláusula a declarações de domínio.



Um **predicado** é uma qualidade, uma característica. Na computação, entendam predicado como **uma condição que deve ser satisfeita**. A **cláusula CHECK** permite especificar **uma condição que deve ser satisfeita** por qualquer valor atribuído a uma variável de determinado domínio.

Exemplo:

```
ALTER TABLE TELEFONE
ADD CONSTRAINT TIPO_TELEFONE
CHECK (TIPO IN('Fixo', 'Movel'));
```

Letra a.

Criação de um Índice

Os índices aceleram a recuperação dos dados. Para criar um índice em uma tabela, o comando SQL é:

```
CREATE INDEX nome_do Índice
ON nome_da_tabela(nome_da_coluna);
```

O comando **CREATE INDEX** constrói o índice *nome_da_idx* na tabela especificada.

Os **índices** são utilizados, principalmente, para melhorar o desempenho do banco de dados (embora a utilização não apropriada possa resultar em uma degradação desse desempenho).

Você pode criar tantos índices quantos desejar em qualquer tabela. Pode-se ter um índice para cada coluna da tabela, assim como um índice para uma combinação de colunas.

Quantos índices e de que tipos você criará para uma determinada tabela dependerá do tipo de consultas que você espera que sejam dirigidas ao banco de dados e do tamanho deste.

Excesso de índices pode ser tão prejudicial quanto sua falta!

Alteração de Tabelas

Conforme ocorram novas situações ou novos dados sejam armazenados no banco de dados, a definição original de uma tabela pode tornar-se insuficiente.

A SQL permite-nos realizar diversas alterações em uma tabela, como:

- **incluir** novas colunas em uma tabela;
- **excluir** colunas existentes em uma tabela;
- adicionar a definição de uma **restrição** em uma tabela;
- excluir a definição de uma restrição existente em uma tabela;
- **modificar** uma coluna.

A sintaxe para **INCLUIR** uma coluna é:

```
ALTER TABLE      nome_da_tabela
                ADD            nome_da_coluna tipo_do_dado;
```

A sintaxe para **ALTERAR A LARGURA** de uma coluna já existente é:

```
ALTER TABLE      nome_da_tabela
                MODIFY        nome_da_coluna tipo_do_dado
                            nova_largura;
```

Exemplos:

```
ALTER TABLE Telefone
    DROP COLUMN Tipo;

ALTER TABLE TELEFONE
    ADD CONSTRAINT TIPO_TELEFONE
    CHECK (TIPO IN('Fixo', 'Movel'));

ALTER TABLE TELEFONE
    DROP CONSTRAINT TIPO_TELEFONE;

ALTER TABLE TELEFONE
    MODIFY (TIPO VARCHAR2(5));
```

Nota: para diminuir tamanho a coluna precisa estar **vazia**.

Exclusão de Tabelas

Para excluir uma tabela de um banco de dados, use o comando

DROP TABLE seguido do nome da tabela:

```
DROP TABLE nomedatabela;
```

Quando uma tabela é excluída por um comando SQL, como acima, todas as visões e índices definidos sobre a tabela são automaticamente excluídos.

Exemplo:

```
DROP TABLE EMP;
```

Obs.: O comando **DROP** remove a tabela inteira da base de dados que inclui a sua estrutura e registros. Remove, portanto, não somente os dados da tabela, mas também a própria definição do objeto. A tabela, seus dados e restrições deixam de existir no banco de dados.

DIRETO DO CONCURSO

004. (CESPE/FUB/2018) 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.



O comando **DROP TABLE** é utilizado para **excluir uma tabela do banco de dados**. Deve ser usado com cuidado porque ele **apaga a tabela com todos os seus dados**.

SINTAXE DO COMANDO:

```
DROP TABLE NOME_DA_TABELA;
```

Exemplo:

```
DROP TABLE teste;
```

O comando **DROP TABLE** remove a tabela inteira da base de dados que inclui a sua estrutura e registros.

Certo.

Exclusão de Índices

Para EXCLUIR um índice, use o comando SQL **DROP INDEX** seguido do nome do índice:

```
DROP INDEX nome_do Índice  
    ON nome_da_tabela;
```

Se você tem índices definidos com o mesmo nome em tabelas diferentes, deve usar a cláusula **ON**, como no exemplo acima, para indicar o índice que quer eliminar. Se há apenas um índice com este nome, então você não precisa especificar o nome da tabela. O comando **DROP** abaixo será suficiente:

```
DROP INDEX nome_do Índice;
```

Obs.: A eliminação de um índice **não** elimina as **tabelas** ou **visões** relacionadas com o índice.

Inserção

A maioria dos sistemas trata a carga inicial no banco de dados de um grande conjunto de dados numa operação genérica de carga. O comando SQL **INSERT** geralmente é usado para inserir linhas individuais de dados num banco de dados já existente.

A sintaxe desse comando é:

```
INSERT INTO nome_da_tabela (nome_da_coluna1,  
                           nome_da_coluna2,...)  
VALUES ('valor1', 'valor2', ...);
```

Se a lista de valores está na mesma sequência que as colunas na tabela e há um valor para cada coluna da tabela, então a lista de nomes das colunas pode ser omitida. Caso contrário, os nomes das colunas devem ser especificados como mostrado acima.

Os valores inseridos **devem** ter o tipo compatível com o tipo da coluna na qual estão sendo inseridos. Valores do tipo CHAR devem estar entre aspas simples; valores do tipo NUM ou do tipo NULL não devem ser colocados entre aspas simples.

Exemplos:

```
INSERT INTO PESSOA (CPF, NOME, SEXO)  
VALUES ('11122233344','Patricia', 'F');  
INSERT INTO PESSOA (CPF, NOME, SEXO);  
SELECT CPF, NOME, SEXO FROM Aluno;
```

Atualização

O comando **UPDATE** é usado para alterar valores em linhas já existentes.

Sua forma geral é:

```
UPDATE nome_da_tabela  
SET     coluna1=novo_valor,  
          coluna2=novo_valor,  
          ...  
          colunaN=novo_valor,  
WHERE   condição;
```

A cláusula **SET** do comando **UPDATE** indica as colunas a serem alteradas e quais os novos valores.

O comando **UPDATE** atua em todas as linhas que satisfazem a condição especificada pela cláusula **WHERE**. A cláusula **WHERE** é opcional, mas, se for omitida, todas as linhas serão atualizadas.

Você pode atualizar várias colunas em cada linha com um único comando **UPDATE** listando as várias colunas após a cláusula **SET**, conforme visto acima.

A cláusula **WHERE** no comando **UPDATE** pode conter uma subconsulta.

Exemplo:

```
UPDATE PESSOA  
SET     idade = 20  
WHERE   nome = 'Maria';
```

Exclusão

O comando **DELETE** é usado para remover linhas de uma tabela. Sua forma geral:

```
DELETE  
FROM    nome_da_tabela  
WHERE   condição;
```

Você não pode excluir parcialmente uma linha, portanto **não** precisa especificar os nomes das colunas no comando **DELETE**.

A cláusula **WHERE** determina que linhas serão eliminadas. Ela pode ser complexa e incluir várias condições, conectores e/ou subconsultas.

Se você deseja excluir todas as linhas de uma tabela, omita a cláusula WHERE, como listado a seguir:

DELETE

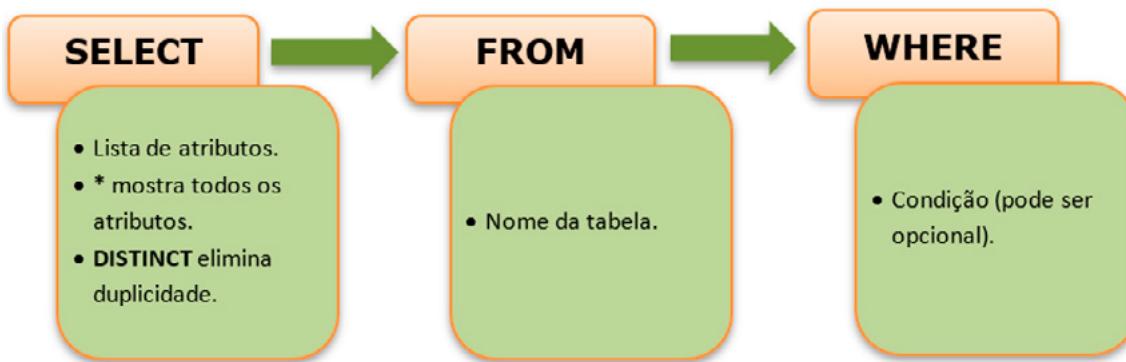
```
FROM nome_da_tabela;
```

O comando anterior eliminará todas as linhas, deixando apenas as especificações das colunas e o nome da tabela.

Recuperação Usando SELECT

A estrutura básica de uma consulta em SQL consiste em TRÊS cláusulas: **SELECT**, **FROM** e **WHERE**.

Uma **sintaxe básica** de uma **consulta SQL** é esquematizada a seguir:



- A cláusula **SELECT** relaciona as colunas que se quer presentes no resultado da consulta. Essa cláusula corresponde à operação de projeção da álgebra relacional.
- A cláusula **FROM** corresponde à operação de produto cartesiano da álgebra relacional. Ela associa a tabela ou tabelas que serão pesquisadas durante a avaliação de uma expressão. Em outras palavras, é uma lista de relações a serem varridas na execução da expressão.
- A cláusula **WHERE** corresponde à seleção do predicado da álgebra relacional. Consiste em um predicado envolvendo atributos da relação que aparece na cláusula **FROM**.

Assim, uma consulta típica em SQL tem a forma:

SELECT coluna1, coluna2,..., colunaN

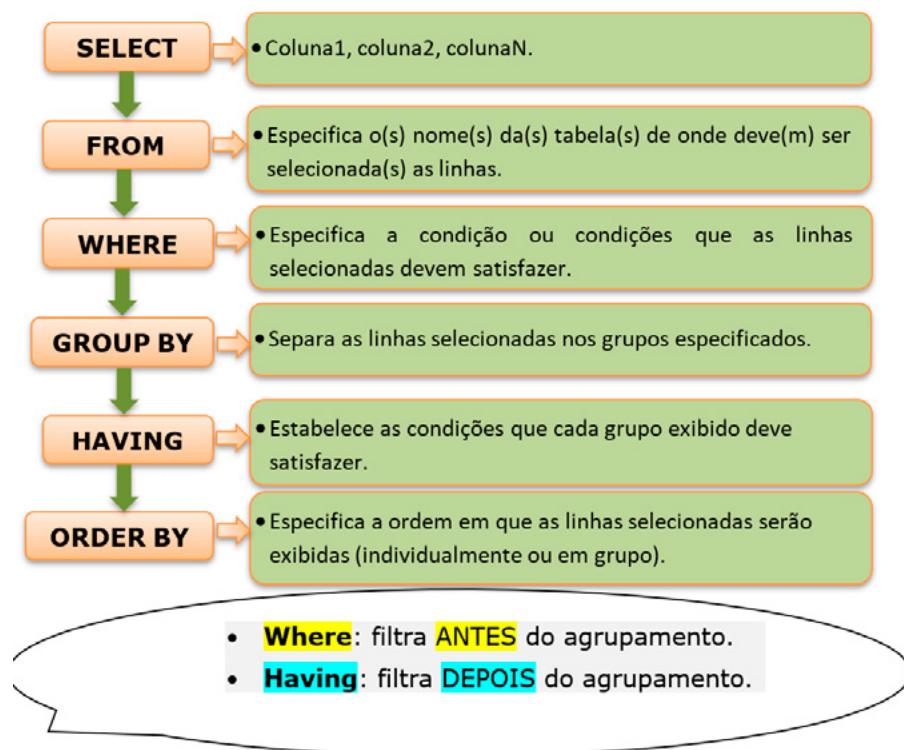
FROM nome_da_tabela

WHERE condição;

Obs.: | **SELECT** e a cláusula **FROM** são necessárias em todas as consultas SQL. Devem aparecer antes de qualquer outra cláusula na consulta.
 | O resultado de uma consulta SQL é SEMPRE uma tabela.

Estrutura de uma Consulta Simples

Uma **consulta SQL** pode conter outras cláusulas como esquematizadas a seguir:



Expressões Tabulares (Cláusulas)

Expressões tabulares são cláusulas que são usadas para produzir tabelas. As expressões tabulares suportadas pela SQL e seus propósitos são:

FROM	Especifica o(s) nome(s) da(s) tabela(s) de onde deve(m) ser selecionada(s) as linhas.
WHERE	Especifica a condição ou condições que as linhas selecionadas devem satisfazer.
GROUP BY	Separa as linhas selecionadas nos grupos especificados.
HAVING	Estabelece as condições que cada grupo exibido deve satisfazer.
ORDER BY	Especifica a ordem em que as linhas selecionadas serão exibidas (individualmente ou em grupo).

A cláusula FROM é necessária para uma consulta SQL; as cláusulas WHERE, GROUP BY, HAVING e ORDER BY são opcionais.

Exemplos:

```

SELECT * FROM PESSOA;

SELECT CPF, NOME, SEXO, IDADE
FROM PESSOA
WHERE sexo = 'M' OR sexo = 'F';
SELECT CPF, NOME, SEXO, IDADE
FROM PESSOA
WHERE IDADE > ANY (10,20,30);
ou
WHERE IDADE > ALL (10,20,30);

SELECT CPF, NOME, SEXO
FROM PESSOA
WHERE nome LIKE 'P%';

```

LIKE e NOT LIKE

Observe o uso do **LIKE** no exemplo acima. **LIKE determina se uma cadeia de caracteres específica corresponde a um padrão especificado.** Um padrão pode incluir caracteres normais e **coringas**.

Durante a correspondência de padrões, os caracteres normais devem corresponder exatamente aos caracteres especificados na cadeia de caracteres.

No entanto, os **caracteres coringas** podem ser correspondidos a fragmentos arbitrários da cadeia de caracteres.

Ex.: o caractere **%** (combina qualquer substring, independentemente do tamanho) e **_** (combina caractere a caractere).

Os operadores **NOT LIKE** são usados de forma oposta ao operador **LIKE**.

Expressão	Explicação
LIKE 'A% '	Todas as palavras que iniciem com a letra A .
LIKE '%A'	Todas as palavras que terminem com a letra A .
LIKE '%A% '	Todas que tenham a letra A em qualquer posição .
LIKE 'A_'	<i>String de dois caracteres que tenham a primeira letra A e o segundo caractere seja qualquer outro.</i>

Expressão	Explicação
LIKE '_A'	<i>String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja A.</i>
LIKE '_A_'	<i>String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.</i>
LIKE '%A_'	Todos que tenham a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '_A% '	Todos que tenham a letra A na segunda posição e o primeiro caractere seja qualquer um.

DICA!

Leitura complementar

<https://www.devmedia.com.br/sql-server-2005-operadores-like-e-not-like/17292>

Criação de Visões

Uma **visão (view)** pode ser considerada como uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas). Pode ser considerada também como uma **tabela virtual ou uma consulta armazenada**.

Uma vez que a **view** é gerada, **o seu conjunto de dados é armazenado em uma tabela temporária (virtual)**, tornando o acesso às informações mais rápido.

Deve-se ressaltar que uma view não existe fisicamente, é uma tabela virtual. No entanto, os dados contidos em uma **view** podem ser modificados normalmente.

View (visão) é uma tabela única derivada de outras tabelas. Por não existir fisicamente, pode ser considerada uma **tabela virtual**. A visão, diferentemente das tabelas básicas, não contém suas tuplas armazenadas no banco de dados.

As vantagens de se usar **views** são:

- permite economizar tempo, evitando retrabalho;
- aumenta a velocidade de acesso aos dados;
- esconde a complexidade do banco de dados;
- simplifica a gerência de permissão de usuários; e
- organiza os dados a serem exportados.

Para criar uma visão, você seleciona apenas as colunas da tabela (ou tabelas) básica em que está interessado.

Obs.: **Visões** podem ser criadas usando expressões de consulta para extrair dados de outras visões.

Além disso, é possível criar uma visão que contém **apenas** dados calculados e, que, portanto, não possuem dados armazenados em uma tabela base.

Para definir uma visão, você deve dar um nome para a visão e então estabelecer a consulta contendo os nomes das colunas e as especificações que constituirão a visão. A sintaxe é:

```
CREATE VIEW nome-de-visão
[ (nomes_das_colunas_da_visão) ]
AS (expressão da consulta);
```

Em que (expressão da consulta) é um comando **SELECT FROM**.

Exemplo:

```
CREATE VIEW Empregados_Sede
AS
SELECT (Nome, Endereco, Sexo, Data-nasc)
FROM EMPREGADO;
```

Mais comandos:

Grant	Concessão de privilégios a tabelas e visões.
Revoke	Revogação de privilégios a tabelas e visões.
COMMIT	Efetiva alterações feitas na base de dados.
ROLLBACK	Desfaz alterações na base de dados.
SAVEPOINT	Estabelece pontos de retorno dentro de uma transação.

Esquematizando

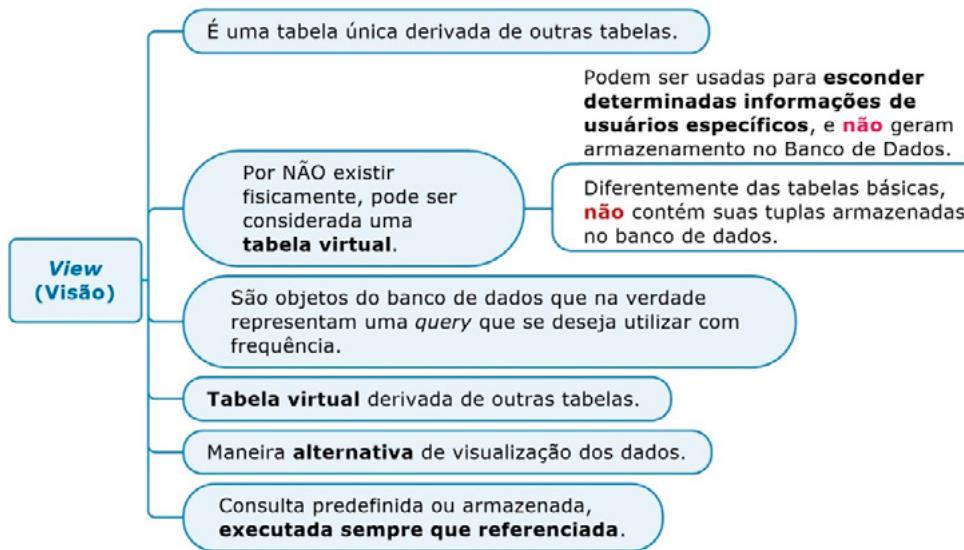


Figura. View (Visão). Fonte: (QUINTÃO, 2021)

** Restrições de Integridade Usando...

STORED PROCEDURES (procedimentos armazenados no banco)

Stored Procedure é uma sequência de comandos em SQL para realização de diferentes tarefas repetitivas no banco, aceitando parâmetros de entrada e retornando valores. Algumas das vantagens das **Stored Procedures** são:

- redução do tráfego na rede;
- melhora do desempenho;
- criação de mecanismos de segurança e backup.

TRIGGERS (gatilhos)

Uma *trigger* é uma **sub-rotina**, parecida com uma *stored procedure*, que tem como característica **ser executada automaticamente a partir de alguma ação** realizada no banco de dados.

Geralmente utilizada com um tipo de proteção ao acesso indiscriminado a dados de uma tabela. Quando há uma tentativa de inserir, atualizar ou excluir os dados em uma tabela, e uma *trigger* tiver sido definida na tabela para essa ação específica, ela será executada **automaticamente**, não podendo ser ignorada.

Comando Select

A sintaxe mais simples do comando **SELECT** é:

```

SELECT [DISTINCT | ALL] [* | coluna [, coluna, ...]}
FROM tabela
  
```

O conteúdo deste livro eletrônico é licenciado para DANIEL MARTINS FRANCA - 03653309140, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

Em que:

Cláusula	Descrição
DISTINCT	Não mostra eventuais valores repetidos de colunas.
ALL	Mostra todos os valores, ainda que sejam repetidos. Esse é o padrão se DISTINCT não for definido.
*	Indica que devem ser mostradas todas as colunas da tabela.
Coluna	Lista de colunas que devem ser mostradas.
Tabela	Nome da tabela em que será realizada a busca.

Seleção de Colunas Específicas

Exemplo: Para retornar todos os nomes e códigos dos clientes da tabela cliente:

```
SELECT cod_cliente, nome_cliente
FROM cliente;
```

Seleção de Todas as Colunas

Exemplo:

```
SELECT * FROM Empregado;
```

Esse exemplo utiliza o **coringa** “*” para selecionar as colunas na ordem em que foram criadas. A instrução **select** seleciona um grupo de registros de uma (ou mais) tabela(s). No caso, a instrução **from** indica a necessidade de pesquisarmos tais dados apenas na tabela Empregado.

Eliminação de Duplicações

A cláusula **DISTINCT** não mostra eventuais valores repetidos de colunas.

```
SELECT DISTINCT Nome
FROM Empregado;
```

Utilização de Pseudônimos

A SQL oferece um modo de renomear os atributos de uma relação resultante. Para isso, usa a cláusula **as**, tomando a forma:

nome-antigo as nome-novo

Exemplo:

```
SELECT Matricula AS Matricula_Empregado  
FROM Empregado;
```

A cláusula **as** pode aparecer nas cláusulas **select** e **from**. Considere a consulta seguinte como exemplo:

```
SELECT nome AS nome_instrutor, id_curso  
FROM instrutor, ministra  
WHERE instrutor.ID = ministra.ID;
```

Nesse exemplo, o nome do atributo nome foi substituído pelo nome nome_instrutor.

A cláusula **as** é particularmente útil na renomeação de relações. Um motivo para renomear uma relação é **substituir um nome de relação longo por uma versão abreviada**, mais conveniente para usar em outro ponto da consulta. Veja o exemplo: “Para todos os instrutores na universidade que ministraram algum curso, ache seus nomes e a ID do curso para todos os cursos que eles ministraram”.

```
SELECT T.nome, S.id_curso  
FROM instrutor as T, ministra as S  
WHERE T.ID = S.ID;
```

CLÁUSULA WHERE

A cláusula “**where**” corresponde ao **operador de seleção da álgebra relacional**. Contém a condição que as tuplas devem obedecer a fim de serem listadas. Ela pode comparar valores em colunas, literais, expressões aritméticas ou funções.

