TI TOTAL

TI PARA CONCURSOS

Professor
Ramon Souza

Tecnologia da Informação

TEORIA

SQL (DML)

SUMÁRIO

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

DIRETIVAS DA AULA

Esta aula aborda um dos temas mais importantes para concursos de Tecnologia da Informação: a linguagem SQL.

Inicialmente, abordaremos as sublinguagens que compõem a SQL, como DDL, DML, DCL e DTL e os principais comandos de cada uma delas. Essa é uma parte mais direta, mas importantíssima, pois várias questões requerem apenas que você saiba diferenciar essas sublinguagens e identificar a qual delas pertence algum comando.

Em seguida abordamos a DML em detalhes, iniciando pela sintaxe básica de uma consulta com SELECT. Fique bastante atento as cláusulas que são usadas e na ordem em que elas são declaradas nos comandos.

Nos tópicos seguintes abordamos a definição de alias, a ordenação de resultados e as funções de agregação, bem como o agrupamento de resultados. Até esse tópico trabalhamos a sintaxe com base em uma tabela. As questões mais simples e diretas cobram até essa parte da aula.

Evoluindo no entendimento da sintaxe, passamos a estudar as operações que permitem unir tabelas como o produto cartesiano e as operações de junções. Fique muito atento na diferença dos retornos dos tipos de join, pois as bancas exploram bastante.

Após, abordamos os operadores de conjuntos que servem para operar mais de uma consulta.

Depois, entramos nas chamadas consultas aninhadas, que são consultas dentro de outras. Essa é provavelmente a parte mais complexa da aula e requer bastante atenção para o entendimento.

Para finalizar o SELECT, abordamos algumas cláusulas especiais que vez por outra caem em alguma questão. “Passe o olho” na sintaxe básica dessas cláusulas.

Continuando a DML, abordamos as cláusulas DELETE, UPDATE e INSERT. Tratamos apenas da sintaxe básica delas, mas tenha em mente que praticamente tudo que foi estudado para DML, se aplica a esses comandos também.

Para fechar a aula, resumimos a lógica de três estados que trabalha com os valores TRUE, FALSE e UNKNOWN. Montamos a tabela verdade para essas operações. Você precisa ter noção de como são os resultados de operações envolvendo esses três valores.

ORIENTAÇÕES ESPECIAIS PARA AS BANCAS

CEBRASPE

A banca CEBRASPE costuma trazer questões mais diretas e pontuais. Há muitas questões conceituais sobre as definições e funções das cláusulas. Mesmo nas questões com código, geralmente são códigos bem mais curtos do que os trazidos em outras bancas e que questionam sobre cláusulas bem específicas.

FCC

A FCC costuma abordar o tema dando opções de código para serem avaliadas. Normalmente, dentre os itens há pelo menos três com sintaxe errada, então se você conhecer bem as sintaxes, já conseguirá eliminar os itens e chegar mais próximo das respostas corretas.

FGV

A banca FGV costuma ir um pouco além na cobrança desses conteúdos, focando bastante na análise dos resultados de códigos e no número de linhas retornadas. Para essa banca, você precisará treinar bastante a sua capacidade de raciocínio e velocidade de resolução.

Inclusive essa banca possui uma cláusula queridinha: EXISTS. Então, preste bastante atenção no tópico 2.1.9 que trata das consultas aninhadas. Além disso, atenção também a como realizar contagem de linhas nos joins (item 2.1.7).

VUNESP

A banca VUNESP aborda SQL de forma teórico-conceitual, mas também apresenta questões práticas sobre os comandos principais. Tenha atenção especial aos joins e funções de agregação.

GLOSSÁRIO DE TERMOS

Atributo ou campo: coluna de uma tabela no modelo relacional.

Alias: apelido para uma coluna ou tabela.

Cláusula: palavra chave ou termo utilizado em uma linguagem para compor um comando. Ex.: SELECT, WHERE, IN.

Consulta aninhada ou subconsulta: uma consulta interna a outra.

Consulta externa: uma consulta que possui subconsulta.

Expressão booleana: expressão que pode ser avaliada em verdadeiro ou falso.

Função de agregação: usadas para resumir informações de várias tuplas em uma síntese de tupla única.

Linguagem declarativa: expressa a lógica do cálculo sem descrever o fluxo de controle.

Linguagem não declarativa ou procedural: expressa o procedimento para o acesso aos dados.

Join: junção em inglês.

NULL: representa vazio ou ausência de valor.

Query: consulta em inglês.

Registros ou tuplas: linhas de uma tabela no modelo relacional.

Storage: armazenamento em inglês.

SQL (**Structured Query Language – Linguagem de Consulta Estruturada**): linguagem de consulta padrão para acesso e manipulação em bancos de dados relacionais.

1. INTRODUÇÃO AO SQL

A **linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada)** é a **linguagem padrão para acesso e manipulação de bancos de dados**. Por ser um padrão, ela foi um dos principais motivos para o sucesso dos bancos de dados comerciais, pois os usuários possuem liberdade para alternar entre os SGBDs, sem ter que mudar os códigos SQL. A padronização foi realizada pelo American National Standarts Institute (ANSI) e, por isso, a linguagem padrão é chamada SQL/ANSI. Contudo, vale mencionar que existem diferentes versões da linguagem SQL que fogem um pouco do padrão implementadas por cada SGBD.

Uma das principais características da **linguagem SQL** é que ela é **declarativa ou não procedural**. Uma linguagem declarativa expressa a lógica de um cálculo sem descrever seu fluxo de controle, ou ainda, uma linguagem em que apenas são declarados os dados que precisam ser acessados, sem necessidade de especificar como obter esses dados. A linguagem declarativa é oposta a uma linguagem procedural em que se precisa informar como obter os dados também, além dos dados desejados.

A **linguagem SQL** possui comandos para acesso e manipulação dos dados, mas também para a criação, exclusão e alteração das tabelas, índices e demais estruturas de bancos de dados, além de comandos para controlar os dados e transações. Dada essa amplitude, geralmente se fala em subdivisões da linguagem, geralmente tratando-se dos seguintes subgrupos:

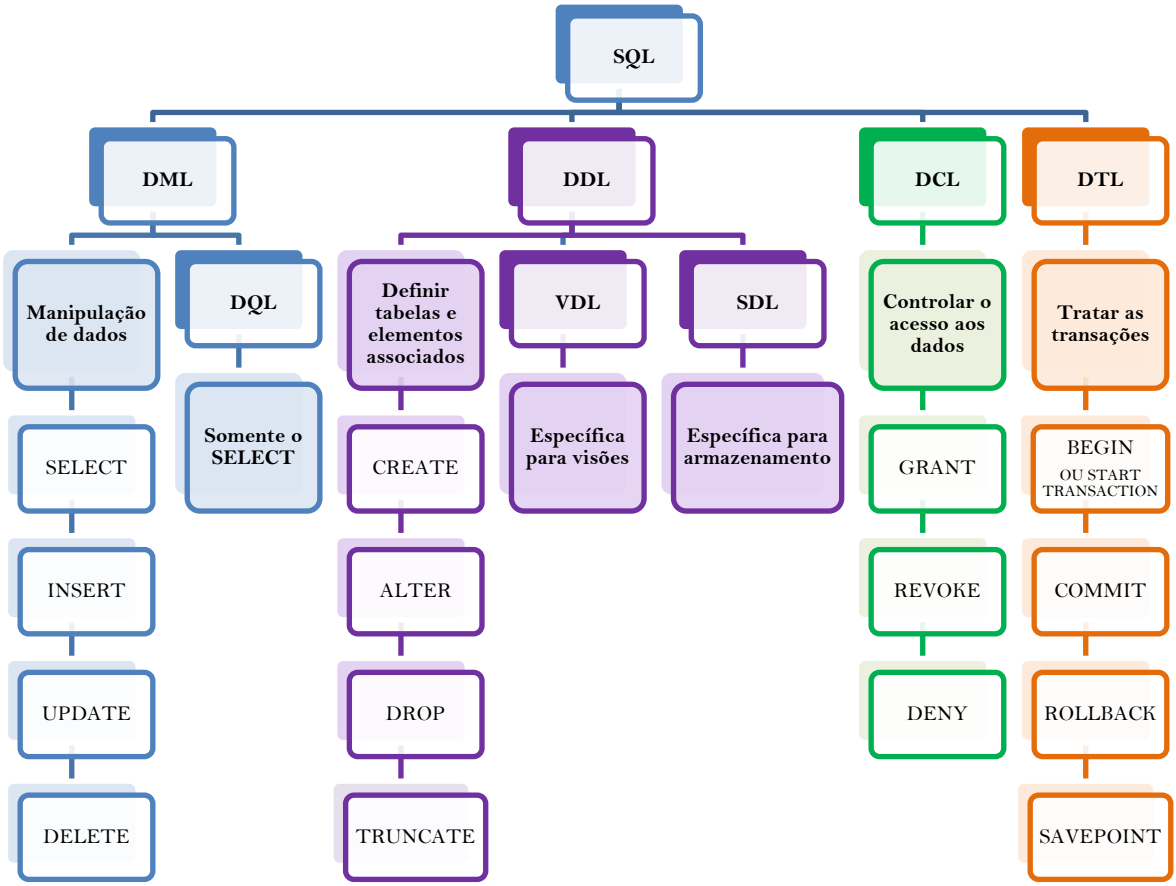
- **DML (Data Manipulation Language)**: utilizado para realizar **consultas, inclusões, alterações e exclusões de dados presentes em registros**. Estas tarefas podem ser executadas em vários registros de diversas tabelas ao mesmo tempo. Os comandos que realizam respectivamente as funções referidas são **SELECT, INSERT, UPDATE e DELETE**.
    - **DQL (Data Query Language)**: por vezes, o comando **SELECT** é dito como integrante da DQL e permite ao usuário **especificar uma consulta ("query") como uma descrição do resultado desejad**o. Esse comando é composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas.

- **DDL (Data Definition Language)**: permite ao utilizador **definir tabelas novas e elementos associados**. Os comandos desta linguagem são **CREATE, ALTER** e **DROP**, além do comando **TRUNCATE**.
    - **VDL (View Definition Language)**: há autores que falam em uma linguagem específica para a **definição de visões**.
    - **SDL (Storage Definition Language)**: há autores que falam em uma linguagem específica para a **definição do armazenamento ou especificação do esquema interno**.

- **DCL (Data Control Language)**: controla os aspectos de autorização de dados e licenças de usuários para **controlar quem tem acesso para ver ou manipular dados** dentro do banco de dados. Inclui os comandos **GRANT** e **REVOKE,** além do comando **DENY**.

- **DTL (Data Transaction Language)**: linguagem para **tratar as transações**. Os principais comandos desta linguagem são **COMMIT** e o **ROLLBACK**, além do **START TRANSACTION (ou BEGIN)** e do **SAVEPOINT**.

Esquematizando:



Esquema 1 – Linguagem SQL e subdivisões.

DICA DO PROFESSOR!!!

Se você souber **para que servem as subdivisões e quais comandos estão em cada uma delas**, já irá acertar algumas questões sobre esse assunto ou, pelo menos, eliminar alguns itens em questões de múltipla escolha.

EXEMPLIFICANDO!!!

A seguir temos um comando simples em SQL.

SELECT codigo, nome **FROM** cliente

ORDER BY nome **DESC**

Basicamente, esse comando declara que devem ser retornados códigos e nomes de todos os clientes, apresentados por ordem decrescente de nome. Não se preocupe em entender a sintaxe agora, pois vamos estudar os elementos necessários para você compreender os comandos que podem cair na sua prova.

1- (CESPE / CEBRASPE - 2024 – FINEP - Analista) Considerando-se que determinada empresa possui vários tipos de banco de dados para armazenamentos de dados estruturais, é correto afirmar que a linguagem SQL, nesse caso, tem a finalidade de

a) desenvolver aplicações baseadas na linguagem Java.

b) realizar cálculos matemáticos simples e complexos.

c) gerenciar sistemas operacionais.

d) desenvolver aplicativos para dispositivos móveis.

e) manipular dados em banco de dados.

Resolução:

Vamos analisar cada um dos itens:

a) **Incorreto:** SQL não é uma linguagem de desenvolvimento de aplicações em Java ou em qualquer outra linguagem de programação. As aplicações desenvolvidas em Java (ou em outra linguagem) costumam interagir com SQL para acesso e manipulação dos dados.

b) **Incorreto:** embora seja possível realizar cálculos simples com SQL, sua finalidade principal não é realizar cálculos matemáticos, mas sim manipular dados em banco de dados. Cálculos mais complexos são geralmente realizados por linguagens de programação que interagem com SQL.

c) **Incorreto:** SQL não tem nenhuma relação com o gerenciamento de sistemas operacionais. Ela é usada para gerenciar dados em sistemas de banco de dados.

d) **Incorreto**: SQL não é uma linguagem para desenvolver aplicativos para dispositivos móveis ou para qualquer outra plataforma. Ela é usada para acesso e manipulação dos dados. Para desenvolver os aplicativos, são usadas linguagem de programação.

e) **Correto:** a **linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada)** é a **linguagem padrão para acesso e manipulação de bancos de dados**.

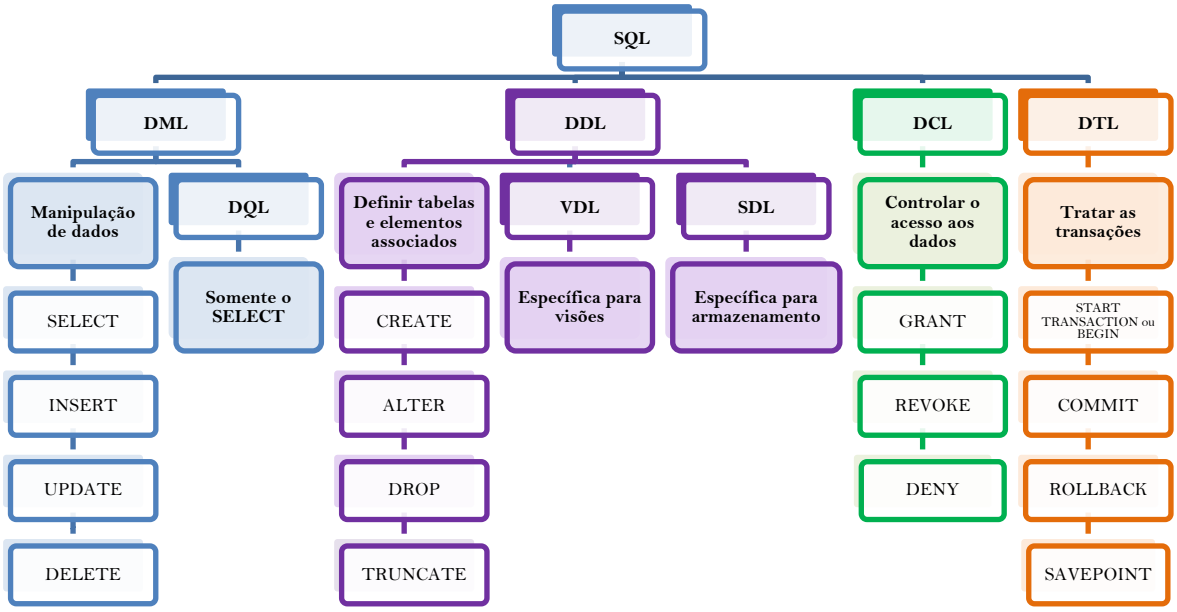
Gabarito: **Letra E**.

2- (CESPE / CEBRASPE - 2024 – CTI - Tecnologista Júnior) Considerando as linguagens e os fundamentos de bancos de dados relacionais, julgue o item subsequente.

No DML (data manipulation language), a instrução TRUNCATE elimina todas as linhas de uma tabela simultaneamente, enquanto a instrução DELETE oferece a possibilidade de excluir dados específicos ou todos os dados.

Resolução:

As descrições dos comandos estão corretas, porém TRUNCATE é um comando da DDL e não da DML. Os comandos são esquematizados a seguir conforme as subdivisões SQL:



Gabarito: **Errado.**

3- (FGV - 2024 – Câmara Municipal de Fortaleza – Analista Legislativo) No contexto das linguagens de manipulação de dados, relacione as linguagens com seus respectivos comandos:

1. DDL
2. DML
3. DQL
4. DTL
5. DCL

( ) GRANT

( ) TRANSACTION

( ) SELECT

( ) INSERT

( ) DROP

Assinale a opção que indica a relação correta na ordem apresentada.

a) 1 – 2 – 3 – 4 – 5.

b) 2 – 1 – 5 – 3 – 4.

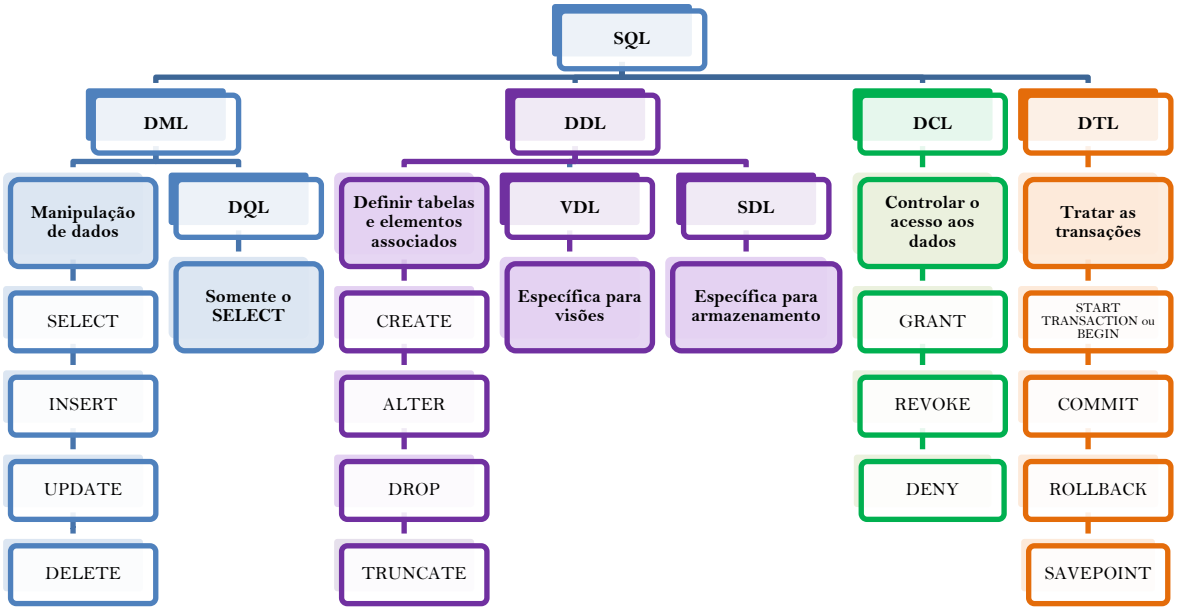
c) 3 – 4 – 1 – 5 – 2.

d) 4 – 3 – 2 – 1 – 5.

e) 5 – 4 – 3 – 2 – 1.

Resolução:

Os comandos são esquematizados a seguir conforme as subdivisões SQL:



Logo, podemos associar corretamente como:

(**5-DCL**) GRANT

(**4-DTL**) TRANSACTION

(**3-DQL ou 2-DML**) SELECT

(**2-DML**) INSERT

(**1-DDL**) DROP

Gabarito: **Letra E.**

2. SQL (DML)

2.1 DML: instrução SELECT

2.1.1 Sintaxe básica do SELECT

A instrução básica para **recuperar informações de um banco de dados** é a instrução **SELECT**. É importante ressaltar que esta instrução **não realiza a mesma função que a operação de SELEÇÃO da álgebra relacional**, mas sim a função da **operação de PROJEÇÃO**, pois após o SELECT são informadas as colunas que se deseja retornar.

A sintaxe básica de uma instrução SELECT é da seguinte forma:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição;

Esse comando permite recuperar os valores das colunas de uma tabela que cumprem uma determinada condição. A cláusula **WHERE** realiza a mesma função da **operação de SELEÇÃO** da álgebra relacional, pois filtra linhas.

É possível realizar uma consulta sem indicar colunas específicas, retornando-se **todas as colunas**, basta usar o * do seguinte modo:

SELECT * **FROM** nome_da_tabela **WHERE** condição;

É possível realizar uma instrução SELECT mesmo **sem indicar nenhuma condição** e, portanto, a cláusula **WHERE** é **opcional**. Assim, teremos a forma mais simples:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela;

EXEMPLIFICANDO!!!

Vamos supor que a partir de uma tabela de Clientes com vários atributos, tenhamos interesse em apenas o nome e a cidade de cada cliente. Poderemos realizar esta consulta com base no comando a seguir:

SELECT Nome_Cliente, Cidade **FROM** Clientes;

CLIENTES

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

SELECT Nome_Cliente, Cidade

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

Note que após o SELECT são indicados os atributos que aparecerão no resultado. Caso seja utilizado *, então todos os atributos serão exibidos.

Em uma tabela, uma coluna pode conter valores duplicados e, algumas vezes, estamos interessados somente nos valores diferentes. Para isso, usa-se a cláusula **DISTINCT** após o SELECT. Ao usar **SELECT DISTINCT**, somente serão retornados os **valores diferentes ou distintos**, isto é, não são exibidos valores duplicados. A sintaxe é:

SELECT DISTINCT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Se desejar saber em que países você possui clientes, a consulta a seguir pode ser utilizada:

SELECT Pais **FROM** Clientes;

O retorno dessa consulta será:

Pais
Germany
Mexico
Mexico

} Mesmo os duplicados são retornados.