Veja exemplos:

Nome=’José’

Sexo = ’Masculino’

Salário >= 1000

- **Objetivo:** filtrar um conjunto de linhas de uma tabela

```
SELECT colunas
FROM tabela
[WHERE condição]
```

- Uma cláusula **WHERE** pode conter mais de uma condição, sendo possível usar **operadores lógicos** para indicar a união entre as condições:
 - **AND**: exibe os registros em que todas as condições são verdadeiras.

```
SELECT coluna1, coluna2, coluna3 ...
FROM nome_da_tabela
WHERE condição1 AND condição2 AND condição3 ...;
```

- **OR**: exibe os registros em que pelo menos uma condição é verdadeira.

```
SELECT coluna1, coluna2, coluna3 ...
FROM nome_da_tabela
WHERE condição1 OR condição2 OR condição3 ...;
```

- **Operadores**

Operador	Significado	Exemplo
=	Igualdade	Nome = "Maria"
<>	Diferente	Sexo <> "M"
>	Maior que	Salario > 1500,00
>=	Maior ou igual a	Salario >= 1500,00
<	Menor que	Salario < 1500,00
<=	Menor ou igual a	Salario <= 1500,00
IS NULL	Valor nulo	E-mail IS NULL
IN	Conjunto	Valor IN (10, 25, 52, 67)
BETWEEN	Intervalo	Data BETWEEN '15/03/1980' AND '14/03/2000'
NOT	Negação	NOT País='Alemanha'
LIKE	Comparação String	Nome LIKE 'J%'

O conteúdo deste livro eletrônico é licenciado para DANIEL MARTINS FRANCA - 03653300140, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

- Exemplos:

Conjunção de condições

```
SELECT *
FROM Empregado
WHERE Nome = 'Paulo' AND Salario > 1000
```

Disjunção de condições

```
SELECT *
FROM Empregado
WHERE Matricula = 1 OR Nome LIKE 'J%'
```

CLÁUSULA ORDER BY

- Objetivo: ordenar o resultado de uma consulta.

```
SELECT colunas
FROM tabela
[WHERE condição]
[ORDER BY {coluna1,...} [ASC|DESC]]
```

- Exemplo:

```
SELECT *
FROM Empregado
ORDER BY Nome;
```

JUNÇÃO DE TABELAS

A junção de tabelas tem o objetivo de **combinar linhas de tabelas diferentes, por meio de valores comuns em colunas correspondentes.**

Exemplo:

```
SELECT Empregado.Nome, Filial.Nome
FROM Empregado, Filial
WHERE Empregado.CodFilial = Filial.CodFilial
OU
SELECT Empregado.Nome, Filial.Nome
FROM Empregado e, Filial f
WHERE e.CodFilial = f.CodFilial
```

Joins são instruções em SQL usadas para obter informações de duas ou mais tabelas, baseado no [relacionamento entre certas colunas dessas tabelas](#), em outras palavras, **baseado em chaves primárias e estrangeiras**.

Essa é a primeira grande diferença entre os *joins* e o **produto cartesiano**. Neste eu posso relacionar tabelas que não têm nenhum campo em comum (apesar de isso não fazer muito sentido). Já **nos joins, eu preciso ter um relacionamento entre as tabelas, senão não tem join**.

Temos os seguintes [tipos de joins](#):

Inner Join	Retorna linhas quando existe pelo menos uma combinação em ambas as tabelas.
Left Join	Retorna todas as linhas da tabela da esquerda , mesmo que as linhas não combinem com a tabela da direita.
Right Outer Join <i>(ou simplesmente Right Join)</i>	Retorna todas as linhas da tabela da direita , mesmo que as linhas não combinem com a tabela da esquerda.
Full join	Retorna linhas quando têm uma combinação em uma das tabelas.

Vejamos o exemplo de duas tabelas abaixo:

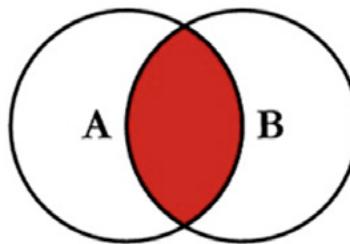
ID_AUTOR	NOME_AUTOR
1	José de Alencar
2	Paulo Coelho
3	Ana Beatriz Barbosa Silva
4	Jorge Amado
5	Clarice Lispector
6	Carlos Drummond de Andrade
7	Eros Grau
8	Fernando Gabeira
9	Zibia Gasparetto

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR
107	A Escrava Isaura	12345	06/22/1979	79.4	-
104	A Paixão Segundo G.H.	12345	11/23/1980	46.9	5
105	Farewell	12345	07/30/1967	37.8	6
100	Iracema	12345	11/11/2009	56.5	1
103	Mar Morto	12345	11/04/1965	102.5	4
102	Mentes Perigosas	12345	02/12/1998	88.5	3
101	O Diário de um Mago	12345	05/14/2007	86.5	2
108	O Inverno das Fadas	12345	12/07/1999	22.9	-
106	Triângulo no Ponto	12345	03/15/1980	153.7	7

Para entender melhor a funcionalidade do **JOIN** na linguagem SQL, podemos visualizar os diagramas listados a seguir que representam a **teoria dos conjuntos**.

INNER JOIN

É usando para **juntar** dados de mais de uma tabela em que haja correspondência entre as linhas das tabelas. **Retorna somente a intersecção entre as duas tabelas**, conforme mostra a área em vermelho na figura.

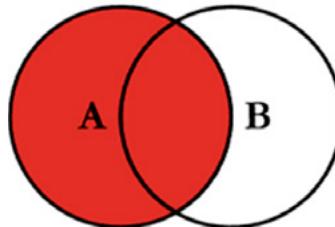


```
SELECT *
FROM LIVRO A
INNER JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR;
```

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
100	Iracema	12345	11/11/2009	56.5	1	1	José de Alencar
101	O Diário de um Mago	12345	05/14/2007	86.5	2	2	Paulo Coelho
102	Mentes Perigosas	12345	02/12/1998	88.5	3	3	Ana Beatriz Barbosa Silva
103	Mar Morto	12345	11/04/1965	102.5	4	4	Jorge Amado
104	A Paixão Segundo G.H.	12345	11/23/1988	46.9	5	5	Clarice Lispector
105	Farewell	12345	07/30/1967	37.8	6	6	Carlos Drummond de Andrade
106	Triângulo no Ponto	12345	03/15/1980	153.7	7	7	Eros Grau

LEFT JOIN

É usando para **juntar dados de duas ou mais tabelas retornando todas as linhas da tabela à esquerda (A) MESMO QUE NÃO HAJA correspondência na tabela da direita (B).**



```
SELECT *
FROM LIVRO A
LEFT JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR;
```

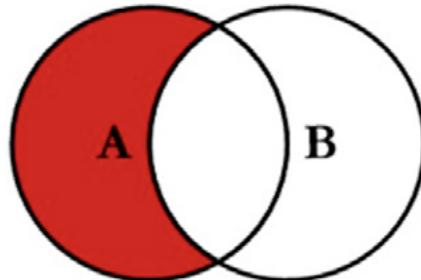
ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
100	Iracema	12345	11/11/2009	56.5	1	1	José de Alencar
101	O Diário de um Mago	12345	05/14/2007	86.5	2	2	Paulo Coelho
102	Mentes Perigosas	12345	02/12/1998	88.5	3	3	Ana Beatriz Barbosa Silva
103	Mar Morto	12345	11/04/1965	102.5	4	4	Jorge Amado
104	A Paixão Segundo G.H.	12345	11/23/1988	46.9	5	5	Clarice Lispector
105	Farewell	12345	07/30/1967	37.8	6	6	Carlos Drummond de Andrade
106	Triângulo no Ponto	12345	03/15/1980	153.7	7	7	Eros Grau
107	A Escrava Isaura	12345	06/22/1979	79.4	-	-	-
108	O Inverno das Fadas	12345	12/07/1999	22.9	-	-	-

Podemos ainda excluir os registros exibindo os itens da tabela à esquerda (A) que NÃO tem correspondências na tabela a direita (B).

```

SELECT *
FROM LIVRO A
LEFT JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR
WHERE A.ID_AUTOR IS NULL;

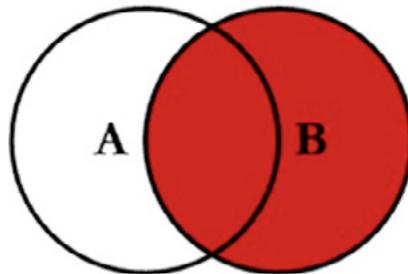
```



ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
108	O Inverno das Fadas	12345	12/07/1999	22.9	-	-	-
107	A Escrava Isaura	12345	06/22/1979	79.4	-	-	-

RIGTH JOIN

É usado para juntar dados de duas ou mais tabelas retornando todas as linhas da tabela a direita AINDA QUE NÃO HAJA correspondência na tabela da esquerda.



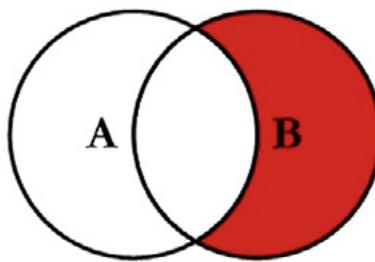
```

SELECT *
FROM LIVRO A
RIGHT JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR;

```

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
100	Iracema	12345	11/11/2009	56.5	1	1	José de Alencar
101	O Diário de um Mago	12345	05/14/2007	86.5	2	2	Paulo Coelho
102	Mentes Perigosas	12345	02/12/1998	88.5	3	3	Ana Beatriz Barbosa Silva
103	Mar Morto	12345	11/04/1965	102.5	4	4	Jorge Amado
104	A Paixão Segundo G.H.	12345	11/23/1988	46.9	5	5	Clarice Lispector
105	Farewell	12345	07/30/1967	37.8	6	6	Carlos Drummond de Andrade
106	Triângulo no Ponto	12345	03/15/1980	153.7	7	7	Eros Grau
-	-	-	-	-	-	8	Fernando Gabeira
-	-	-	-	-	-	9	Zibia Gasparetto

RIGTH OUTER JOIN (Excluindo correspondência) exibe todos os itens da tabela à direita que NÃO tem correspondências na tabela à esquerda.

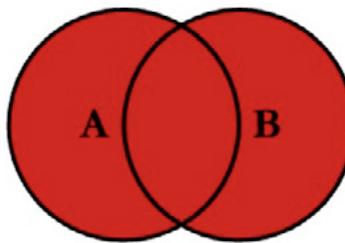


```
SELECT *
FROM LIVRO A
RIGHT JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR
WHERE A.ID_AUTOR IS NULL;
```

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
-	-	-	-	-	8		Fernando Gabeira
-	-	-	-	-	9		Zibia Casparetto

FULL OUTER JOIN

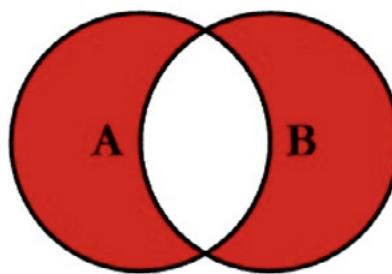
É usado para **juntar duas ou mais tabelas nas quais HÁ OU NÃO correspondências entre as tabelas. TODOS OS DADOS!**



```
SELECT *
FROM LIVRO A
FULL JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR;
```

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
100	Iracema	12345	11/11/2009	56.5	1	1	José de Alencar
102	Mentes Perigosas	12345	02/12/1998	88.5	3	3	Ana Beatriz Barbosa Silva
104	A Paixão Segundo G. H.	12345	11/23/1988	46.9	5	5	Clarice Lispector
106	Triângulo no Ponto	12345	03/15/1980	153.7	7	7	Eros Grau
108	O Inverno das Fadas	12345	12/07/1999	22.9	-	-	-
101	O Diário de um Mago	12345	05/14/2007	86.5	2	2	Paulo Coelho
103	Mar Morto	12345	11/04/1965	102.5	4	4	Jorge Amado
105	Farewell	12345	07/30/1967	37.8	6	6	Carlos Drummond de Andrade
107	A Escrava Isaura	12345	06/22/1979	79.4	-	-	-
-	-	-	-	-	-	8	Fernando Gabeira
-	-	-	-	-	-	9	Zibia Gasparetto

FULL OUTER JOIN (Excluindo correspondências) é usando para juntar duas ou mais tabelas onde HÁ OU NÃO correspondências entre as tabelas.



```

SELECT *
FROM LIVRO A
FULL JOIN AUTOR B ON A.ID_AUTOR = B.ID_AUTOR
WHERE A.ID_AUTOR IS NULL
OR B.ID_AUTOR IS NULL;
    
```

ID_LIVRO	NOME_LIVRO	ISSN	DATA_PUB	PRECO_LIVRO	ID_AUTOR	ID_AUTOR	NOME_AUTOR
108	O Inverno das Fadas	12345	12/07/1999	22.9	-	-	-
107	A Escrava Isaura	12345	06/22/1979	79.4	-	-	-
-	-	-	-	-	-	8	Fernando Gabeira
-	-	-	-	-	-	9	Zibia Gasparetto

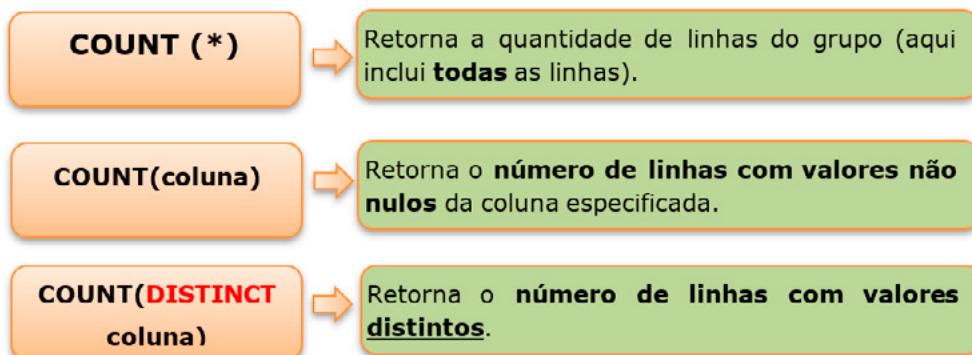
FUNÇÕES DE GRUPO

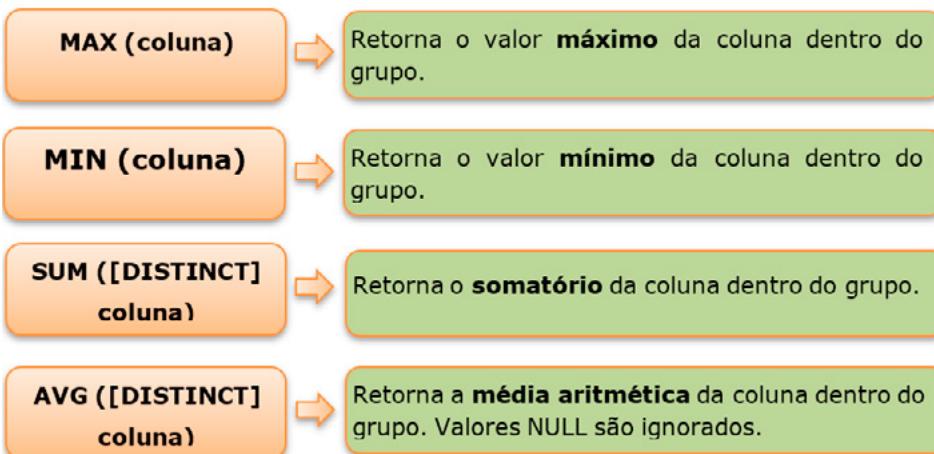
- **Objetivo:** retornar um valor para um grupo de linhas.

```

SELECT função_grupo (coluna)
FROM tabela
[WHERE condição]
[ORDER BY coluna]
    
```

- Exemplos de **funções de grupo**:





- **Exemplos:**

Calcula o salário médio dos empregados da filial de código 1

```
SELECT AVG (Salario)
FROM Empregado
WHERE CodFilial = 1
```

Calcula o preço médio das revistas por assunto.

```
SELECT ASSUNTO, AVG(Preco)
FROM Revista
GROUP BY assunto;
```

Calcula a quantidade de livros por assunto.

```
SELECT ASSUNTO, COUNT(*)
FROM Livro
GROUP BY assunto;
```

CLÁUSULA GROUP BY

- **Objetivo:** dividir as linhas de uma consulta em grupos menores.

```
SELECT colunas, função_grupo (coluna)
FROM tabela
[WHERE condição]
[GROUP BY coluna]
```

- **Exemplo:**

Calcula o salário médio dos empregados de cada filial

```
SELECT AVG(Salario)
FROM Empregado
GROUP BY CodFilial;
```

Calcula a quantidade de empregados de cada filial

```
SELECT COUNT(Nome)
FROM Empregado
GROUP BY CodFilial;
```

CLÁUSULA HAVING

- **Objetivo: restringir funções de grupo!!** Esta opção só é utilizada combinada com a opção GROUP BY.

```
SELECT colunas, função_grupo (coluna)
FROM tabela
[WHERE condição]
[GROUP BY coluna]
[HAVING função_grupo (coluna)]
[ORDER BY coluna]
```

- **Exemplo:**

Seleciona o código da final e o salário médio dos empregados das filiais que tenham o salário médio maior que 2000.

```
SELECT CodFilial, AVG (Salario)
FROM Empregado
GROUP BY CodFilial
HAVING AVG(Salario) > 2000;
```

Obs.: | A cláusula GROUP BY deve ser colocada ANTES da HAVING, pois os grupos são formados e as funções de grupos são calculadas antes de se resolver a cláusula HAVING.

SUBQUERIES (SUBCONSULTAS)

No processamento deste comando SQL composto, a **subconsulta** é efetuada primeiro, e então o resultado é aplicado à consulta principal.

São comandos SQL utilizados em condições de cláusulas WHERE ou HAVING para prover resultados que são utilizados para completar a consulta principal.

Operadores:

> ANY	Maior que algum valor da lista.
< ANY	Menor que algum valor da lista.
= ANY	Igual a algum valor da lista (equivalente a IN).
> ALL	Maior que todos os valores da lista.
< ALL	Menor que todos os valores da lista.
<> ALL	Diferente de todos os valores da lista.
IN	Existe na lista.
NOT IN	Não existe na lista.
EXISTS	Retorna verdadeiro se determinada subquery retorna ao menos uma linha, e falso em caso contrário.
NOT EXISTS	Produz resultado contrário.

- **Sintaxe de exemplo:**

- ... Salario > ANY (subquery)
- ... Salario < ANY (subquery)
- ... Salario > ALL (subquery)
- ... Salario < ALL (subquery)

- **Veja exemplos:**

Selecionar os empregados (nome) das filiais cujo nome seja ABC.

```
SELECT Nome
FROM Empregado
WHERE CodFilial IN (SELECT CodFilial
                     FROM Filial
                     WHERE Nome='ABC' );
```

Selecionar os empregados (nome) com salário maior ou igual a todos os valores de salário da tabela empregados.

```
SELECT Nome
FROM Empregado
WHERE Salario >= ALL (SELECT Salario
                      FROM Empregado);
```

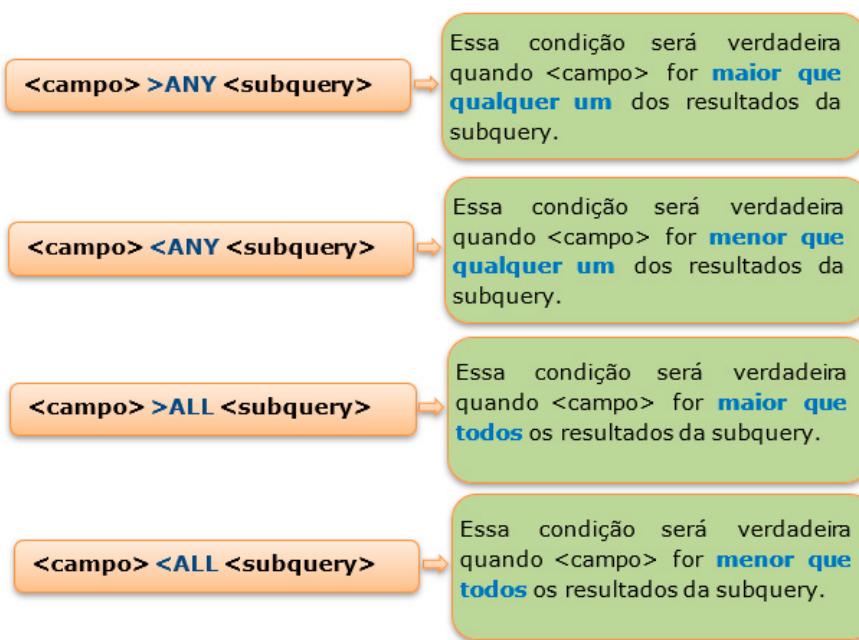
Selecionar os empregados (nome) com salário maior que algum salário da tabela empregados.

```
SELECT Nome
FROM Empregado
WHERE Salario > ANY (SELECT Salario
                      FROM Empregado);
```

Vamos reforçar alguns comandos SQL muito empregados em consultas!

Operadores ANY e ALL

Quando a subquery retornar mais de um valor, os operadores ANY e ALL podem ser utilizados para compatibilizar o resultado da subquery com o tipo do operador de comparação.



Veja exemplo:

Listar os produtos que tenham preço unitário superior ao preço unitário de todos os produtos de hardware.

```

SELECT Descricao
FROM Produto
WHERE Precounitario > ALL (SELECT Precounitario
                            FROM Produto
                            WHERE Tipo='HARDWARE') ;
    
```

Operadores IN e NOT IN

Verifica se o dado faz parte ou não da lista fornecida. A lista pode ser formada por valores retornados por uma subquery.

Veja exemplo:

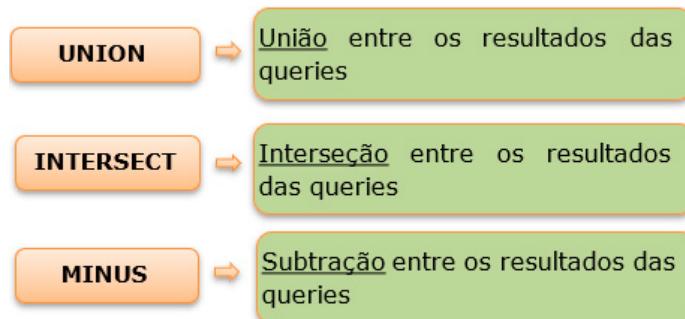
Listar as notas de venda e os respectivos vendedores, em que constem mais de 2 produtos vendidos.

```

SELECT IdNV, NomeVendedor
FROM NotaVenda
WHERE IdNV IN (SELECT IdNV
                FROM ItemNota
                GROUP BY IdNV
                HAVING COUNT(IdProduto) > 2) ;
    
```

Operadores de Conjuntos

Como o resultado de um query é um conjunto de linhas você pode realizar operações de conjuntos entre queries.



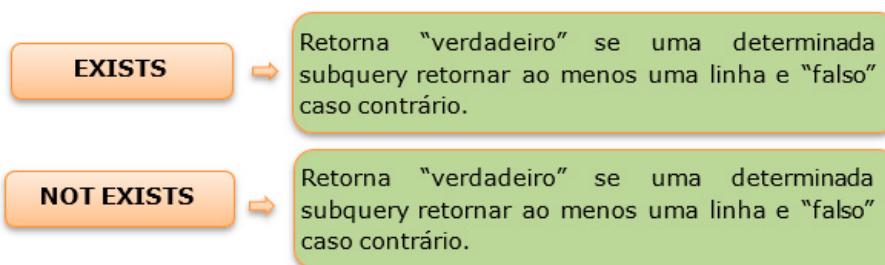
A operação de união permite reunir os resultados de duas consultas distintas em um só resultado. Equivale à operação de União da Álgebra Relacional. A operação de união elimina as linhas duplicadas.

Veja exemplo:

Listar todas as Notas de venda que sejam do vendedor 'JUCA', ou em que conste o produto 14.

```
SELECT IdNV FROM NotaVenda WHERE NomeVendedor=' JUCA'
UNION
SELECT IdNV FROM ItemNota WHERE IdProduto=14;
```

Operadores EXISTS e NOT EXISTS



Veja exemplo:

Listar a descrição de todos os produtos que já foram vendidos.

```
SELECT Descricao
FROM Produto p
WHERE EXISTS (SELECT IdProduto FROM ItemNota i
WHERE p.IdProduto = i.IdProduto);
```

TIPOS DE DOMÍNIO NA SQL

Domínio: conjunto de **valores válidos para uma determinada coluna.**

Alguns exemplos:

char(n)	String de caracteres de tamanho fixo com tamanho n especificado pelo usuário.
varchar(n)	String de caracteres de tamanho variável com tamanho n máximo especificado pelo usuário.
int	Inteiro (um subconjunto finito de inteiros que é dependente da máquina).
numeric(p,d)	Número de ponto fixo , com precisão de p dígitos especificada pelo usuário, com n dígitos à direita do ponto decimal.
float(n)	Número de ponto flutuante , com precisão de pelo menos n dígitos.

LINGUAGENS DE BANCO DE DADOS

A partir de agora, vamos pensar sobre o que faremos com o nosso Modelo Relacional? Como já destacado anteriormente, **o próximo passo é elaborar o projeto físico do Banco de Dados.**

Mas o que é o **projeto físico**, afinal? É simplesmente pegar nosso modelo relacional pronto e transformá-lo em um Banco de Dados, dentro de um SGBD.

Então, como vamos informar ao SGBD que nosso Banco de Dados tem determinadas tabelas, com campos, chaves primárias, etc. Nesse ponto, é que entram as linguagens que foram criadas para “dialogar” com o Banco de Dados.

Vamos reforçar a seguir as principais:

DDL – DATA DEFINITION LANGUAGE OU LINGUAGEM DE DEFINIÇÃO DE DADOS

Quando um banco de dados é criado, ele inicialmente está “vazio”. Assim, antes de começar a consultar e alterar dados é preciso definir onde e como as informações serão gravadas dentro do novo banco; então criam-se diversas tabelas explicitando o tipo de dados de cada campo, as chaves estrangeiras, os índices, as regras, etc.

Para a realização dessa definição dos dados é utilizada uma **DDL (Data Definition Language – Linguagem de Definição de dados).**

Os comandos DDL são armazenados no dicionário de dados (ou **catálogo**). Logo, o dicionário de dados contém os metadados (dados a respeito das estruturas de armazenamento) do banco.

Os principais comandos da DDL são:

- **create table**: cria uma nova tabela com seus campos e define as restrições de campo.
- **create index**: cria um novo índice em uma tabela existente.
- **alter table**: altera as definições de campos e de restrições.
- **create domain**: cria um tipo de dado definido pelo usuário.
- **drop table**: exclui uma tabela existente de um banco de dados
- **drop index**: exclui um índice existente de uma tabela.

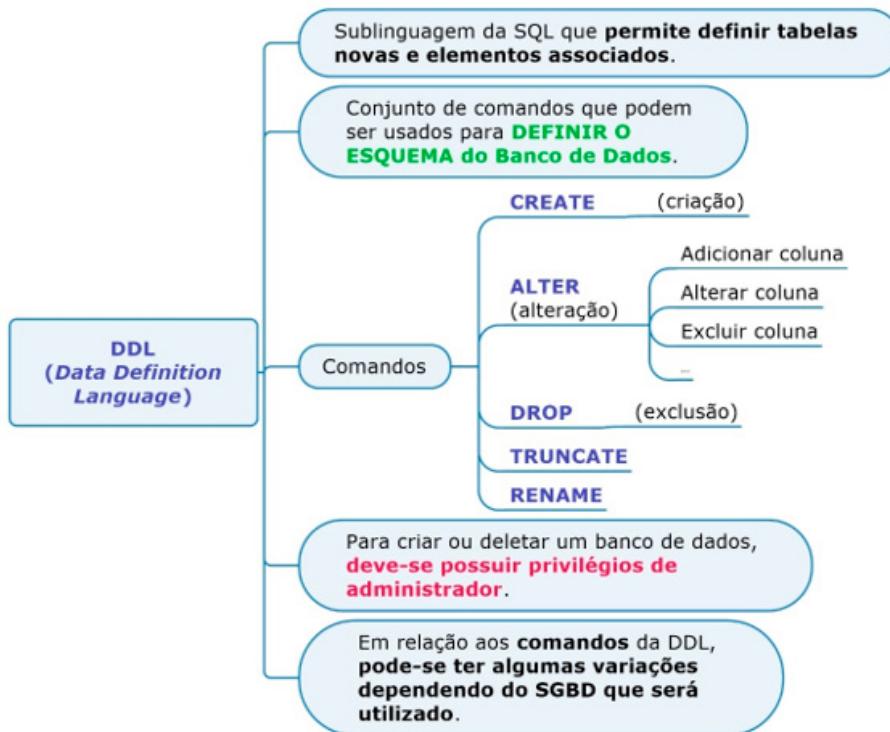


Figura. DDL. Fonte: (QUINTÃO, 2021)

DML – DATA MANIPULATION LANGUAGE OU LINGUAGEM DE MANIPULAÇÃO DE DADOS

Após a carga dos dados nas tabelas criadas pelos comandos DDL, os comandos de manipulação de dados tornam possível sua manipulação, incluindo **inserções**, **atualizações**, **exclusões** e **consultas** (com a utilização do comando SELECT da SQL).

A **DML** visa à manipulação de dados (incluir, alterar, excluir e consultar) por meio do usuário.

Principais comandos:

- **select**: seleção de registros;
- **insert**: inserção de registros;
- **update**: atualização de registros;
- **delete**: deleção de registros

DQL – DATA QUERY LANGUAGE OU LINGUAGEM DE CONSULTA DE DADOS

A linguagem de consulta de dados é um subconjunto da **DML** que possui apenas a instrução de **SELECT**.

DCL – DATA CONTROL LANGUAGE OU LINGUAGEM DE CONTROLE DE DADOS

Permite controlar o acesso dos usuários aos dados em um banco de dados.

Principais comandos:

- **GRANT**: concessão de privilégios a tabelas e visões.
- **REVOKE**: revogação de privilégios a tabelas e visões.

DTL – DATA TRANSACTION LANGUAGE OU LINGUAGEM DE TRANSAÇÃO DE DADOS

Faz **controle de transações** no banco de dados. Principais comandos:

COMMIT	Efetiva alterações feitas na base de dados.
ROLLBACK	Desfaz alterações na base de dados.
SAVEPOINT	Estabelece pontos de retorno dentro de uma transação.

SDL – STORAGE DEFINITION LANGUAGE OU LINGUAGEM DE DEFINIÇÃO DE ARMAZENAMENTO

Utilizada para **especificar o esquema interno de armazenamento**. Pertencia às versões antigas do SQL, mas foi absorvida pela DDL. Alguns autores nem citam mais esta modalidade.

VDL – VIEW DEFINITION LANGUAGE OU LINGUAGEM DE DEFINIÇÃO DE VISÕES

Voltada para **especificar as visões dos usuários** e seus mapeamentos para o esquema conceitual. Ex: *CREATE VIEW, DROP VIEW*.

DIRETO DO CONCURSO

005. (CESPE/MEC/2015) 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.



A linguagem **SQL** possui diversos **comandos** para acessar e manipular dados, criação, exclusão e alteração das tabelas, índices e demais estruturas de bancos de dados, além de comandos para controlar os dados e transações. Podemos subdividir a linguagem nos seguintes subgrupos:

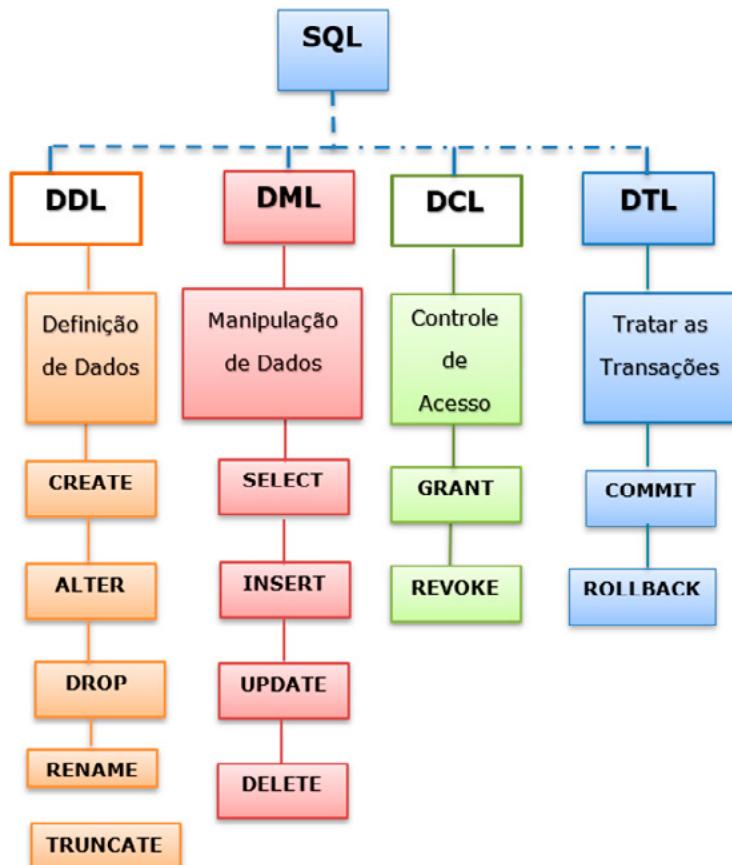


Figura. Comandos da SQL

Conforme visto, a **Linguagem de Definição de Dados (DDL)** é um conjunto de comandos dentro da SQL utilizada para a definição de estrutura dos dados e tabelas. Eles permitem criar, remover ou modificar objetos do banco de dados. Os comandos DDL mais comuns são **CREATE**, **ALTER**, **DROP**, **RENAME** e **TRUNCATE**.

A **Linguagem de Modificação de Dados (DML)** é um conjunto de comandos de modificação de dados. Incluem funcionalidades para a criação, remoção ou alteração de registros nas tabelas. Por exemplo, **INSERT**, **UPDATE**, **DELETE**. Assim, o comando **CREATE é um comando DDL para criar um novo objeto em um banco de dados.**

Errado.

006. (CESPE/TJ-SE/2014) Julgue os itens seguintes, acerca da segurança e do tuning de banco de dados. Para cancelar os privilégios de um usuário a uma tabela do banco de dados, deve-se utilizar o comando **REVOKE**.



A **DCL (Data Control Language ou Linguagem de Controle de Dados)** serve para controlar dados! E qual é o significado dela? Essa linguagem possui um conjunto de comandos para lidar com **autorizações de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados.**

São exemplos de **comandos** dessa linguagem:

- **GRANT**: utilizado para conceder permissão a um usuário em relação a algum objeto, assim altera as permissões em objetos-esquema;
- **REVOKE**: utilizado para remover ou restringir a capacidade de um usuário de executar operações. É o inverso do comando GRANT.

Ao executarmos o comando **REVOKE <privilegio> FROM <usuário>**, estamos **cancelando** a permissão que aquele usuário tem de realizar a operação.

Certo.

007. (CESPE/STM/2011) Os comandos do grupo DDL (Data Definition Language) do SQL permitem gerar os dados das tabelas que formam um banco de dados.



A questão deixa dúvidas quando menciona “gerar os dados das tabelas”. Quando falamos em **dados** pensamos nas informações armazenadas nas tabelas e isso se trata de comandos **DML (Data Manipulation Language)**, portanto, estaria errada a afirmação. Por outro lado, a palavra “dados” pode ser interpretada como a estrutura da tabela, ou seja, os campos, as restrições de integridade e os tipos de dados.

Diante da polêmica, essa questão teve seu gabarito alterado de **ERRADO** para **CERTO** usando o seguinte argumento como justificativa: “*Os comandos do grupo DDL - Data Definition Language - do SQL permitem gerar tabelas que formam um banco de dados, porém, as estruturas e os conteúdos das tabelas devem ser definidos anteriormente. Dessa forma, opta-se pela alteração do gabarito*”

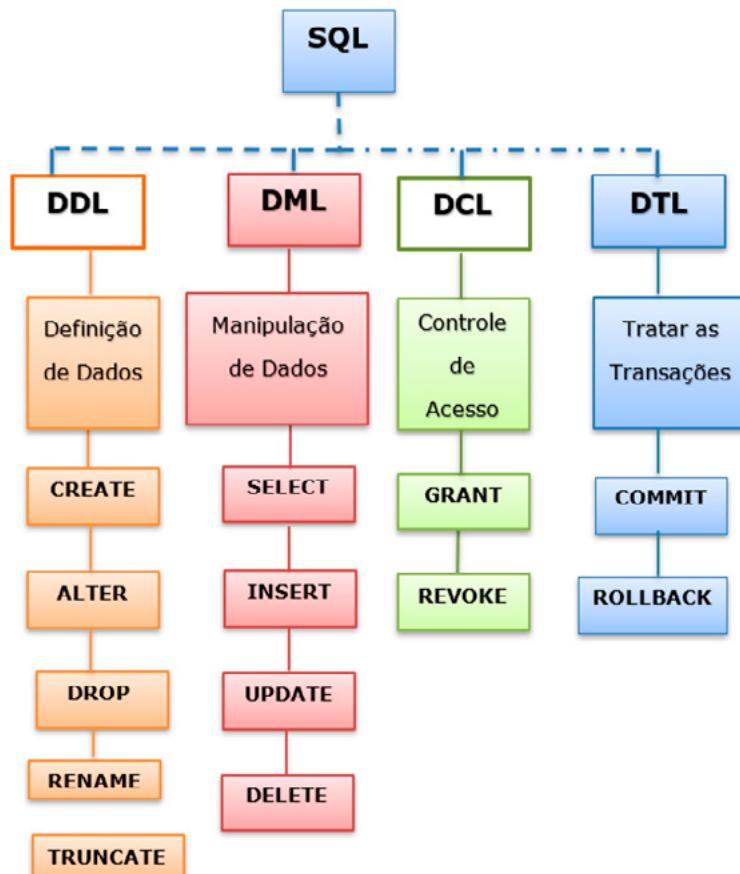
Certo.

008. (UFPR/ITAIPU/2015) Sobre a Linguagem SQL (DDL e DML), assinale a alternativa correta segundo o padrão SQL ANSI 92.

- a) Insert, Delete e Update são operações DDL.
- b) Inner join, left outer join, right outer join, full outer join e cross join são cláusulas de junções suportadas pelo padrão.
- c) Modify Table e Truncate Table são operações DDL.
- d) ADD, EXISTS, BETWEEN, LIKE, IS e IN são operadores relacionais válidos segundo o padrão.
- e) CASE, NVL, TRIM e UPPER são operações DDL.



a) Errada. A linguagem SQL pode ser subdividida conforme a estrutura organizada no diagrama a seguir:



b) Certa. As cláusulas de **junção** são determinadas exatamente por:

Inner Join	Retorna linhas quando <u>existe pelo menos uma combinação em ambas as tabelas.</u>
Left Join	Retorna todas as linhas da tabela da esquerda , mesmo que as linhas não combinem com a tabela da direita.
Right Join	Retorna todas as linhas da tabela da direita , mesmo que as linhas não combinem com a tabela da esquerda.
Full join	Retorna todas as linhas quando têm ou não uma combinação entre elas.
Cross Join	Retorna a união de duas ou mais tabelas por cruzamento. Ou seja, para cada linha de uma tabela faz-se a junção com todas as linhas de outra tabela.

É importante observar que a instrução pode ser escrita de forma resumida, suprimindo a palavra OUTER em left outer join, right outer join e full outer join.

- c) Errada. Os comandos Truncate e Modify não alteram objetos na estrutura da base de dados e, portanto, não podem ser classificados como DDL.
- d) Errada. Os comandos ADD, EXISTS, BETWEEN, LIKE, IS e IN não são considerados operadores relacionais porque não comparam elementos. Exemplos de operadores de relação são: diferente(<>), maior ou igual (>=), igualdade (=), etc.
- e) Errada. Os comandos CASE, NVL, TRIM e UPPER são exemplos de funções implementadas em alguns sistemas gerenciadores de banco de dados e não podem ser classificadas como DDL.

Letra b.

ÍNDICES

- **Os índices aceleram a recuperação dos dados.**

Por exemplo, um índice para um arquivo em um sistema de banco de dados funciona da mesma forma que um índice em um livro de 800 páginas. Para procurar determinado conteúdo, você lê o índice no início do livro, acha o item desejado e pula diretamente para a página do assunto, para depois realizar a pesquisa de maneira mais refinada. Uma pesquisa talvez não fosse tão preocupante nesse contexto, no entanto se você precisar de várias pesquisas, seria muito desagradável ficar horas procurando o conteúdo que deseja estudar. No banco de dados, é a “mesma coisa”.

- Exemplos de **variáveis que precisam ser consideradas antes de se escolher o tipo de índice mais apropriado:**

Tipos de acesso	Maneira pela qual se localiza um registro e o seu grau de eficiência.
Tempo de inserção	Tempo contabilizado para a inserção do registro. Envolve encontrar o local para inserir o novo item de dados e o tempo para atualizar a estrutura de índice.
Tempo de exclusão	Tempo contabilizado para a exclusão do registro. A depender do algoritmo escolhido, excluir um registo pode ser muito mais trabalhoso do que inseri-lo.
Espaço adicional	Estruturas de índices ocupam espaço em disco. Geralmente sua implementação é compensadora, pois é realizada em bancos de dados volumosos.

Considere as seguintes observações (DEVMEDIA,2020):

- Os **índices** são utilizados, principalmente, para **melhorar o desempenho do banco de dados** (embora a utilização não apropriada possa resultar em uma **degradação desse desempenho**).
- **Você pode criar tantos índices quantos desejar em qualquer tabela** (É possível ter um índice para cada coluna da tabela, assim como um índice para uma combinação de colunas).
- **A eliminação de um índice não elimina as tabelas ou visões relacionadas com o índice.**
- Os índices são muito bons no sentido de *performance* do banco de dados, otimizam as buscas de dados, mas, por outro lado, consomem muito espaço em disco, o que pode se tornar concorrente do próprio banco se você o detém em um espaço generoso ou pode se tornar caro quando está armazenado em um equipamento *storage*.
- **Excesso de índices pode ser tão prejudicial quanto sua falta!**
- Quando colunas indexadas são modificadas, o SGBD desloca recurso internamente para manter esses índices atualizados e associados.
- **A manutenção de índices requer tempo e recursos**, portanto, não crie índices que não serão usados efetivamente.
- Quando se contém grande quantidade de dados duplicados, índices apresentam mais custo que benefícios, assim como usar índices com atributos de pouca variação, como “sexo”.
- Para possibilitar acessos aleatórios rápidos aos registros de um arquivo, uma estrutura de índice pode ser utilizada.
- Em um arquivo organizado com índice sequencial, o desempenho dos acessos ao arquivo tipicamente piora na medida em que o arquivo cresce. Para evitar essa degradação, há SGBDs que usam uma estrutura árvore-B+ para implementar índices de múltiplos níveis.

RESUMO

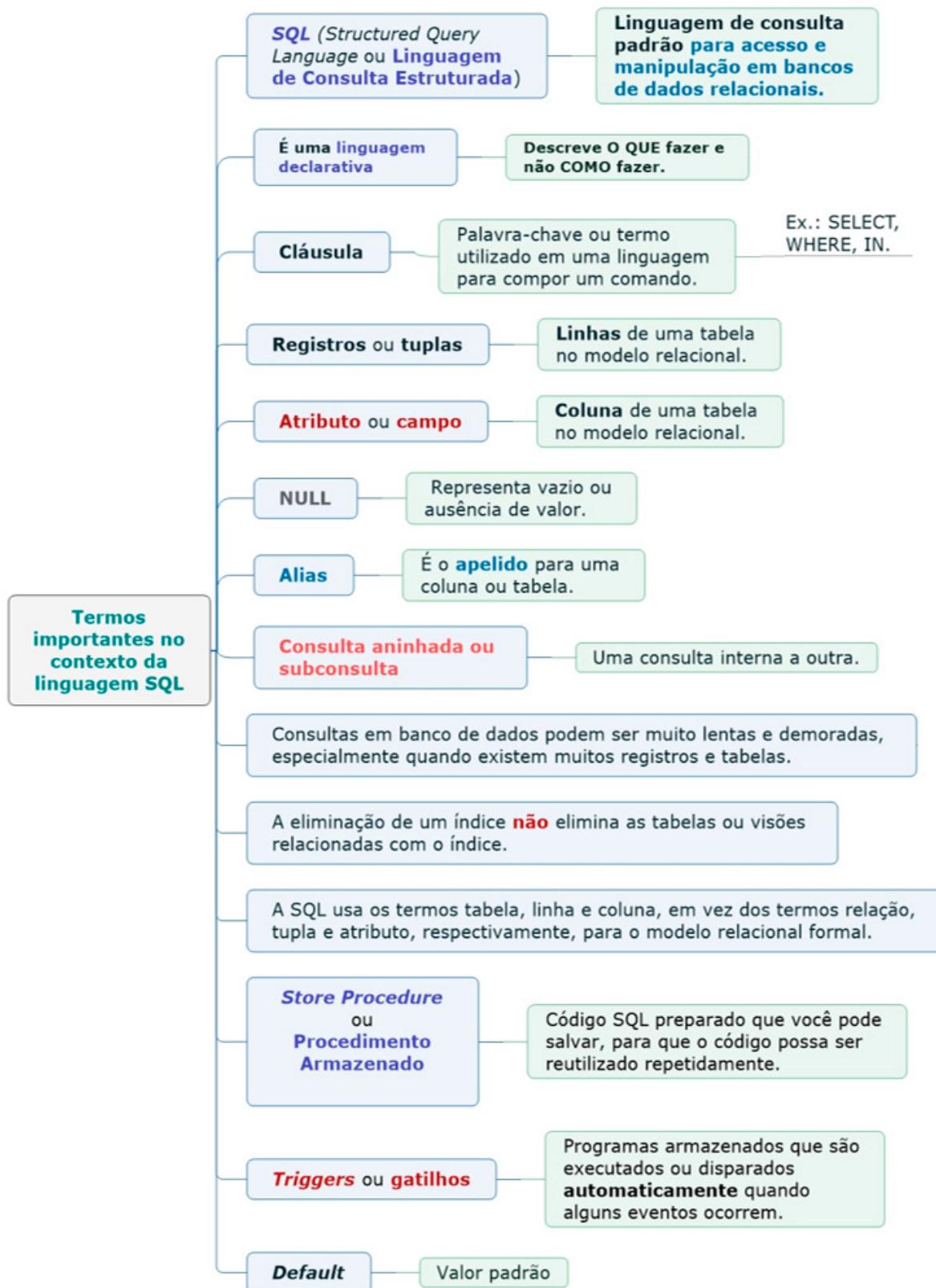


Figura. SQL. Fonte: (QUINTÃO, 2021)

DROP x DELETE x TRUNCATE

O comando **DROP** remove a tabela inteira da base de dados que inclui a sua estrutura e registros. Remove, portanto, não somente os dados da tabela, mas também a própria definição do objeto. A tabela, seus dados e restrições deixam de existir no banco de dados. Veja exemplo:

DROP TABLE teste;

O comando **DELETE** remove os registros, mas a estrutura da tabela permanece a mesma. Podemos deletar apenas algumas linhas usando a cláusula WHERE. Usando o comando **DELETE** podemos reverter a operação com o comando ROLLBACK. Veja exemplo:

```
DELETE FROM teste
WHERE x=10;
ROLLBACK;
```

O comando **TRUNCATE TABLE** irá remover TODAS as linhas de uma tabela sem registrar as exclusões de linhas individuais. O **TRUNCATE TABLE** funciona como uma instrução **DELETE**, mas sem usar a cláusula WHERE.

Veja exemplo:

TRUNCATE TABLE teste;

A tabela-resumo seguinte destaca comandos principais da linguagem, MUITO cobrados em provas!

Linguagem	Comando	Descrição
DDL - Data Definition Language ou Linguagem de Definição de Dados	Create	Cria objetos na base de dados.
	Alter	Altera a estrutura de um objeto da base.
	Drop	Elimina determinado objeto da base de dados.
DML - Data Manipulation Language ou Linguagem de Manipulação de Dados	Select	Seleciona dados de uma base de dados.
	Insert	Insere linhas em uma tabela.
	Update	Altera os valores das linhas de uma tabela.
	Delete	Exclui linhas em uma tabela.