Note, porém, que neste resultado, o Pais Mexico foi retornado duas vezes. Para eliminar essa redundância, podemos usar a cláusula DISTINCT:

SELECT DISTINCT Pais **FROM** Clientes;

O novo resultado será:

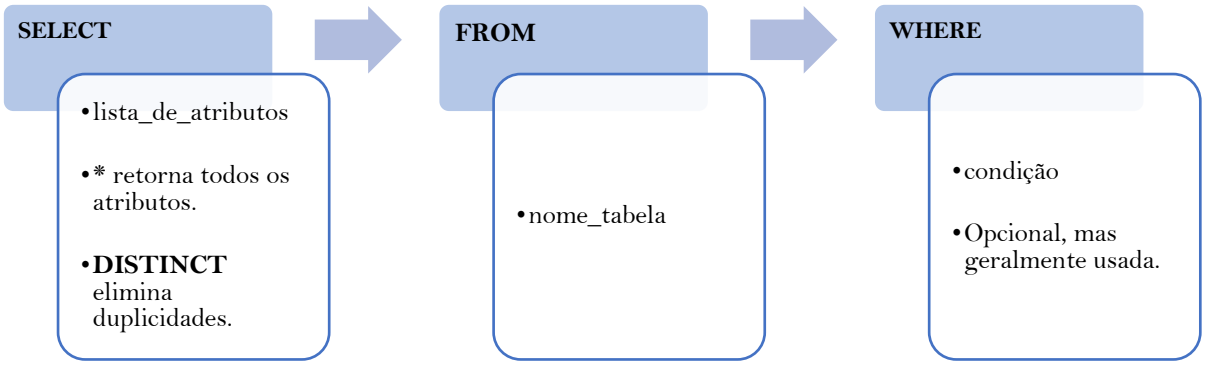
Pais
Germany
Mexico

} Os duplicados aparecem somente uma vez no resultado.

ATENÇÃO!!!

Cuidado, pois a avaliação de redundância considera todos os valores retornados. Assim, se mais de um atributo for retornado, somente serão consideradas duplicadas as linhas que possuírem os mesmos valores para todos os atributos. Por exemplo, em **SELECT DISTINCT** Nome, Pais **FROM** Clientes, os pares (João, México) e (João, Espanha) não são duplicados, pois embora o nome seja igual, o país é diferente.

Diante do visto até agora, a sintaxe básica é esquematizada a seguir:



Esquema 2 – Sintaxe básica da instrução SELECT.

4- **(CESPE / CEBRASPE - 2024 – LNA - Tecnologista)** Assinale a opção em que é apresentada a palavra-chave, em SQL, que deve ser incluída em uma instrução de SELECT para evitar a apresentação de resultados duplicados da tabela.

a) UNIQUE

b) NOTDUPLICATE

c) NOTALL

d) DISTINCT

e) ONLY

Resolução:

Em uma tabela, uma coluna pode conter valores duplicados e, algumas vezes, estamos interessados somente nos valores diferentes. Para isso, usa-se a cláusula **DISTINCT** após o SELECT. Ao usar **SELECT DISTINCT**, somente serão retornados os **valores diferentes ou distintos**, isto é, não são exibidos valores duplicados. A sintaxe é:

SELECT DISTINCT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição;

Gabarito: **Letra D**.

5- **(CESPE / CEBRASPE - 2024 – MPO - Analista de Planejamento e Orçamento)** Acerca de fundamentos dos bancos de dados relacionais, normalização, diagrama entidade-relacionamento e linguagem SQL, julgue o item a seguir.

Em uma consulta SQL, a cláusula FROM corresponde à seleção do predicado que envolve atributos da relação determinada pela cláusula SELECT.

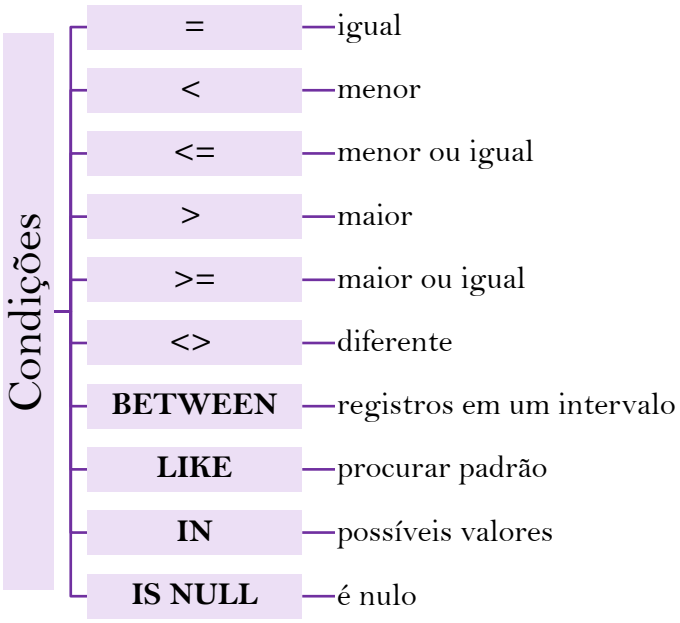
Resolução:

A cláusula FROM indica a(s) tabela(s) de onde os dados serão consultados. Os atributos são definidos na cláusula SELECT. O predicado (condição) é definido na cláusula WHERE.

Gabarito: **Errado.**

Condições

A **condição** é uma expressão condicional (booleana) que **identifica tuplas a serem recuperadas**. Em SQL, os operadores básicos de comparação na clásula WHERE são:



Esquema 3 – Condições na cláusula WHERE.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje recuperar os clientes que residem no Mexico, então poderá realizar a seguinte consulta:

```
SELECT * FROM Clientes WHERE Pais='Mexico';
```

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

6- (CESPE / CEBRASPE - 2022 - MP TCE-SC - Analista de Contas Públicas) Acerca do conceito de view, da modelagem dimensional, do modelo de referência CRISP-DM e da linguagem SQL, julgue o item subsequente.

O comando a seguir tem a finalidade de mostrar o nome e o email de todos os procuradores que recebem salários acima de R$ 30.000,00.

SELECT nome, email

FROM procurador

WHEN salario >= 30.000,00;

Resolução:

Primeiramente, a cláusula correta para definir uma condição no comando é WHERE e não WHEN. WHEN existe, porém é usada para condições dentro da instrução CASE.

Em segundo lugar, note que o operador utilizado é >=, então na verdade, o comando retorna o nome e e-mail dos procuradores com salário MAIORES OU IGUAIS a 30.000,00 e não apenas acima desse valor. Por isso, a questão está errada.

Gabarito: **Errado.**

7- (FCC - 2022 – TJ CE - Analista Judiciário) Uma tabela chamada cliente possui os campos abaixo

id - int (Primary Key)

nome - varchar(70)

cidade - varchar(40)

estado - varchar(40)

Em condições ideais, para exibir os dados de todos os clientes, cujo nome da cidade não seja igual ao nome do estado, utiliza-se a instrução SQL:

SELECT * FROM cliente WHERE

a) cidade =! estado;

b) cidade !EQUALS estado;

c) cidade UNLIKE(estado);

d) cidade <> estado;

e) cidade UNLIKE estado;

Resolução:

O operador para diferente no SQL padrão é o <>. Logo, o item d) realiza o que a questão busca. É possível também usar != em alguns SGBDS (o item a inverte como =!, por isso não pode ser a resposta).

Gabarito: **Letra D.**

Note que praticamente todas as condições são bem intuitivas, então vamos detalhar somente as condições BETWEEN, LIKE, IN e IS NULL.

Operador BETWEEN

O operador **BETWEEN** recupera os **registros que estão em um determinado intervalo**. Os valores podem ser números, texto ou data e tanto os **valores de início como o de fim são incluídos**.

A sintaxe básica com esse operador é:

> **SELECT** coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **BETWEEN** valor1 **AND** valor2;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje recuperar somente os clientes que possuam CEP entre 00000 e 06000, então poderá usar a seguinte consulta:

SELECT * **FROM** Clientes **WHERE** CEP **BETWEEN** 00000 **AND** 06000;

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Nesse exemplo, temos poucos dados, mas note que o operador BETWEEN pode ser bastante útil para um banco com uma infinidade de dados em que queremos separar dados com base em intervalos.

8- **(CESPE / CEBRASPE - 2024 – FINEP - Analista)** Assinale a opção que apresenta a cláusula SQL que permite extrair dados em determinado intervalo.

a) find in

b) like

c) alias

d) between

e) ranges

Resolução:

O operador **BETWEEN** recupera os **registros que estão em um determinado intervalo**. Os valores podem ser números, texto ou datas e tanto os valores de início como o de fim são incluídos.

Gabarito: **Letra D**.

9- **(CESPE / CEBRASPE - 2023 – AGER-MT - Analista Regulador)** Em linguagem de manipulação de dados DML, o operador SQL BETWEEN serve para

a) delimitar o valor de uma coluna na cláusula WHERE.

b) delimitar as colunas a serem apresentadas na cláusula WHERE.

c) delimitar os limites de um campo para a cláusula INSERT.

d) restringir a quantidade de linhas a serem recuperadas na cláusula UPDATE.

e) restringir a quantidade de campos na cláusula DELETE.

Resolução:

O operador **BETWEEN** recupera os **registros que estão em um determinado intervalo**. Dito isto, vamos analisar cada um dos itens:

a) **Incorreto**: não delimita um único valor, mas um intervalo.

b) **Incorreto**: não delimita as colunas, mas sim registros com base em um intervalo.

c) **Correto**: isso mesmo, o BETWEEN irá delimintar um intervalo de valores para um ou mais campos, podendo ser usado em um SELECT, UPDATE, DELETE ou INSERT após a cláusula WHERE.

d) **Incorreto**: a cláusula UPDATE não serve para recuperar, mas sim para atualizar.

e) **Incorreto**: não delimita os campos, mas sim registros com base em um intervalo.

Gabarito: **Letra C**.

Operador IN

O operador **IN** permite especificar **múltiplos valores para uma condição**, isto é, a condição será testada com base na lista de valores indicada.

A sua sintaxe básica é:

> **SELECT** coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **IN** (valor1, valor2, ...);

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje consultar os clientes de cidades específicas, digamos que de Strasbourg e Berlin. Então poderá usar a seguinte consulta:

SELECT * **FROM** Clientes **WHERE** Cidade **IN** ('Berlin', 'Strasbourg');

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

Note que somente foram retornados os registros cuja cidade está informada na cláusula IN, que foram Berlin e Strasbourg.

10- (FCC - 2018 - SEGEP-MA - Analista Executivo - Programador de Sistemas) Um Programador de Sistemas recuperou todos os dados dos países Brasil, Argentina e Peru gravados no campo Pais da tabela Cliente, abaixo especificada:

Tabela Cliente:

IdCliente

NomeCliente

Endereco

CEP

Cidade

SiglaUF

Pais

A sintaxe SQL correta que ele usou para realizar essa atividade foi SELECT * FROM Cliente

a) WHERE Pais IN ('Argentina', 'Peru', 'Brasil');

b) WHEN Pais = ('Argentina', 'Peru', 'Brasil');

c) WHERE Pais = ('Argentina', 'Peru', 'Brasil');

d) WHEN Pais IN ('Argentina', 'Peru', 'Brasil');

e) WHERE Pais BETWEEN ('Argentina', 'Peru', 'Brasil');

Resolução:

O **operador IN** permite especificar **múltiplos valores para uma condição**, isto é, a condição será testada com base na lista de valores indicada. A sua sintaxe básica é:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **IN** (valor1, valor2, ...);

Logo,

a) **Correto**: WHERE Pais IN ('Argentina', 'Peru', 'Brasil');

b) **Incorreto**: ~~WHEN~~ Pais ~~=~~ ('Argentina', 'Peru', 'Brasil');

A cláusula não é WHEN, mas sim WHERE. Além disso, o = não pode ser usado para comparação com mais de um valor.

c) **Incorreto**: WHERE Pais ~~=~~ ('Argentina', 'Peru', 'Brasil');

O = não pode ser usado para comparação com mais de um valor.

d) **Incorreto**: ~~WHEN~~ Pais IN ('Argentina', 'Peru', 'Brasil');

A cláusula não é WHEN, mas sim WHERE.

e) **Incorreto**: WHERE Pais ~~BETWEEN~~ ('Argentina', 'Peru', 'Brasil');

A cláusula BETWEEN é para intervalos e não para listas de valores.

Gabarito: **Letra A**.

Operador LIKE

O operador **LIKE** é utilizado para **procurar um padrão em uma coluna**. Este operador permite a comparação com parte de uma cadeia de caracteres. Este operador é usado em conjunto com dois elemenos curinga (wildcard):

- **%** substitui um número qualquer de 0 ou mais caracteres.
- _ substitui um único caractere.

A sintaxe com esse operador é:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **LIKE** padrão;

Você pode definir uma série de padrões para a consulta, mas o quadro a seguir apresenta alguns significados básicos para os padrões:

Operador LIKE	Procurar padrão em uma coluna
%	Substitui um número qualquer de 0 ou mais caracteres.
_	Substitui um único caractere.
LIKE 'A%'	Qualquer string que inicie com A.
LIKE '%A'	Qualquer string que termine com A.
LIKE '%A%'	Qualquer string que tenha A em qualquer posição.
LIKE 'A_'	String de dois caracteres que tenha a primeira letra A e o segundo caractere seja qualquer outro.
LIKE '_A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja a letra A.
LIKE '_A_'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
LIKE '%A_'	Qualquer string que tenha a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '_A%'	Qualquer string que tenha a letra A na segunda posição e o primeiro caractere seja qualquer outro caractere.
LIKE '_ _ _'	Qualquer string com exatamente três caracteres.
LIKE '_ _ _%'	Qualquer string com pelo menos três caracteres.
LIKE '%"%'	Qualquer string que tenha o caractere " em qualquer posição.

Esquema 4 – Operador LIKE e exemplos.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje consultar o nome dos clientes que iniciem com 'An'. A seguinte consulta servirá a esse propósito:

SELECT * **FROM** Clientes **WHERE** Nome_Cliente **LIKE** 'An%';

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Note que somente foram retornados os registros dos clientes que começam com 'An' e terminam com qualquer sequência de caracteres.

Outros exemplos de consultas e padrões possíveis seriam:

- Clientes que tenham 'Ana' em qualquer parte do nome:
  - **SELECT** * **FROM** Clientes **WHERE** Nome_Cliente **LIKE** '%Ana%';

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

- Clientes que tenham 'l' como a segunda letra:
  - **SELECT** * **FROM** Clientes **WHERE** Nome_Cliente **LIKE** '_l%';

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

As possibilidades são inúmeras e dependem do seu objetivo ao realizar a consulta.

11- (FGV - 2023 – DPE RS – Técnico) A programadora Fabiana elaborou um relatório com o nome do autor dos processos utilizando o seguinte comando SQL:

SELECT Autor FROM tab_processo

Ao ver o relatório gerado, o chefe da Fabiana solicitou um relatório contendo apenas os autores que possuem o primeiro nome Elizabeth. Contudo, Fabiana observou que havia diferentes grafias, como: Elisabeth, Elizabete etc.

No MySQL, para garantir que qualquer Autor cujo nome comece pelas letras "eli" sejam recuperados, Fabiana deve complementar o comando SQL com a cláusula WHERE Autor LIKE:

a) 'eli%'

b) '%eli%'

c) 'eli_'

d) '_eli'

e) '_eli%'

Resolução:

Como o padrão desejado é todo nome que comece com eli, então podemos utilizar 'eli%'.

Texto começado com Eli. ← **Eli%** → Seguido de quaisquer 0 ou mais caracteres

Gabarito: **Letra A.**

12- (FCC - 2023 – COPERGÁS- Analista) Em uma tabela chamada user de um banco de dados aberto e em condições ideais, para selecionar todos os registros que possuem nomes (campo nome) iniciados com a letra E e terminados com a letra l utiliza-se a instrução SQL SELECT * FROM user

a) LIKE = 'E*l';

b) WHERE nome = 'E%l';

c) LIKE nome CONTAINS 'E%l';

d) WHERE nome LIKE 'E*l';

e) WHERE nome LIKE 'E%l';

Resolução:

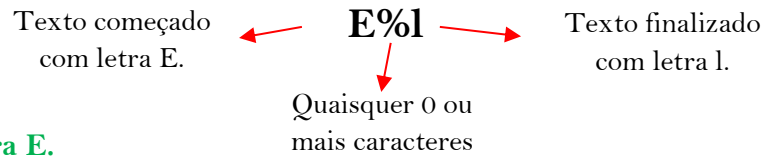
O operador **LIKE** é utilizado para **procurar um padrão em uma coluna**. A sintaxe com esse operador é:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **LIKE** padrão;

Logo, podemos eliminar os itens a), b) e c) por não apresentarem a palavra-chave LIKE.

Entre d) e e), a única diferença é o uso de * e %. O operador LIKE pode ser usado com dois curingas, o % para representar quaisquer quantidades de caracteres e o _ para representar um único caractere. Logo, o item correto é o e).

Explicando o padrão:



Gabarito: **Letra E.**

13- (CESPE / CEBRASPE - 2022 - BANRISUL - Técnico de Tecnologia da Informação) No que se refere à álgebra relacional e a SQL, julgue o item a seguir.

Considerando-se uma tabela nomeada empregados que contém, entre outras colunas, uma identificada como nome, o comando SQL que retorna o nome de todos os empregados que tenham o nome luiz ou luis é o seguinte

SELECT nome FROM empregados WHERE nome like '%lui_%'

Resolução:

Vamos analisar o comando:

SELECT nome

-- selecionar o nome

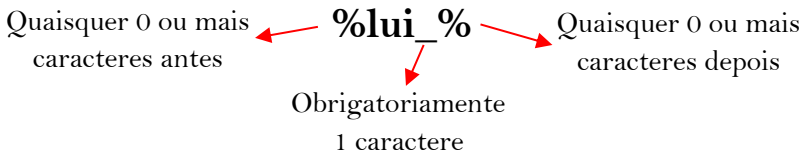
FROM empregados

-- a partir da tabela empregados

WHERE nome like '%lui_%'

-- cujo nome possua 'lui' seguido obrigatoriamente de um caractere (_) e que possua quaisquer caracteres antes (%) e quaisquer caracteres depois (%).

Explicando o padrão:



Nesse caso, luis ou luiz respeitam o padrão.

Gabarito: **Certo**.

Operador IS NULL

O operador **IS NULL** **testa se um valor é NULO**, isto é, se o atributo não possui um valor específico.

A sua sintaxe básica é:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **IS NULL;**

Caso queira os não nulos, então basta usar **IS NOT NULL.**

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** coluna **IS NOT NULL;**

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje consultar os clientes que possuam o Nome_Contato nulo. A seguinte consulta servirá a esse propósito:

SELECT * FROM Clientes **WHERE** Nome_Contato **IS NULL**;

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
5	Donald Pateta	null	null	null	null	null

E se você quiser retornos os clientes que não possuam o Nome_Contato nulo:

SELECT * FROM Clientes **WHERE** Nome_Contato **IS NOT NULL**;

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

ATENÇÃO!!!

A **comparação com NULL não deve ser feita com os operadores lógicos = ou <>**, mas sim **com IS NULL e IS NOT NULL**. Ao comparar qualquer coisa com NULL usando os operadores lógicos comuns, será retornado um resultado desconhecido na comparação (UNKNOWN) e, por isso, não serão retornadas linhas.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

O comando

SELECT * FROM Clientes **WHERE** CEP = **NULL**;

Não irá retornar nenhuma linha, pois ainda que exista um CEP nulo, a comparação não deve ser realizada com operadores lógicos.

Agora, o comando

SELECT * FROM Clientes **WHERE** CEP **IS NULL**;

Irá retornar o resultado a seguir:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
5	Donald Pateta	null	null	null	null	null

Nesse caso, são retornadas as linhas que possuam CEP nulo. Na tabela, somente a linha do IDCliente = 5.

Então, muito cuidado na hora de analisar os comandos, principalmente para as questões que perguntam quantas linhas são retornadas.

14- (FGV - 2022 – TCU – Auditor Federal de Controle Externo – Controle Externo – Auditoria Governamental) Na questão abaixo, considere as tabelas de banco de dados T, TX e DUAL, exibidas com suas respectivas instâncias a seguir.

T

sequencia	caracteristica
1	23987
2	9845
3	NULL
4	40983
6	48750
7	NULL
8	NULL
10	48750
12	48750

TX

sequencia	caracteristica
2	9845
3	998034
4	50932
5	24390
6	48750
6	50296
7	NULL
8	998746
9	32746
9	NULL
9	22798

DUAL

x
NULL

Analise os cinco comandos SQL exibidos abaixo, utilizando a tabela DUAL apresentada anteriormente.

(1) select * from dual where x = null

(2) select * from dual where x <> null

(3) select * from dual where x > 10

(4) select * from dual where not x > 10

(5) select * from dual where x > 10

union

select * from dual where x <= 10

Se os resultados desses comandos fossem separados em grupos homogêneos, de modo que em cada grupo todos sejam idênticos e distintos dos elementos dos demais grupos, haveria:

a) apenas um grupo;

b) apenas dois grupos;

c) apenas três grupos;

d) apenas quatro grupos;

e) cinco grupos.

Resolução:

A tabela DUAL possui apenas um registro com valor nulo (NULL).

A **comparação com NULL não deve ser feita com os operadores lógicos = ou <>**, mas sim **com IS NULL e IS NOT NULL**. Ao comparar qualquer coisa com NULL usando os operadores lógicos comuns, será retornado um resultado desconhecido na comparação (UNKNOWN) e, por isso, não serão retornadas linhas.

Dito isto, vamos analisar os comandos:

(1) **select * from dual where x = null** retorna **vazio**, isto é, nenhuma linha, pois está se comparação NULL usando o comparador lógico =.

(2) **select * from dual where x <> null** retorna **vazio**, isto é, nenhuma linha, pois está se comparação NULL usando o comparador lógico <>.

(3) **select * from dual where x > 10** retorna **vazio**, pois não existe nenhum x maior que 10.

(4) **select * from dual where not x > 10** retorna **vazio**, pois não existe nenhum x que não é maior que 10.

(5) **select * from dual where x > 10** retorna **vazio**, pois não existe nenhum x maior que 10.

union

select * from dual where x <= 10 retorna **vazio**, pois não existe nenhum x menor ou igual a 10.

A união dos dois conjuntos **vazio** será **vazio** também.

Logo, todos os comandos retornam resultados vazios, isto é, sem nenhuma linha. Portanto, podemos classificá-los em apenas um grupo, o de resultados vazios.