Linguagem	Comando	Descrição
DCL - Data Control Language ou Linguagem de Controle de Dados	Grant	Fornece permissão de acesso em objetos do banco de dados.
	Revoke	Retira permissão de acesso em objetos do banco de dados.
DTL -Data Transaction Language ou Linguagem de Transação de Dados	Commit	Efetiva (torna permanente!) as alterações feitas na base de dados.
	Rollback	Defaz, em caso de falha do sistema, qualquer alteração que ainda não tenha sido efetivada, de modo que a integridade do banco de dados seja mantida. Permite também ao usuário desfazer qualquer alteração feita no banco de dados antes que tenha sido executado um comando COMMIT.
	Savepoint	Estabelece pontos de retorno dentro de uma transação.

Restrição de Integridade (RI):

- Descreve as **condições que cada instância legal de uma relação deve satisfazer**.
- Inserts/deletes/updates que violam RI's **não** são permitidas.
- Podem ser usadas para **assegurar a semântica da aplicação** (ie., *cod_autor* é uma chave), ou **prevenir inconsistências** (ie., *titulo* tem que ser um string).
- **Tipos de RI's**: restrições de domínios, restrições de chave primária, restrições de chave estrangeira, restrições gerais.
- **NOT NULL** especifica que uma coluna não pode receber valores nulos.
- **UNIQUE** define que uma coluna não pode ter valores duplicados.
- **DEFAULT** atribui um valor padrão para uma determinada coluna a ser utilizado, caso nenhum valor seja passado como parâmetro no momento da inserção.

Veja mais: <https://www.inf.ufsc.br/~mario.dantas/cap6.pdf>

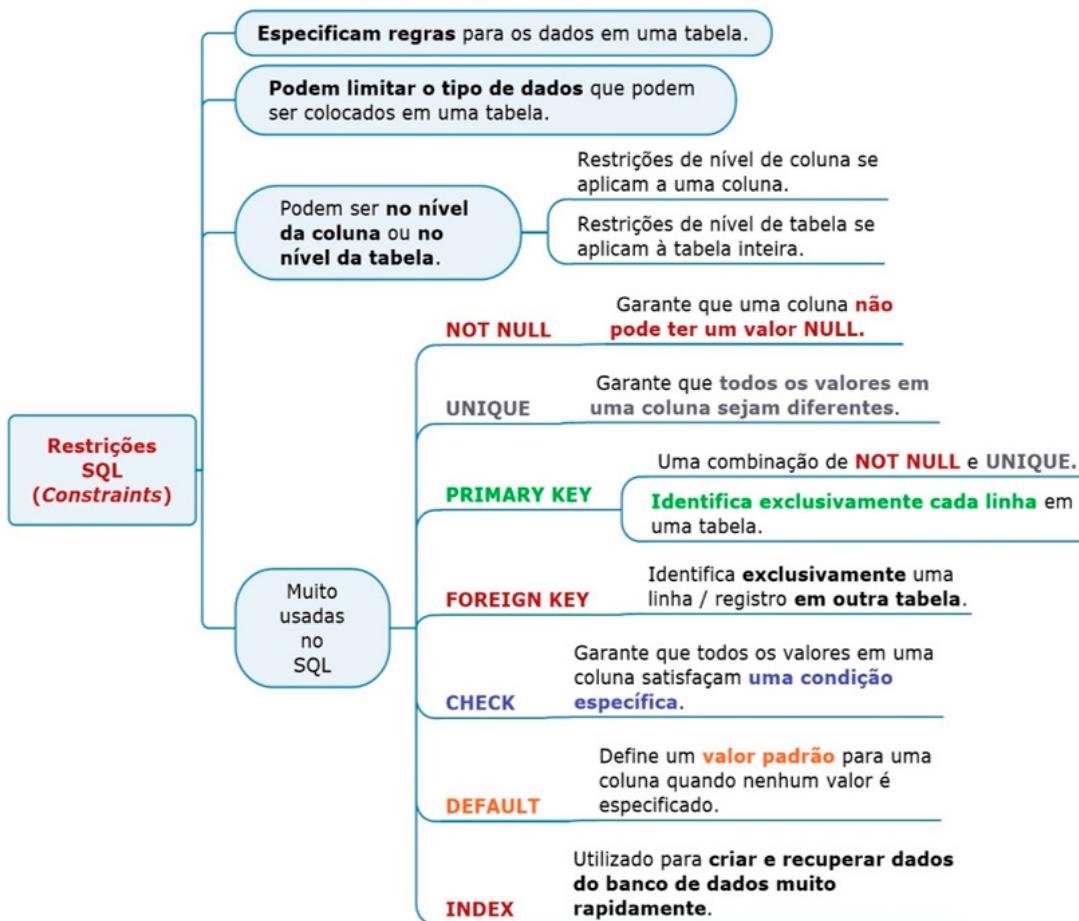


Figura. Restrições (QUINTÃO, 2021)

Mais observações/exemplos sobre SQL, IMPORTANTES para fixação.

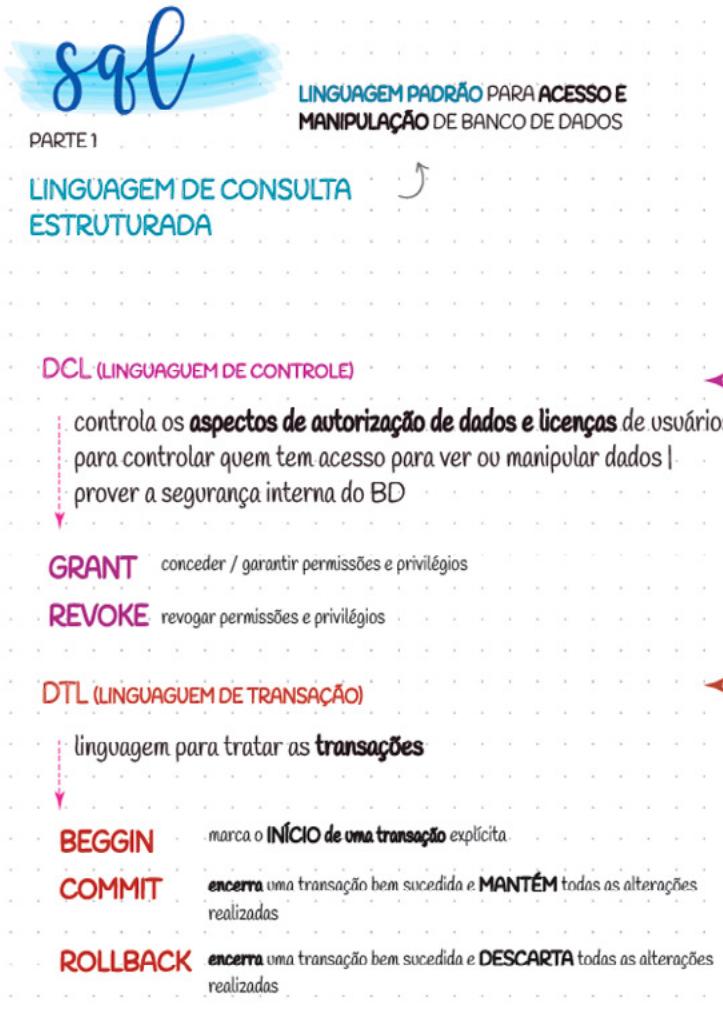


Figura. SQL. Fonte: Clube dos Mapas por @paola.tuzani

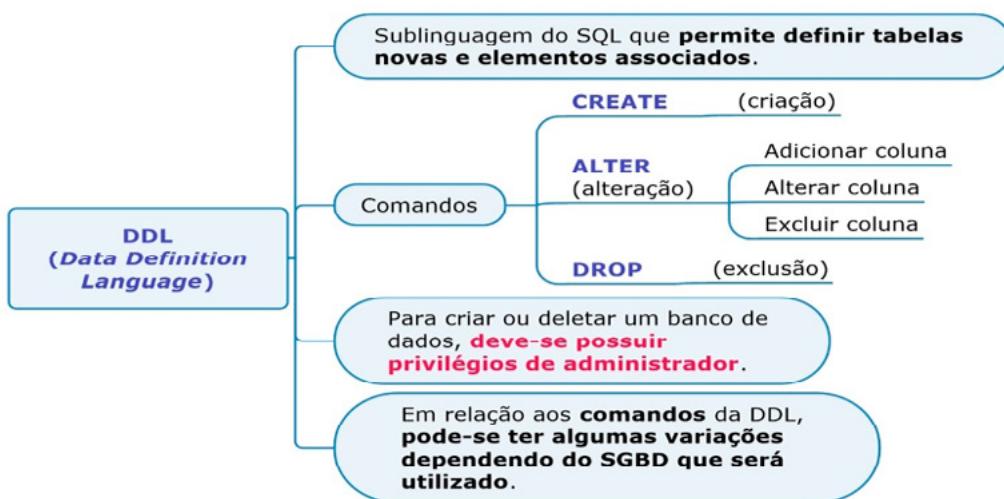


Figura. DDL. Fonte: (QUINTÃO, 2021)

sql

PARTE 2

SELECT (DML - MANIPULAÇÃO)

recuperar informações de um BD | operação de projeção

sintaxe básica

`SELECT coluna1, coluna2, ... FROM nome_da_tabela WHERE condição`

= lista de atributos

* para não especificar colunas | retorna todos os atributos

SELECT DISTINCT

retorna apenas valores diferentes ou distintos
elimina duplicidades

cláusula opcional!
consulta pode ser realizada sem condição

cláusula AS alias | apelido

NOME TEMPORÁRIO A UMA TABELA OU COLUNA RETORNADOS

PARA TORNAR OS NOMES DAS COLUNAS MAIS LEGÍVEIS

EXISTE APENAS PARA A DURAÇÃO DA CONSULTA

PODE SER OMITIDA

condição

cláusula WHERE

mais de uma condição

operadores

=

<

<=

>

>=

≠ (diferente)

between (intervalo | entre)

like (padrão | % qualquer caractere | _ único caractere específico)

in (possíveis valores)

and TODAS AS CONDIÇÕES SÃO VERDADEIRA

or PELO MENOS UMA CONDIÇÃO É VERDADEIRA

not NENHUMA DAS CONDIÇÕES É VERDADEIRA

cláusula ORDER BY

ordenação das tuplas/linhas no resultado da consulta

desc ORDEM DECRESCENTE

asc ORDEM CRESCENTE (padrão)

`SELECT * FROM Clientes WHERE NOT País='Mexico' ORDER BY País DESC`

retornará todos os atributos/colunas da tabela clientes exceto méxico em ordem decrescente de países

sql

PARTE 3

SELECT (DML - MANIPULAÇÃO)

produto cartesiano

combição de linhas/tuplas de duas ou mais tabelas/relações, independentemente de ter os mesmos valores em atributos/columnas comuns | o resultado do produto cartesiano de duas tabelas é uma terceira tabela contendo todas as combinações possíveis entre as linhas das tabelas originais, de modo que a tabela resultante possuirá um número de colunas que é igual à soma das quantidades de colunas das duas relações iniciais, e um número de linhas igual ao produto do número de suas linhas

SELECT tabela1.coluna1, tabela2.coluna2, ... **FROM** tabela1, tabela2
WHERE condição

É INDICADO PELO USO DE **VÍRGULA (,)**

ENTRE AS TABELAS REFERENCIADAS

para verificar quais as tabelas em que os dados serão buscados

funções de agregação

são usadas para resumir informações de várias tuplas em uma síntese de tupla única

SELECT FUNCAO(coluna) FROM nome_da_tabela WHERE condição

min MENOR VALOR DE UMA COLUNA

max MAIOR VALOR DE UMA COLUNA

count NÚMERO DE LINHAS QUE ATENDE A UM CRITÉRIO | QUANTIDADE

avg MÉDIA DOS VALORES DE UMA COLUNA NUMÉRICA

sum SOMA DOS VALORES DE UMA COLUNA NUMÉRICA

agrupamentos - GROUP BY

especifica os atributos de agrupamento, que também devem aparecer na cláusula SELECT

haven CONDIÇÃO COM FUNÇÃO AGREGADORA

SELECT FornecedorID, COUNT(ProdutoID) FROM Produtos GROUP BY FornecedorID HAVING COUNT(ProdutoID) <= 2

consulta a quantidade de produtos que são fornecidos por cada fornecedor, porém somente aqueles que fornecem uma quantidade de produtos menor ou igual a 2.

Figura. SQL. Fonte: Clube dos Mapas por @paola.tuzani

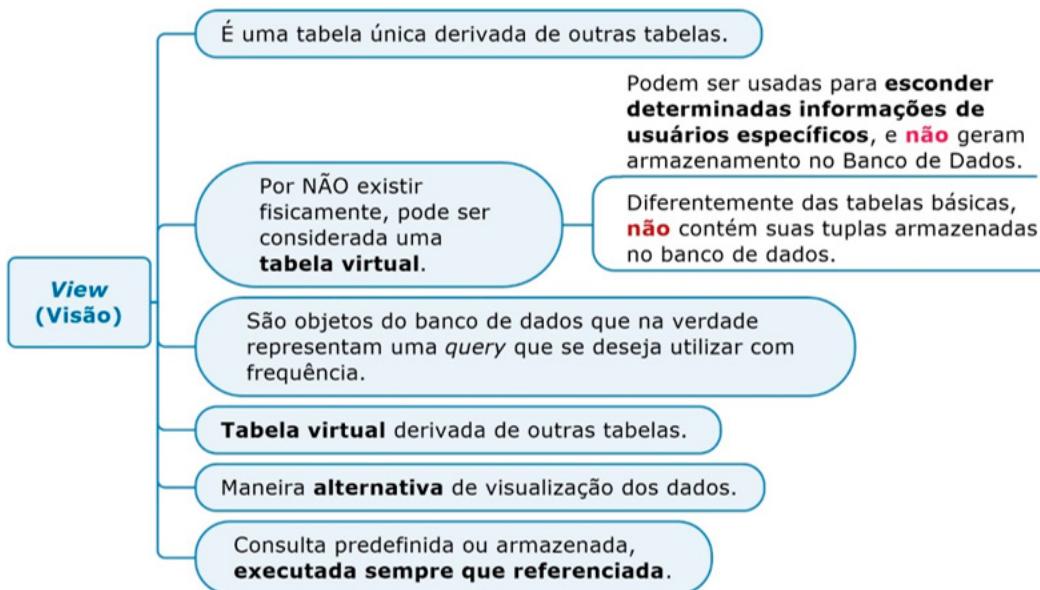


Figura. View (Visão). Fonte: (QUINTÃO, 2021)

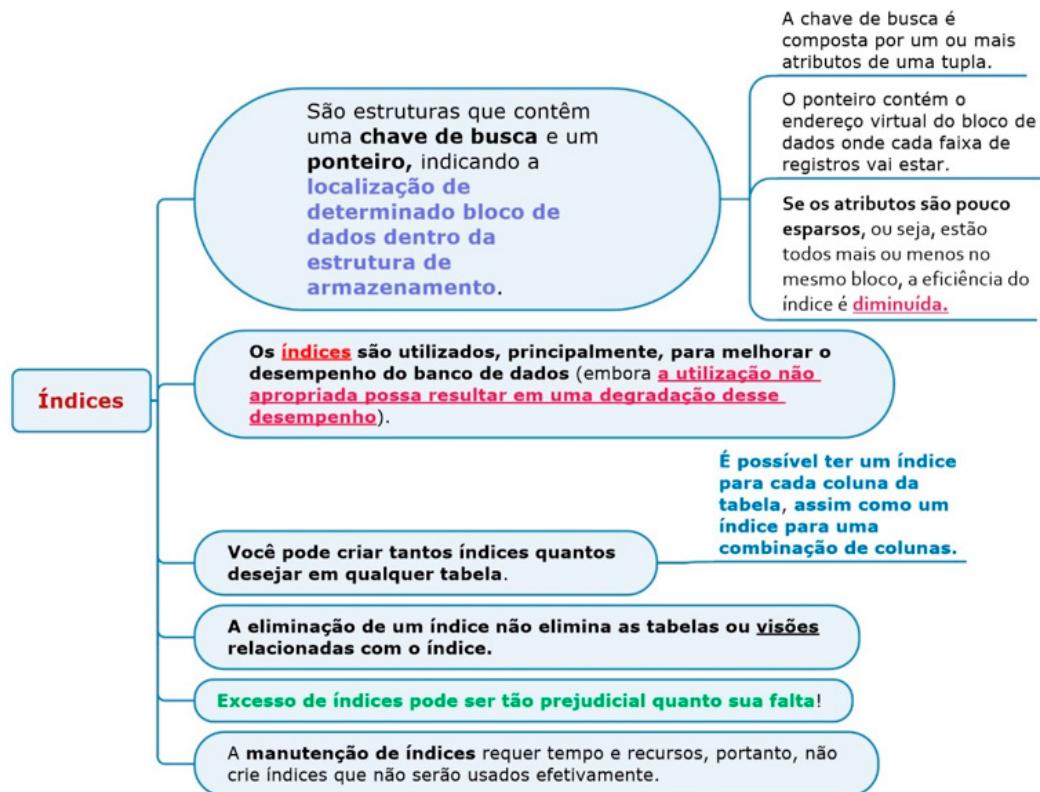


Figura. Índices. Fonte: (QUINTÃO, 2021)

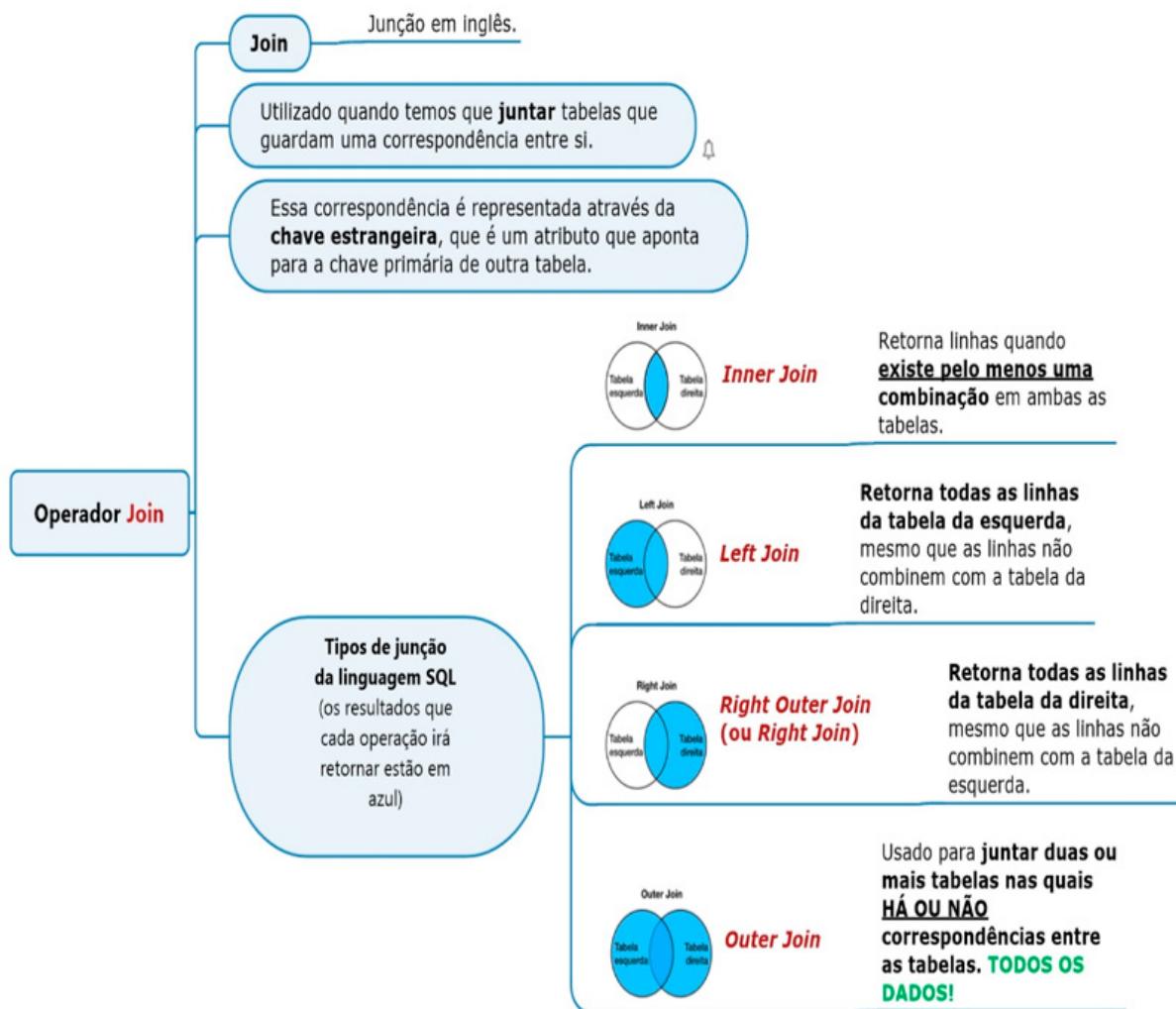


Figura. Operador Join. Fonte: (QUINTÃO, 2021)

QUESTÕES COMENTADAS EM AULA

001. (CESPE/TRE-BA/TÉCNICO JUDICIÁRIO/TELECOMUNICAÇÕES E ELETRICIDADE/2010) Uma linguagem de acesso a bancos de dados relacionais amplamente usada é o SQL.

002. (FCC/TCE-SE/ANALISTA DE CONTROLE EXTERNO/COORDENADORIA DE INFORMÁTICA/2011) Durante a criação de uma tabela - Create Table, em SQL, deseja-se especificar que uma coluna só possa incluir, por exemplo, valores maiores que zero. Uma constraint utilizada para isso é

- a) Verify.
- b) Check.
- c) Max.
- d) Avg.
- e) Having.

003. (ESAF/ANA/ANALISTA ADMINISTRATIVO/TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO/2009) Em SQL, a cláusula check aplicada a uma declaração de domínio

- a) permite especificar um predicado que deve ser satisfeito por qualquer valor atribuído a uma variável de determinado domínio.
- b) especifica um predicado que deve ser satisfeito por uma tupla em uma relação.
- c) proíbe a inserção de um valor nulo para as variáveis do domínio.
- d) verifica se os atributos considerados formam uma chave candidata.
- e) não tem efeito, pois não se aplica esta cláusula a declarações de domínio.

004. (CESPE/FUB/2018) 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.

005. (CESPE/MEC/2015) 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.

006. (CESPE/TJ-SE/2014) Julgue os itens seguintes, acerca da segurança e do tuning de banco de dados. Para cancelar os privilégios de um usuário a uma tabela do banco de dados, deve-se utilizar o comando REVOKE.

007. (CESPE/STM/2011) Os comandos do grupo DDL (Data Definition Language) do SQL permitem gerar os dados das tabelas que formam um banco de dados.

008. (UFPR/ITAIPU/2015) Sobre a Linguagem SQL (DDL e DML), assinale a alternativa correta segundo o padrão SQL ANSI 92.

- a)** Insert, Delete e Update são operações DDL.
- b)** Inner join, left outer join, right outer join, full outer join e cross join são cláusulas de junções suportadas pelo padrão.
- c)** Modify Table e Truncate Table são operações DDL.
- d)** ADD, EXISTS, BETWEEN, LIKE, IS e IN são operadores relacionais válidos segundo o padrão.
- e)** CASE, NVL, TRIM e UPPER são operações DDL.

QUESTÕES DE CONCURSO

009. (CESPE/CEBRASPE/MINISTÉRIO DA ECONOMIA/TECNOLOGIA DA INFORMAÇÃO/CIÊNCIA DE DADOS/2020) 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.



Para criar uma tabela, utilizaremos o comando **CREATE TABLE TAB**; o comando **CREATE DATABASE** é usado para criar bancos de dados. Dessa forma, o comando CREATE DATABASE TAB irá realizar a criação de um banco de dados chamado “TAB”.

Errado.

010. (CESPE/CEBRASPE/TJ-AM/2019) Julgue o próximo item, relativos a sistema gerenciador de banco de dados (SGBD). Um SGBD trata do acesso ao banco e pode ser executado independentemente pelo Oracle, MySQL ou PostgreSQL; no entanto, cada SGBD utiliza DML (data manipulation language) e DDL (data definition language) específicas.



A **linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada)** é a **linguagem padrão para acesso e manipulação de bancos de dados**. Os SGBDs (como Oracle, MySQL, PostgreSQL, etc.) costumam utilizar o padrão definido pelo *American National Standards Institute (ANSI)* para a linguagem SQL, mas pode-se implementar algumas variações dos comandos definidos nesse padrão, além de apresentar comandos próprios e funções predefinidas. Portanto, a **assertiva está correta**.

Certo.

011. (CESPE/TRE-PE/ANALISTA JUDICIÁRIO/ANÁLISE DE SISTEMAS/2017) 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 ponto e vírgula
- b) ALTER TABLE tbproduto ADD COLUMN idcategoria INT ponto e vírgula
- c) UPDATE TABLE tbproduto ADD COLUMN idcategoria INT ponto e vírgula
- d) ADD COLUMN idcategoria INT IN TABLE tbproduto ponto e vírgula
- e) UPDATE TABLE ADD COLUMN idcategoria INT IN tbproduto ponto e vírgula



A sintaxe de SQL para adicionar um campo em uma tabela é:

ALTER TABLE nome_da_tabela ADD nome_da_coluna tipo_da_coluna;

Das alternativas, a que adiciona um campo chamado idcategoria é a letra B: ALTER TABLE tbproduto ADD COLUMN idcategoria INT;

As outras alternativas estão com a sintaxe incorreta.

Letra b.

012. (CESPE/FUB/TÉCNICO DE TECNOLOGIA DA INFORMAÇÃO/2016) A respeito das principais instruções da linguagem SQL, julgue o item subsecutivo. SELECT é uma instrução de controle de banco de dados que permite recuperar o conteúdo de uma ou mais tabelas.



Conforme visto a seguir, select é uma instrução de manipulação de dados, usada para recuperar dados do banco de dados.

Linguagem	Comando	Descrição
DML - Data Manipulation Language ou Linguagem de Manipulação de Dados	Select	Seleciona dados de uma base de dados.
	Insert	Insere linhas em uma tabela.
	Update	Altera os valores das linhas de uma tabela.
	Delete	Exclui linhas em uma tabela.
DCL - Data Control Language ou Linguagem de Controle de Dados	Grant	Fornece permissão de acesso em objetos do banco de dados.
	Revoke	Retira permissão de acesso em objetos do banco de dados.

A estrutura básica de uma consulta em SQL consiste em TRÊS cláusulas: **SELECT, FROM e WHERE**. Assim, uma consulta típica em SQL tem a forma:

```
SELECT coluna1, coluna2, ..., colunaN
FROM nome_da_tabela
WHERE condição;
```

A cláusula **SELECT** relaciona as **colunas que se quer presentes no resultado da consulta**. Usado para recuperar dados do banco de dados.

A cláusula FROM associa a tabela ou tabelas que serão pesquisadas durante a avaliação de uma expressão. Em outras palavras, é uma **lista de relações a serem varridas na execução da expressão**.

A cláusula WHERE consiste em um **predicado envolvendo atributos da relação que aparece na cláusula FROM**.

Errado.

013. (CESPE/FUB/TÉCNICO DE TECNOLOGIA DA INFORMAÇÃO/2016) A respeito das principais instruções da linguagem SQL, julgue o item subsecutivo.

O operador BETWEEN-AND retornará verdadeiro se o valor da coluna na cláusula WHERE for maior ou igual ao primeiro valor e menor ou igual ao segundo valor.



Os operadores BETWEEN e AND permitem especificar um critério durante uma determinada Consulta. Tal critério se materializa no formato de um **intervalo de valores** nos registros de uma tabela.

Dessa forma, foi solicitada a interpretação da cláusula BETWEEN-AND utilizando operadores lógicos.

Vide a seguir um exemplo de cada um dos métodos aqui destacados:

Select * from discos where disco_id Between 10 And 12

=> Serão solicitadas todas as linhas cujo valor da coluna disco_id esteja compreendido entre 10 e 12. Poderíamos utilizar também:

Select * from discos where disco_id >= 10 And disco_id <=12

Certo.

014. (CESPE/SERPRO/TÉCNICO/PROGRAMAÇÃO E CONTROLE DE SERVIÇOS DE TECNOLOGIA DA INFORMAÇÃO/2013) Julgue os itens seguintes, relativos à manipulação de dados em sistemas de computação. Nesse sentido, considere que a sigla SGBD, sempre que empregada, se refere a sistema gerenciador de banco de dados.

A linguagem de consulta estruturada, ou *Structured Query Language* (SQL), que serve para descrever estruturas de dados e esquemas, é uma linguagem de pesquisa declarativa padrão para banco de dados relacionais.



SQL é uma linguagem de pesquisa declarativa para Bancos de Dados Relacionais em oposição a outras linguagens procedurais. **Por ser não procedural, você especifica QUAL informação quer, e não como trazê-la.** Em outras palavras, não é necessário especificar o método de acesso aos dados. O SGBD usa o “otimizador” para interpretar o comando SQL e escolher o melhor caminho para acesso aos dados.

Certo.

015. (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

Considerando-se os campos e dados contidos na tabela 3A6AAA, denominada tbproduto, é correto afirmar que o comando S Q L

a) `SELECT MAX(preco) FROM tbproduto WHERE idtipo`

igual a 5 ponto e vírgula

retornará 14,00 como resultado.

b) `SELECT sum(preco) FROM tbproduto WHERE idtipo`

igual a 5

`GROUP BY preco`

`HAVING preco menor que 14 ponto e vírgula`

retornará dois registros.

c) `SELECT sum(preco) as total FROM tbproduto`

`WHERE idtipo in (1,5) and nome like abre aspa`

`simples porcentagem e fecha aspa simples group`

`by idtipo having sum(preco) maior que 13 ponto`

e vírgula

retornará 26,00 como resultado.

d) `SELECT nome FROM tbproduto WHERE idtipo not in`

`(5) and preco maior que (select min(preco))`

from tbproduto where idtipo igual a 1) ponto

e vírgula

retornará apenas leite como resultado.

e) `SELECT asterisco FROM tb produto WHERE preco`

`BETWEEN 10 AND 12 ponto e vírgula`

retornará cinco registros.



a) Errada. Este comando procura pelo maior preço, indicado por `MAX(preco)`, dentro da tabela `tbproduto` em que o `idtipo` tem o valor 5. Ao procurar todos os registros cujo `idtipo` é igual a 5, encontramos 2 registros. Um deles é o sabão e tem o preço de 11,00 e o outro é o shampoo, cujo preço é de 12,30. Queremos o que tem o maior preço de todos. Logo a resposta é 12,30.

b) Certa. O comando utilizado na assertiva B é interessante. Mas, cuidado para não confundir o comando da alternativa com o comando a seguir: “`SELECT sum(preco) FROM tbproduto WHERE idtipo=5`”. Este último comando mostra a soma de todos os preços dos registros cujo `idtipo` seja 5.

Observe ainda que o comando da alternativa não é exatamente isso. Ele tem a cláusula “`GROUP BY`” seguido do “`HAVING`”. O “`GROUP BY`” vai fazer com que o comando mostre todos os preços antes de fazer a soma. Neste caso, ele irá mostrar os valores 11,00 e 12,30. Estes valores são menores do que 14, o que satisfaz a condição do `HAVING`.

c) Errada. Quando colocamos a condição “`nome like '%e'`”, significa que o campo nome pode dar “match” com qualquer sequência de caracteres, seguida do caractere e. O % é um caractere coringa e significa uma sequência qualquer de caractere. Outro exemplo seria “`nome like '%Oliveira'`”. Esta condição indica que qualquer sequência de caracteres seguida de um “Oliveira” vai resultar em verdadeiro.

Analizando a consulta inteira, vemos que ela irá atuar da seguinte forma: para cada `idtipo` de 1 até 5 (`idtipo in (1, 5)`), considerando somente os produtos que tenham o nome terminado pela letra ‘e’ (`nome like '%e'`), serão somados todos os preços relativos a cada `idtipo`. A soma de cada `idtipo` só aparecerá no resultado final caso ela seja maior do que 13 (`having sum(preco) > 13`).

Então o resultado será:

Total

40,00

Portanto, item errado.

Observação da letra c):

O resultado parcial, antes de “`having sum(preco) > 13`” é:

40,00

11,50

O valor 40 vem da soma dos preços 14 (leite), 12,75 (carne) e 13,25 (azeite), todos do idtipo 1, cujos nomes terminam com a letra 'e'. O 11,50 vem do creme, que tem o idtipo igual a 3. Porém, a linha relativa a 11,50 não entrará por causa do "having sum(preco) > 13".

d) Errada. Primeiro temos que analisar a consulta interna "select min(preco) from tbproduto where idtipo = 1".

Ela é uma consulta simples e procura pelo menor preço (min(preco)) de todos os produtos cujo idtipo é igual a 1. Este resultado é 12,75.

A consulta então passa a ser: SELECT nome FROM tbproduto WHERE idtipo not in (5) and preco > 12,75.

A alternativa quer o nome do produto, cujo idtipo não esteja no conjunto (5), isto é, que esteja dentro de (1, 2, 3 e 4) e cujo preço seja maior do que 12,75.

O resultado da consulta terá dois registros, que são o leite e o azeite.

e) Errada. A consulta dessa alternativa procura por todos os registros que têm o preço entre 10,00 e 12,00. De todos, o creme, cujo preço é 11,50, e o sabão, cujo preço é 11,00, são retornados pela consulta.

Letra b.

016. (CESPE/TCE-PA/AUDITOR DE CONTROLE EXTERNO/ÁREA INFORMÁTICA/ANALISTA DE SUPORTE/2016) No que concerne à linguagem SQL, julgue o item seguinte:

Para que um usuário possa executar o comando select em uma view, não é necessário que ele tenha esse privilégio diretamente na view, mas apenas na tabela a que a view faz referência.



Um **Sistema Gerenciador de Banco de Dados (SGBD)** concede privilégios ou permissões para views, tabelas, procedures, funções de maneira independente. Deste modo, ter o privilégio para visualizar uma tabela não implica que terá acesso à view derivada.

Os **privilégios** são concedidos de forma **independente** nas **tabelas** e **views**. Isso quer dizer que um usuário pode ter permissão na tabela e não na views ou vice-versa. **Isso funciona de forma indiscriminada para qualquer comando da linguagem SQL e não apenas para o comando SELECT.**

Vale lembrar que as views são criadas como mecanismos importantes que podem reduzir a complexidade e aumentar a segurança da base de dados, exibindo apenas estruturas e elementos essenciais a determinado usuário.

Quanto à questão dos privilégios, este pode ser dado usando o comando **GRANT** da linguagem SQL.

No exemplo a seguir tem-se a sequência de comandos para a criação de uma view e, em seguida, a concessão de privilégios de seleção ao usuário escola.

1) Criação da view:

```
CREATE VIEW dados_alunos
AS SELECT nome, sexo, email
FROM aluno;
```

2) Concessão de privilégio SELECT na view para o usuário escola:

```
GRANT SELECT ON dados_alunos TO escola;
```

Errado.

017. (VUNESP/TJM-SP/TÉCNICO DE COMUNICAÇÃO E PROCESSAMENTO DE DADOS/2017) Considere a seguinte estrutura de uma tabela de um banco de dados relacional: Teste (Chave, Nome, Peso, Profissão); e um resultado de uma consulta feita a essa tabela:

Profissão	Peso
Dentista	70
Engenheiro	75
Juiz	65

Uma consulta SQL que tem como resultado a tabela acima apresentada é

a) `SELECT *`

```
FROM Teste WHERE
```

```
Profissão LIKE '%a%
```

b) `SELECT Peso, Profissão`

```
FROM Teste
```

```
HAVING Profissão ORDER BY Desc
```

c) `SELECT Profissão, Peso`

```
FROM Teste
```

```
WHERE Peso BETWEEN 65 and 75
```

d) `SELECT Chave, Profissão, Peso`

```
FROM Teste
```

```
OCULT Chave
```

e) `SELECT Chave, Profissão, Peso`

```
FROM Teste
```

```
SHOW Profissão, Peso
```



Os campos exibidos como resultados são somente **Profissão** e **Peso**, respectivamente. Deste modo, eles precisam aparecer na consulta logo após o comando **SELECT** indicando o nome dos campos ou colunas da tabela.

A condição na cláusula WHERE é que o valor de Peso esteja no intervalo de 65 - 75 e para isso usamos a cláusula BETWEEN (entre) os valores passados na consulta.

Observem que as **restrições ou predicados** em uma consulta SQL devem aparecer na cláusula **WHERE**. Para compor as restrições, podemos usar a cláusula **BETWEEN**, que **nos permite selecionar intervalos de dados ao retornar os resultados de uma consulta**.

A sintaxe para uso da cláusula BETWEEN é a seguinte:

SELECT colunas **FROM** tabela

WHERE coluna **BETWEEN** valor1 **AND** valor2;

Usamos o operador lógico AND para auxiliar na criação do código de consulta.

Letra c.

018. (VUNESP/MPE-SP/ANALISTA TÉCNICO CIENTÍFICO/ENGENHEIRO DE COMPUTAÇÃO/2016) Considere a seguinte tabela de um banco de dados relacional:
Funcionário (ID, Nome, Função, Salário)

O comando SQL para obter a média dos salários agrupados por Função, apenas para médias superiores a R\$ 2.000,00 é

```
SELECT Função, AVG (Salário)
FROM Funcionário
GROUP BY Função
X – AVG(Salário) > 2000,00
```

Para que a consulta atenda ao especificado, o valor de X deve ser substituído por:

- a) CASCADE
- b) HAVING
- c) ORDER BY
- d) TOTAL
- e) WHERE



O valor de X deve ser substituído pela cláusula HAVING que é aplicada às linhas no conjunto de resultados. Só os grupos que atendem os parâmetros de HAVING são exibidos na saída da consulta.

É importante lembrar que só aplicamos uma cláusula HAVING em colunas da tabela que também são exibidas na cláusula GROUP BY ou em uma função de agregação (MIN, MAX, AVG, etc.).

NOTAS**1) Consultas SQL:**

```
SELECT[ DISTINCT| ALL] { * | <select list> }
FROM<table reference> [ , <table reference> ] . . .
[ WHERE<searchcondition> ]
[ GROUPBY<groupingspecification> ]
[ HAVING<searchcondition> ]
[ ORDERBY<ordercondition> ]
```

2) Observe a seguir exemplos de **funções agregadas** que operam sobre conjuntos de valores de uma coluna de uma relação e retornam um valor para cada conjunto: COUNT (), SUM (), MIN (), MAX (), AVG () .

Quando utilizamos funções agregadas, devemos utilizar o **GROUP BY** para os atributos na cláusula SELECT que não aparecem como parâmetros nas funções agregadas e a opção **HAVING** para predicados que são aplicados após a formação dos grupos.

Letra b.

019. (VUNESP/SAEG/ANALISTA DE SERVIÇOS ADMINISTRATIVOS/TECNOLOGIA DA INFORMAÇÃO/2015) Um programador desenvolveu um formulário web em que os dados digitados pelo usuário são concatenados a uma instrução SQL INSERT, que é enviada diretamente ao Sistema Gerenciador de Banco de Dados.

Tal procedimento produz uma falha de segurança, pois permite que um usuário mal intencionado insira

- a)** caracteres em codificação diversa da utilizada pelo banco de dados, causando erro na visualização dos dados.
- b)** dados duplicados nas tabelas do banco de dados, causando conflitos de chave primária.
- c)** identificadores de elementos que não correspondem a registros existentes em outras tabelas, violando a restrição da chave estrangeira.
- d)** instruções SQL nos campos do formulário para alterar o comportamento do sistema ou danificá-lo.
- e)** tipos de dados incorretos nos campos do formulário, produzindo inconsistências na base de dados.



SQL Injection é uma técnica que permite a um criminoso injetar comandos SQL nos campos do formulário para alterar o comportamento do sistema ou danificá-lo. Dessa forma, ele consegue roubar informações importantes ou ainda apagá-las.

Do ponto de vista da segurança, **SQL Injection é uma falha grave**, que deve ser evitada. O primeiro passo para isso é entender que os dados concatenados em um comando SQL podem modificar sua lógica de execução.

A técnica fundamental para evitar o SQL Injection é **remover qualquer comando SQL dos dados informados pelo usuário da aplicação.**

Letra d.

020. (VUNESP/CETESB/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO/ADMINISTRADOR DE BANCO DE DADOS/2013) Sistemas de Gerenciamento de Bancos de Dados desempenham inúmeras funções. Dentre tais funções, é correto afirmar que esses sistemas

- a)** contêm software que permite a comunicação em tempo real por meio da internet.
- b)** implementam ferramentas de gestão de projetos como, por exemplo, gráficos de Gantt.
- c)** mantêm a integridade dos dados inseridos no banco de dados, por exemplo, impedindo a duplicação do valor de chaves primárias.
- d)** possuem dicionários de línguas, permitindo a tradução imediata do conteúdo de qualquer banco de dados.
- e)** são capazes de fazer a compilação de todas as linguagens de programação orientadas a objetos.



Um **sistema de gerenciamento de banco de dados (SGBD)** possui vários recursos como interpretar e compilar uma linguagem de banco de dados (exemplo: SQL), além de prover segurança, controle de acesso e integridade dos dados armazenados.

Entre as restrições de integridade, as chaves primárias garantem que um registro ou instância seja único em uma tabela no banco de dados. Isso elimina duplicação de registros e permite a recuperação da informação.

Embora sejam vastas as atribuições dos SGBDs, eles não contêm por natureza softwares que permitem comunicação em tempo real, não são ferramentas de gerência de projeto, não possuem dicionários de línguas nem são capazes de compilar linguagens orientadas a objetos. Deste modo, a alternativa correta é letra C.

Letra c.

021. (FUNIVERSA/IPHAN/ANALISTA/TECNOLOGIA DA INFORMAÇÃO/2009) A linguagem SQL (Structured Query Language) foi desenvolvida para ser uma linguagem padrão, independentemente do *hardware* ou *software*. Porém, ela é uma linguagem específica para manipulação de

- a)** banco de dados.
- b)** arquivos de texto.
- c)** cálculos matemáticos.
- d)** imagens.
- e)** dados estatísticos.



A linguagem SQL é a linguagem para manipulação de banco de dados mais difundida nos dias de hoje. Ela permite por exemplo efetuar consultas a banco de dados, definir a estrutura dos dados, modificar dados em um banco e especificar restrições de segurança.

Letra a.

022. (FCC/PREFEITURA DE SÃO LUÍS-MA/AUDITOR-FISCAL DE TRIBUTOS I/TECNOLOGIA DA INFORMAÇÃO (TI)/2018) Um Auditor está executando operações em uma tabela chamada cidadao de um banco de dados aberto e em condições ideais. Para exibir os dados de todas as pessoas que possuem na segunda letra do campo nome a vogal a, deve-se utilizar a instrução SQL:

```
SELECT * FROM cidadao WHERE
```

- a) nome = '*a';
- b) nome LIKE '_a%';
- c) nome CONTAINS('a',2);
- d) nome LIKE '*a';
- e) nome HAVE(2,'a');



O operador “**LIKE**” é usado para **buscar uma determinada string na linguagem SQL**. Podemos utilizar também o **caractere “%” para indicar um “coringa”, ou seja, um texto qualquer que pode aparecer no campo**.

A alternativa B é correta, pois destaca qualquer coisa que apareça após a letra “a” % estando ela na segunda posição (*underline* indicando a primeira posição).

Letra b.

023. (FCC/TRT-11ª/TÉCNICO JUDICIÁRIO DE TI/2017) Considere que no TRT exista, em um banco de dados, a tabela TRAB que possui como campos: nome, sexo, salario de vários trabalhadores. Um Técnico foi solicitado a escrever um comando SQL para obter a média salarial dos trabalhadores do sexo FEMININO. O comando correto é:

- a) SELECT sexo="FEMININO" FROM TRAB WHERE AVG(salario);
- b) SELECT sexo, AVG(salario) as MediaSalarial FROM TRAB GROUP BY sexo;
- c) SELECT AVG(salario) FROM TRAB WHERE SEXO='FEMININO';
- d) SELECT sexo, AVG(salario) FROM TRAB GROUP BY sexo="FEMININO";
- e) SELECT * FROM TRAB WHERE sexo='FEMININO' as AVG(salario);



Uma consulta típica em SQL tem a forma:

```
SELECT colunas  
FROM tabela  
[WHERE condição]
```

A cláusula **SELECT** relaciona as colunas que se quer presentes no resultado da consulta.

Aqui estamos interessados em obter a **média salarial** dos trabalhadores! Então, utilizaremos a função **AVG(coluna)**, que retorna a **média aritmética** da coluna dentro do grupo. Valores NULL são ignorados. A coluna a ser utilizada é a de **salario**.

A cláusula **FROM** estará associada à tabela **TRAB** que será percorrida na execução da expressão.

A cláusula **WHERE** é utilizada para filtrar um conjunto de linhas de uma tabela. Utilizaremos o filtro **SEXO='FEMININO'**.

SELECT e a cláusula **FROM** são necessárias em todas as consultas SQL. Devem aparecer antes de qualquer outra cláusula na consulta.

O comando `SELECT AVG(salario) FROM TRAB WHERE SEXO='FEMININO';` irá obter na tabela TRAB a média salarial dos trabalhadores do sexo FEMININO.

Letra c.

024. (FCC/TRE-SP/ANALISTA JUDICIÁRIO/ANÁLISE DE SISTEMAS/2017) Em uma situação hipotética, ao ser designada para atender aos requisitos de negócio de um usuário, uma Analista de Sistemas do TRE-SP escreveu expressões e comandos para serem executados em um Banco de Dados Relacional que visavam

(1) criar uma tabela que contivesse dados de processos partidários,
(2) controlar a segurança e o acesso a ela e
(3) manipular dados nela. Desta forma ela, se valeu, correta e respectivamente, por exemplo, de alguns elementos de expressões tais como:

- a) CREATE, GRANT e ALTER
- b) DROP, ALTER e UPDATE
- c) INSERT, INDEX e CREATE
- d) INSERT, REVOKE e SELECT
- e) CREATE, REVOKE e INSERT



Essa questão é bem simples, vamos analisar o enunciado para respondê-la.

A primeira ação que a analista fez foi “(1) criar uma tabela que contivesse dados de processos partidários”, o que significa que ela utilizou a cláusula **CREATE** para criar uma tabela,

lembrando que a cláusula CREATE ou REPLACE é utilizada para criar ou substituir tabelas em uma base de dados.

A segunda ação que ela fez foi “(2) controlar a segurança e o acesso a ela”, o que significa que ela poderia ter utilizado a cláusula GRANT para atribuir permissão ao usuário para realizar uma operação ou REVOKE para remover uma permissão de GRANT ou DENY.

E por último ela foi “(3) manipular dados nela”, o que significa que ela pode ter atualizado (UPDATE), selecionado (SELECT) ou inserido dados na tabela (INSERT). A alternativa que possui a combinação de linguagens utilizados é a letra E, CREATE, REVOKE e INSERT.

Os comandos CREATE, DROP e ALTER são comandos de definição de dados e objetos em um banco de dados relacional, e eles são comumente chamados de (DDL - *Data Definition Language*).