A pegadinha da questão era saber que a comparação com nulo não dá erro, pois se desse erro, teríamos dois grupos, um de erros e um de vazios.

Gabarito: **Letra A**.

Mais de uma condição e negação de 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, ... **FROM** nome_da_tabela **WHERE** condição1 **AND** condição2 **AND** condição3 ...;

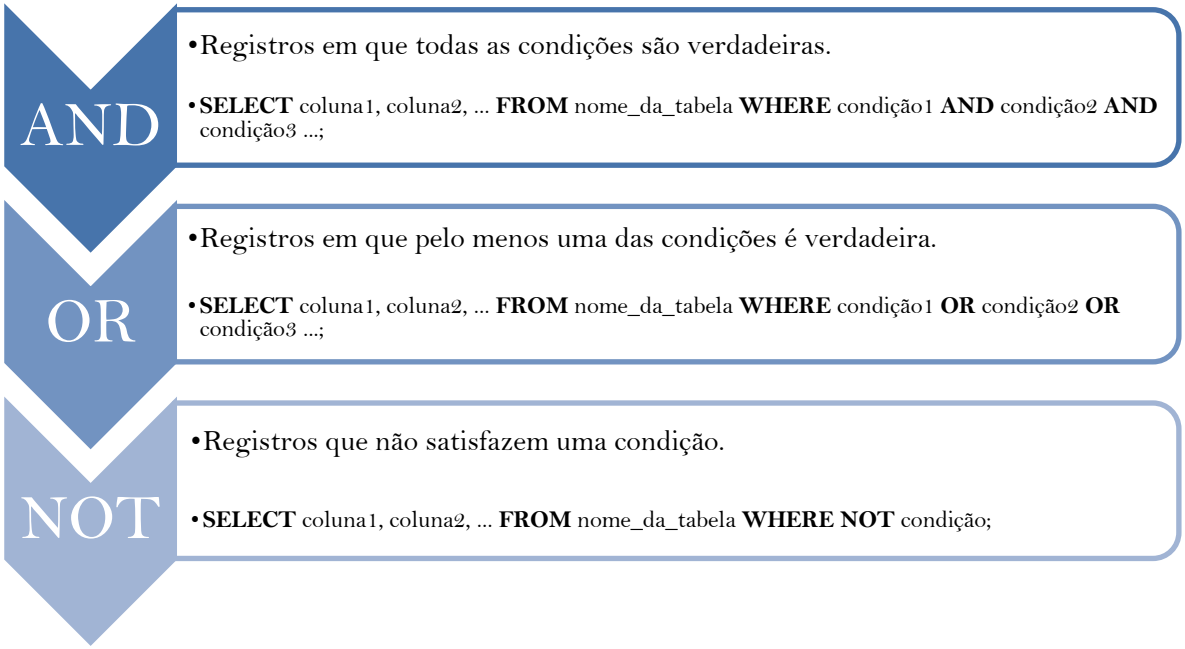
- **OR**: exibe os registros em que pelo menos uma condição é verdadeira.

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

Também é possível recuperar os registram que não safisfazem uma determinada condição. Basta usar o **NOT** antes da condição.

- **NOT**: exibe os registros que não satisfazem uma condição.

  SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE NOT** condição;



Esquema 5 – Cláusulas para definir mais de uma condição e negação de condição.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Para recuperar somente os clientes do Mexico e cujo CEP seja maior ou igual a 05023, podemos usar a seguinte consulta:

SELECT * **FROM** Clientes **WHERE** Pais='Mexico' **AND** CEP>=05023;

O retorno dessa consulta será:

As duas condições são atendidas

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Para retornar os clientes da Alemanha (Germany) OU os que possuam CEP 05021, podemos usar a consulta:

SELECT * **FROM** Clientes **WHERE** Pais='Germany' **OR** CEP=05021;

O retorno dessa consulta será:

Uma ou outra condição é atendida

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

Para retornar os clientes que não são do Mexico, podemos utilizar a consulta:

SELECT * **FROM** Clientes **WHERE NOT** Pais='Mexico';

Teremos como retorno:

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

15- (FGV - 2024 – Câmara Municipal de Fortaleza – Analista Legislativo) Em um banco de dados relacional, considere a tabela a seguir, que possui informações sobre diferentes tipos de produtos, incluindo eletrônicos, roupas, eletrodomésticos, entre outros:

Produto (ID, Nome, Tipo, Preço, Fabricante)

Assinale a alternativa que corresponde à consulta que retornará o nome e o preço dos produtos que possuem a palavra "Smart" em seu tipo, somente do fabricante "Banana Inc." e preço abaixo de R$2000,00.

a) SELECT Nome

```
FROM Produto
WHERE Fabricante = 'Banana Inc.'
AND Nome = 'Smart'
AND Preço < 2000;
```

b) SELECT Nome, Preço

```
FROM Produto
WHERE Tipo = 'Smart'
AND Fabricante = 'Banana Inc.'
AND Preço < 2000;
```

c) SELECT Nome, Preço

```
FROM Produto
WHERE Tipo LIKE '%Smart%'
AND Fabricante = 'Banana Inc.'
AND Preço < 2000;
```

d) SELECT Fabricante, Preço

```
FROM Produto
WHERE Fabricante = 'Banana Inc.'
 AND Tipo = 'Smart'
 AND Preço < 2000;
```

e) SELECT Tipo, Preço

```
 FROM Produto
WHERE Tipo LIKE '%Smart%'
```

AND Fabricante = 'Banana Inc.'

AND Preço > 2000;

Resolução:

Queremos uma consulta que retorne o nome e o preço dos produtos que possuem a palavra "Smart" em seu tipo, somente do fabricante "Banana Inc." e preço abaixo de R$2000,00. Vamos montar esse comando passo a passo:

Passo 1 -> Retornar nome e preço:

SELECT nome, preço ...

Note que definindo só isso, já conseguimos eliminar os itens a), d) e e), pois eles trazem atributos diferentes no SELECT. Somente os itens b) e c) possuem os atributos desejados.

Passo 2 -> Da tabela produto:

FROM Produto ...

Passo 3 -> 1ª condição de possuir a palavra "Smart" no tipo:

WHERE Tipo LIKE '%Smart%' ...

Aqui vale um cuidado, pois como queremos quaisquer produtos que POSSUEM a palavra "Smart" no tipo, não devemos utilizar o =. Se utilizássemos o igual, apenas seriam retornados os produtos que tivessem exatamente o tipo "Smart" e não com essa palavra em qualquer parte do tipo. Com o igual, "TV Smart" não seria retornada, por exemplo. Com isso eliminamos o item b).

Por isso, devemos usamos o operador LIKE para definir o padrão de ter a palavra em qualquer parte do tipo:

Passo 4 -> 2ª condição de que o fabricante seja "Banana Inc.":

AND Fabricante = 'Banana Inc.'

Nesse caso, queremos exatamente 'Banana Inc.', por isso usamos o =.

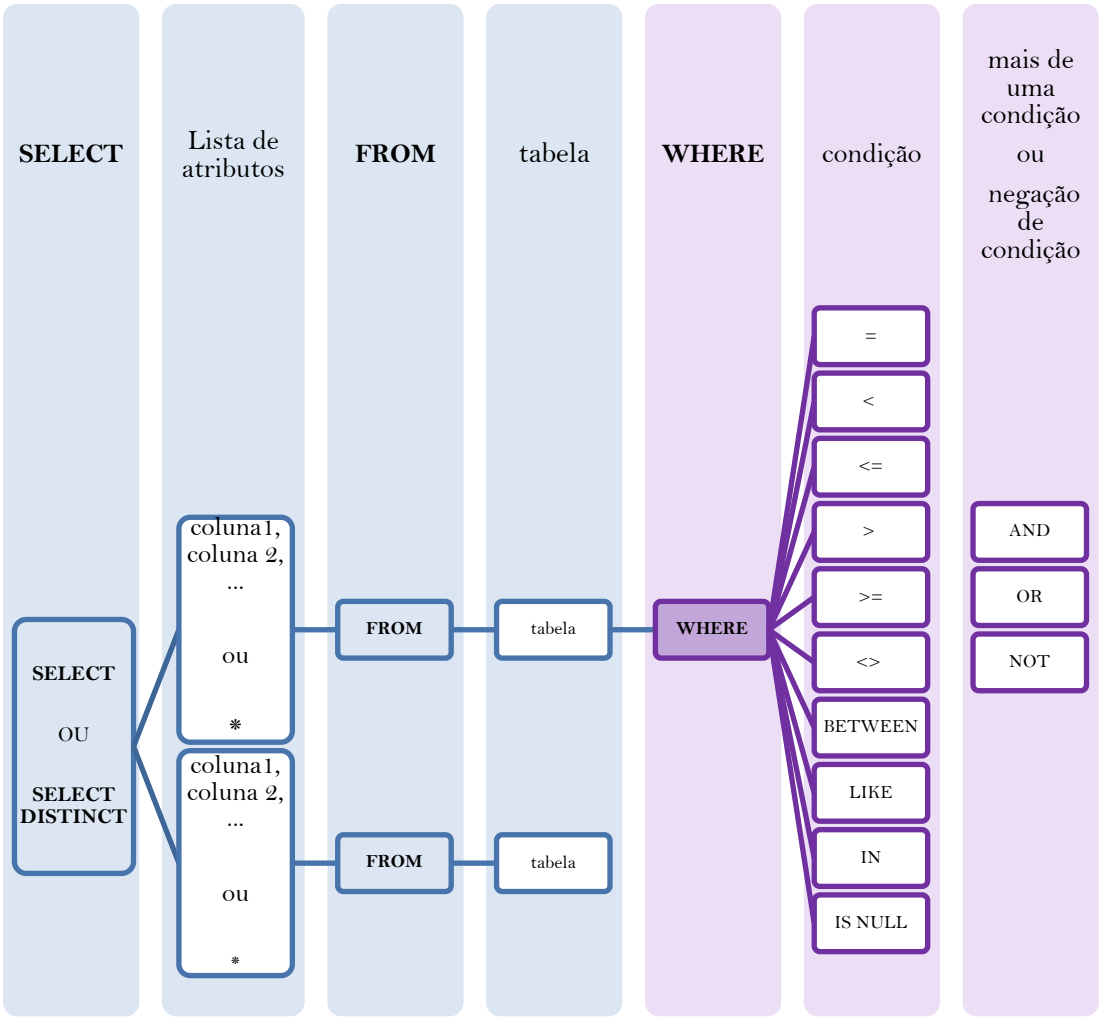
Passo 5 -> 3ª condição de que o preço seja menor que 2000:

AND Preço < 2000;

Gabarito: **Letra C.**

Esquema geral sobre a sintaxe básica do SELECT

Um esquema geral sobre a sintaxe básica da instrução SELECT é:



Esquema 6 – Instrução SELECT.

2.1.2 Definição de alias

Algumas vezes é interessante atribuir um **alias (apelido)** para uma tabela ou um atributo para facilitar a consulta ou mesmo para evitar ambiguidades. Sobre os alias:

- São usados para fornecer um **nome temporário a uma tabela ou coluna** em uma tabela.
- Costumam ser usados para **tornar os nomes das colunas mais legíveis**.
- Existe **apenas para a duração da consulta**.

A cláusula **AS** é usada para atribuição de uma alias, embora seja possível omití-la:

SELECT coluna1 **AS** nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

OU

SELECT coluna1 nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

SELECT coluna1, coluna2... **FROM** nome_da_tabela **AS** nova **WHERE** condição;

OU

SELECT coluna1, coluna2... **FROM** nome_da_tabela nova **WHERE** condição;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Para recuperar o nome dos clientes e o "Nome_Contato" como "Nome_do _Representante" podemos usar:

SELECT Nome_Cliente, Nome_Contato **AS** Nome_do_Representante **FROM** Clientes;

O retorno dessa consulta será:

Nome_Cliente	Nome_do_Representante
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno Taquería	Antonio Moreno

Perceba que o resultado apresenta o nome informado no alias para a coluna Nome_do_Representante e não o nome original na tabela.

ATENÇÃO!!!

Ao definir um alias em uma tabela, então pode-se usá-lo para fazer referência a tabela e seus atributos. Por exemplo, ao usar um alias p para produto, podemos referenciar um atributo com p.id_produto.

SELECT p.id_produto **FROM** produto **AS** p **WHERE** p.preco > 1000;

OU

SELECT p.id_produto **FROM** produto p **WHERE** p.preco > 1000;

Atribuição de alias

- Nome temporário a uma tabela ou coluna
- Tornar os nomes das colunas mais legíveis
- Existe apenas para a duração da consulta
- Cláusula AS (pode ser omitida)

Esquema 7 – Atribuição de alias.

16- **(FUNDATEC - 2023 – CAU-RS – Analista Superior)** Na linguagem SQL, um alias para uma coluna ou tabela pode ser criado utilizando a palavra-chave:

a) HOW

b) AS

c) FOR

d) ON

e) LIKE

Resolução:

A cláusula **AS** é usada para atribuição de uma alias, embora seja possível omití-la:

SELECT coluna1 **AS** nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

OU

SELECT coluna1 nova, coluna2 **FROM** nome_da_tabela **WHERE** condição;

SELECT coluna1, coluna2... **FROM** nome_da_tabela **AS** nova **WHERE** condição;

OU

SELECT coluna1, coluna2... **FROM** nome_da_tabela nova **WHERE** condição;

Gabarito: **Letra B**.

2.1.3 Ordenação com SELECT

A linguagem SQL permite que o usuário **ordene as tuplas no resultado de uma consulta** pelos valores de um ou mais atributos que aparecem, usando a cláusula **ORDER BY**.

A ordem padrão está em ordem crescente de valores. A palavra-chave **DESC** pode ser usada para ordenar os resultados em **ordem decrescente** de valores. A palavra-chave **ASC** pode ser usada para especificar a **ordem crescente** explicitamente.

A sintaxe básica para esse comando é:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição **ORDER BY** coluna **ASC/DESC**;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje consultar os clientes ordenados pelo país, exceto os clientes do Mexico. A seguinte consulta servirá a esse propósito:

SELECT * **FROM** Clientes **WHERE NOT** Pais='Mexico' **ORDER BY** Pais **ASC**; OU

SELECT * **FROM** Clientes **WHERE NOT** Pais='Mexico' **ORDER BY** Pais;

O retorno dessa consulta será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais ⇩
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

Se quisesse retornar em ordem decrescente, bastaria usar a consulta:

SELECT * **FROM** Clientes **WHERE NOT** Pais='Mexico' **ORDER BY** Pais **DESC**;

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais ⇧
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

É importante destacar que é possível ordenar por mais de uma coluna, bastando indicar as colunas e a ordem desejada. Assim, por exemplo, valem as seguintes sintaxes:

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição **ORDER BY** coluna1, coluna2 **ASC**;

SELECT coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição **ORDER BY** coluna1 **ASC** coluna2 **DESC**;

ATENÇÃO!!!

Uma sintaxe possível para a cláusula ORDER BY é a que indica o número da coluna ao invés de seu nome. Assim, não se assuste se você encontrar ORDER BY 1 ou coisa do tipo.

O número indica qual coluna da cláusula SELECT será usada para a ordenação. Assim, se for 1, será usada a primeira coluna, se for 2, a segunda, e, assim, sucessivamente.

EXEMPLIFICANDO!!!

Dado o comando a seguir:

SELECT cpf, nome **FROM** funcionario **WHERE** salario > 1000 **ORDER BY** 1 **ASC**;

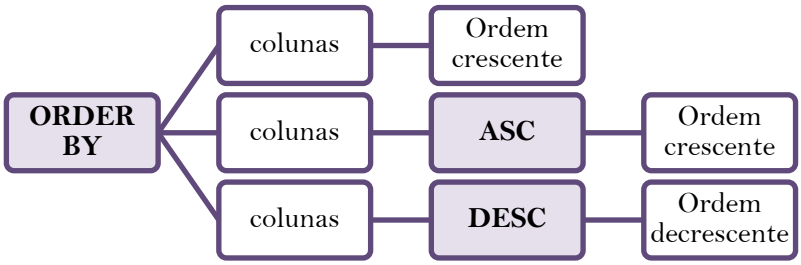
Haverá a ordenação com base no campo cpf, pois é o primeiro campo após o SELECT.

Já no comando a seguir:

SELECT cpf, nome **FROM** funcionario **WHERE** salario > 1000 **ORDER BY** 2 **ASC**;

Haverá a ordenação com base no campo nome, pois é o segundo campo após o SELECT.

Esquematizando:



Esquema 8 – Cláusula ORDER BY.

17- (VUNESP - 2023 – TJM SP – Técnico em Comunicação e Processamento de Dados Judiciário) Considere a seguinte tabela de um banco de dados relacional:

Passagem (ID, Origem, Destino, Valor)

O comando SQL para obter a Origem e Destino para valores situados entre 200,00 e 1.000,00, ordenado pela Origem, é:

a) SELECT Origem, Destino

FROM Passagem

HAVING Valor BETWEEN (200.00; 1000.00);

b) SELECT Origem, Destino

FROM Passagem

ORDER BY Origem AND

(Valor >= 200.00 OR Valor <= 1000.00);

c) SELECT Origem, Destino

FROM Passagem

WHERE Origem ASC AND

Valor BETWEEN (200.00 <> 1000.00);

d) SELECT Origem, Destino

FROM Passagem

WHERE Valor BETWEEN (200.00 AND 1000.00)

ORDER BY Origem;

e) SELECT Origem, Destino

FROM Passagem

HAVING Origem ASC AND

Valor BETWEEN (200.00 UNTIL 1000.00);

Resolução:

Queremos uma consulta que retorne a Origem e o Destino de passagens que possuem valores entre 200,00 e 1000,00, ordenado pela origem. Vamos montar esse comando passo a passo:

Passo 1 -> Retornar Origem e Destino:

```
SELECT Origem, Destino ...
```

Passo 2 -> Da tabela Passagem:

FROM Passagem ...

Passo 3 -> 1ª condição de valor entre 200 e 1000:

WHERE Valor BETWEEN 200 AND 1000 ...

Até aqui já conseguimos eliminar alguns itens, pois os a), b) e e) não possuem a cláusula WHERE. Um detalhe é que nos itens os valores estão utilizando 2 casas decimais com o separador ponto (200.00 e 1000.00). E outro detalhe é que os itens estão colocando as condições entre parênteses.

Passo 4 -> ordenado por origem:

ORDER BY origem

Gabarito: **Letra D**.

18- **(FCC - 2022 – TRT 14ª Região - Técnico Judiciário)** Considere a seguinte sintaxe SQL para obter todas as colunas da tabela Clientes, classificando a coluna Estado em ordem decrescente:

..**I**.. * FROM Clientes ..**II**.. Estado ...**III**... ;

As lacunas I, II e III devem ser preenchidas, respectivamente, por:

a) SELECT – CLASS – ORDER DECR

b) CLASS – ORDER BY – INV

c) SELECT – ORDER BY – DESC

d) OBTAIN – CLASS DESC – ORDER

e) SELECT CLASS – BY – DESC

Resolução:

Para classificar os clientes em ordem decrescente de Estado, a sintaxe será:

SELECT * **FROM** Clientes **ORDER BY** Estado **DESC;**

Gabarito: **Letra C.**

2.1.4 Funções de agregação

As **funções de agregação** são usadas para **resumir informações de várias tuplas em uma síntese de tupla única**. Existem diversas funções de agregação embutidas no SQL: **COUNT, SUM, MAX, MIN e AVG**.

A sintaxe básica para essas funções é:

> **SELECT FUNCAO**(coluna1) **FROM** nome_da_tabela **WHERE** condição; em que **FUNCAO** é qualquer uma das funções de agregação.

O quadro a seguir apresenta as definições dessas funções:

FUNÇÃO	RETORNO
MIN	Menor valor de uma coluna.
MAX	Maior valor de uma coluna.
COUNT	Número de linhas que atendem a um critério.
AVG	Média dos valores de uma coluna numérica.
SUM	Soma dos valores de uma coluna numérica.

Esquema 9 – Funções de agregação.

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4
5	Suco	2	2	Litros	8

Vamos supor que você deseje consultar a quantidade de produtos que existem, então poderá usar a seguinte consulta:

SELECT COUNT(ProdutoID) **FROM** Produtos;

O retorno dessa consulta será:

COUNT(ProdutoID)
5

Continuando o exemplo para mais algumas consultas com função de agregação:

- Consultar o menor preço:

```
SELECT MIN(Preco) FROM Produtos;
```

MIN(Preco)
3

- Consultar o maior preço:

```
SELECT MAX(Preco) FROM Produtos;
```

MAX(Preco)
8

- Consultar o preço médio:

```
SELECT AVG(Preco) FROM Produtos;
```

AVG(Preco)
5,2

Dada a tabela DetalheVendas:

DetalheVendasID	VendaID	ProdutoID	Quantidade
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

Para selecionar a quantidade vendida, podemos usar o agregador SUM:

```
SELECT SUM(Quantidade) FROM DetalheVendas;
```

O retorno será:

SUM(Quantidade)
76

ATENÇÃO!!!

A cláusula **COUNT** pode ser usada com o nome da coluna, * ou com 1:

- **COUNT(nome_da_coluna)**: retorna o número de linhas excluindo-se da contagem as linhas que possuem nulo para a coluna desejada.
- **COUNT(*) ou COUNT(1)**: retorna o número total de linhas, independentemente de valores nulos registrados para qualquer campo.

A cláusula **SUM** pode ser usada com o nome da coluna ou com um número indicativo da quantidade a ser somada:

- **SUM(nome_da_coluna)**: retorna o somatório dos valores presentes em nome_da_coluna.
- **SUM(1)**: retorna um somatório, sendo somado 1 para cada registro encontrado. Resultado similar a COUNT(*) ou COUNT(1), porém retorna NULL se não encontrar nenhum registro.
- **SUM(2)**: retorna um somatório, sendo somado 2 para cada registro encontrado.
- **SUM(N)**: retorna um somatório, sendo somado N para cada registro encontrado.

EXEMPLIFICANDO!!!