Na questão foram citados também comandos pertencentes ao conjunto de manipulação de dados de um banco (comumente chamada de Instruções DML - *Data Manipulation Language*), e os comandos mais comuns são INSERT, UPDATE, SELECT e DELETE.

E ainda temos as instruções que fazem parte da linguagem que permite o acesso ao banco de dados (DCL - *Data Control Language*), garantindo a segurança dos dados que são apresentados na questão como GRANT e o REVOKE.

Letra e.

025. (FCC/TRT-1^a REGIÃO/ANALISTA JUDICIÁRIO/DESENVOLVIMENTO DE

SISTEMAS/2014) Considere a seguinte consulta em SQL sobre uma base de dados:

SELECT Produto

FROM Lista

WHERE Produto LIKE “c%a”

Um dos possíveis resultados produzidos por essa consulta é

- a) Cabo, cabide, calça
- b) Ábaco, ácido, apito.
- c) Faca, laca, isca.
- d) Caneta, caixa, cabana.
- e) Rocha, tocha, mecha.



O LIKE busca padrões entre caracteres e o “%” é utilizado para aceitar qualquer caractere em seu lugar. Sendo assim, c%a retornará qualquer palavra que comece com c e termine com a.

Letra d.

026. (FCC/SABESP/ANALISTA DE GESTÃO/SISTEMAS/2014) Analise o trecho de SQL abaixo:

```
SELECT      *      FROM      Clientes
WHERE Cidade...l..... ('Araraquara','Limeira');
```

Para que este comando retorne às linhas cuja coluna Cidade seja Araraquara ou Limeira, a lacuna l deve ser substituída por:

- a) IN
- b) WHERE
- c) CHECK
- d) ALIAS
- e) RANGE



Uma consulta típica em SQL tem a forma:

```
SELECT coluna1, coluna2,..., colunaN
FROM nome_da_tabela
WHERE condição;
```

A cláusula **SELECT** relaciona as colunas que se quer presentes no resultado da consulta. Essa cláusula corresponde à operação de projeção da álgebra relacional.



A cláusula **FROM** corresponde à operação de produto cartesiano da álgebra relacional. Ela associa a tabela ou tabelas que serão pesquisadas durante a avaliação de uma expressão. Em outras palavras, é uma lista de relações a serem varridas na execução da expressão.

A cláusula **WHERE** corresponde à seleção do predicado da álgebra relacional. Consiste em um predicado envolvendo atributos da relação que aparece na cláusula **FROM**.

SELECT e a cláusula FROM são necessárias em todas as consultas SQL. Devem aparecer antes de qualquer outra cláusula na consulta.

SELECT* FROM Clientes irá selecionar todas as colunas da tabela Clientes.

```
SELECT      *      FROM      Clientes
WHERE Cidade...l..... ('Araraquara','Limeira');
```

irá selecionar todas as colunas da tabela Clientes, nas quais Cidade esteja na lista (`Araraquara`, `Limeira`).

Letra a.

027. (FCC/TRT-6ª REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2012)

Considere os seguintes comandos em PL/SQL:

```
CREATE TABLE VALORES (ID INT NOT NULL PRIMARY KEY,VALOR INT);
```

```
INSERT INTO VALORES VALUES (1,5);
```

```
INSERT INTO VALORES VALUES (1,6);
```

```
INSERT INTO VALORES VALUES (NULL,7);
```

```
INSERT INTO VALORES VALUES (2,1);
```

```
INSERT INTO VALORES VALUES (2,8);
```

```
INSERT INTO VALORES VALUES (3,9);
```

```
INSERT INTO VALORES VALUES (NULL,10);
```

```
SELECT COUNT(VALOR) FROM VALORES WHERE VALOR >= 5;
```

A consulta retornará o valor

- a) 2.
- b) 14.
- c) 6.
- d) 29.
- e) 5.



Durante a inserção dos valores na tabela Valores os seguintes comandos não são executados devido a erro:

```
INSERT INTO VALORES VALUES (1,6); [erro]
```

```
INSERT INTO VALORES VALUES (NULL,7); [erro]
```

```
INSERT INTO VALORES VALUES (2,8); [erro]
```

```
INSERT INTO VALORES VALUES (NULL,10); [erro]
```

É emitido erro porque uma chave primária não pode ser nula, e porque não se pode inserir valores em registros que já têm valor inserido (para modificar o conteúdo faz-se necessário o uso do comando UPDATE).

A seguir é mostrada a tabela, após a execução dos comandos acima apresentados.

ID	VALOR
1	5
2	1
3	9

Como pode ser observado só existem dois registros na tabela Valores que possuem valores maiores ou igual a 5, que são os de ID 1 e 3.

Letra a.

028. (COPS/AFPR/ANALISTA DE TI/2010) Assinale a alternativa que apresenta o comando SQL para apagar as entradas da tabela pessoa onde o valor do campo idade é maior do que 20.

- a) drop from pessoa where idade > 20;
- b) remove where idade > 20 from pessoa;
- c) erase from pessoa where idade > 20;
- d) delete from pessoa where idade > 20;
- e) delete where idade > 20 from pessoa;



A sintaxe correta para remover uma determinada linha ou um conjunto de linhas de uma determinada tabela é a seguinte:

DELETE FROM [TABELA] WHERE [CAMPO] [CONDICAO];

No caso do exemplo, a sintaxe correta seria a apresentada na letra D.

DELETE FROM PESSOA WHERE IDADE > 20;

Letra d.

029. (FCC/TRT-15^a REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2013)
Considere o comando SQL abaixo.

SELECT	*	FROM	Clientes
WHERE Cidade _____ ('Paris','Londres');			

Para retornar informações da tabela Clientes, cuja coluna Cidade seja igual a Paris ou Londres, a lacuna deve ser preenchida com o operador

- a) FIND
- b) EQ
- c) IN
- d) =
- e) RANGE



Na questão, observe que pretendemos verificar quais valores fazem parte de um conjunto. Logo, utiliza-se o **IN**.

Letra c.

030. (FUNCAB/EMPRESA DE DESENVOLVIMENTO URBANO/EMDUR/ANALISTA DE INFORMÁTICA) Alguns termos utilizados na terminologia de banco de dados estão

disponibilizados na Coluna I. Estabeleça a correta correspondência com os seus significados, disponibilizados na Coluna II.

Coluna I	Coluna II
1. DDL	() Linguagem utilizada para extrair porções de dados dos bancos de dados.
2. DML	() Conjunto de valores possíveis para uma entidade e seus atributos.
3. Domínios	() Linguagem utilizada para criar, excluir e alterar estruturas de bancos de dados.
4. Tabela	() Conjunto não ordenado de linhas.

A sequência correta é:

- a) 2, 3, 1 e 4.
- b) 1, 3, 2 e 4.
- c) 2, 4, 1 e 3.
- d) 1, 4, 2 e 3.
- e) 4, 3, 2 e 1.



A primeira alternativa refere-se à **Linguagem de Manipulação de Dados (DML - Data Manipulation Language)**, que permite a manipulação dos dados armazenados em um banco de dados.

Possui 4 comandos básicos:

- **INSERT**: inserção de registros;
- **DELETE**: remoção de registros;
- **UPDATE**: atualização de registros;
- **SELECT**: seleção (consulta) de registros.

Logo, a primeira afirmativa da coluna II refere-se à opção 2 da coluna I.

Neste ponto, já sabemos que somente as alternativas A e C podem ser corretas...

Vamos analisar a segunda afirmativa. O conjunto de valores possíveis é chamado de **domínio**. Ao definirmos que um atributo é de determinado tipo (inteiro, por exemplo), são restringidos os valores que o campo pode assumir. Sendo assim, a segunda afirmativa da coluna II refere-se à opção 3 da coluna I.

Já sabemos então que a resposta da questão é a letra A.

Mas vamos analisar as outras afirmativas para conferir e treinar.

A terceira afirmativa refere-se ao conceito de **Linguagem de Definição de Dados (DDL - Data Definition Language)**.

Comandos básicos da DDL:

- **CREATE**: criação de novas estruturas
- **ALTER**: alteração de estruturas existentes
- **DROP**: remoção de estruturas

E, finalmente, a quarta afirmativa define uma tabela. Lembre-se de que os dados armazenados em uma tabela não precisam estar ordenados.

Letra a.

031. (ESAF/STN/DESENVOLVIMENTO DE SISTEMAS/2008) SBGD (Sistema Gerenciador de Bancos de Dados) possui um compilador para uma determinada linguagem, cuja função é o processamento de declarações, a fim de identificar as descrições dos componentes do esquema conceitual do Banco de Dados. Tal linguagem é de

- a) consulta estrutura – SQL.
- b) definição de armazenamento – SDL.
- c) manipulação de dados – DML.
- d) definição de visão – VDL.
- e) definição de dados – DDL.



Linguagem de definição de dados (DDL, do Inglês *Data Definition Language*) é uma linguagem de computador usada para a definição de estruturas de dados.

Uma vez compilados, os parâmetros DDL são armazenados num conjunto de arquivos denominado **dicionário de dados**.

O dicionário de dados contém os metadados (dados a respeito das estruturas de armazenamento). O SGBD sempre consulta os **metadados** a cada operação sobre o banco de dados. Por exemplo, um determinado programa precisa recuperar alguns campos (nome, CPF) de um arquivo de clientes. O SGBD irá verificar se os campos "nome" e "CPF" estão definidos para este arquivo. O interpretador DDL processa os comandos alimentados pelos DBAs na definição dos esquemas.

Os comandos básicos da **DDL** são poucos

- **CREATE**: cria um objeto (uma Tabela, por exemplo) dentro da base de dados;
- **DROP**: apaga um objeto do banco de dados.

Alguns sistemas de banco de dados usam o comando **ALTER**, que permite ao usuário alterar um objeto, por exemplo, adicionando uma coluna a uma tabela existente. Outros comandos **DDL**:

- **ALTER TABLE**
- **CREATE INDEX**
- **ALTER INDEX**
- **DROP INDEX**
- **CREATE VIEW**
- **DROP VIEW**

Letra e.

032. (COPS/AFPR/ANALISTA DE TI/2010) Assinale a alternativa que apresenta corretamente o comando SQL para atualizar as entradas da tabela pessoa, fazendo com que o campo idade contenha o valor 23 quando o valor ali armazenado for maior do que 20.

- a) update pessoa set idade=23 where idade > 20;
- b) change pessoa set idade=23 where idade > 20;
- c) upadte pessoa set 23 where idade > 20;
- d) change pessoa set 23 where idade > 20;
- e) update from pessoa idade= 23 where idade >20;



A sintaxe correta para atualizar uma determinada linha ou um conjunto de linhas de uma determinada tabela é a seguinte:

UPDATE [TABELA] SET [CAMPO] [OPERADOR] [NOVO_VALOR] WHERE [CONDICAO];

No caso do exemplo, a sintaxe correta seria a apresentada na letra A.

UPDATE PESSOA SET IDADE=23 WHERE IDADE > 20;

Letra a.

033. (COPS/AFPR/ANALISTA DE TI/2010) O comando SQL para selecionar todas as entradas da tabela pessoa, onde o campo idade possui um valor menor do que 20 é:

- a) select * from pessoa where idade < 20;
- b) find * from pessoa where idade < 20;
- c) search * from pessoa where idade < 20;
- d) select where idade < 20 from pessoa;
- e) find where idade < 20 from pessoa;



A sintaxe correta para selecionar uma determinada linha ou um conjunto de linhas de uma determinada tabela é a seguinte:

SELECT [CAMPO(S)] FROM [TABELA] WHERE [CONDICAO];

No caso do exemplo, a sintaxe correta seria a apresentada na letra A.

SELECT * FROM PESSOA WHERE IDADE < 20;

Note bem: o caractere asterisco (*) é utilizado para representar todos os campos de uma tabela ou um conjunto de tabelas em uma consulta, na sintaxe SQL.

Letra a.

034. (ESAF/ANAC/ANALISTA ADMINISTRATIVO/ANÁLISE DE SISTEMAS/2016) Em SQL, algumas consultas precisam de que os valores existentes no banco de dados sejam buscados

e depois usados em uma condição de comparação. Elas podem ser formuladas por meio de consultas

- a) concorrentes.
- b) comparadas.
- c) segmentadas.
- d) hierarquizadas.
- e) aninhadas.



Consultas aninhadas é um dos recursos mais poderosos e úteis da linguagem SQL. **Uma consulta aninhada é uma consulta secundária embutida dentro de uma consulta principal. Esta consulta embutida também é chamada de subconsulta.**

Uma **subconsulta** geralmente aparece na cláusula WHERE da consulta principal, embora ela também possa fazer parte das cláusulas FROM ou HAVING.

Obs.: Em SQL, uma consulta é **aninhada** quando ela está dentro de outra consulta SQL.

Referência: <http://www.ic.unicamp.br/~geovane/mo410-091/Ch05-Resumo.pdf>.

Letra e.

035. (ESAF/MPOG-TI/ANALISTA DE PLANEJAMENTO E ORÇAMENTO/2010) Em uma SQL

- a) a Linguagem de Manipulação de Relacionamentos comprehende os comandos para construir tabelas em um banco de dados.
- b) a Linguagem de Definição de Dados fornece tabelas para criação e modificação de comandos.
- c) os comandos básicos da Linguagem de Definição de Dados são *Select, Insert, Update* e *Delete*.
- d) a Linguagem de Manipulação de Dados comprehende os comandos para inserir, remover e modificar informações em um banco de dados.
- e) os comandos básicos da Linguagem de Definição de Dados são *Sort, Insert, Undo* e *Store*.



A **DML (Data Manipulation Language – Linguagem de Manipulação de Dados)** visa à manipulação de dados (incluir, alterar, excluir e consultar) por meio do usuário.

Os principais comandos da **DML**:

- SELECT: seleção de registros;
- INSERT: inserção de registros;
- UPDATE: atualização de registros;
- DELETE: deleção de registros.

Para a definição dos dados é utilizada uma **DDL (Data Definition Language – Linguagem de Definição de dados)**. Os comandos DDL são armazenados no dicionário de dados (ou

catálogo). Logo, o dicionário de dados contém os metadados (dados a respeito das estruturas de armazenamento) do banco.

Os principais comandos da DDL são:

- CREATE: criação de novas estruturas;
- ALTER: alteração de estruturas;
- DROP: remoção de estruturas.

Existe ainda a **DCL (Data Control Language - Linguagem de Controle de Dados)** para controlar o acesso dos usuários aos dados em um banco de dados. Principais comandos:

- GRANT: concessão de privilégios a tabelas e visões.
- REVOKE: revogação de privilégios a tabelas e visões.

Logo, de acordo com as definições acima, a letras A, B, C e E estão incorretas. Já a letra D faz a referência a uma DML.

Letra d.

036. (ESAF/CVM/ANALISTA DE TIC/INFRAESTRUTURA/PROVA 2/2010) A linguagem de definição de dados permite a especificação do esquema do banco de dados.



A **Linguagem de Definição de Dados (DDL)** é usada para especificar o esquema conceitual.

Certo.

037. (ESAF/CVM/ANALISTA DE TIC/INFRAESTRUTURA/PROVA 2/2010) A linguagem de definição de dados permite expressar as consultas e atualizações do banco de dados.



A **Linguagem de Definição de Dados (DDL)** não é utilizada para consultas e atualizações do banco de dados. A linguagem que permite consultas e atualizações é a **DML (Linguagem de Manipulação de Dados)**.

Errado.

038. (ESAF/CVM/ANALISTA DE TIC/INFRAESTRUTURA/PROVA 2/2010) A linguagem de manipulação de dados permite a especificação do esquema do banco de dados.



A **Linguagem de Manipulação de Dados, DML, não** permite a especificação de esquema de banco de dados. Mas a DML permite a atualização, inserção e exclusão de dados. A **Linguagem de Definição de Dados (DDL)** é usada para especificar o esquema conceitual.

Errado.

039. (IADES/FHB-DF/TECNOLOGIA DA INFORMAÇÃO/ANALISTA DE SISTEMAS/2017)

Considere as tabelas aluno (matricula INT, nome CHAR, cod_curso INT) e curso (cod_curso INT, curso CHAR, area CHAR), apresentadas a seguir.

matricula	nome	cod_curso
1	Joao	1
2	Pedro	1
3	Maria	2
4	Jose	2
5	Ana	3
6	Paulo	3

cod_curso	curso	area
1	Engenharia Civil	Exatas
2	Lingua Portuguesa	Humanas
3	Historia	Humanas
4	Matematica	Exatas

Qual dos comandos Structured Query Language (SQL) pode ser utilizado para listar somente os alunos que cursam História?

- a) select aluno.Nome from aluno,curso where aluno.cod_curso = curso.cod_curso AND curso.curso = 'Historia';
- b) select aluno.Nome from aluno,curso where curso.curso = 'Historia';
- c) select aluno.Nome from aluno,curso where aluno.cod_curso = 'Historia';
- d) select curso.curso from aluno,curso where aluno.cod_curso = curso.cod_curso AND curso.curso = 'Historia';
- e) select aluno.Nome from aluno,curso where curso.curso = 'Historia' and aluno.cod_curso!=curso.cod_curso;



a) Certa. É uma prática altamente recomendada fazer referência <tabela.campo>. Nesse contexto, como exemplo “aluno” é o nome da tabela e “nome” o campo da tabela.

b) Errada. select aluno.Nome from aluno,curso where aluno.cod_curso = curso.cod_curso AND curso.curso = 'Historia';

Faltou adicionar na cláusula WHERE uma relação que conecta as tabelas ALUNO e CURSO, de modo que fosse possível a recuperação de todos os alunos que cursam história.

c) Errada. O atributo cod_curso é numérico e não do tipo caracter. Caso fosse o número 3 no lugar da string história essa alternativa estaria correta.

d) Errada. Essa consulta retorna o nome do curso e não o nome dos alunos.

e) Errada. Está completamente errada no uso do operador diferente de (!=) no lugar de igual a (=), além disso com a adição da condição curso.curso = 'Historia' faz com que ele nunca retorne nada uma vez que não existe tal relação na tabela.

Letra a.

040. (FUMARC/SAÚDE/2007) Sobre o comando SQL a seguir, escolha a afirmativa **correta**:

SELECT depto.nome, funcionario.nome

FROM depto, funcionario

WHERE depto.codigo = funcionário.depto_codigo AND

funcionário.salario > 5000 AND

depto.codigo = 'Diretoria' OR depto.codigo = 'Tecnologia'

a) Seleciona os funcionários da Diretoria e da Tecnologia que tenham salário maior que 5.000.

b) Seleciona os funcionários da Diretoria com salário maior que 5.000 e todos os funcionários da Tecnologia, independente de salário.

c) Seleciona os funcionários que fazem parte da Diretoria e da Tecnologia, simultaneamente, e que tenham salário maior que 5.000.

d) Seleciona todos os funcionários com salário maior que 5.000 e ainda os funcionários da Diretoria e Tecnologia, independente de seus salários.



SQL é a abreviatura de *Structured Query Language*. É uma linguagem para criação de bancos de dados e para recuperação e manutenção dos dados armazenados nestes. A estrutura básica de uma consulta em SQL, utilizada na questão, consiste em três cláusulas: SELECT, FROM E WHERE.

SELECT. A cláusula **SELECT** é usada para listar os atributos desejados como resultado de uma consulta. Em outras palavras, relaciona as colunas que se quer presentes no resultado da consulta.

FROM. A cláusula FROM relaciona a tabela ou tabelas que devem ser pesquisadas/varridas na execução da expressão.

WHERE. Especifica a condição ou condições que as linhas selecionadas devem satisfazer.

Uma cláusula WHERE pode conter **mais de uma condição**, sendo possível usar operadores lógicos para indicar a união entre as condições:

- **AND**: exibe os registros em que todas as condições são verdadeiras.

SELECT coluna1, coluna2, coluna3... FROM nome_da_tabela WHERE condição1 **AND** condição2 **AND** condição3 ...;

- **OR**: exibe os registros em que pelo menos uma condição é verdadeira.

SELECT coluna1, coluna2, coluna3... FROM nome_da_tabela WHERE condição1 **OR** condição2 **OR** condição3 ...;

Na questão, observe a condição dada na cláusula WHERE, em que se pode obter o que está especificado na letra B (selecionar os funcionários da Diretoria com salário maior que 5.000 e todos os funcionários da Tecnologia, independente de salário).

WHERE depto.codigo = funcionário.depto_codigo **AND**

funcionário.salario > 5000 **AND**

depto.codigo = 'Diretoria' **OR** depto.codigo = 'Tecnologia'

Exemplos de operadores que podem ser utilizados =, <>, >, <, IS NULL, IN, BETWEEN, NOT, LIKE.

Letra b.

041. (COPS/ANALISTA DE DESENVOLVIMENTO MUNICIPAL/TI/ÁREA DE PROGRAMAÇÃO/2008) Observe o esquema de banco de dados a seguir:

Cartão	(cod_cartão, nome_cartão)
Empresa	(cod_empresa, nome_empresa)
Compra	(cod_cartão, cod_empresa, data, hora, valor)
– cod_cartão referencia Cartão	
– cod_empresa referencia Empresa	

Com base no esquema de banco de dados, assinale a alternativa correta.

- a) INSERT INTO Cartao (cod_cartao, nome_cartao) VALUES ('001', 'credicard');
 INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES ('002', 'Dom Juliano');
 INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES ('001', '002', '01/04/2008', '14:00', 15,00);
- b) INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES ('002', 'Dom Juliano');
 INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES ('001', '002', '01/04/2008', '14:00', 15,00)
 INSERT INTO Cartao (cod_cartao, nome_cartão) VALUES ('001', 'credicard');
- c) INSERT INTO Cartao (cod_cartao, nome_cartão) VALUES ('001', 'credicard');
 INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES ('001', '002', '01/04/2008', '14:00', 15,00);
 INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES ('002', 'Dom Juliano');
- d) INSERT INTO Cartao (cod_cartao, nome_cartao) VALUES (null, 'credicard');
 INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES ('002', 'Dom Juliano');
 INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES (null, '002', '01/04/2008', '14:00', 15,00);
- e) INSERT INTO Cartao (cod_cartao, nome_cartao) VALUES ('001', 'credicard');
 INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES (null, 'Dom Juliano');
 INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES ('001', null, '01/04/2008', '14:00', 15,00);



A sintaxe correta para adicionar uma determinada linha ou um conjunto de linhas em uma determinada tabela é a seguinte:

INSERT INTO [TABELA] VALUES (ATRIBUTOS);

A sintaxe acima requer que seja lembrada a ordem das colunas da tabela que se deseja adicionar linhas.

A sintaxe abaixo pode ser utilizada como alternativa para quando não é lembrada a ordem correta das colunas em uma dada tabela.

INSERT INTO [TABELA] (COLUNAS) VALUES (ATRIBUTOS);

No caso da questão, a sintaxe correta seria a apresentada na letra A.

`INSERT INTO Cartao (cod_cartao, nome_cartao) VALUES ('001', 'credicard');`

`INSERT INTO Empresa (cod_empresa, nome_empresa) VALUES ('002', 'Dom Juliano');`

`INSERT INTO compra (cod_cartao, cod_empresa, data, hora, valor) VALUES ('001', '002', '01/04/2008', '14:00', 15,00);`

Ainda, observe algumas dicas:

1. **Não** se pode ter chave primária vazia (null).
2. Deve-se seguir a ordem lógica de inserção dos valores nas colunas das tabelas.
3. Para inserir valores na tabela compra, é imprescindível que as tabelas Cartão e Empresa tenham sido populadas (cod_cartão e cod_empresa fazem parte da chave primária composta da tabela Compra).