Dada a tabela DetalheVendas:

DetalheVendasID	VendaID	ProdutoID	Quantidade
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	NULL

Vejamos quais os resultados dos comandos:

SELECT COUNT(Quantidade) **FROM** DetalheVendas;

Resultado: 4 (total de linhas que não possui Quantidade com NULL).

SELECT COUNT(*) **FROM** DetalheVendas; ou **SELECT COUNT**(1) **FROM** DetalheVendas;

Resultado: 5 (total de linhas incluindo as que possuem valores NULL).

SELECT SUM(Quantidade) **FROM** DetalheVendas;

Resultado: 36 (soma dos valores da coluna quantidade).

SELECT SUM(1) **FROM** DetalheVendas;

Resultado: 5 (soma 1 para cada registro da tabela).

SELECT SUM(3) **FROM** DetalheVendas;

Resultado: 15 (soma 3 para cada registro da tabela).

19- (CESPE / CEBRASPE - 2024 –MPO - Analista de Planejamento e Orçamento) No que se refere à qualidade e visualização de dados, julgue o item a seguir.

Soma(SUM), Média(AVG), Máximo(MAX), Mínimo(MIN), Contagem(COUNT) e Agrupamento(GROUP BY) são exemplos de técnicas de agregação de dados.

Resolução:

As **funções de agregação** são usadas para **resumir informações de várias tuplas em uma síntese de tupla única**. Existem diversas funções de agregação embutidas no SQL: **COUNT, SUM, MAX, MIN e AVG**. O quadro a seguir apresenta as definições dessas funções:

FUNÇÃO	RETORNO
MIN	Menor valor de uma coluna.
MAX	Maior valor de uma coluna.
COUNT	Número de linhas que atendem a um critério.
AVG	Média dos valores de uma coluna numérica.
SUM	Soma dos valores de uma coluna numérica.

Gabarito: **Certo**.

20- (CESPE / CEBRASPE - 2023 – MPE-RO – Analista) Assinale a opção em que a instrução SQL apresentada, quando executada, permite mostrar a quantidade de contratos ativos de determinado órgão no ano de 2023.

a) SELECT SUM (*)

FROM contrato

WHERE situacao IN 'ATIVO'

AND ano= 2023;

b) SELECT AVG (*)

FROM contrato

WHERE situacao = 'ATIVO'

AND ano= 2023;

c) SELECT COUNT (*)

FROM contrato

WHERE situacao IN 'ATIVO'

OR ano= 2023;

d) SELECT MAX (*)

FROM contrato

WHERE situacao = 'ATIVO'

OR ano= 2023;

e) SELECT COUNT (*)

FROM contrato

WHERE situacao = 'ATIVO'

AND ano= 2023;

Resolução:

Queremos mostrar a quantidade de contratos ativos de determinado órgão no ano de 2023. Vamos montar o comando passo a passo:

Passo 1 -> Retornar quantidade de contratos:

SELECT COUNT(*) ...

Para retornar a quantidade de registros, podemos usar a função de agregação COUNT. Logo, COUNT(*) irá retornar a quantidade de linhas da consulta.

Aqui já conseguimos eliminar os itens a), b) e d) que trazem funções de agregação com finalidades diferentes. A) usa SUM, que seria para o somatório de valores, b) usa AVG que seria para a média, e d) usa MAX que seria para o valor máximo.

Passo 2 -> Da tabela contrato:

FROM contrato ...

Passo 3 -> 1ª condição deque a situação seja ativa:

WHERE situacao = 'ATIVO' ...

Queremos uma situação exata, então usamos o =.

Vale ressaltar que o IN até poderia ser utilizado, porém a sintaxe exige (), por isso o item c) não seria aplicável. Esse trecho no item c) seria correto se fosse WHERE situacao IN ('ATIVO').

Passo 4 -> 2ªcondição de que o ano seja 2023:

AND ano = 2023;

Nesse caso, queremos exatamente o ano 2023, por isso usamos o =.

Uma ressalva é que devemos usar o AND porque todas as condições devem ser atendidas conforme comando da questão. O OR presente não seria adequado, pois indicaria que apenas uma das condições poderia ser verdadeira.

Gabarito: **Letra E.**

21- (FCC - 2023 – TRT 12ª Região- Analista Judiciário) Considere uma tabela denominada Cidadao e as colunas Nome_Cidadao e Valor_Recebido.

Para obter a média dos valores maiores que 200 recebidos pelos cidadãos, um Analista deve utilizar o comando

a) SELECT AVG (Valor_Recebido > 200) FROM Cidadao;

b) SELECT AVG (Valor_Recebido) FROM Cidadao WHERE Valor_Recebido > 200;

c) SELECT AVG (Valor_Recebido) > 200 FROM Cidadao;

d) SELECT FROM Cidadao AVG (Valor_Recebido) > 200;

e) SELECT Nome Cidadao and AVG (Valor_Recebido) FROM Cidadao WHERE Valor_Recebido > 200;

Resolução:

Queremos obter a média dos valores maiores que 200 recebidos pelos cidadãos. Vamos montar o comando passo a passo:

Passo 1 -> Retornar a média de valores recebidos:

SELECT AVG(Valor_Recebido) ...

Para retornar a média de valores, usamos a função agregadora AVG. Logo, AVG(Valor_Recebido) irá retornar a média dos valores da coluna Valor_Recebido.

Passo 2 -> Da tabela Cidadao:

FROM Cidadao ...

Passo 3 -> condição de valor recebido maior que 200:

WHERE Valor_Recebido > 200;

Logo:

a) **Incorreto**: SELECT AVG (Valor_Recebido ~~> 200~~) FROM Cidadao;

A condição deve vir após o WHERE e não no SELECT.

b) **Correto**: SELECT AVG (Valor_Recebido) FROM Cidadao WHERE Valor_Recebido > 200 conforme comando que construímos passo a passo.

c) **Incorreto**: SELECT AVG (Valor_Recebido) ~~> 200~~ FROM Cidadao;

A condição deve vir após o WHERE e não no SELECT.

d) **Incorreto**: SELECT FROM Cidadao ~~AVG (Valor_Recebido) > 200~~;

A função AVG deve vir após o SELECT e a condição deve vir após o WHERE.

e) **Incorreto**: SELECT Nome_Cidadao and , AVG (Valor_Recebido) FROM Cidadao WHERE Valor_Recebido > 200; Esse seria correto com uso de , ao invés de and para separar os atributos no SELECT.

Gabarito: **Letra B.**

22- (FGV - 2015 - DPE-RO - Analista da Defensoria Pública - Analista Programador) Analise o comando SQL a seguir.

select max(A1) X, count(*) Y, sum(A1) Z from T

Executado quando a instância da tabela T estiver vazia (com zero registros), esse comando produz como resultado:

a)

X	Y	Z
NULL	0	NULL

b)

X	Y	Z
NULL	NULL	NULL

c)

X	Y	Z
0	0	0

d)

X	Y	Z
0	NULL	0

e)

X	Y	Z
NULL	NULL	0

Resolução:

Questão bem interessante e diferente. Basicamente ela quer saber qual o retorno das funções de agregação MAX, COUNT e SUM em uma tabela vazia.

As funções MAX(A1) e SUM(A1) retornarão **NULL**, pois o valor do número máximo ou soma dos registros não é conhecido. Não podemos dizer que é 0, pois nem mesmo temos registros para fazer qualquer comparativo ou somatório.

Já a função COUNT(*) retornará 0, pois o valor do número de registros, embora zerado, é conhecido.

Gabarito: **Letra A.**

2.1.5 Agrupamentos com SELECT

Geralmente queremos aplicar as funções de agregação a subgrupos de tuplas em uma relação, na qual os subgrupos são baseados em alguns valores de atributo. Cada grupo (partição) consistirá nas tuplas que possuem o mesmo valor de algum(ns) atributo(s), chamado(s) atributo(s) de agrupamento.

A linguagem SQL tem uma cláusula **GROUP BY** para **aplicar agrupamentos**. Esta cláusula especifica os atributos de agrupamento, que também devem aparecer na cláusula SELECT, de modo que o valor resultante da aplicação de cada função de agregação a um grupo de tuplas apareça junto com o valor do(s) atributo(s) de agrupamento.

A sintaxe básica da cláusula GROUP BY é:

SELECT colunas **FROM** nome_da_tabela **WHERE** condição **GROUP BY** coluna;

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4
5	Suco	2	2	Litros	8

Vamos supor que você deseje consultar a quantidade de produtos que são fornecidos por cada fornecedor, então poderá usar a seguinte consulta:

SELECT FornecedorID, **COUNT**(ProdutoID) **FROM** Produtos **GROUP BY** FornecedorID;

O retorno dessa consulta será:

FornecedorID	COUNT(ProdutoID)
1	3
2	2

A cláusula **HAVING** pode ser usada para **definir uma condição para um agrupamento** com GROUP BY. A sintaxe básica da cláusula GROUP BY com HAVING é:

SELECT colunas **FROM** nome_da_tabela **WHERE** condição **GROUP BY** coluna **HAVING** condição;

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4
5	Suco	2	2	Litros	8

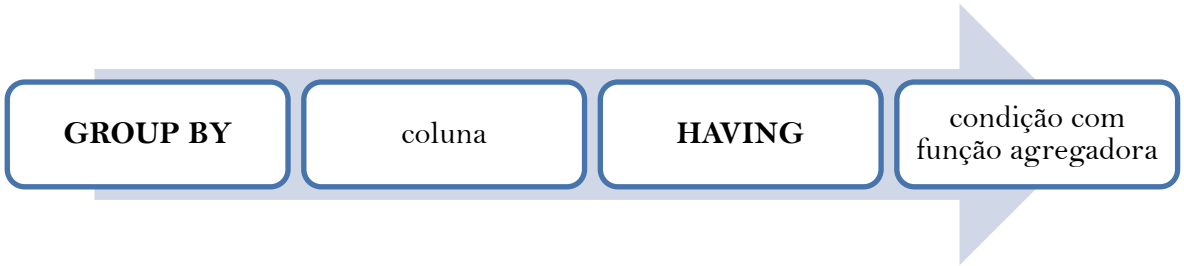
Vamos supor que você deseje consultar a quantidade de produtos que são fornecidos por cada fornecedor, porém somente aqueles que forneçam uma quantidade de produtos menor ou igual a 2, então poderá usar a seguinte consulta:

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

O retorno dessa consulta será:

FornecedorID	COUNT(ProdutoID)
2	2

Note que o fornecedor 1 não foi retornado na consulta, pois fornece 3 produtos.



Esquema 10 – Cláusula GROUP BY e HAVING.

23- (CESPE / CEBRASPE - 2024 –FINEP - Analista) Na tabela SQL criada pela expressão a seguir, sessao corresponde a uma sessão e a variável duracao corresponde à duração da sessão de certo usuário.

```
create table sessao (
  id integer primary key,
  userid integer not null,
  duracao decimal not null
)
```

A partir dessas informações, assinale a opção que corresponde ao script utilizado para se obter o tempo médio de duração das sessões dos usuários que tenham mais de uma sessão.

a) select userid, avg(duracao) from sessao group by userid having count(*)>1

b) select id, userid, avg(duracao) from sessao where count(*)>1

c) select userid, avg(duracao) from sessao having count(id)>1

d) select id, userid, avg(duracao) from sessao where count(*)>1 group by id, userid

e) select id, avg(duracao) from sessão group by id having count(*)>1

Resolução:

Queremos obter o tempo médio de duração das sessões dos usuários que tenham mais de uma sessão. Vamos montar o comando passo a passo para isso:

Passo 1 -> retornar o tempo médio:

```
SELECT userid, avg(duracao) ...
```

Para retornar a média de um atributo, usamos a função AVG. Logo, AVG(duração) trará a média de durações das sessões. Trouxemos também o userid, pois vamos precisar da duração média de cada usuário.

Passo 2 -> da tabela sessao:

```
FROM sessao ...
```

Passo 3 -> agrupado por usuário:

```
GROUP BY userid ...
```

A cláusula GROUP BY é usada para definir os grupos. Como queremos agrupar por usuário, então podemos usar o atributo userid (identificador de cada usuário).

Passo 4 -> filtrar grupos de usuários que tenham mais de uma sessão:

```
HAVING count(*)>1;
```

A assertiva solicitou que somente sejam retornados os grupos de usuários que possuam mais de uma sessão. Para isso, é necessário aplicar a cláusula HAVING para filtrar os grupos.

Vamos analisar os itens:

a) **Correto**: select userid, avg(duracao) from sessao group by userid having count(*)>1

A sintaxe está correta e realiza o que se pede.

b) **Incorreto**: select id, userid, avg(duracao) from sessao where ~~count(*)>1~~

Uma função de agregação como COUNT não deve ser utilizada diretamente na cláusula WHERE, devendo ser usada após o SELECT ou após o HAVING da cláusula GROUP BY.

c) **Incorreto**: select userid, avg(duracao) from sessao ??? having count(id)>1

Faltou a cláusula GROUP BY.

d) **Incorreto**: select id, userid, avg(duracao) from sessao where ~~count(*)>1~~ group by id, userid

Uma função de agregação como COUNT não deve ser utilizada diretamente na cláusula WHERE, devendo ser usada após o SELECT ou após o HAVING da cláusula GROUP BY.

e) **Incorreto**: select id, avg(duracao) from sessão group by id having count(*)>1

Embora a sintaxe aqui seja correta, usar o id como agrupador não realizaria o que a questão pede. A questão solicita o agrupamento por usuário e não por sessão. Id é o identificador da sessão. Além disso, é chave primária e, portanto, não se repete. Logo, nunca haverá um grupo que possua mais de um registro com mesmo id.

Gabarito: **Letra A.**

24- **(CESPE / CEBRASPE - 2023 – POLC-AL - Perito Criminal)** Com relação aos componentes de um computador, aos barramentos de E/S, à aritmética computacional e à linguagem SQL, julgue o próximo item.

Em SQL, para que não haja erro de construção (sintaxe), as cláusulas GROUP BY e HAVING, quando usadas, devem ser definidas sempre antes da cláusula WHERE.

Resolução:

As cláusulas GROUP BY e HAVING devem vir após a cláusula WHERE. A ordem correta segue a seguinte sintaxe:

SELECT colunas **FROM** nome_da_tabela **WHERE** condição **GROUP BY** coluna **HAVING** condição;

Gabarito: **Errado.**

25- (FGV - 2024 –TJ-MS – Técnico de Nível Superior) Observe script SQL a seguir.

SELECT COUNT(*) AS [Quantidade], Tipo_Processo

FROM Processo

GROUP BY Tipo_Processo;

O resultado da execução desse script é:

a) a lista dos registros da tabela quantidade;

b) a quantidade de processos por tipo;

c) a contagem dos registros da tabela de tipos de processos;

d) o agrupamento de processos que realizam contagem;

e) a contagem dos processos relacionados à quantidade de valores.

Resolução:

Vamos analisar o comando:

SELECT COUNT(*) AS [Quantidade], Tipo_Processo

-- Seleciona o número de registros (COUNT(*)) e o Tipo de Processo. Em AS [Quantidade] temos a definição de um alias para a contagem, ou seja, no resultado, o título da coluna será [Quantidade] ao invés de COUNT(*).

FROM Processo

-- Da tabela Processo.

GROUP BY Tipo_Processo;

-- Agrupando os resultados pelo Tipo de Processo.

Portanto, esse comando serve para exibir o número de processos agrupados por tipo.

Gabarito: **Letra B.**

2.1.6 Produto Cartesiano

O **Produto Cartesiano** seleciona **todos os pares de linhas das duas relações de entrada** (independentemente de ter ou não os mesmos valores em atributos comuns). A nova relação possui todos os atributos que compõem cada uma das relações que fazem parte da operação.

A quantidade de linhas resultantes de um produto cartesiano será exatamente o produto entre a quantidade de linhas das relações de entrada.

Em SQL, o produto cartesiano é indicado com o uso de vírgulas entre as tabelas desejadas.

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

Como temos mais de uma tabela sendo referenciada, é importante informar a qual tabela pertence um atributo. Assim, tabela1.coluna1 indica que desejamos obter o valor de coluna1 que está na tabela 1.

EXEMPLIFICANDO!!!

Dadas as tabelas Cliente e Pedido a seguir:

CLIENTE

CD_CLIENTE	NM_CLIENTE	DT_CADASTRO
1	Pedro da Silva	25/05/2009
2	João de Souza	28/05/2009

PEDIDO

NR_PEDIDO	CD_CLIENTE	QT_TOTAL	VL_TOTAL
10	1	20	34
20	2	50	100

O produto cartesiano destas tabelas será obtido pela seguinte sintaxe:

SELECT * FROM Cliente**,** Pedido**;**

O retorno dessa consulta será:

CD_CLIENTE	NM_CLIENTE	DT_CADASTRO	NR_PEDIDO	CD_CLIENTE	QT_TOTAL	VL_TOTAL
1	Pedro da Silva	25/05/2009	10	1	20	34
1	Pedro da Silva	25/05/2009	20	2	50	100
2	João de Souza	28/05/2009	10	1	20	34
2	João de Souza	28/05/2009	20	2	50	100

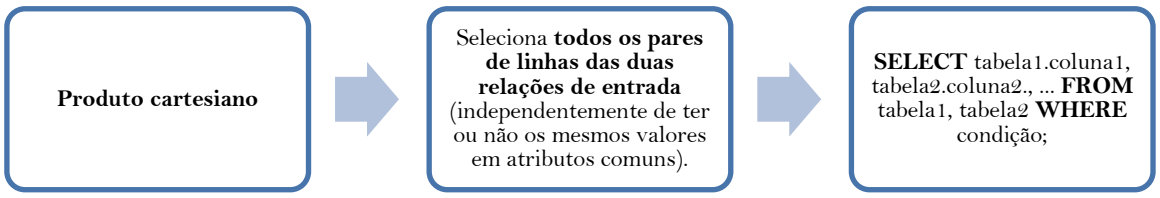
Note que foi retornada uma linha para cada cliente, relacionando-o com cada pedido, mesmo aqueles que não são deste cliente. Houve um cruzamento total entre as linhas das duas tabelas. Por exemplo, "Pedro da Silva" foi relacionado tanto com o pedido de nr 10 quanto com o pedido de nr 20, mesmo que o seu ID esteja relacionado apenas ao pedido 10.

DICA DO PROFESSOR!!!

A quantidade de linhas do resultado do produto cartesiano é dada pela multiplicação da quantidade de linhas das tabelas de entrada. Logo, se A possui 10 linhas e B possui 100 linhas, então SELECT * FROM A, B irá possuir 1000 linhas.

Porém, tome muito cuidado, pois pode haver alguma outra condição após o WHERE ou mesmo nas tabelas de entrada que altere essa quantidade de linhas do resultado.

Esquematicamente temos que:



Esquema 11 – Produto Cartesiano.

26- (CESPE / CEBRASPE - 2022 - TELEBRAS - Especialista em Gestão de Telecomunicações) Julgue o seguinte item, pertinentes a bancos de dados.

Conforme os conceitos de SQL (ANSI), em uma expressão SQL o produto cartesiano resulta que algumas linhas da primeira tabela são unidas a todas as linhas da segunda tabela.

Resolução:

O produto cartesiano resulta de TODAS (e não apenas algumas) as linhas da primeira com todas as linhas da segunda, ou seja, há um cruzamento completo das linhas das tabelas de entrada.

Gabarito: **Errado.**

27- (CESPE - 2019 - TJ-AM - Assistente Judiciário - Programador)

```
SELECT P.P#,
   P.COR,
   MAX (FP.QDE) AS QDEMAX
FROM P, FP
WHERE P.P# = FP.P#
AND (P.COR = COR ('Azul') OR P.COR = COR ('Rosa'))
AND FP.QDE > QDE (100)
GROUP BY P.P#, P.COR
HAVING SUM (FP.QDE) > QDE (250)
```

Considerando a formulação do algoritmo conceitual da consulta em SQL precedente, julgue o item a seguir.

A cláusula FROM é avaliada para produzir uma nova tabela, e essa nova tabela não é produzida a partir do produto cartesiano das tabelas P e FP.

Resolução:

A cláusula **FROM** é avaliada para verificar quais as tabelas em que os dados serão buscados.

O **Produto Cartesiano** seleciona **todos os pares de linhas das duas relações de entrada** (independentemente de ter ou não os mesmos valores em atributos comuns).

Nesse exemplo da questão, a cláusula **FROM P, FP** indica que a busca ocorrerá no produto cartesiano das tabelas P e FP.

Gabarito: **Errado.**

28- (FGV - 2015 - TJ-PI - Analista Judiciário - Analista de Sistemas / Desenvolvimento) Atenção:

Na questão a seguir, considere a tabela T mostrada abaixo com a respectiva instância.

T

a	b	c
1	2	NULL
2	3	NULL
5	NULL	NULL
4	2	NULL

O número de linhas produzidas, além da linha de títulos, pelo comando SQL

select *

from t t1, t t2, t t3

where t1.b is null

é:

a) 1

b) 4

c) 8

d) 16

e) 32

Resolução:

O comando apresenta o produto cartesiano de três cópias da mesma tabela, contudo com a condição de que b em t1 seja NULL. Para encontrar a quantidade de linhas de um produto cartesiano, temos que multiplicar o número de linhas das tabelas. Sendo assim,

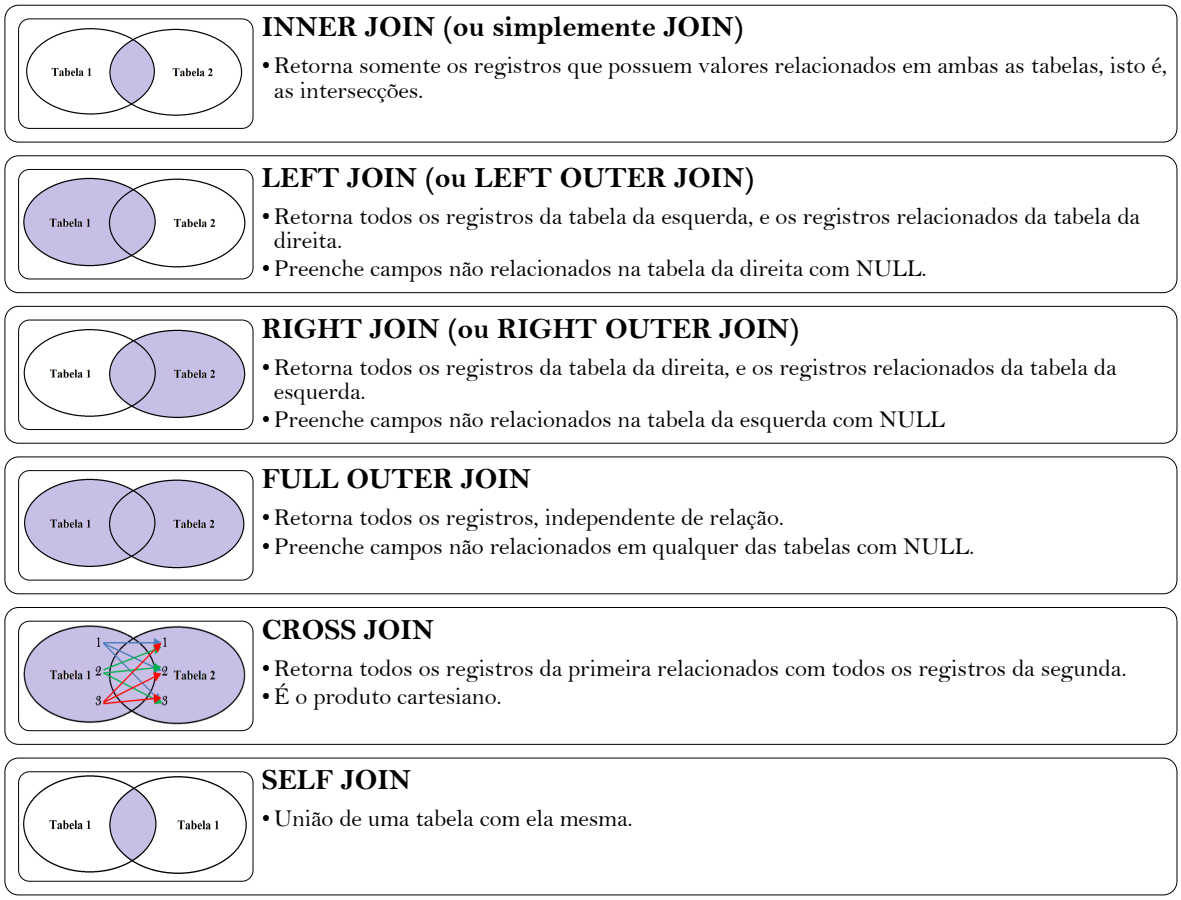
t1 (somente as linhas com b igual a NULL) x t2 x t3 = 1 x 4 x 4 = 16 linhas

Observe que além do produto cartesiano, há uma condição WHERE que limita a quantidade de linhas de t1 que formará o resultado.

Gabarito: **Letra D.**

2.1.7 Junções (joins)

As **tabelas de junção**, que **combinam duas ou mais tabelas baseando-se em colunas relacionadas**. As tabelas de junção são especificadas segundo a cláusula **JOIN**, que podem ser dos seguintes tipos:



Esquema 12 – Tipos de JOIN.

A sintaxe básica para consultas com junções é:

```
SELECT colunas FROM tabela1 JOIN tabela2 ON tabela1.coluna = tabela2.coluna;
```

Para o INNER JOIN, se as colunas em ambas as tabelas tiverem o mesmo nome, podemos usar a cláusula USING:

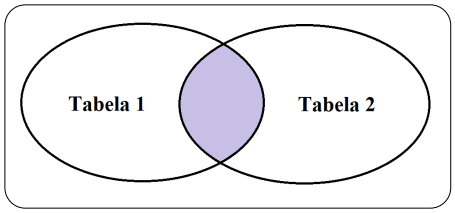
```
SELECT colunas FROM tabela1 INNER JOIN tabela2 USING (coluna);
```

ATENÇÃO!!!

O SQL ANSI especifica cinco tipos INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER e CROSS. Mas alguns SGBDS podem implementar outros tipos a exemplo do SELF JOIN.

INNER JOIN (ou simplesmente JOIN)

A cláusula **INNER JOIN** retorna somente os registros que possuem valores relacionados em ambas as tabelas, isto é, as intersecções.



EXEMPLIFICANDO!!!

Dada as tabelas Pessoas e Veículos a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

Vamos supor que você deseje identificar todas as pessoas que possuem veículos e exibir essa relação em uma única tabela de junção. Para isso, poderá usar a consulta a seguir:

SELECT * **FROM** PESSOAS **INNER JOIN** VEICULOS **ON** PESSOAS.CPF = VEICULOS.CPF; OU

SELECT * **FROM** PESSOAS **JOIN** VEICULOS **ON** PESSOAS.CPF = VEICULOS.CPF;

O retorno dessa consulta será:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001

Note que somente foram retornados os dados referentes as pessoas que possuem veículos, isto é, que possuem CPF nas duas tabelas. Neste exemplo, é possível notar que somente Fernando possui um veículo relacionado, que é o Carro de Placa SB-0001.

Como as duas tabelas possuem os campos a serem relacionados com o mesmo nome, então as sintaxes a seguir podem ser utilizadas:

SELECT * **FROM** PESSOAS **INNER JOIN** VEICULOS **USING**(CPF); OU

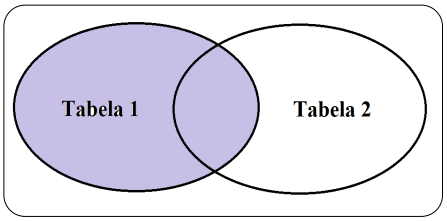
SELECT * **FROM** PESSOAS **JOIN** VEICULOS **USING**(CPF);

O resultado será exibido sem a repetição do campo CPF:

NOME	CPF	ESTADO	VEICULO	PLACA
Fernando	111.111. 111-11	PR	Carro	SB-0001

LEFT JOIN (ou LEFT OUTER JOIN)

A cláusula **LEFT JOIN** retorna todos os registros da tabela da esquerda, e os registros relacionados da tabela da direita. Caso não haja valores relacionados na tabela da direita, os seus campos serão preenchidos com NULL.



EXEMPLIFICANDO!!!

Dada as tabelas Pessoas e Veículos a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

Vamos supor que você deseje identificar todas as pessoas e saber se elas posuem ou não veículos. Caso elas possuam, você deseja identificar qual o veículo que elas são proprietárias. Para isso, poderá usar a consulta a seguir:

```
SELECT * FROM PESSOAS LEFT JOIN VEICULOS ON PESSOAS.CPF = VEICULOS.CPF;
```

O retorno dessa consulta será:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL

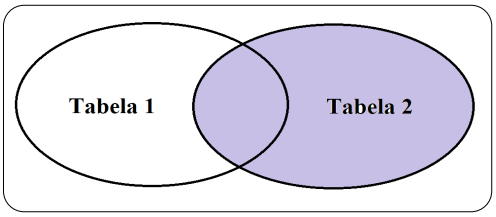
Note que foram retornadas todas as pessoas e, para as que possuem veículos, as informações dos veículos foram informadas nas colunas da direita.

Para aquelas que não possuem veículos, os dados referentes a estes foram preenchidos com NULL.

Neste exemplo, Fernando possui o veículo de Placa SB-0001, Guilherme não possui nenhum veículo.

RIGHT JOIN (ou RIGHT OUTER JOIN)

A cláusula **RIGHT JOIN** retorna todos os registros da tabela da direita, e os registros relacionados da tabela da esquerda. Caso não haja valores relacionados na tabela da esquerda, os seus campos serão preenchidos com NULL.



EXEMPLIFICANDO!!!

Dada as tabelas Pessoas e Veículos a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

Vamos supor que você deseje identificar todos os veículos e saber se eles posuem ou não proprietários. Caso eles possuam, você deseja identificar qual o proprietário deles. Para isso, poderá usar a consulta a seguir:

SELECT * **FROM** PESSOAS **RIGHT JOIN** VEICULOS **ON** PESSOAS.CPF = VEICULOS.CPF;

O retorno dessa consulta será:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
NULL	NULL	NULL	NULL	Carro	SB-0002

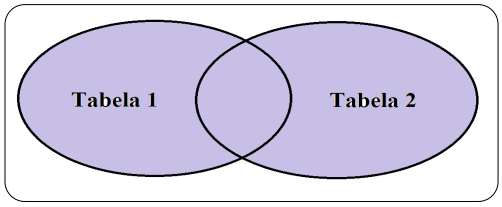
Note que foram retornados todos os veículos e, para as que possuem proprietários, as informações destes foram informadas nas colunas da esquerda.

Para aquelas que não possuem proprietários, os dados referentes a estes foram preenchidos com NULL.

Neste exemplo, o veículo de Placa SB-0001 está relacionado a Fernando, e veículo de Placa SB-0002 não possui pessoa relacionada.

FULL OUTER JOIN

A cláusula **FULL OUTER JOIN** retorna todos os registros, relacionando aqueles que tiverem relação. Caso não haja valores relacionados na tabela da esquerda ou da direita, os seus campos serão preenchidos com NULL.



EXEMPLIFICANDO!!!

Dada as tabelas Pessoas e Veículos a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

Vamos supor que você deseje identificar todos os proprietários e veículos e as relações entre eles, caso haja. Para isso, poderá usar a consulta a seguir:

SELECT * **FROM** PESSOAS **FULL OUTER JOIN** VEICULOS **ON** PESSOAS.CPF = VEICULOS.CPF;

O retorno dessa consulta será:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
NULL	NULL	NULL	NULL	Carro	SB-0002

Note que foram retornados todas as pessoas e todos os veículos.

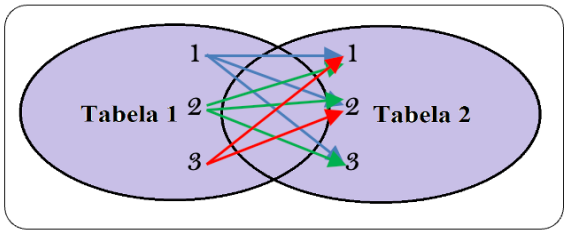
Quando há uma relação entre uma pessoa e um veículo, esta é indicada no mesmo registro.

Quando não há relação, os valores são preenchidos com NULL.

Neste exemplo, Fernando possui o veículo de Placa SB-0001, Guilherme não possui veículo, e o veículo de Placa SB-002 não possui proprietário.

CROSS JOIN

O **CROSS JOIN** realiza o produto cartesiano das tabelas, isto é, retorna **todos os pares de linhas das duas relações de entrada** (independentemente de ter ou não os mesmos valores em atributos comuns). A nova relação possui todos os atributos que compõem cada uma das relações que fazem parte da operação.



ATENÇÃO!!!

Muito cuidado para não pensar que CROSS JOIN (produto cartesiano) é similar ao FULL OUTER JOIN. Para explicar a diferença, vamos usar um exemplo.

Dada as tabelas Pessoas e Veículos a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

O CROSS JOIN (produto cartesiano) trará o seguinte resultado:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Fernando	111.111.111-11	PR	NULL	Carro	SB-0002
Guilherme	222.222.222-22	SP	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	Carro	SB-0002

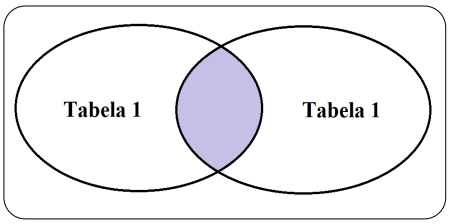
Já o FULL OUTER JOIN trará o seguinte resultado:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
NULL	NULL	NULL	NULL	Carro	SB-0002

Observe que no CROSS JOIN, há duas linhas para Fernando, pois há uma linha para cada veículo, mesmo para o veículo que não é dele. Já no FULL OUTER JOIN, há apenas uma linha, pois as relações são consideradas e, portanto, Fernando só é associado ao seu veículo.

SELF JOIN

A cláusula **SELF JOIN** é similar a um JOIN, contudo relaciona uma tabela com ela mesma.



EXEMPLIFICANDO!!!

Dada a tabela Pessoas a seguir:

PESSOAS

NOME	CPF	ESTADO	INDICADO
Fernando	111.111.111-11	PR	NULL
Guilherme	222.222.222-22	SC	111.111.111-11

Vamos supor que você deseje identificar as pessoas que foram indicadas por outras. Para isso, poderá usar a consulta a seguir:

SELECT A.NOME, B.NOME **AS** INDICADO_POR **FROM** PESSOAS A **JOIN** PESSOAS B **ON** A.INDICADO = B.CPF**;**

O retorno dessa consulta será:

NOME	INDICADO_POR
GUILHERME	FERNANDO

Note que somente foram retornados os dados referentes as pessoas que foram indicadas por alguém.

Neste exemplo, Guilherme foi indicado por Fernando.

DICA DO PROFESSOR!!!

CONTAGEM DE LINHAS NOS RESULTADOS DE CONSULTAS COM JOINS

Um tipo de questão relativamente comum é o que solicita a contagem de linhas em comandos SQL. Para os comandos em geral, você precisará avaliar o resultado, mas existem alguns bizus para avaliar a quantidade de linhas dos comandos com junções. São elas:

- **INNER JOIN**: exato número de linhas que possuem correspondência.
- **LEFT JOIN**: no mínimo o número de linhas da tabela da esquerda. Linhas adicionais aparecem se houver correspondências múltiplas na tabela da direita.
- **RIGHT JOIN**: no mínimo o número de linhas da tabela da direita. Linhas adicionais aparecem se houver correspondências múltiplas na tabela da esquerda.
- **FULL OUTER JOIN**: no mínimo o número de linhas da maior tabela e no máximo a soma total de linhas das tabelas. Pode-se usar a soma das linhas das tabelas menos o número de linhas relacionadas.
- **CROSS JOIN**: multiplicação do número de linhas das tabelas envolvidas.

***IMPORTANTE**: qualquer condição adicional no **WHERE** ou nos filtros aplicados às tabelas pode alterar a contagem de linhas.

Vamos analisar um exemplo com base nas tabelas a seguir:

PESSOAS

NOME	CPF	ESTADO
Fernando	111.111.111-11	PR
Guilherme	222.222.222-22	SC
Maria	333.333.333-33	CE

VEICULOS

CPF	VEICULO	PLACA
111.111.111-11	Carro	SB-0001
NULL	Carro	SB-0002

Os retornos para cada um dos tipos de JOIN será:

INNER JOIN: 1 linha (apenas a linha que tem correspondência)

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001

LEFT JOIN: 3 linhas (no mínimo o número de linhas da tabela da esquerda).

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
Maria	333.333.333-22	CE	NULL	NULL	NULL

CONTINUAÇÃO...

Mas como assim "no mínimo"? Isso mesmo, pois pode acontecer de haver mais de um registro relacionado. Imagine que exista um terceiro veículo com valores 111.111.111-11, Carro, SB-0003. Nesse caso, o resultado do LEFT JOIN teria 4 linhas, sendo duas linhas para Fernando e seus respectivos veículos:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0002
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
Maria	333.333.333-22	CE	NULL	NULL	NULL

RIGHT JOIN: 2 linhas (no mínimo o número de linhas da tabela da direita).

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
NULL	NULL	NULL	NULL	Carro	SB-0002

O mesmo raciocínio do "no mínimo" serve também para o Right Join.

FULL OUTER JOIN: 4 linhas (entre 3, que é o número de linhas da maior tabela e 5, que é o somatório de linhas das duas tabelas) (5-1, somatório de linhas – linhas relacionadas).

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
Maria	333.333.333-22	CE	NULL	NULL	NULL
NULL	NULL	NULL	NULL	Carro	SB-0002

No caso do FULL OUTER JOIN, temos um intervalo de números de linhas possíveis. Vamos avaliar os extremos casos para entender esse intervalo.

Extremo 1: todos os registros relacionados. Nesse caso, teremos o menor número de linhas possível que é igual ao número de linhas da maior tabela envolvida. No exemplo, temos uma tabela com 3 linhas e outra com 2 linhas, logo teremos 3 linhas no resultado. Por exemplo, se o Carro de Placa SB-0002 fosse de Maria (CPF = 333.333.333-33), o resultado seria:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
Maria	333.333.333-22	CE	NULL	NULL	NULL

CONTINUAÇÃO...

Extremo 2: nenhum registro relacionado. Nesse caso, teremos o maior número de linhas possível que é igual ao somatório da quantidade de linhas das tabelas envolvidas. No exemplo, temos uma tabela com 3 linhas e outra com 2 linhas, logo teremos 3+2 = 5 linhas no resultado. Por exemplo, se o Carro de Placa SB-0001 tivesse o CPF NULL, teríamos:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	NULL	NULL	NULL
Guilherme	222.222.222-22	SP	NULL	NULL	NULL
Maria	333.333.333-22	CE	NULL	NULL	NULL
NULL	NULL	NULL	NULL	Carro	SB-0001
NULL	NULL	NULL	NULL	Carro	SB-0002

CROSS JOIN: 6 linhas (3x2, multiplicação do número de linhas das tabelas envolvidas):

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Fernando	111.111.111-11	PR	NULL	Carro	SB-0002
Guilherme	222.222.222-22	SP	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	Carro	SB-0002
Maria	333.333.333-22	CE	111.111.111-11	Carro	SB-0001
Maria	333.333.333-22	CE	NULL	Carro	SB-0002

Essas regrinhas irão te ajudar na resolução de questões para contagem de linhas com junções, mas lembre-se sempre de avaliar as outras condições. Por exemplo, o seguinte comando terá como resultado 4 linhas e não 6:

```
SELECT * FROM PESSOAS CROSS JOIN VEICULOS WHERE PESSOA.ESTADO <> 'CE';
```

Nesse comando, perceba que houve um filtro na tabela PESSOA, que seleciona apenas as pessoas que não sejam do Estado 'CE, logo, Maria não será considerada no CROSS JOIN e o resultado será:

NOME	CPF	ESTADO	CPF	VEICULO	PLACA
Fernando	111.111.111-11	PR	111.111.111-11	Carro	SB-0001
Fernando	111.111.111-11	PR	NULL	Carro	SB-0002
Guilherme	222.222.222-22	SP	111.111.111-11	Carro	SB-0001
Guilherme	222.222.222-22	SP	NULL	Carro	SB-0002

29- **(FGV - 2024 – CGE-PB – Auditor de Contas Públicas)** Observe as tabelas A e B a seguir, que possuem, respectivamente, 9 e 10 registros.

Tabela A	
ID	**DESCRICAO**
1	Descrição1
2	Descrição2
3	Descrição3
4	Descrição4
5	Descrição5
6	Descrição6
7	Descrição7
8	Descrição8
10	Descrição10

Total de Registros: 9

Tabela B	
ID	**VALOR**
1	10
2	20
3	30
5	50
6	60
7	70
9	90
11	110
12	120
13	130

Total de Registros: 10

Após executar diferentes tipos de junções entre essas tabelas, o total de registros retornados em cada caso, sendo eles INNER JOIN, RIGHT JOIN, FULL OUTER JOIN, CROSS JOIN e LEFT JOIN, é, respectivamente:

a) 9, 10, 13, 90 e 9;

b) 6, 10, 13, 90 e 9;

c) 6, 10, 10, 90 e 9;

d) 6, 10, 13, 19 e 9;

e) 6, 12, 13, 90 e 9.

Resolução:

A contagem de linhas do resultado de junções pode ser feita com base nas seguintes regras:

- **INNER JOIN**: exato número de linhas que possuem correspondência.

  Na questão são **6 linhas**, pois é exatamente o número de linhas relacionadas pelo ID nas tabelas, quais sejam IDs 1, 2, 3, 5, 6 e 7.

- **LEFT JOIN**: no mínimo o número de linhas da tabela da esquerda. Linhas adicionais aparecem se houver correspondências múltiplas na tabela da direita.

  Na questão são **9 linhas**, que é o número de linhas da tabela da esquerda. Como não há nenhum ID repetido na tabela da direita, então não há mais de uma linha para um mesmo valor da tabela da esquerda.

- **RIGHT JOIN**: no mínimo o número de linhas da tabela da direita. Linhas adicionais aparecem se houver correspondências múltiplas na tabela da esquerda.

  Na questão são **10 linhas**, que é o número de linhas da tabela da direita. Como não há nenhum ID repetido na tabela da esquerda, então não há mais de uma linha para um mesmo valor da tabela da direita.

- **FULL OUTER JOIN**: no mínimo o número de linhas da maior tabela e no máximo a soma total de linhas das tabelas. Pode-se usar a soma das linhas das tabelas menos o número de linhas relacionadas.

  Logo, temos 19 linhas no total, sendo 6 relacionadas. Portanto, **13 linhas**.

- **CROSS JOIN**: multiplicação do número de linhas das tabelas envolvidas.

  Na questão são **90 linhas**, 10 x 9.

Na ordem desejada: **6** (INNER), **10** (RIGHT), **13** (FULL), **90** (CROSS) e **9** (LEFT).

Gabarito: **Letra B.**

30- **(FGV - 2024 –TJ-MS – Técnico de Nível Superior)** João está escrevendo uma consulta que envolve várias tabelas e precisa garantir que todas as suas linhas sejam incluídas no resultado, mesmo que não haja correspondências entre elas.

Para tanto, João deverá utilizar o seguinte operador de junção:

a) LEFT JOIN;

b) INNER JOIN;

c) RIGHT JOIN;

d) CROSS JOIN;

e) FULL OUTER JOIN.