No caso, as tabelas Cartão e Empresa podem ser criadas em qualquer ordem, uma vez que as mesmas não possuem nenhum tipo de dependência. Já a tabela Compra não pode ser criada em uma ordem qualquer uma vez que é necessário que as tabelas Cartão e Empresa tenham sido criadas a priori. Observe que para se realizar uma compra são necessários um Cartão e uma Empresa, já que as chaves primárias simples de ambas fazem parte da chave primária composta da tabela Compra, logo a tabela Compra possui uma relação de dependência com as tabelas Cartão e Empresa.

Letra a.

042. (ESAF/SUSEP-TI/2006) Analise as seguintes afirmações relacionadas a conceitos básicos sobre Banco de Dados.

I – O comando SQL responsável por fechar uma transação confirmando as operações feitas é o INSERT.

II – O comando SQL responsável por fechar uma transação e desfazer todas as operações é o COMMIT.

III – Quando uma transação ainda está aberta para um usuário, enquanto não é executado um comando COMMIT, o próprio usuário pode ver as suas alterações, mas outros usuários não podem vê-las.

IV – Uma transação assegura um espaço de trabalho que contém várias alterações, inclusões e exclusões de dados em uma ou mais tabelas, com a possibilidade de confirmação ou cancelamento das operações sem comprometimento dos dados.

Indique a opção que contenha todas as afirmações verdadeiras.

- a) I e II
- b) II e III
- c) III e IV
- d) I e III
- e) II e IV



O comando COMMIT fecha a transação, confirmando todas as operações. Logo, as afirmações I e II estão incorretas.

A afirmação III está correta, pois enquanto o comando Commit não for executado, as operações são visíveis apenas para o usuário que está executando a transação.

O item IV refere-se ao conceito de transação, a qual pode conter várias operações que precisam ser confirmadas para serem efetivadas no banco de dados. Item correto.

Letra c.

043. (COPESE/UFPI/UFPI/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO/ADAPTADA/2017) Um banco de dados relacional consiste em uma coleção de tabelas, cada uma com um nome único atribuído. Sobre o modelo relacional, é correto afirmar que a linguagem HTML é uma linguagem de consulta amigável que possui como base formal a álgebra relacional.



Quando falamos em “relações no modelo relacional”, estamos falando em tabelas cujas operações são baseadas na álgebra relacional (seleção, projeção, união, subtração, junção e produto cartesiano) e que manipulam conjuntos de dados ao invés de um único registro. A **linguagem SQL** foi baseada nas operações de álgebra relacional.

Errado.

044. (FCC/IF-PE/IF-PE/TÉCNICO EM TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO/2016) Leia as afirmativas abaixo e responda à questão proposta.

I – Retorna linhas quando há, pelo menos, uma correspondência entre duas tabelas.

II – Operador usado para combinar o resultado do conjunto de duas ou mais instruções SELECT.

III – Operador usado em uma cláusula WHERE para pesquisar um padrão específico em uma coluna.

I, II e III correspondem, em SQL, respectivamente, aos comandos:

- a) INNER JOIN, UNION e LIKE.
- b) INNER JOIN, JOIN e DISTINCT.
- c) LEFT JOIN, UNIQUE e LIKE.
- d) SELECT, JOIN e BETWEEN.
- e) SELECT, UNIQUE e BETWEEN.



Item I – O comando **INNER JOIN** realiza a junção e recupera a informação de intercessão entre duas tabelas.

Item II – O operador **UNION** é usado para combinar o resultado de execução das duas *queries* e então executa um **SELECT DISTINCT** com o objetivo de eliminar as linhas duplicadas.

Item III – O operador **LIKE** é utilizado para buscar por uma determinada *string* dentro de uma coluna com valores textuais.

Assim, conforme visto, a alternativa correta é **letra A (INNER JOIN, UNION, LIKE)**.

Letra a.

045. (FCC/TRF-5^a REGIÃO/TÉCNICO JUDICIÁRIO/INFORMÁTICA/2017) Após constatar que todos os dados em uma tabela estavam incorretos, foi solicitado ao Técnico em Informática para limpar os registros desta tabela mantendo sua estrutura, para que os dados corretos fossem posteriormente inseridos. Para realizar este trabalho o Técnico terá que utilizar a instrução SQL

- a) DROP TABLE table_name.
- b) REDO * FROM table_name.
- c) DELETE TABLE table_name.
- d) ERASE * FROM table_name.
- e) TRUNCATE TABLE table_name.



O comando capaz de remover todos os dados de uma tabela de forma rápida e eficiente é o comando **TRUNCATE**. Porém, devemos ter muito cuidado ao usá-lo porque não há como recuperar os dados excluídos.

Outro comando que também remove registros em uma tabela é o comando **DELETE**, porém, é necessário usar a cláusula **FROM** e passar quais os registros passíveis de exclusão.

O comando **DROP** remove também os registros em uma tabela, entretanto, a estrutura (própria tabela) da tabela também é removida.

Os demais comandos listados como alternativas na questão não estão ligados à remoção de informações.

Letra e.

046. (FCC/SABESP/TÉCNICO EM GESTÃO 01/INFORMÁTICA/2018) O RH da empresa deseja emitir um relatório de todos os funcionários que ganham comissão. Deseja mostrar o nome, salário e comissão organizando os dados por salário de forma descendente. Considerando a existência de um banco de dados Oracle aberto e em condições ideais, que possui uma tabela chamada Funcionario com os campos nome, cargo, salario e comissao, a instrução SQL que deverá ser utilizada é

- a) `SELECT ALL nome, salario, comissao FROM funcionario WHERE comissao IS NOT EMPTY ORDER BY salario DESCENDING;`
- b) `SELECT nome, salario, comissao FROM funcionario WHERE comissao IS NOT NULL ORDER BY salario DESC;`
- c) `SELECT nome, salario, comissao FROM funcionario WHERE NOT_EMPTY(comissao) ORDER BY salario DESC;`
- d) `SELECT * FROM funcionario WHERE comissao IS NOT NULL ORDER BY salario DESC;`
- e) `SELECT ONLY nome, salario, comissao FROM funcionario WHERE NOT_NULL(comissao) ORDER BY salario DESC;`



- a) Errada. Há dois erros na instrução. O primeiro, não usamos ALL seguido dos campos da tabela. O segundo, a forma correta para indicar que há informações (não esteja vazio) é IS NOT NULL e não IS NOT EMPTY.
- b) Certa. Selecione os campos nome, salario, comissao corretamente, indicamos que a comissao não deve conter valores vazios e a ordenação pelo campo salario de forma decrescente (DESC)
- c) Errada. O comando NOT_EMPTY(comissao) não é usado para identificar que o campo não deve estar vazio.
- d) Errada. O asterisco (*) indica o retorno de todos os campos na tabela e o campo CARGO não é solicitado como retorno na consulta.
- e) Errada. Há dois erros na instrução. O primeiro, não usamos ONLY após a cláusula SELECT para indicar que somente tais campos serão selecionados. O segundo, a forma correta para indicar que há informações (não esteja vazio) é IS NOT NULL e não NOT_NULL.

Letra b.

047. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017)

Atenção: Para responder à questão, considere as informações abaixo.

Considere a existência de um banco de dados com as tabelas criadas pelos comandos abaixo.

`CREATE TABLE Partido (`

```
    idPartido VARCHAR(4) NOT NULL,  
    nomePartido VARCHAR(70),  
    presidentePartido VARCHAR(50),
```

```
PRIMARY KEY (idPartido)
);
CREATE TABLE Filiado (
    idFiliado INT NOT NULL,
    nomeFiliado VARCHAR(50),
    dataFiliacao DATE,
    idPartido VARCHAR(4) NOT NULL,
    PRIMARY KEY (idFiliado),
    FOREIGN KEY (idPartido)
    REFERENCES Partido (idPartido)
);
```

Considere, ainda, que estas tabelas contêm os registros abaixo.

idPartido	nomePartido	presidentePartido	
PNC	Partido Nacional Constitucionalista	Paulo Prates	
PRC	Partido Republicano Constitucionalista	Mauro Gomes	
PTP	Partido Trabalhista Popular	Andrea Machado	
idFiliado	nomeFiliado	dataFiliacao	idPartido
1	Marcos Paulo Andrade	12/01/2017	PNC
2	Maria Silva Prates	30/08/2015	PNC
3	Marcelo Rocha Nunes	20/06/2016	PNC
4	Adriana Soares	30/12/2012	PRC
5	Juarez Fraciolli	18/02/2013	PRC
6	Zilda Gomes	01/08/2014	PRC
7	Jefferson Frade	12/01/2017	PTP
8	Ricardo Monteiro	21/02/2017	PTP
9	Thiago Brandão	17/10/2015	PTP
10	José Marques	28/01/2016	PTP
11	Murilo Coutinho	Null	PTP
12	Juca Souza	Null	PRC

Para alterar o idPartido para PNC na tabela Filiado, apenas para filiados com data de filiação entre 01/01/2017 e 30/06/2017, utiliza-se o comando

- a) ALTER Filiado SET idPartido='PNC' WHERE dataFiliacao BETWEEN '2017-01-01' AND '2017-06-30';
 - b) UPDATE idPartido='PNC' FROM FILIADO WHERE dataFiliacao BETWEEN '2017-01-01' AND '2017-06- 30';
 - c) SELECT * FROM FILIADO WHERE dataFiliacao BETWEEN '2017-01-01' AND '2017-06-30' AND ALTER idPartido TO 'PNC';
 - d) UPDATE Filiado SET idPartido='PNC' WHERE dataFiliacao BETWEEN '2017-01-01' AND '2017-06- 30';

e) UPDATE * FROM Filiado SET idPartido='PNC' WHERE dataFiliacao BETWEEN '2017-01-01' AND '2017-06-30';



- a) Errada. O comando **ALTER** altera a estrutura da tabela e não os registros.
- b) Errada. Para atualizar as informações contidas em um campo é necessário usar o comando **SET** seguido pelo campo e a informação dos registros que serão alterados.
- c) Errada. Novamente o comando **ALTER** é usado para alterar registros, o que não deve ser feito.
- d) Certa. Comando **UPDATE** usado com o comando **SET** e o valor que deve ser alterado de maneira correta. A data de filiação está também sendo corretamente usada na cláusula WHERE.
- e) Errada. O comando **UPDATE** não deve ser usado com (*) asterisco para atualização de registros em uma tabela e sim usando o comando **SET**.

Letra d.

048. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017) Ao executar um comando SQL, foram exibidos os dados abaixo.

Nome	Partido
Murilo Coutinho	PTP
Juca Souza	PRC

O comando utilizado foi

- a) SELECT nomeFiliado, idPartido FROM Filiado WHERE dataFiliacao IS NULL;
- b) SELECT nomeFiliado Nome, idPartido Partido FROM Filiado WHERE idFiliado>11;
- c) SELECT nomeFiliado as Nome, idPartido as Partido FROM Filiado WHERE dataFiliacao=NULL
- d) SELECT nomeFiliado AS Nome, idPartido AS Partido FROM Filiado WHERE nomeFiliado='Murilo Coutinho' AND nomeFiliado='Juca Souza';
- e) SELECT nomeFiliado as Nome, idPartido as Partido FROM Filiado WHERE dataFiliacao IS NULL;



- a) Errada. O nome do campo que aparece na exibição (Nome, Partido) não é o mesmo da tabela (nomeFiliado, idPartido). Se não for renomeado irá retornar o mesmo nome do campo da tabela.
- b) Errada. Para renomear o nome de exibição é necessário usar a cláusula "AS".
- c) Errada. A forma correta de verificar que não há dados em um campo da tabela (NULL) deve ser usado como dataFiliacao is NULL e não **dataFiliacao=NULL**.
- d) Errada. O comando AND é usado indicando que duas ou mais situações são verdadeiras. Isso quer dizer que um filiado não pode se chamar 'Murilo Coutinho' AND 'Juca Souza' simultaneamente.
- e) Certa. A forma correta de indicar que não há dados de dataFiliacao **IS NULL**;

Letra e.

049. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017) Para excluir os filiados ao partido PNC utiliza-se a instrução SQL

- a) ERASE * FROM Filiado WHERE idPartido='PNC';
- b) DELETE FROM Filiado WHERE idPartido='PNC';
- c) REMOVE * FROM Filiado WHERE idPartido='PNC';
- d) DELETE RECORD FROM Filiado WHERE idPartido='PNC';
- e) DELETE Filiado WHERE idPartido='PNC';



a) Errada. O comando **ERASE** não é um comando válido para exclusão de registros em um banco de dados relacional.

b) Certa. O comando **DELETE** é usado para apagar registros em tabelas e a cláusula **WHERE** indica quais registros serão removidos.

c) Errada. O comando **REMOVE** também não é um comando válido para apagar registros em tabelas. Para isso usamos o comando **DELETE**.

d) Errada. O comando **DELETE** deve ser usado seguido da cláusula **FROM**, indicando em qual tabela os dados serão apagados. O comando **RECORD** não é válido para se referenciar a registros em uma tabela.

e) Errada. Sempre usamos o comando **DELETE** seguido da cláusula **FROM**, senão é impossível saber a origem dos dados a serem apagados.

Letra b.

050. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017) The SQL statement SELECT * FROM Partido WHERE presidentePartido LIKE '%tes'; finds any values in the presidentePartido field that

- a) don't have "tes".
- b) have "tes" in the second position.
- c) begin with "tes".
- d) have "tes" in any position.
- e) ends with "tes".



O operador **LIKE** é utilizado para buscar por uma determinada *string* dentro de um campo com valores textuais.

O caracter "%" é usado como coringa no operador LIKE. Deste modo, a consulta irá retornar qualquer "presidentePartido" cujo nome comece com **qualquer coisa (quaisquer caracteres)** e termine com "tes". No exemplo da questão irá retornar o registro "Paulo Prates". A **alternativa correta é E (ends with "tes")**.

Letra e.

051. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017)

Um Técnico criou uma *view* utilizando o comando `CREATE VIEW Filiados_PRC AS SELECT nomeFiliado, dataFiliacao FROM Filiado WHERE idPartido='PRC';`.

Para excluir a *view* criada utiliza-se o comando

- a) `DROP VIEW Filiados_PRC REFERENCES TABLE Filiado;`
- b) `DELETE VIEW Filiados_PRC;`
- c) `DROP VIEW Filiados_PRC;`
- d) `REVOKE VIEW Filiados_PRC REFERENCES TABLE Filiado;`
- e) `ERASE VIEW Filiados_PRC TABLE CASCADE;`



A exclusão de uma **VIEW (tabela virtual)** é similar a exclusão de uma tabela “real” em um banco de dados relacional. Para isso usamos o comando `DROP VIEW` seguido pelo nome dado à *VIEW* no momento de sua criação.

Como ela foi criada como *Filiados_PRC*, a forma correta é **`DROP VIEW Filiados_PRC`**, portanto, alternativa correta é letra C.

Letra c.

052. (FCC/TRE-PR/TÉCNICO JUDICIÁRIO/OPERAÇÃO DE COMPUTADORES/2017) Ao tentar alterar na tabela Partido o idPartido de PNC para PNCT, foi exibida a mensagem “Cannot delete or update a parent row: a foreign key constraint fails”.

Isso ocorreu porque o Sistema Gerenciador de Banco de Dados não conseguiu alterar na tabela *Filiado* o idPartido dos filiados ao PNC para PNCT.

Para que a alteração fosse bem sucedida, no momento da criação da tabela *Filiado*, à cláusula `REFERENCES` deveria ter sido adicionada a cláusula

- a) `CONSTRAINT MODIFY CASCADE.`
- b) `ALTER CASCADE.`
- c) `ON UPDATE CASCADE.`
- d) `EXTEND UPDATE.`
- e) `ON ALTER CASCADE.`



A criação de uma tabela na base de dados é uma tarefa que deve ser bem pensada. O SGBD trabalha pela integridade dos dados de forma automática. A **integridade referencial** é obtida quando tentamos realizar alguma operação (alteração, exclusão, remoção) em uma tabela na qual há registros de chave estrangeira que é a chave primária de outra tabela.

Para fazer o SGBD entender que queremos afetar também a tabela referenciada, usamos o comando **CASCADE** (em cascata).

Como a questão pede que façamos a alteração na tabela referenciada, usamos após a cláusula `REFERENCES` os comandos `ON UPDATE CASCADE`.

Letra c.

053. (FCC/SABESP/ANALISTA DE GESTÃO/SISTEMAS/2014) Ao invés de executar uma consulta toda de uma vez, é possível configurar um objeto que encapsula a consulta e, em seguida, ler o resultado da consulta algumas linhas por vez. Uma razão para seu uso é para evitar estouro de memória quando o resultado contém um grande número de linhas. Em SGBD, este objeto é chamado de

- a) Constraint.
- b) Statement.
- c) Trigger.
- d) Cursor.
- e) Sub Query.



Esta é justamente a definição de **Cursor**. Trata-se de um recurso oferecido pelos SGBDs que permite realizar uma varredura num conjunto de registros, oriundos de uma tabela ou de uma consulta, executando-se um conjunto de operações em cada registro.

Pode-se imaginar um cursor como um comando de repetição. A utilização de cursores permite a realização de um grupo de comando sobre registros de forma mais simples e sem retornar uma grande quantidade de linhas.

Aproveitando a questão, vamos ver outros dois conceitos relevantes de SGBDs.

Stored Procedures são programas armazenados no banco de dados. A linguagem utilizada para criação destes programas é dependente do SGBD utilizado, fornecendo extensões à SQL padrão. Assim, é possível a criação e uso de variáveis, comentários, instruções de declaração, testes de condições, looping, etc. Uma stored procedure possui total acesso às instruções de manipulação SQL, mas geralmente não podem executar instruções DDL. Também pode receber parâmetros de entrada e fornecer resultados para outras aplicações.

Entre as principais vantagens do uso de stored procedures estão:

- Projeto modular: aplicações que acessam o mesmo banco de dados podem compartilhar stored procedures, eliminando código duplicado e diminuindo o tamanho das aplicações.
- Manutenção: quando um procedimento é atualizado, todas as mudanças se refletem automaticamente nas aplicações que usam aquele procedimento.
- Performance: a stored procedure é executada na máquina servidor e não na máquina cliente, reduzindo o tráfego na rede.

Um **trigger** é um programa armazenado dentro do banco de dados, que é executado automaticamente quando certos tipos de eventos acontecem. Estes eventos são baseados em tabelas e colunas. Por exemplo, um trigger pode ser executado quando uma linha é incluída ou excluída de uma tabela.

Os triggers são fundamentais na manutenção da integridade do banco de dados. Podem ser usados para verificar as restrições de integridade, criar valores automáticos para campos e chaves primárias, notificar uma outra aplicação sobre mudanças ocorridas e guardar

informações em tabelas de histórico. Um trigger deve ser definido para disparar antes (BEFORE) ou depois (AFTER) de uma operação baseada em linhas. As operações podem ser: INSERT, UPDATE ou DELETE.

Letra d.

054. (FCC/ICMS-RJ/AUDITOR-FISCAL DA RECEITA ESTADUAL/2014) Uma das tabelas do banco de dados da Receita contém dados sigilosos, quais sejam senhas e números de cartões de crédito de várias pessoas. Como estes dados não podem ficar expostos a todos os usuários que acessam o banco de dados, pois isso violaria as políticas de privacidade da Receita e leis estaduais e federais, deve-se

- a) manter a tabela privada (ou seja, não conferir permissão de consulta a qualquer usuário) e, então, criar uma ou mais views que omitam as colunas sigilosas. Como as views não envolvem armazenamento de dados, não ocupam espaço em disco, o que seria mais uma vantagem.
- b) transformar os campos sigilosos em uma superchave, que é um mecanismo dos bancos de dados que ocultam dados de usuários não autorizados.
- c) criar uma view, que é um mecanismo de ocultação de dados. As views criam novas tabelas que ficam armazenadas em áreas protegidas do disco. Essas tabelas ficariam acessíveis apenas aos usuários autorizados.
- d) manter a tabela de acesso irrestrito, mas criar uma única view que obscureça as colunas sigilosas usando o comando replace view. Também pode-se restringir quais linhas um grupo de usuários pode acessar adicionando uma cláusula constraint à definição da view.
- e) criar uma view chamada ACESSORESTRITO usando uma instrução case when e, em seguida, armazenar as tuplas resultantes em outra tabela de acesso irrestrito. Assim, todos os usuários poderiam usar a view criada consultando diretamente as tabelas.



A assertiva A é a solução para o problema. Uma **visão (view)** pode ser considerada como uma maneira alternativa de observação de dados de uma ou mais entidades (tabelas). Pode ser considerada também como uma **tabela virtual ou uma consulta armazenada**. As vantagens de se usar **views** são:

- permite economizar tempo, evitando retrabalho;
- aumenta a velocidade de acesso aos dados;
- esconde a complexidade do banco de dados;
- simplifica a gerência de permissão de usuários; e
- organiza os dados a serem exportados.

Uma vez que a **view** é gerada, **o seu conjunto de dados é armazenado em uma tabela temporária (virtual)**, tornando o acesso às informações mais rápido. **Deve-se ressaltar que uma view não existe fisicamente, é uma tabela virtual.** No entanto, os dados contidos em uma view podem ser modificados normalmente.

Para criar uma visão, você seleciona apenas as colunas da tabela (ou tabelas) básica em que está interessado, podendo omitir as colunas sigilosas. A sintaxe básica para se criar uma visão é a seguinte:

CREATE VIEW nome_view AS

SELECT coluna(s)

FROM tabela

WHERE condição

Conforme visto, a letra A é a resposta da questão.

Letra a.

055. (CESPE/TCE-PE/ANALISTA DE CONTROLE EXTERNO/AUDITORIA DE CONTAS PÚBLICAS/2017) A respeito de bancos de dados relacionais, julgue os itens subsequentes.

Uma visão (view) é derivada de uma ou mais relações e armazena os dados em uma tabela física do banco de dados, visando tornar ágeis as consultas.



Conforme vimos em nosso curso, quando existem várias tabelas em um banco de dados, e o administrador de banco de dados não quer mostrar todas as informações, ele pode criar o que chamamos de **visão** (ou **view**) para mostrar somente parte das informações da(s) tabela(s). Ele pode fazer isso por questão de segurança, por questão de estruturação e/ou também por questão de performance.

Quando digo “**por questão de segurança**”, quero dizer que pode existir alguns campos em alguma(s) tabela(s) que não podem ser exibidos. Por isso, ao ter que mostrar os dados de uma tabela, especificamente, temos que ocultar certos campos, tais como senha, salários etc.

Com relação à questão de **estruturação**, de vez em quando, acontece que certas informações de uma tabela devem ser relacionadas com outras informações de outras tabelas. Neste caso, facilitaria bastante se pudesse haver somente uma tabela com todas as informações necessárias e já organizadas para, por exemplo, o programador.

Quanto à **performance**, uma **view** é montada de forma que **ela não existe fisicamente dentro do banco de dados**. Ela é colocada em memória para que o acesso seja feito de forma mais rápido. Além disso, todos os “JOIN”, que são pontos de ineficiência em junções de tabelas, são feitos de forma que, ao acessar a **view**, eles não são mais necessários, dando mais eficiência às consultas.

Com isso, o erro do item está em afirmar que os dados são armazenados em uma tabela física no banco de dados.

Errado.

056. (CESPE/TCE-SC/AUDITOR DE TI/2016) Com relação aos bancos de dados relacionais, julgue os próximos itens.

Denomina-se visão uma tabela única derivada de uma ou mais tabelas básicas do banco. Essa tabela existe em forma física e viabiliza operações ilimitadas de atualização e consulta.