Resolução:

Vamos analisar cada um dos itens:

a) **Incorreto**: **LEFT JOIN** retorna todos os registros da tabela da esquerda, e os registros relacionados da tabela da direita. Logo, não garante que todas as linhas serão incluídas no resultado, pois não trará os registros da tabela da direita que não tiverem relação com os registros da tabela da esquerda.

b) **Incorreto**: **INNER JOIN** retorna somente os registros relacionados, logo os registros de ambas as tabelas que não tiverem relação não serão retornados.

c) **Incorreto**: **RIGHT JOIN** retorna todos os registros da tabela da direita, e os registros relacionados da tabela da esquerda. Logo, não garante que todas as linhas serão incluídas no resultado, pois não trará os registros da tabela da esquerda que não tiverem relação com os registros da tabela da direita.

d) **Incorreto**: **CROSS JOIN** retorna o produto cartesiano, ou seja, todas as combinacões de linhas das tabelas de entrada. Embora esse comando garanta que todas as linhas sejam retornadas, ele não é o mais adequado para o objetivo da questão, pois não exibirá as relações entre as linhas quando elas existirem, trazendo um simples cruzamento de todas as linhas.

e) **Correto**: **FULL OUTER JOIN** retorna todos os registros das tabelas independente de relação, trazendo na mesma linha aqueles que tiverem relação.

Gabarito: **Letra E.**

31- **(CESPE / CEBRASPE - 2023 – DATAPREV - Analista de Processamento)** Em relação às linguagens de banco de dados SQL, DDL e DML, julgue o item a seguir.

Somente são possíveis os seguintes quatro tipos de JOIN em SQL, segundo o padrão ANSI: INNER JOIN, LEFT JOIN, RIGHT JOIN e CROSS JOIN.

Resolução:

O SQL ANSI define também o FULL OUTER JOIN. Ou seja, são cinco tipos.

Gabarito: **Errado.**

32- **(CESPE / CEBRASPE - 2022 - TRT 8ª Região - Técnico Judiciário)** Considere-se que as tabelas Produto e Categoria, a seguir, tenham sido implementadas em um banco de dados SQL.

Produto

idProduto	DeProduto	idCategoria	ValorProduto
1	Arroz	2	9
2	Feijão	2	9
3	Detergente	1	7
4	Sabão	1	7
5	Escova	1	7

Categoria

idCategoria	DeCategoria
1	Limpeza
2	Alimentos

Considere-se, ainda, que o script SQL a seguir tenha sido executado no Postgres12.

```
SELECT C.DeCategoria, AVG(P.ValorProduto) total
FROM Produto P
LEFT OUTER JOIN Categoria C
ON P.idCategoria = C.idCategoria
GROUP BY P.idCategoria
HAVING total > 7;
```

Assinale a opção que contenha a tabela com o resultado correto do script supracitado.

a)

DeCategoria	total
Alimentos	9

b)

DeCategoria	total
Alimentos	9
Limpeza	7

c)

DeCategoria	total
Alimentos	18
Limpeza	21

d)

DeCategoria	total
Alimentos	18

e)

DeCategoria	total
Limpeza	21
Alimentos	18

Resolução:

Vamos analisar o comando passo a passo:

```
SELECT C.DeCategoria, AVG(P.ValorProduto) total
-- selecionar os atributos DeCategoria da tabela C e a média do ValorProduto da tabela P.
FROM Produto P LEFT OUTER JOIN Categoria C
-- a partir do LEFT JOIN das tabelas Produto e Categoria.
ON P.idCategoria = C.idCategoria
-- com base na condição de igualdade entre os idCategoria.
GROUP BY P.idCategoria
-- e agrupar os resultados por idCategoria.
HAVING total > 7;
-- desde que o total para o grupo seja maior que 7
```

Vamos agora entender a execução do comando:

1. O comando combina as tabelas Produto e Categoria usando a condição P.idCategoria = C.idCategoria. Como é um LEFT OUTER JOIN, todas as linhas da tabela Produto são mantidas, mesmo que não tenham correspondência na tabela Categoria. A tabela resultante do JOIN é:

idProduto	DeProduto	idCategoria	ValorProduto	idCategoria	DeCategoria
1	Arroz	2	9	2	Alimentos
2	Feijão	2	9	2	Alimentos
3	Detergente	1	7	1	Limpeza
4	Sabão	1	7	1	Limpeza
5	Escova	1	7	1	Limpeza

2. A cláusula GROUP BY P.idCategoria agrupa os produtos por idCategoria. Em cada grupo, a função AVG(P.ValorProduto) é usada para calcular a média dos valores dos produtos. O resultado desse agrupamento é:

idCategoria	DeCategoria	AVG(P.valorProduto)
1	Limpeza	7
2	Alimentos	9

3. A cláusula HAVING total > 7 filtra os grupos onde a média dos valores dos produtos é maior que 7. Nesse caso:

- O grupo "Limpeza" (média = 7) é eliminado, pois não atende à condição.
- O grupo "Alimentos" (média = 9) é mantido.

idCategoria	DeCategoria	AVG(P.valorProduto)
1	Alimentos	9

4. A consulta retorna as categorias que passaram pelo filtro HAVING, junto com a média dos valores dos produtos.

DeCategoria	total
Alimentos	9

Gabarito: **Letra A.**

2.1.8 Operadores de conjuntos

O SQL permite trabalhar no resultado de duas consultas através dos operadores UNION (e UNION ALL), INTERSECT e EXCEPT.

UNION e UNION ALL

O operador **UNION** **combina os resultados de duas ou mais consultas**, retornando todas as linhas pertencentes a todas as consultas envolvidas na execução. Para utilizar o UNION, o número e a ordem das colunas precisam ser idênticos em todas as consultas e os tipos de dados precisam ser compatíveis.

Existem dois tipos de operador UNION, sendo eles UNION e UNION ALL.

- O operador **UNION**, por padrão, executa o **equivalente a um SELECT DISTINCT**. Em outras palavras, ele combina o resultado de execução das duas consultas e então executa um SELECT DISTINCT a fim de **eliminar as linhas duplicadas**. Este processo é executado mesmo que não haja registros duplicados.

  A sintaxe básica é:

  SELECT colunas **FROM** tabela1

  UNION

  SELECT colunas **FROM** tabela2;

- O operador **UNION ALL** tem a mesma funcionalidade do UNION, porém, **não executa o SELECT DISTINCT no resultado** e apresenta todas as linhas, inclusive as linhas duplicadas.
  A sintaxe básica é:

  SELECT colunas **FROM** tabela1

  UNION ALL

  SELECT colunas **FROM** tabela2;

ATENÇÃO!!!

As colunas de ambas as consultas não precisam ser exatamente as mesmas. Elas só precisam aparecer na mesma quantidade e ordem. E cada coluna deve ter o mesmo tipo de dado.

Assim, na primeira consulta podemos ter uma coluna de cadeia de caracteres chamada "nome_cliente" e na segunda consulta podemos ter uma coluna de cadeia de caracteres chamada "nome_fornecedor". Ainda assim, será possível realizar a união, pois elas possuem o mesmo tipo de dados e estão na mesma ordem no conjunto de colunas de cada subconsulta.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

E a tabela Fornecedores a seguir:

IDFornecedor	Nome_Fornecedor	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

A consulta a seguir irá retornar as cidades de ambas as tabelas, comuns ou não:

SELECT Cidade **FROM** Clientes **UNION SELECT** Cidade **FROM** Fornecedores;

Cidade	
Berlin	Linhas da 1ª consulta, excluindo a repetição de México D.F.
México D.F.	
London	Linhas da 2ª consulta
New Orleans	
Ann Arbor	

Como a cláusula UNION realiza um SELECT DISTINCT implícito, então México D.F não foi retornado duas vezes. Se ao invés de UNION, tivessemos usado UNION ALL, o resultado seria:

Cidade	
Berlin	Linhas da 1ª consulta, incluindo a repetição de México D.F.
México D.F.	
México D.F.	
London	Linhas da 2ª consulta
New Orleans	
Ann Arbor	

INTERSECT

O operador **INTERSECT** **permite a intersecção entre consultas**, retornando as linhas que existem tanto na primeira quanto na segunda consulta. INTERSECT possui um SELECT DISTINCT implícito, logo **não retorna linhas repetidas**.

A sintaxe básica é:

```
SELECT colunas FROM tabela1
INTERSECT
SELECT colunas FROM tabela2;
```

EXEMPLIFICANDO!!!

Dada a tabela ClientesA a seguir:

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

E a tabela ClientesB a seguir:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	England
3	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

A consulta a seguir irá retornar as cidades que estão presentes nas duas tabelas ao mesmo tempo:

```
SELECT Cidade FROM ClientesA INTERSECT SELECT Cidade FROM ClientesB ORDER BY Cidade;
```

Cidade
Berlin

Linhas que estão em ambas

Somente as cidades que são retornadas nas duas consultas vão no resultado. No caso, somente a cidade de Berlin.

EXCEPT

O operador **EXCEPT** **retorna os registros que aparecem na primeira consulta e não aparecem na segunda**. EXCEPT possui um SELECT DISTINCT implícito, logo **não retorna linhas repetidas**.

A sintaxe básica é:

```
SELECT colunas FROM tabela1
EXCEPT
SELECT colunas FROM tabela2;
```

EXEMPLIFICANDO!!!

Dada a tabela ClientesA a seguir:

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

E a tabela ClientesB a seguir:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	England
3	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

A consulta a seguir irá retornar as cidades que estão presentes na primeira consulta, mas não na segunda:

```
SELECT Cidade FROM ClientesA EXCEPT SELECT Cidade FROM ClientesB ORDER BY Cidade;
```

Cidade
México D.F.

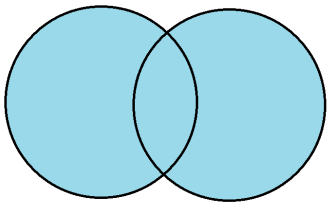
Linha somente na 1ª consulta

Perceba que não foi retornada uma linha para Belin, pois ela está também na segunda consulta.

As operações de união, intersecção e diferença podem ser representadas em diagramas:

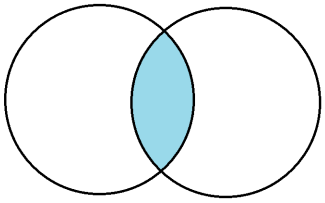
União

A U B



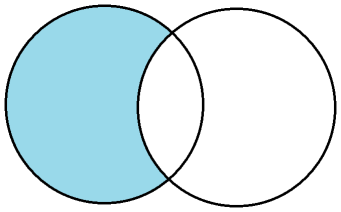
Intersecção

A ∩ B



Diferença

A - B



O quadro a seguir apresenta um resumo desses operadores:

OPERADOR	RETORNO
UNION	Todas as linhas pertencentes as consultas envolvidas, sem as repetições.
UNION ALL	Todas as linhas pertencentes as consultas envolvidas, incluindo as repetições.
INTERSECT	Linhas que estão tanto na primeira quanto na segunda consulta. Intersecção, sem repetições.
EXCEPT	Linhas que estão na primeira, mas não estão na segunda, sem repetições.

Esquema 13 – Operadores de conjuntos.

33- (CESPE / CEBRASPE - 2023 – DATAPREV - Analista de Tecnologia da Informação) Com relação a SQL, julgue o item a seguir.

O comando UNION é utilizado para combinar as linhas de duas tabelas, mesmo que as colunas dessas tabelas sejam de tipos e tamanhos diferentes.

Resolução:

Embora as colunas não precisem ser exatamente as mesmas nas consultas com os operadores de conjuntos (UNION, INTERSECT, EXCEPT), elas precisam aparecer na mesma quantidade, na mesma ordem e serem do mesmo tipo.

Por exemplo, na primeira consulta podemos ter uma coluna de cadeia de caracteres chamada "nome_cliente" e na segunda consulta podemos ter uma coluna de cadeia de caracteres chamada "nome_fornecedor". Ainda assim, será possível realizar a união, pois elas possuem o mesmo tipo de dados e estão na mesma ordem no conjunto de colunas de cada subconsulta. Agora, se a primeira coluna da primeira consulta for do tipo data, por exemplo, "data_nascimento" e a primeira coluna da segunda consulta for do tipo textual, por exemplo, "mês do nascimento", então não será possível realizar a união dessa forma.

Gabarito: **Errado.**

34- (CESPE/CEBRASPE - 2022 - TCE-SC- Auditor Fiscal de Controle Externo)

tabela1
campo
0
2
3
3
4
4
6
7

tabela2
campo
5
5
7
8
8
9

Considerando as tabela1 e tabela2 apresentadas, julgue o item que se segue, referentes a banco de dados.

Considere que o comando a seguir seja executado sem erro.

select campo from tabela2

except

select campo from tabela1

Nesse caso, o resultado obtido será a tabela seguinte.

campo
5
5
8
8
9

Resolução:

Vamos interpretar o comando parte a parte:

select campo from tabela2 -- seleção dos valores de campo da tabela2

except – com exceção dos valores que constem na consulta a seguir

select campo from tabela1-- seleção do valores de campo da tabela1

Ou seja, o resultado será os valores da tabela2 (**5, 5**, **7**, **8, 8, 9**) que não estão na tabela1 (0, 2, 3, 3, 4, 4, 6, **7**), logo: (5, ~~5,~~ 8, ~~8,~~ 9). A cláusula EXCEPT possui um DISTINCT implícito e, portanto, os valores repetidos não serão retornados. Assim, o retorno correto seria:

campo
5
8
9

Gabarito: **Errado.**

35- (FGV - 2018 - Prefeitura de Niterói - RJ - Analista de Políticas Públicas e Gestão Governamental - Gestão de Tecnologia) A questão deve ser respondida a partir das tabelas de banco de dados t1 e t2, a seguir.

T1

A	B	C
1	2	4
2	3	5
4	2	4
6	2	NULL

T2

D	E	F
1	2	NULL
2	5	5
4	2	1
7	12	1

Analise o comando SQL exibido abaixo.

```
select * from T1 where C > 5
UNION
select * from T1 where C <= 5
```

A execução desse comando no MS SQL Server produz um resultado que contém, além da linha de títulos, n linhas.

Assinale o valor de n.

a) 3 b) 4 c) 5 d) 6 e) 8

Resolução:

Vamos analisar o comando:

```
select * from T1 where C > 5
-- seleção dos registros da tabela T1 cujo valor de C seja maior que 5.
-- Logo, não será retornado nenhum registro, pois o C máximo da tabela é 5.
UNION
-- cláusula para unir os resultados, eliminando as duplicações implicitamente.
select * from T1 where C <= 5
-- seleção dos registros da tabela T1 cujo valor de C seja menor ou igual a 5.
-- Logo, serão retornados os seguintes valores:
```

A	B	C
1	2	4
2	3	5
4	2	4

Em relação ao registro com o valor C igual a NULL, vale ressaltar que NULL não é menor, maior ou igual a 5, por isso não é retornado em nenhuma das duas partes da consulta.

Logo, temos 3 registros sendo retornados.

Gabarito: **Letra A**.

2.1.9 Consultas aninhadas

Em alguns casos, precisamos realizar uma consulta que é comparada com o resultado de outra consulta. Aqui teremos o uso de uma **consulta dentro de outra consulta** ou **consulta aninhada**. A consulta que é realizada dentro de outra é chamada subconsulta.

Uma **subconsulta, consulta interna ou seleção interna** é uma consulta que está aninhada dentro de uma instrução SELECT, INSERT, UPDATE ou DELETE ou em outra subconsulta. Uma subconsulta pode ser usada em qualquer lugar em que é permitida uma expressão.

As subconsultas podem ser comparadas com a consulta externa com o uso de operadores IN (ou NOT IN), ANY, ALL e EXISTS (ou NOT EXISTS), além dos operadores básicos =, <, <=, >, >=, <>.

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4
5	Suco	2	2	Litros	8

E a tabela Produtos Líquidos a seguir:

ProdutoID	Nome_do_Produto
1	Leite
5	Suco

Vamos supor que você deseje consultar o preço médio de seus produtos líquidos. Para isso, você pode fazer uma consulta aninhada:

SELECT AVG(Preco) **FROM** Produtos **WHERE** ProdutoID **IN (SELECT** ProdutoID **FROM** ProdutosLiquidos);

O retorno dessa consulta será 5,5, pois a subconsulta retornará os ids 1 e 5 que serão utilizados pela consulta externa para fazer a média dos preços.

ANY e ALL

Os operadores **ANY** e **ALL** permitem realizar uma comparação entre o valor de uma única coluna e um conjunto de outros valores.

- **ANY**: TRUE se **QUALQUER um dos valores** da subconsulta **atender a condição**.

  SELECT colunas **FROM** tabela WHERE coluna operador **ANY** (subconsulta);

- **ALL**: TRUE se **TODOS os valores** da subconsulta **atenderem a condição**.

  SELECT colunas **FROM** tabela WHERE coluna operador **ALL** (subconsulta);

* O operador deve ser um dos operadores padrão de comparação: =, <>, !=, >, >=, <, ou <=.

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4
5	Suco	2	2	Litros	8

E a tabela Pedidos a seguir:

PedidoID	ProdutoID	Data	Quantidade	MeioDeEntrega
1	1	15/01/2025	10	1
2	2	20/01/2025	15	2
3	1	24/01/2025	20	2
4	3	27/01/2025	25	1
5	3	28/01/2025	30	1

Vamos supor que você deseje consultar somente os produtos que já foram pedidos alguma vez. Para isso, você pode fazer uma consulta aninhada:

SELECT Nome_do_Produto **FROM** Produtos **WHERE** ProdutoID = **ANY** (**SELECT** ProdutoID **FROM** Pedidos);

O retorno dessa consulta será:

Nome_do_Produto
Leite
Banana
Melancia

CONTINUAÇÃO...

Vamos explicar usando um passo a passo com os recortes das tabelas envolvidas:

1. Isolando o primeiro valor da tabela Produto e comparando o seu ProdutoID com os retornados na subconsulta (que são os ProdutoID da tabela Pedido):

ProdutoID	Nome_do_Produto
1	Leite
2	Banana
3	Melancia
4	Pão
5	Suco

PedidoID	ProdutoID
1	1
2	2
3	1
4	3
5	3

1=1? SIM

Como existe algum valor que cumpra a condição, então o ANY retorna TRUE, e portanto o primeiro produto será selecionado.

2. Os produtos 2 e 3 também irão cumprir a condição.

3. Agora vamos analisar o que acontece com o produto 4:

ProdutoID	Nome_do_Produto
1	Leite
2	Banana
3	Melancia
4	Pão
5	Suco

PedidoID	ProdutoID
1	1
2	2
3	1
4	3
5	3

4=1? NÃO
4=2? NÃO
4=1? NÃO
4=3? NÃO
4=3? NÃO

Nesse caso, não há nenhum ProdutoID que faça correspondência com o externo. Logo, o ANY retorna FALSE e o produto 4 não é retornado.

4. O produto 5 também não será retornado. Por isso, o resultado conterá apenas Leite, Banana e Melancia.

Agora vamos supor que você deseje encontrar produtos que nunca foram pedidos. Uma possibilidade é usar a seguinte consulta:

```
SELECT Nome_do_Produto FROM Produtos WHERE ProdutoID <> ALL (SELECT ProdutoID FROM Pedidos);
```

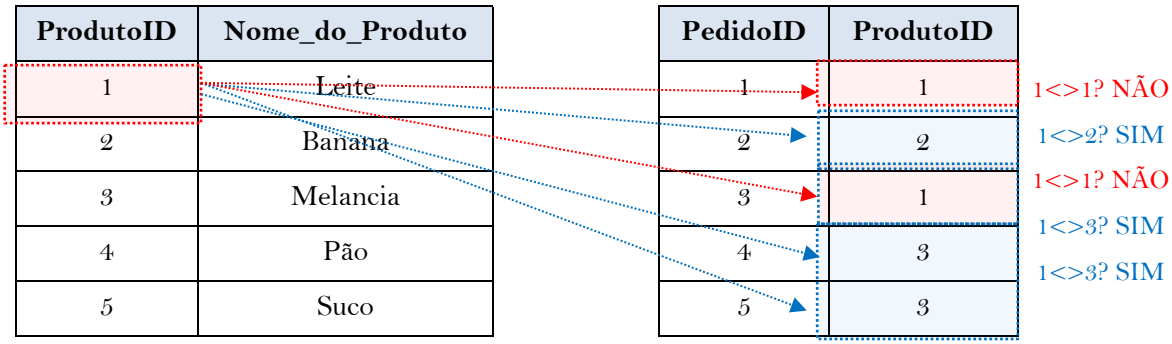
O retorno dessa consulta será:

Nome_do_Produto
Pão
Suco

CONTINUAÇÃO...

Vamos explicar o passo a passo:

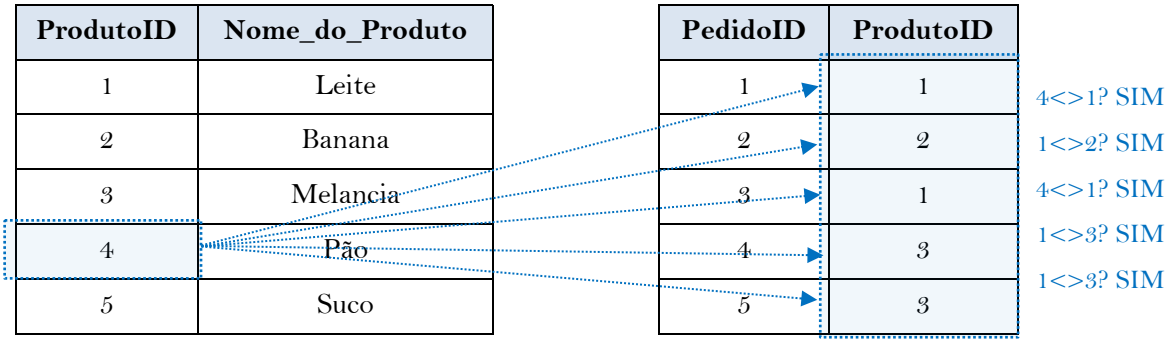
1. Isolando o primeiro valor da tabela Produto e comparando o seu ProdutoID com os retornados na subconsulta (que são os ProdutoID da tabela Pedido):



Nesse caso, nem todos os valores retornados na subconsulta cumprem a condição e, por isso, o ALL retorna FALSE. Logo, o primeiro elemento não será retornado na consulta externa.

2. Para os produtos 2 e 3 acontecerá o mesmo.

3. Agora vamos analisar o que acontece com o produto 4.



Nesse caso, todos os valores retornados na subconsulta cumprem a condição de serem diferentes do valor externo e, portanto, o ALL retorna TRUE e o produto 4 será retornado pela consulta externa.

4. O produto 5 também será retornado. Por isso, o retorno da consulta será Pão e Suco.

EXISTS (e NOT EXISTS)

A cláusula **EXISTS** faz uma **verificação se existe algum resultado para a subconsulta informada**. **Caso haja**, o **resultado da consulta principal é exibido**. É muito comum sua utilização quando se deseja trazer resultados onde um valor específico existe dentro de outra tabela. A sintaxe básica é:

> **SELECT** colunas **FROM** tabela **WHERE EXISTS (SELECT** colunas **FROM** tabela WHERE condicão);

Da mesma forma, também há a cláusula **NOT EXISTS**, somente **retorna o resultado da consulta principal, se não houver nenhum resultado para a subconsulta**.

EXEMPLIFICANDO!!!

Considere as tabelas a seguir:

PRODUTO

id	nome	preco	id_categoria
1	Bola	35.00	1
2	Patinete	120.00	1
3	Carrinho	15.00	1
4	Skate	296.00	1
5	Notebook	3500.00	2
6	Monitor LG 19	450.00	2
7	O Diário de Anne Frank	45.00	3
8	O dia do Curinga	65.00	3
9	O mundo de Sofia	48.00	3
10	Através do Espelho	38.00	3

VENDA_PRODUTO

id	id_produto	valor	data
1	1	35.00	15/05/2018
2	1	35.00	15/06/2018
3	1	35.00	15/07/2018
4	2	120.00	15/07/2018
5	2	120.00	14/07/2018
6	3	15.00	15/07/2018
7	7	45.00	15/07/2018
8	8	65.00	15/07/2018
9	8	65.00	16/07/2018
10	9	48.00	16/07/2018
11	5	3500.00	16/07/2018
12	5	3500.00	16/07/2018
13	6	450.00	16/07/2018

Suponha que seja necessário trazer em uma consulta na tabela de produtos, todos aqueles registros que tiveram alguma venda. Para isso podemos utilizar o EXISTS, além de testar se a condição é verdadeira, traz como retorno os dados da consulta. A sintaxe a seguir poderá ser utilizada:

```
SELECT p.id, p.nome FROM produto p WHERE EXISTS (SELECT v.id_produto FROM venda_produto v WHERE v.id_produto = p.id);
```

Vamos analisar o comando por partes:

```
SELECT p.id, p.nome FROM produto p
-- seleção do id e nome da tabela produto
WHERE EXISTS
-- a consulta só será efetuada para o registro que cumprir a condição da subconsulta
(SELECT v.id_produto FROM venda_produto v WHERE v.id_produto = p.id);
-- isto é, somente se o id do produto constar na tabela venda_produto.
```

CONTINUAÇÃO...

O resultado desse comando será:

id	nome
1	Bola
2	Patinete
3	Carrinho
5	Notebook
6	Monitor LG 19
7	O Diário de Anne Frank
8	O dia do Curinga
9	O mundo de Sofia

Vamos entender o que aconteceu no comando passo a passo:

1. Isolando o primeiro produto e comparando-o com todos os resultados da subconsulta com base na condição WHERE, ou seja, verificar se o valor do seu id é igual ao valor de algum id da tabela venda_produto (v.id_produto = p.id).

PRODUTO

id	nome	preco	id_categoria
1	Bola	35.00	1
2	Patinete	120.00	1
3	Carrinho	15.00	1
4	Skate	296.00	1
5	Notebook	3500.00	2
6	Monitor LG 19	450.00	2
7	O Diário de Anne Frank	45.00	3
8	O dia do Curinga	65.00	3
9	O mundo de Sofia	48.00	3
10	Através do Espelho	38.00	3

VENDA_PRODUTO

id	id_produto	valor	data	
1	1	35.00	15/05/2018	1=1? SIM
2	1	35.00	15/06/2018	
3	1	35.00	15/07/2018	
4	2	120.00	15/07/2018	
5	2	120.00	14/07/2018	
6	3	15.00	15/07/2018	
7	7	45.00	15/07/2018	
8	8	65.00	15/07/2018	
9	8	65.00	16/07/2018	
10	9	48.00	16/07/2018	
11	5	3500.00	16/07/2018	
12	5	3500.00	16/07/2018	
13	6	450.00	16/07/2018	

Como existe um valor correspondente, então o EXISTS retorna TRUE, e o SELECT será executado para essa primeira linha de id = 1.

2. Perceba que com 2 e 3 acontecerá a mesma coisa. Ilustrando com o 2:

PRODUTO

id	nome	preco	id_categoria
1	Bola	35.00	1
2	Patinete	120.00	1
3	Carrinho	15.00	1
4	Skate	296.00	1

VENDA_PRODUTO

id	id_produto	valor	data	
1	1	35.00	15/05/2018	2=1? NÃO
2	1	35.00	15/06/2018	2=1? NÃO
3	1	35.00	15/07/2018	2=1? NÃO
4	2	120.00	15/07/2018	2=2? SIM

CONTINUAÇÃO...

O fato é que mesmo que alguns valores não cumpram a condição, basta que um cumpra.

3. Agora vamos analisar o que acontecerá na linha do produto 4.

PRODUTO

id	nome	preco	id_categoria
1	Bola	35.00	1
2	Patinete	120.00	1
3	Carrinho	15.00	1
4	Skate	296.00	1
5	Notebook	3500.00	2
6	Monitor LG 19	450.00	2
7	O Diário de Anne Frank	45.00	3
8	O dia do Curinga	65.00	3
9	O mundo de Sofia	48.00	3
10	Através do Espelho	38.00	3

VENDA_PRODUTO

id	id_produto	valor	data	
1	1	35.00	15/05/2018	4=1? NÃO
2	1	35.00	15/06/2018	4=1? NÃO
3	1	35.00	15/07/2018	4=1? NÃO
4	2	120.00	15/07/2018	4=2? NÃO
5	2	120.00	14/07/2018	4=2? NÃO
6	3	15.00	15/07/2018	4=3? NÃO
7	7	45.00	15/07/2018	4=7? NÃO
8	8	65.00	15/07/2018	4=8? NÃO
9	8	65.00	16/07/2018	4=8? NÃO
10	9	48.00	16/07/2018	4=9? NÃO
11	5	3500.00	16/07/2018	4=5? NÃO
12	5	3500.00	16/07/2018	4=5? NÃO
13	6	450.00	16/07/2018	4=6? NÃO

Nessa situação, não há nenhum valor retornado na subconsulta, pois nenhum valor cumpre a condição de igualdade dos ids. Assim, o EXISTS retornará FALSE e o produto de id=4 não será retornado na consulta externa.

4. O processo seguirá para todos os registros de produto, mas nós já podemos perceber que os valores que serão retornados são aqueles de produtos cujos id estão também na tabela venda_produto. Logo, 1, 2, 3, 5, 6, 7, 8 e 9. Somente 4 e 10 não serão retornados.

ATENÇÃO!!!

A cláusula EXISTS retorna TRUE (verdadeiro) se existir algum registro que cumpre a subconsulta. A consulta externa, por sua vez, retorna o registro. Caso contrário, o EXISTS retorna FALSE (falso) e a consulta externa não irá exibir o registro.

Do mesmo modo, a cláusula NOT EXISTS retorna TRUE (verdadeiro) se não existir nenhum registro que cumpra a subconsulta. A consulta externa, por sua vez, retorna o registro. Caso contrário, o NOT EXISTS retorna FALSE (falso) e a consulta externa não irá exibir o registro.

Esquematizando:

Consulta aninhada

- Uma subconsulta, consulta interna ou seleção interna é uma consulta que está aninhada dentro de uma instrução SELECT, INSERT, UPDATE ou DELETE ou em outra subconsulta.
- As subconsultas podem ser comparadas com a consulta externa com o uso de operadores IN (ou NOT IN), ANY, ALL ou EXISTS (ou NOT EXISTS), além dos operadores básicos =, <, <=, >, >=, <>.

Cláusulas Especiais

- **ANY**: retorna TRUE se qualquer um dos valores da subconsulta atender a condição.
- **ALL**: retorna TRUE se todos os valores da subconsulta atenderem a condição.
- **EXISTS**: retorna TRUE se a subconsulta retornar um ou mais registros.
- **NOT EXISTS**: retorna TRUE se a subsconsulta não retornar nenhum registro.

Esquema 14 – Consultas aninhadas.

36- **(FGV - 2023 – TJ SE – Analista Judiciário)** Considere uma tabela relacional TAB, com colunas A e B. A coluna A constitui a chave primária de TAB. A instância de TAB contém 100 linhas, e em todas as linhas o valor da coluna B é 10. Nesse contexto, analise o comando SQL a seguir.

```
select * from TAB t
where not exists
    (select * from TAB tt
      where t.B = tt.B and t.A > tt.A)
```

Além da linha de títulos, o número de linhas produzidas pelo comando acima é:

a) 0;

b) 1;

c) 98;

d) 99;

e) 100.

Resolução:

Primeiro de tudo, vamos montar uma tabela TAB fictícia com base nos dados da questão. Como a questão afirma que A é chave primária e a tabela possui 100 linhas, então termos 100 valores diferentes para A, pois uma chave não pode ter valores repetidos. Para fins de resolução, podemos considerar valores de 1 a 100 (poderiam ser quaisquer outros). O valor de B para todas as linhas é 10.

Logo, chegamos a seguinte tabela TAB:

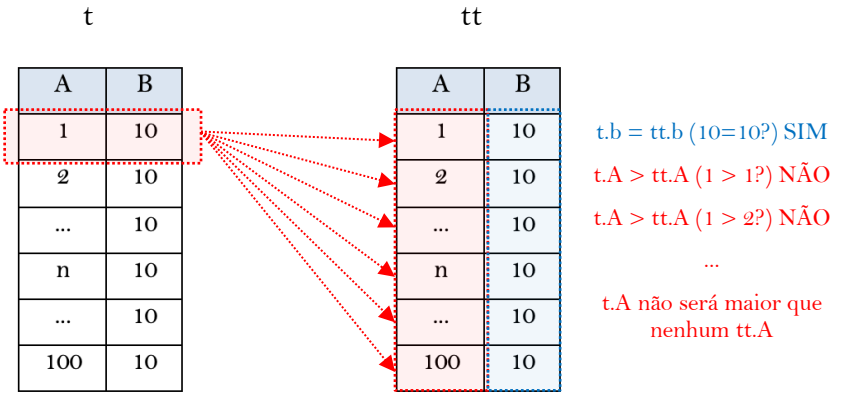
A	B
1	10
2	10
...	10
n	10
...	10
100	10

Agora vamos ao comando:

```
select * from TAB t
-- selecionar os valores da tabela TAB referenciada como t.
where not exists
-- quando não existir retorno para a subconsulta a seguir
    (select * from TAB tt
      -- selecionar os valores da tabela TAB referenciada como tt. Nesse caso, teremos, em tempo de execução, duas referências para a mesma tabela TAB, ou seja, teremos duas cópias da mesma tabela durante a consulta.
      where t.B=tt.B and t.A > tt.A)
      -- e haverá uma comparação dos atributos das referências dessas tabelas.
```

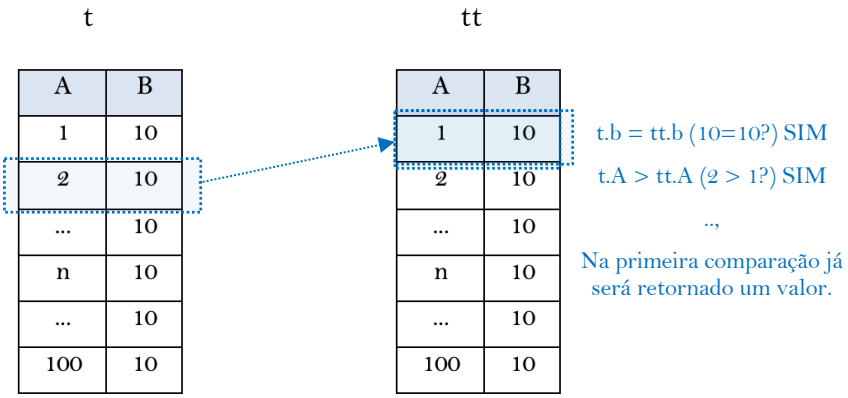
Vamos à execução:

1. Isolando o primeiro valor da referência t e comparando-o com todos os resultados da subconsulta com base na condição WHERE, ou seja, verificar se o valor de t.b é igual a tt.b e se t.A é maior que tt.A.



Como não existe nenhum registro que cumpra as condições, o NOT EXISTS retornará TRUE e, portanto, essa **primeira linha será selecionada**.

2. Vamos a segunda linha:



Nesse caso, será retornado um valor, pois existe um valor em t cujo valor de A seja maior que o valor de tt e seu B seja igual. Como existe pelo menos um registro que cumpre a condição, o NOT EXISTS retorna FALSE e, assim, o valor da **segunda linha não será retornado na consulta externa**.

3. Perceba que para todas as demais linhas, sempre haverá um retorno, pois o valor de A em t será maior do que algum valor de A da referência tt. Logo, NOT EXISTS será FALSE para todas elas e, portanto, **não será mais retornado nenhum valor**.

Embora tenhamos usado um exemplo de tabela com números de exemplo, essa conclusão pode ser obtida pela lógica. Como A é chave primária e nunca terá dois valores iguais, então teremos um menor valor que, portanto, não será maior que nenhum outro valor dessa tabela. O caso do B é mais simples, pois como sempre é 10, seu valor sempre será igual a algum valor dessa tabela (no caso a todos).

Gabarito: **Letra B.**

37- (CESPE / CEBRASPE - 2022 - TCE-SC - Auditor Fiscal de Controle Externo).

tabela1
campo
0
2
3
3
4
4
6
7

tabela2
campo
5
5
7
8
8
9

Considerando as tabela1 e tabela2 apresentadas, julgue o item que se segue, referentes a banco de dados.

Considere que o comando a seguir seja executado sem erro.

select campo from tabela2

where exists

(select campo from tabela1)

Nesse caso, o resultado será a tabela seguinte.

campo
5
5
8
8
9

Resolução:

A cláusula **EXISTS** faz uma **verificação se existe algum resultado para a subconsulta informada**. **Caso haja**, o **resultado da consulta principal é exibido**.

Dado o comando:

```
select campo from tabela2
-- selecionar o valor de campo da tabela2
where exists
-- desde que esse valor exista em algum retorno da subsconsulta subsequente
(select campo from tabela1)
-- selecionar o valor de campo da tabela1
```

O ponto desse comando é que a subconsulta apenas verifica se existe algum valor para campo na tabela1, não fazendo qualquer comparação entre os valores das tabelas. Nesse caso, a subconsulta sempre retorna TRUE, pois existem valores na tabela1. Logo, o SELECT externo será realizado para todos os valores da tabela2.

Para entender melhor, vamos isolar o primeiro valor da tabela2 que é 5. A consulta será executada com a seguinte lógica: "retorne o valor 5, desde que exista algum retorno no EXISTS, ou seja, desde que seja retornado algum valor de campo da tabela1". A subconsulta retornará TRUE, então o valor 5 será retornado. O mesmo acontecerá para todos os demais elementos de tabela2. Logo, o resultado será uma tabela com os mesmos valores da tabela2.

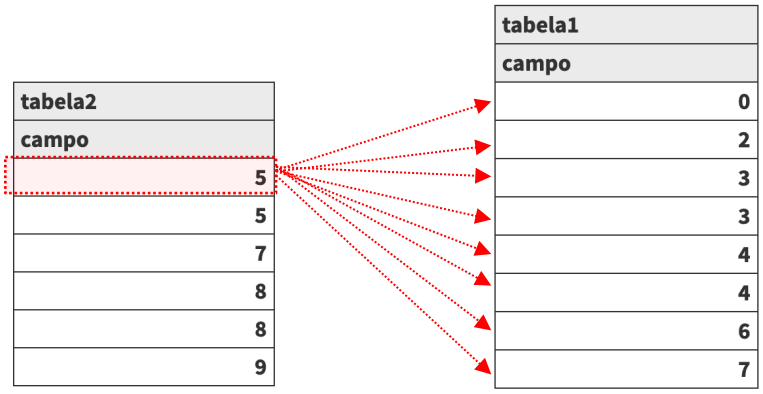
campo
5
5
7
8
8
9

A situação seria diferente caso houvesse alguma condição de comparação:

```
select campo from tabela2
where exists
(select campo from tabela1 where tabela1.campo=tabela2.campo)
```

Vejamos como seria a execução:

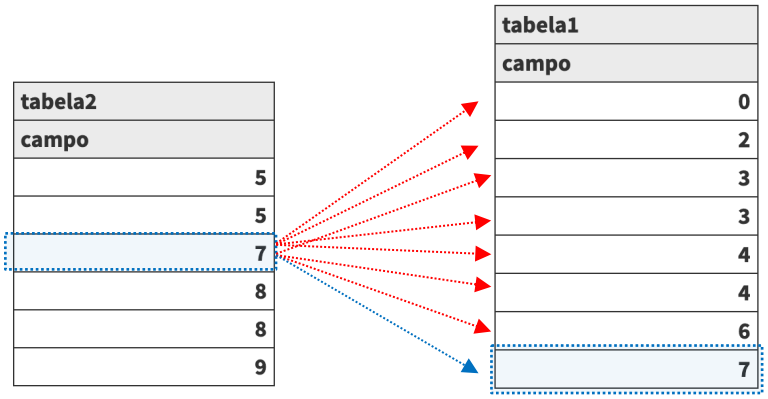
1. Isolando o primeiro valor da tabela2 e comparando-o com todos os resultados da subconsulta (que são simplemente os valores da tabela1).



Como não existe nenhum valor correspondente, então o EXISTS retorna FALSE, e o SELECT não será executado para essa primeira linha de valor = 5.

2. Da mesma forma, já sabemos que não será executado também para a segunda linha.

3. Agora vejamos o que acontece na terceira linha:



Nesse caso, o valor 7 existe em algum dos valores da subconsulta, então o EXISTS retorna TRUE e o SELECT será executado para a linha de valor 7.

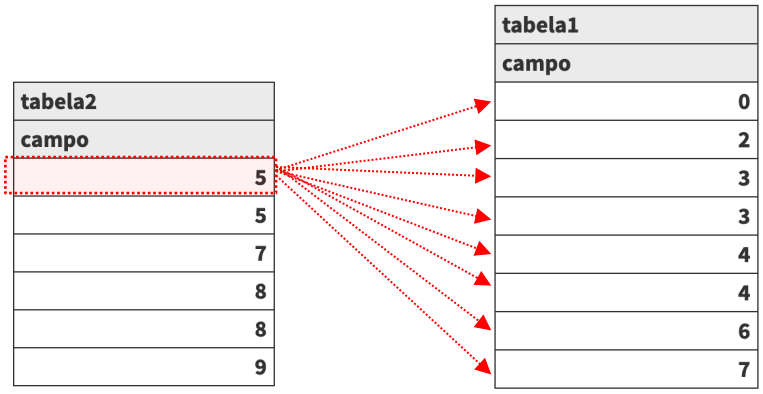
4. O processo seguirá para todos os registros da tabela2, mas nós já podemos perceber que os valores que serão retornados são aqueles da tabela2 que também existem na tabela1. Portanto somente o 7.

O resultado da assertiva seria obtido com a seguinte comparação:

```
select campo from tabela2
where not exists
(select campo from tabela1 where tabela1.campo=tabela2.campo)
```

Vejamos:

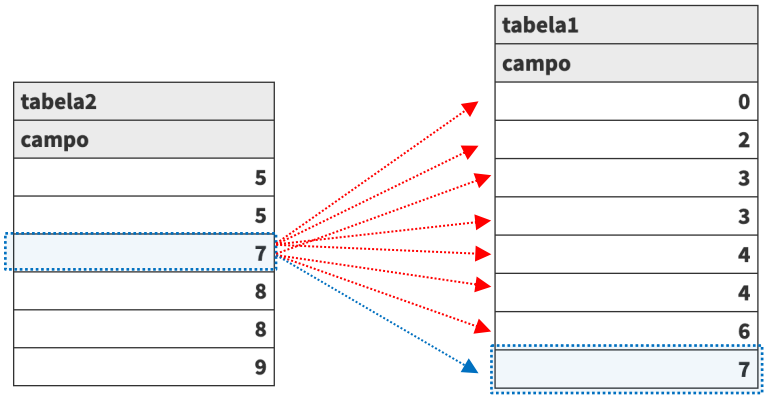
1. Isolando o primeiro valor da tabela2 e comparando-o com todos os resultados da subconsulta (que são simplemente os valores da tabela1).



Como não existe nenhum valor correspondente, então o NOT EXISTS retorna TRUE, e o SELECT será executado para essa primeira linha de valor = 5.

2. Da mesma forma, já sabemos que será executado também para a segunda linha.

3. Agora vejamos o que acontece na terceira linha:



Nesse caso, o valor 7 existe em algum dos valores da subconsulta, então o NOT EXISTS retorna FALSE e o SELECT não será executado para a linha de valor 7.

4. O processo seguirá para todos os registros da tabela2, mas nós já podemos perceber que os valores que serão retornados são aqueles da tabela2 que não existem na tabela1. Portanto somente o 5, 5, 8, 8 e 9.

Contudo, como vimos, o comando da questão retorna 5, 5, 7, 8, 8 e 9.

Gabarito: **Errado.**

2.1.10 Cláusulas especiais

CASE

A cláusula **CASE** **percorre condições e retorna um valor quando a primeira condição é atendida**. Assim, uma vez que uma condição seja verdadeira, a expressão para de avaliar as demais e retorna o resultado. Se nenhuma condição for verdadeira, ela retorna o valor especificado na cláusula **ELSE**. Caso não haja uma parte **ELSE** e nenhuma condição seja verdadeira, ela retornará **NULL**.

A sintaxe da cláusula básica da cláusula CASE é:

```
CASE
   WHEN condição1 THEN resultado1
   WHEN condição2 THEN resultado2
   WHEN condiçãoN THEN resultadoN
   ELSE resultado
END;
```

EXEMPLIFICANDO!!!