Em banco de dados, uma **visão**, também conhecida pela palavra “view”, é uma tabela simples que é derivada de outras tabelas, e que não existe necessariamente em sua forma física. A intenção de uma **view** é unir algumas tabelas, montar uma só para facilitar as consultas. Uma **view** pode, ou não, ser criada permitindo atualização dos dados nas tabelas originais.

Analizando a questão, temos: “Denomina-se **visão** uma tabela única derivada de uma ou mais tabelas básicas do banco (correto). Essa tabela existe em forma física (**não** necessariamente) e viabiliza operações ilimitadas de atualização (**não** necessariamente) e consulta (correto).”.

Portanto, item **errado**.

Quando se diz que a visão não existe em forma física, entenda que a tabela em si, derivada de outras tabelas, **não** está presente no banco de dados, e sim, em memória. Já as tabelas originais, a partir das quais será formada a visão, estão em suas formas físicas, isto é, presentes no banco de dados.

Errado.

057. (FCC/MPE-PE/ANALISTA MINISTERIAL/INFORMÁTICA/2018) Considere a consulta escrita em SQL abaixo.

```
SELECT Item
  FROM Lista
 WHERE Item LIKE "A%S";
```

O resultado da execução dessa consulta será composto por registros da tabela Lista, exibindo o campo Item, com a condição de que seus valores devem

- a) ter em sua formação a sequência A%S e ter número de caracteres variando entre 3 e o número máximo especificado para esse campo, quando da criação da tabela Lista.
- b) ter pelo menos duas letras A e duas letras S e ter número de caracteres variando entre 5 e o número máximo especificado para esse campo, quando da criação da tabela Lista.
- c) ter ou uma letra A, ou uma letra S e ter número de caracteres variando entre 3 e o número máximo especificado para esse campo, quando da criação da tabela Lista.
- d) começar com a letra A, terminar com a letra S e ter número de caracteres variando entre 2 e o número máximo especificado para esse campo, quando da criação da tabela Lista.
- e) começar com as letras A ou S e terminar também com uma dessas duas letras, além de ter número de caracteres variando entre 5 e o número máximo especificado para esse campo, quando da criação da tabela Lista.



O operador **LIKE** é utilizado na cláusula **WHERE** para pesquisar um padrão específico em uma coluna (faz a busca por uma determinada *string* dentro de um campo com valores textuais). Existem dois coringas utilizados em conjunto com o operador LIKE:

%	Percentual (%) representa zero, um, ou múltiplos caracteres.
_	Underscore (_) representa um simples caractere.

Assim, a consulta irá retornar qualquer “Item” cujo nome comece com o caractere “A” e termine com “S”, tendo zero, um ou múltiplos caracteres entre “A” e “S”. Dentre essa possibilidade, temos a letra D como a resposta da questão.

Letra d.

058. (FCC/MPE-MA/2013/ANALISTA MINISTERIAL/BANCO DE DADOS) Uma das formas de impor restrições em um banco de dados relacional é por meio das chaves primárias, sobre as quais pode-se afirmar que

- a) não se aplicam para conjuntos de entidades com menos de 5 atributos.
- b) o tamanho mínimo de seus atributos deve ser de 10 caracteres.
- c) devem ser formadas por, no mínimo, 3 atributos.
- d) os valores de seus atributos devem ser distintos para cada entidade de um conjunto de entidades.
- e) não podem conter atributos do tipo alfanumérico.



Uma das formas de impor restrições em um banco de dados relacional é por meio das **chaves primárias (Primary Key)**, sobre as quais pode-se afirmar que **os valores de seus atributos devem ser DISTINTOS para cada entidade de um conjunto de entidades**.

Exemplo:

NUMFUNC	NOMEFUNC	CPFFUNC	DEPTOFUNC
Chave primária			

Em chaves primárias, **não** pode haver valores nulos e nem repetição de tuplas. Quando a **chave primaria é simples** ela é formada por um único campo da tabela (esse campo não pode ter dois ou mais registros de mesmo valor e também não pode conter nenhum registro nulo). Se a **chave primária é composta** ela é formada por mais de um campo, os valores de cada campo podem se repetir, mas não a combinação desses valores. As chaves primárias não têm limitação da quantidade de atributos para serem formadas e nem tipo de dados que as forme.

Letra d.

059. (FGV/MPE-BA/ANALISTA TÉCNICO/TECNOLOGIA/2017) Considere um banco de dados no qual tenham sido criadas e instanciadas duas tabelas, como mostrado a seguir.

```
create table T2(b1 int primary key, b2 int)
create table T1(a1 int, a2 int,
               constraint FK1 foreign key (a1)
               references T2(b1))

insert into T2 values (1, 1)
insert into T2 values (2, 1)
insert into T1 values (1,2)
insert into T1 values (2,2)
```

O comando de inserção que provoca erro quando executado nesse banco de dados é:

- a)** insert into T1 values (2,2)
- b)** insert into T2 values (1,NULL)
- c)** insert into T1 values (1,NULL)
- d)** insert into T2 values (3,NULL)
- e)** insert into T1 values (NULL,NULL)



Um ponto que deve ser notado é que os valores (1,2) e (2,2) já foram inseridos na tabela T1. Esta tabela não apresenta nenhuma restrição de inserção em seus atributos a1 e a2. Por isso, qualquer valor pode ser inserido ou até mesmo reinserido nesta tabela. Isso significa que as alternativas A, C e E não emitem erro ao serem executadas.

Quanto à tabela T2, temos que seu atributo b1 é chave primária. Isto, em si, é uma restrição, já que significa que não pode haver dois registros com a mesma chave primária. Observe que os valores (1,1) e (2,1) já foram adicionados na tabela, isto é, os valores 1 e 2 já são chaves em T2. Uma nova inserção de chave com algum destes valores provocaria um erro. A alternativa D insere um novo registro com a chave primária de valor 3. Isso não é problema, já que é a primeira vez que o número 3 é inserido como chave primária. Já a alternativa B não pode ser executada, pois o número 1 já foi inserido na tabela T2 como chave primária. Portanto, alternativa B é a nossa resposta.

Alguns devem estar se perguntando se o valor nulo não é problema. Ele não é problema algum, pois, ao criar as tabelas T1 e T2, nenhuma restrição foi colocada nos atributos em relação a ele aceitar o valor nulo ou não.

Para efeitos didáticos, caso haja a necessidade de criar uma tabela e colocar uma restrição de não pode aceitar nulo em um determinado atributo, a sintaxe ficaria assim: “**CREATE TABLE** Teste (atributo1 int NOT NULL, atributo2 int)”. Isso significa que o atributo1 da tabela Teste não pode ser nulo, porém o atributo2 pode, já que a cláusula NOT NULL não foi adicionada neste atributo.

Letra b.

060. (FGV/SEPOG-RO/ANALISTA EM TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO/2017) A consulta SQL a seguir retorna uma série de nomes da tabela usuarios:
select nome from usuarios

Para obter a relação de nomes em ordem alfabética reversa você deve acrescentar ao final da consulta

- a) sort nome.
- b) sort reverse nome.
- c) order by nome desc.
- d) order by nome.
- e) order desc by nome.



A sintaxe, em SQL, para ordenar: **select * from Tabela ORDER BY atributo**.

A palavra “**desc**” indica que é ordenado do maior para o menor. Caso queira ordenar do menor para o maior, no lugar de **desc**, é só usar “**asc**”. Você pode utilizar sem o “**asc**” ou “**desc**” na consulta SQL, deixando somente “**order by nome**”. Nesse caso, o “**asc**” será utilizado por padrão. Com isso, temos que a sintaxe correta é: **select nome from usuarios order by nome desc**. Isso indica que será mostrado o nome de todos os usuários sendo ordenado alfabeticamente, começando do último para o primeiro, que foi chamado, pela questão, de “ordem alfabética reversa”.

Letra c.

061. (FGV/SEPOG-RO/ANALISTA EM TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO/2017) A figura a seguir mostra a estrutura das tabelas Produto, Venda e Cliente pertencentes a um banco de dados de uma empresa comercial.

Tabela:	Produto	Venda	Cliente
Coluna 1:	nome	id_cliente	id_cliente
Coluna 2:	id_produto	id_produto	nome
Coluna 3:	preço	data	telefone
Coluna 4:	peso	quantidade	endereço
Coluna 5:	origem	imposto	email
Coluna 6:			cep

A tabela Venda contém um registro para cada venda efetuada pela companhia. A fim de preservar a integridade referencial do banco de dados, assinale a opção que indica a coluna ou colunas dessa tabela que deveria(m) ser chaves estrangeiras.

- a) id_produto.
- b) imposto e quantidade.
- c) quantidade.
- d) id_produto e quantidade.
- e) id_cliente e id_produto.



Chave estrangeira é um dado que estabelece um relacionamento entre duas tabelas distintas do mesmo banco de dados. Ela pertence a uma tabela, fazendo referência a uma chave primária, situada em outra tabela. O respeito a esta regra é o que se denomina **integridade referencial**. É comum o fato de a chave estrangeira na tabela A ser o campo ID na tabela B, a qual é referenciada. Por exemplo: temos a tabela Departamento, com o campo ID_DEPARTAMENTO. Na tabela Funcionário, temos a chave estrangeira ID_DEPARTAMENTO indicando em qual departamento aquele funcionário trabalha.

Para cada valor de chave estrangeira do campo ID_DEPARTAMENTO em Funcionário, deve haver o mesmo na tabela Departamento. Um campo de chave estrangeira pode ter valor nulo. Isso não é problema. O problema é ter um valor de chave estrangeira que não existe na tabela em que ela é chave primária.

Voltando à questão, para manter a integridade referencial, a tabela Venda necessita ter um campo que faça referência à tabela Produto e outro campo que faça referência à tabela Cliente. Estes campos são: id_produto e id_cliente, respectivamente. Com isso, temos, como resposta, a alternativa E.

Letra e.

062. (FGV/SEPOG-RO/ANALISTA EM TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO/2017) Observe as tabelas a seguir:

Tabela: animais

familia	nome
mamifero	cachorro
mamifero	leao
peixe	linguado
passaro	aquia
reptil	cobra
passaro	pelicano
peixe	linguado

Tabela: tipos

familia	tipo_sangue
mamifero	quente
peixe	frio
passaro	quente

Assinale a opção que indica o número de linhas retornadas pela consulta SQL a seguir.

SELECT DISTINCT nome FROM animais a, tipos t

WHERE a.familia = t.familia

- a) 0**
- b) 1**
- c) 3**
- d) 4**
- e) 5**



Esta consulta tem o objetivo de selecionar, de forma única, todos os nomes de animal, cuja família na tabela “animais” esteja também na tabela “tipo”. Isso é visto em “WHERE a.familia = t.familia”. A consulta retorna somente o nome, já que somente isso foi pedido, como pode ser visto em “SELECT DISTINCT nome”.

Olhando na tabela animais, vemos quatro famílias distintas (mamífero, peixe, pássaro e réptil). Na tabela tipos, vemos três famílias distintas (mamífero, peixe e pássaro). Com isso, temos, como retorno, os seguintes 5 animais: cachorro, leão, linguado, aquia e pelicano. Nota-se a ausência do animal “cobra”, já que sua família (réptil) não está presente na tabela tipos. Perceba também que o animal linguado aparece duas vezes na tabela animais, porém ele deve ser contado somente uma única vez, já que a cláusula DISTINCT foi usada na consulta.

Com isso, temos a alternativa E como gabarito, isto é, 5 animais serão retornados.

Letra e.

063. (FGV/TCM-SP/AGENTE DE FISCALIZAÇÃO/TECNOLOGIA DA INFORMAÇÃO/2015) Views criadas nos bancos podem, de acordo com alguns critérios, ser naturalmente atualizáveis, o que significa, por exemplo, que podem ser objeto de comandos update do SQL sem a necessidade de mecanismos auxiliares ou triggers. Essa característica depende da expressão SQL que define a view e das tabelas/views de origem.

Considere alguns tipos de construções SQL que podem ser empregadas na definição de uma coluna de uma view:

- I – funções de agregação, tais como sum, avg
- II – funções escalares, tais como sin, trim
- III – expressões aritméticas
- IV – expressões condicionais, tais como case
- V – literais
- VI – subconsultas

Está correto concluir que uma determinada coluna NÃO pode ser objeto de atualização quando resultar de qualquer dos tipos:

- a) apresentados, exceto I, II e III;
- b) apresentados, exceto III e IV;
- c) apresentados, exceto V;
- d) apresentados, exceto VI;
- e) apresentados.



Uma **visão** na terminologia SQL é **uma tabela que é derivada de outras tabelas**. Uma visão não precisa existir fisicamente. Por isso, pode ser considerada como uma tabela virtual, isto

é, uma tabela que realmente não existe como tal, mas sim como derivação de uma ou mais tabelas básicas.

A definição da visão fica armazenada no dicionário de dados; esta definição mostra como ela é derivada das tabelas básicas.

O objetivo básico no uso de **visões** é restringir o acesso a certas porções dos dados por questões de segurança, além de predefinir certas consultas por meio de tabelas virtuais que poderão ser utilizadas por outras consultas. Pode-se criar uma visão com o comando CREATE VIEW.

CREATE VIEW nomeVisão AS expressão_de_consulta

As views atualizáveis são aquelas que possuem colunas que podem ser atualizadas. No entanto, existem restrições em relação às atualizações que podem ser realizadas. Em geral, views definidas a partir de diferentes tabelas usando junção e subconsultas, e ainda, usando funções de agregação ou expressões não são atualizáveis.

Letra e.

064. (FGV/CM CARUARU/ANALISTA LEGISLATIVO/INFORMÁTICA/2015) Analise o comando SQL mostrado a seguir juntamente com a instância da tabela C.

```
update C
set b = (select max(b) from C)
```

a	b
1	2
2	4
3	7
4	8

Assinale a opção que apresenta o número de registros da instância da tabela C que sofreram alguma alteração em seus atributos, em relação à instância mostrada, devido à execução desse comando.

- a)** zero
- b)** 1
- c)** 2
- d)** 3
- e)** 4



O comando “select max(b) from C” retornar o maior valor da coluna b da tabela C. A execução para estes valores irá retornar 8.

Logo, com a execução da atualização os outros três registros terão seus valores atualizados.

Letra d.

065. (FGV/TCE-SE/ANALISTA DE TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO/2015) No Oracle, a linguagem procedural que permite estreito acoplamento com o SQL é conhecida como:

- a) Data Pump;
- b) PL/SQL;
- c) SQL Explorer;
- d) SQL*Loader;
- e) Transact SQL.



O **PL/SQL (Procedural Language/Structured Query Language)** é uma extensão do padrão **SQL** específica para o SGBD Oracle. O PL/SQL acrescenta à linguagem padrão (SQL/ANSI) um conjunto de funções específicas para o banco de dados Oracle bem como estruturas similares às linguagens de programação como declaração de variáveis, estruturas de controle, procedimentos e objetos.

Letra b.

066. (CESPE/CEBRASPE/MINISTÉRIO DA ECONOMIA/TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO DE SOFTWARE/2020)



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'
```



A expressão SQL aqui destacada não está adequada: **truncate** não utiliza a cláusula **Where**. Além disso, o uso do **truncate** removerá todas as linhas de dados da tabela aluno, no entanto, a questão busca excluir somente as notas relacionadas do aluno de nome Fulano. Assim, poderemos usar o comando **DELETE** para remover os registros, preservando **a estrutura da tabela**. O correto deveria ser:

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

Nota: o nome do aluno não está na tabela matrícula. Pode ser localizado na tabela aluno, assim para deletar os registros será preciso buscar o id desse aluno.

Errado.

GABARITO

- | | |
|-------|-------|
| 1. C | 37. E |
| 2. b | 38. E |
| 3. a | 39. a |
| 4. C | 40. b |
| 5. E | 41. a |
| 6. C | 42. c |
| 7. C | 43. E |
| 8. b | 44. a |
| 9. E | 45. e |
| 10. C | 46. b |
| 11. b | 47. d |
| 12. E | 48. e |
| 13. C | 49. b |
| 14. C | 50. e |
| 15. b | 51. c |
| 16. E | 52. c |
| 17. c | 53. d |
| 18. b | 54. a |
| 19. d | 55. E |
| 20. c | 56. E |
| 21. a | 57. d |
| 22. b | 58. d |
| 23. c | 59. b |
| 24. e | 60. c |
| 25. d | 61. e |
| 26. a | 62. e |
| 27. a | 63. e |
| 28. d | 64. d |
| 29. c | 65. b |
| 30. a | 66. E |
| 31. e | |
| 32. a | |
| 33. a | |
| 34. e | |
| 35. d | |
| 36. C | |

REFERÊNCIAS

BRAGA, Regina. **Notas de aula**, UFJF, 2012.

CAMARGO, W. B. de. **SQL Server 2005 – Operadores Like e Not Like**. Disponível em: <https://www.devmedia.com.br/sql-server-2005-operadores-like-e-not-like/17292>. Acesso em: fev. de 2020.

DEVMEDIA. **Entendendo e usando índices - Parte 1**. Disponível em: <http://www.devmedia.com.br/entendendo-e-usando-indices-parte-1/6567>. Acesso em: fev. 2020.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. 4. ed. Porto Alegre: Sagra, 2001.

HERNANDEZ, Michael J. **Aprenda a projetar seu próprio banco de dados**. Tradução Patrizia Tallia Parenti. São Paulo: Makron, 2000.

KORTH, Henry F.; SILBERSCHATZ, Abraham. **Sistema de banco de dados**. Tradução Mauricio Heihachiro Galvan Abe. 6. ed. São Paulo: Makron, 2011.

LAUDON, K. C; LAUDON, J. P. **Sistemas de Informações Gerenciais**. São Paulo: Pearson Prentice Hall, 2007.

MACHADO, Felipe Nery Rodrigues; ABREU, Maurício Pereira de. **Projeto de banco de dados: uma visão prática**. 6. ed. São Paulo: Érica, 2000.

O'BRIEN, James A. **Sistemas de informação: e as decisões gerenciais na era da Internet**. Tradução Cid Knipel Moreira. São Paulo: Saraiva, 2003.

QUINTÃO, P. L. **Notas de aula da disciplina “Tecnologia da Informação”**. 2021.

Revistas SQL Magazine (ed. 31 e 32).

SOFTBLUE. **Curso SQL Completo**. 2018. Disponível em:

< <http://www.softblue.com.br/site/curso/id/3/CURSO+DE+SQL+COMPLETO+BASICO+AO+AVANÇADO+ON+LINE+BD03+GRATIS> >. Acesso em: 24 set.2020.

ROB, P.; CORONEL, C. **Sistemas de Banco de Dados Projeto, Implementação e Gerenciamento**. 2011.

SETZER, Valdemar W. **Banco de dados: conceitos, modelos, gerenciadores, projeto lógico, projeto físico.** 3. ed. rev. São Paulo: E. Blücher, 2002.

SETZER, http://www.ime.usp.br/~vwsetzer/dado-info.html_2001.

TAKAI, O.K.; ITALIANO,I.C.; FERREIRA, E.F. **Introdução a Banco de W3SCHOOLS. SQL Tutorial.** Disponível em: <<https://www.w3schools.com/sql/>>. Acesso em: 15 nov. 2019.

Patrícia Quintão



Mestre em Engenharia de Sistemas e computação pela COPPE/UFRJ, Especialista em Gerência de Informática e Bacharel em Informática pela UFV. Atualmente é professora no Gran Cursos Online; Analista Legislativo (Área de Governança de TI), na Assembleia Legislativa de MG; Escritora e Personal & Professional Coach.

Atua como professora de Cursinhos e Faculdades, na área de Tecnologia da Informação, desde 2008. É membro: da Sociedade Brasileira de Coaching, do PMI, da ISACA, da Comissão de Estudo de Técnicas de Segurança (CE-21:027.00) da ABNT, responsável pela elaboração das normas brasileiras sobre gestão da Segurança da Informação.

Autora dos livros: Informática FCC - Questões comentadas e organizadas por assunto, 3^a. edição e 1001 questões comentadas de informática (Cespe/UnB), 2^a. edição, pela Editora Gen/Método.

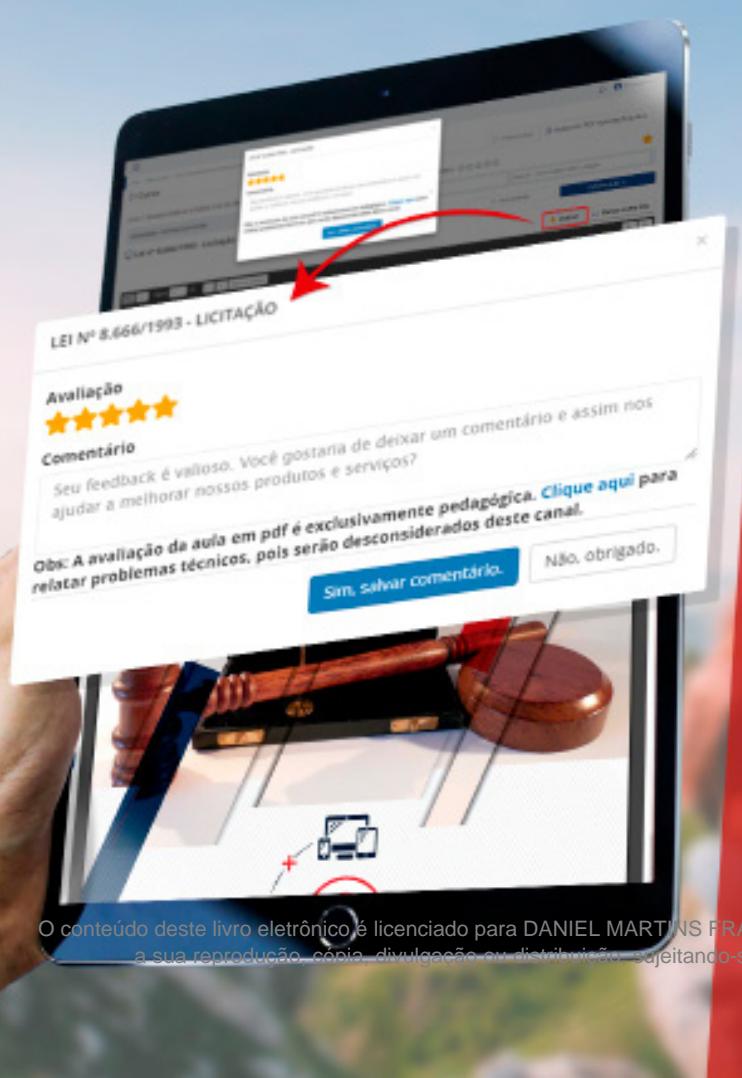
Foi aprovada nos seguintes concursos: Analista Legislativo, na especialidade de Administração de Rede, na Assembleia Legislativa do Estado de MG; Professora titular do Departamento de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia; Professora substituta do DCC da UFJF; Analista de TI/Suporte, PRODABEL; Analista do Ministério Público MG; Analista de Sistemas, DATAPREV, Segurança da Informação; Analista de Sistemas, INFRAERO; Analista - TIC, PRODEMGE; Analista de Sistemas, Prefeitura de Juiz de Fora; Analista de Sistemas, SERPRO; Analista Judiciário (Informática), TRF 2^a Região RJ/ES, etc.

@coachpatriciaquintao

/profapatriciaquintao

@plquintao

t.me/coachpatriciaquintao



NÃO SE ESQUEÇA DE AVALIAR ESTA AULA!

SUA OPINIÃO É MUITO IMPORTANTE
PARA MELHORARMOS AINDA MAIS
NOSSOS MATERIAIS.

ESPERAMOS QUE TENHA GOSTADO
DESTA AULA!

PARA AVALIAR, BASTA CLICAR EM LER
A AULA E, DEPOIS, EM AVALIAR AULA.

AVALIAR