Dada a tabela Produtos a seguir:

ProdutoID	Nome_do_Produto	FornecedorID	CategoriaID	Unidade	Preco
1	Leite	1	1	Litros	3
2	Banana	1	1	Kilogramas	5
3	Melancia	1	2	Unidade	6
4	Pão	2	2	Pacote	4

Ao executar o seguinte comando:

```
SELECT ProdutoID, Preco,
CASE WHEN Preco > 4 THEN 'O preço é MAIOR que 4'
WHEN Preco = 4 THEN 'O preço é 4'
ELSE 'O preço é MENOR que 4'
END AS QuantidadeEmTexto
FROM Produto;
```

O retorno dessa será:

ProdutoID	Preco	QuantidadeEmTexto
1	3	O preço é MENOR que 4
2	5	O preço é MAIOR que 4
3	6	O preço é MAIOR que 4
4	4	O preço é 4

38- (CESPE / CEBRASPE - 2023 – SEPLAN-RR - Analista de Planejamento e Orçamento) Com pertinência à linguagem SQL, julgue o item abaixo.

Considere-se o seguinte script SQL.

```
select report_code, year, month, day, wind_speed,
case
    when wind_speed>= 40 then 'HIGH'
    when wind_speed>= 30 then 'MODERATE'
    else 'LOW'
end as wind_severity
from station_data
```

O resultado da execução do script resultará em erro, pois, caso haja, na tabela station_data, algum registro no campo wind_speed com valor superior a 40, não será possível predizer se o valor da variável wind_severity será igual a 'HIGH'.

Resolução:

Vamos avaliar o comando:

select report_code, year, month, day, wind_speed,

-- seleciona os campos report_code, year, month, day e wind_speed

case – percorre as condições

when wind_speed>= 40 **then** 'HIGH'

– se o valor de wind_speed for maior ou igual a 40, então retorna 'HIGH'.

when wind_speed>= 30 **then** 'MODERATE'

– se o valor de wind_speed for maior ou igual a 30, então retorna 'MODERATE'.

else 'LOW'

– se nenhuma das condições anteriores for verdadeira, então retorna 'LOW'.

end as wind_severity

– finaliza a cláusula case e atribui um apelido wind_severity para o resultado.

from station_data – a partir da tabela station_data.

Em resumo, temos:

wind_speed	wind_severity
1 condicão testada: Maior ou igual a 40	HIGH
2 condicão testada: Maior ou igual a 30	MODERATE
Se não passou em nenhuma condição anterior	LOW

Portanto a assertiva é falsa, pois se houver na tabela um valor de wind_speed maior que 40, a variável wind_severity será HIGH, pois entrará na primeira condição do CASE.

Gabarito: **Errado.**

TOP, LIMIT ou FETCH FIRST

As cláusulas **TOP, LIMIT ou FETCH FIRST** servem para **especificar o número de registros a serem retornados em uma consulta**. A cláusula exata depende do SGBD, mas vou deixar aqui pelo menos a sintaxe básica delas para caso apareçam em questões.

SELECT TOP numero * FROM tabela **WHERE** condição**;** **(SQL Server/MS Access)**

ou

SELECT colunas **FROM** tabela **WHERE** condição **LIMIT numero;** **(MySQL / PostgreSQL)**

ou

SELECT colunas **FROM** tabela **WHERE** condição **FETCH FIRST numero ROWS ONLY;** **(Oracle)**

* numero é a quantidade de linhas que se deseja exibir no resultado.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Caso deseje selecionar apenas os 2 primeiros clientes, é possível usar:

SELECT TOP 2 * FROM Clientes; **(SQL Server / MS Access)**

ou

SELECT * FROM Clientes **LIMIT 2;** **(MySQL / PostgreSQL)**

ou

SELECT * FROM Clientes **FETCH FIRST 2 ROWS ONLY;** **(Oracle)**

O retorno dessa consulta será:

Exibição de apenas os 2 primeiros registros

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

OFFSET

A cláusula **OFFSET** é usada para **pular um número específico de registros antes de começar a exibir** os resultados. A sintaxe básica é:

SELECT colunas **FROM** tabela **WHERE** condição **OFFSET** **numero;**

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Case deseje selecionar os registros, pulando os 2 primeiros, é possível usar:

SELECT * FROM Clientes **OFFSET** **2;**

O retorno dessa consulta será:

Exibição pulando os 2 primeiros registros

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

SELECT INTO

A cláusula **SELECT INTO** é usada para **copiar os dados de uma tabela para uma nova tabela**. A sintaxe básica é:

SELECT colunas **INTO** nova_tabela **FROM** tabela_original **WHERE** condicao**;**

É possível também criar a tabela com a atribuição de valores, usando a sintaxe a seguir:

SELECT valor1 AS coluna1, valor2 AS coluna2... **INTO** nova_tabela**;**

EXEMPLIFICANDO!!!

O comando SELECT * INTO Clientes_Brasileiros FROM Clientes WHERE Pais = 'Brasil'; copia somente os Clientes do Brasil para uma tabela chamada Clientes_Brasileiras.

O comando SELECT 1 AS código, 'João'AS nome INTO alunos; cria a tabela alunos com um registro com código=1 e nome= "João".

O quadro a seguir apresenta um resumo dessas cláusulas especiais:

OPERADOR	RETORNO
CASE	Percorre condições e retorna um valor para a 1ª condição atendida.
TOP, LIMIT ou FETCH FIRST	Especifica o número de registros a serem retornados.
OFFSET	Pula um número de registros antes de começar a exibir.
SELECT INTO	Copia dados de uma tabela para uma nova tabela

Esquema 15 - Cláusulas especiais.

39- (FUNDATEC - 2023 – CAU RS – Analista Superior) Assinale a alternativa que corresponde ao resultado esperado pela execução do comando SQL abaixo:

SELECT *

INTO BKP_VENDAS

FROM VENDAS;

a) Será criada uma nova tabela, nomeada como BKP_VENDAS, com todos os dados da tabela VENDAS

b) Serão atualizados apenas os registros da tabela BKP_VENDAS que tiverem chave correspondente na tabela VENDAS

c) A tabela VENDAS será renomeada para BKP_VENDAS

d) Será criada uma nova tabela vazia, nomeada como BKP_VENDAS

e) Serão copiados os dados da tabela BKP_VENDAS para a tabela VENDAS

Resolução:

A cláusula **SELECT INTO copia os dados de uma tabela em uma nova tabela**. A sintaxe é: **SELECT** colunas **INTO** nova_tabela **FROM** tabela_original **WHERE** condicao**;**

Vamos aos itens:

a) **Correto:** o comando SELECT * INTO BKP_VENDAS FROM VENDAS; cria uma nova tabela BKP_VENDAS e copia todos os registros da tabela VENDAS para ela.

b) **Incorreto:** o comando não atualiza registros, ele cria uma tabela e copia os dados existentes da tabela VENDAS.

c) **Incorreto:** o comando não renomeia a tabela VENDAS, ele apenas cria uma cópia dela. Para renomear uma tabela, seria necessário usar ALTER TABLE ou RENAME TABLE.

d) **Incorreto:** a tabela BKP_VENDAS será criada, mas não estará vazia. Todos os registros da tabela VENDAS serão copiados para ela.

e) **Incorreto:** o comando copia os dados de VENDAS para BKP_VENDAS, não o contrário.

Gabarito: **Letra A**.

2.2 DML: instrução DELETE

A instrução básica para **deletar registros existentes de uma tabela** é a instrução **DELETE**.

A sintaxe básica de uma instrução **DELETE** é da seguinte forma:

DELETE FROM nome_da_tabela **WHERE** condição;

Esse comando permite excluir os valores das colunas de uma tabela que cumprem uma determinada condição.

Vale ressaltar que a condição após a cláusula WHERE pode utilizar os mesmo operadores usados para o SELECT como =, <, <=, >, >=, <>, BETWEEN, LIKE, IN, AND, OR e NOT.

É possível deletar todos os registros **sem indicar nenhuma condição**:

DELETE FROM nome_da_tabela;

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

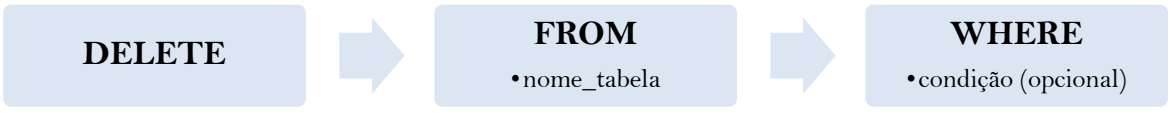
Vamos supor que você deseje excluir o cliente de nome "Alfred Futtekiste". Para isso, poderá usar o seguinte comando:

DELETE FROM Clientes **WHERE** Nome_Cliente='Alfreds Futterkiste';

Registro excluído

Ao realizar uma nova consulta nessa tabela, o resultado será:

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico



Esquema 16 – Sintaxe básica da instrução DELETE.

40- (CESPE / CEBRASPE - 2022 – PETROBRAS - Profissional Petrobras de Nível Superior) Julgue o item abaixo, a respeito da linguagem SQL.

O comando delete alunos permite apagar uma tabela de nome alunos.

Resolução:

O comando DELETE não apaga a tabela, mas sim os dados de uma tabela. Para apagar a tabela, deve-se usar o comando DROP.

Gabarito: **Errado.**

41- (FCC - 2022 – PGE AM - Técnico em Gestão Procuratorial. Para excluir da tabela Advogado_Processo todas as linhas nas quais consta o valor 4378 no campo oabAdvogado, utiliza-se a instrução

a) DELETE * FROM Advogado_Processo AS adp WHERE adp.oabAdvogado=4378;

b) DELETE FROM Advogado_Processo (SELECT oabAdvogado=4378);

c) DELETE * FROM Advogado_Processo WHERE oabAdvogado=4378;

d) DELETE FROM Advogado_Processo WHERE oabAdvogado=4378;

e) DELETE FROM Advogado_Processo (SELECT * FROM Advogado WHERE oabAdvogado=4378);

Resolução:

Vamos analisar cada um dos comandos:

a) **Incorreto**: DELETE ~~*~~ FROM Advogado_Processo AS adp WHERE adp.oabAdvogado=4378;

A sintaxe do DELETE não é com *.

b) **Incorreto**: DELETE FROM Advogado_Processo ~~(SELECT oabAdvogado=4378)~~;

Há a presença de uma subconsulta incompleta, além de não haver nenhuma cláusula para a avaliação como em WHERE oabAdvogado = (subconsulta completa aqui...).

c) **Incorreto**: DELETE ~~*~~ FROM Advogado_Processo WHERE oabAdvogado=4378;

A sintaxe do DELETE não é com *.

d) **Correto**: DELETE FROM Advogado_Processo WHERE oabAdvogado=4378;

Comando em conformidade com a sintaxe do DELETE.

e) **Incorreto**: DELETE FROM Advogado_Processo **WHERE oabAdvogado =** (SELECT * FROM Advogado WHERE oabAdvogado=4378);

Há a presença de uma subconsulta, mas sem nenhuma cláusula de avaliação antes dela como WHERE oabAdvogado = (subconsulta completa aqui...).

Gabarito: **Letra D.**

2.3 DML: instrução UPDATE

A instrução básica para **atualizar os registros de uma tabela** é a instrução **UPDATE**.

A sintaxe básica de uma instrução **UPDATE** é da seguinte forma:

> **UPDATE** nome_da_tabela **SET** coluna1 = valor1, coluna2 = valor2 ... **WHERE** condição;

Esse comando permite atualizar os valores das colunas de uma tabela que cumprem uma determinada condição.

Vale ressaltar que a condição após a cláusula WHERE pode utilizar os mesmo operadores usados para o SELECT como =, <, <=, >, >=, <>, BETWEEN, LIKE, IN, AND, OR e NOT.

Muito cuidado quando for atualizar uma tabela, **caso não seja indicada nenhuma condição**, todos os registros da tabela serão atualizados.

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você deseje alterar o Nome_Contato e a cidade do cliente de IDCliente = 1. Para isso, poderá usar o seguinte comando:

UPDATE Clientes **SET** Nome_Contato = 'Alfred Schmidt', Cidade= 'Frankfurt'

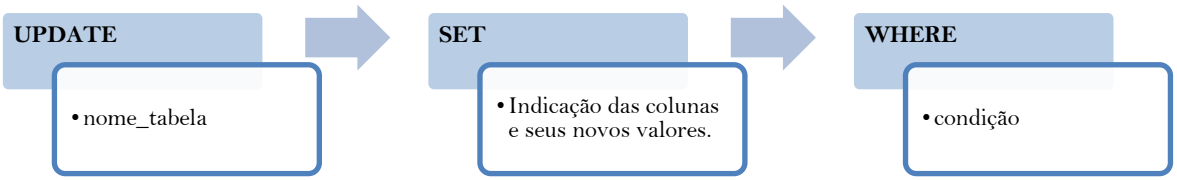
WHERE CustomerID = 1;

Ao realizar uma nova consulta nessa tabela, o resultado será:

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

Caso tivéssemos esquecido de colocar a condição, todos os clientes teriam seu Nome_Contato modificado para 'Alfred Schmidt" e a Cidade modificada para 'Frankfurt'.

A sintaxe básica do comando **UDPATE** pode ser esquematizada a seguir:



Esquema 17 – Sintaxe básica da instrução UPDATE.

42- (VUNESP - 2024 – Pref Santo André – Analista de Tecnologia da Informação) Considere a seguinte tabela de um banco de dados relacional:

Aluno (Número, Nome, Curso)

O comando SQL para transferir o aluno de número 247 para o curso de Física é:

a) SET Aluno.Curso = Física WHERE Número = 247;

b) SET Aluno Curso = Física FOR Número = 247;

c) UPDATE Aluno SET Curso = Física WHERE Número = 247;

d) UPDATE Aluno Curso = Física FOR Número = 247;

e) UPDATE Aluno.Curso = Física, Aluno.Curso = 247;

Resolução:

Vamos analisar cada um dos comandos:

a) **Incorreto**: sem o comando UPDATE.

b) **Incorreto**: sem o comando UPDATE.

c) **Correto**: UPDATE Aluno SET Curso = Física WHERE Número = 247;

De acordo com a sintaxe do UPDATE.

d) **Incorreto**: UPDATE Aluno **SET** Curso = Física ~~FOR~~ **WHERE** Número = 247;

Faltou o SET. Além disso, a condição é definida com WHERE e não com FOR.

e) **Incorreto**: UPDATE Aluno **SET** Aluno.Curso = Física~~,~~ **WHERE** Aluno.~~Curso~~**Numero** = 247;

Faltou indicar o nome da tabela, o SET e o WHERE. Além disso, a condição é que o número do aluno seja 247 e não o curso.

Gabarito: **Letra C**.

43- (FGV - 2015 - Câmara Municipal de Caruaru - PE - Analista Legislativo - Informática) 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.

Resolução:

Vamos analisar o comando:

update C

-- atualização da tabela C

set b = (select max (b) from C).

-- alterar o valor de b para o máximo b.

Logo, serão alterados todos os registros, colocando-se 8 no valor de b, pois este é o valor máximo, à exceção dos registros que já possuem valor 8. Assim, os novos valores serão:

a b

1 8

2 8

3 8

4 8

Dessa forma, foram alterados 3 registros.

Gabarito: **Letra D.**

2.4 DML: instrução INSERT INTO

A instrução para **inserir novos registros em uma tabela** é a instrução **INSERT INTO**.

A sintaxe básica de uma instrução **INSERT INTO** é da seguinte forma:

INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ...) **VALUES** (valor1, valor2, valor3, ...);

Esse comando permite inserir os valores indicados após VALUES nas colunas indicadas.

Caso você esteja inserindo um registro completo, isto é, com todas as colunas, você não precisa indicar quais as colunas, mas apenas inserir os dados na mesma ordem das colunas da tabela. Assim, a sintaxe a seguir também é válida:

INSERT INTO nome_da_tabela **VALUES** (valor1, valor2, valor3, ...);

EXEMPLIFICANDO!!!

Dada a tabela Clientes a seguir:

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

Vamos supor que você inserir dois novos clientes. Para isso, poderá usar os seguintes comandos:

INSERT INTO Clientes (IDCliente, Nome_Cliente, Pais) **VALUES** (4, 'João Aprovado dos Santos', 'Brazil');

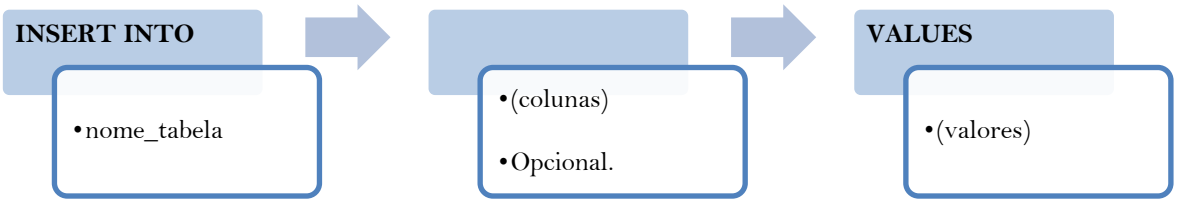
INSERT INTO Clientes **VALUES** (5, 'Maria Concursada', 'Maria Antonieta', 'SGAN 911', 'Brasília', 6000, 'Brazil');

Ao realizar uma nova consulta nessa tabela, o resultado será:

2 registros inseridos

IDCliente	Nome_Cliente	Nome_Contato	Endereco	Cidade	CEP	Pais
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	João Aprovado	NULL	NULL	NULL	NULL	Brazil
5	Maria Concursada	Maria Antonieta	SGAN 911	Brasília	6000	Brazil

A sintaxe básica do comando **INSERT** pode ser esquematizada a seguir:



Esquema 18 – Sintaxe básica da instrução INSERT INTO.

44- (CESPE / CEBRASPE - 2024 – CNPq - Analista em Ciência e Tecnologia I)

categoria	modelo	valor
1	A	40
1	A	18
1	B	76
2	B	97

Tendo como referência a tabela precedente, de nome dados, julgue o item subsequente, com relação à SQL.

As expressões SQL a seguir, após serem executadas, gerarão o mesmo resultado.

insert into dados values (2,'C',40);

insert into dados (categoria, modelo, valor) values (2,'C',40);

Resolução:

Perfeitamente. As duas sintaxes são equivalentes, sendo que a segunda indica as colunas de forma explícita e a primeira utiliza as colunas na ordem definida na tabela.

A sintaxe básica de uma instrução **INSERT INTO** é da seguinte forma:

INSERT INTO nome_da_tabela (coluna1, coluna2, coluna3, ...) **VALUES** (valor1, valor2, valor3, ...);

Esse comando permite inserir os valores indicados após VALUES nas colunas indicadas.

Caso você esteja inserindo um registro completo, isto é, com todas as colunas, você não precisa indicar quais as colunas, mas apenas inserir os dados na mesma ordem das colunas da tabela. Assim, a sintaxe a seguir também é válida:

INSERT INTO nome_da_tabela **VALUES** (valor1, valor2, valor3, ...);

Gabarito: **Certo.**

3. LÓGICA DE TRÊS ESTADOS

No SQL, o tratamento de valores nulos introduz uma lógica de **três estados**, diferentemente da lógica booleana tradicional (que possui apenas TRUE e FALSE). Isso acontece porque um valor NULL representa um dado desconhecido, e qualquer operação envolvendo NULL pode resultar em um terceiro estado chamado "UNKNOWN".

Os estados da lógica de três valores são:

- **TRUE (T - Verdadeiro)** → Quando uma condição é verdadeira.
- **FALSE (F - Falso)** → Quando uma condição é falsa.
- **UNKNOWN (? - Desconhecido)** → Quando há incerteza (geralmente devido a valores NULL).

Vejamos a tabela verdade para a avaliação dessa lógica:

NOT A	A	B	A AND B	A OR B
F	T	T	T	T
F	T	F	F	T
F	T	?	?	T
T	F	T	F	T
T	F	F	F	F
T	F	?	F	?
?	?	T	?	T
?	?	F	F	?
?	?	?	?	?

Esquema 19 - Lógica de três estados.

45- **(FGV - 2023 – SRFB– Auditor Fiscal da Receita Federal)** Os principais Sistemas Gerenciadores de Bancos de Dados oferecem total suporte à linguagem SQL; um aspecto importante da implementação do SQL é o tratamento para valores nulos, quando a lógica admite três estados.

T – true

F – false

? – unknown

Nesse contexto, considere as expressões lógicas a seguir.

I. (T OR F) AND (? OR T)

II. T AND ((? OR F) OR ?)

III. NOT (? AND (? AND ?))

Com relação às expressões acima, está correto afirmar que o valor final é unknown (?) em

a) I, apenas.

b) I e II, apenas.

c) I e III, apenas.

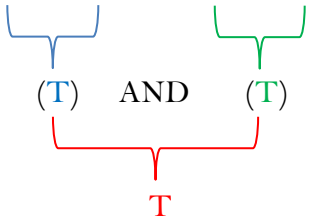
d) II e III, apenas.

e) I, II e III.

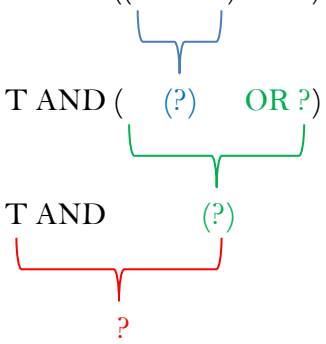
Resolução:

Vamos avaliar as expressões:

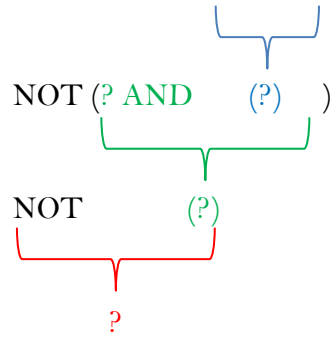
I. (T OR F) AND (? OR T)



II. T AND ((? OR F) OR ?)



III. NOT (? AND (? AND ?))

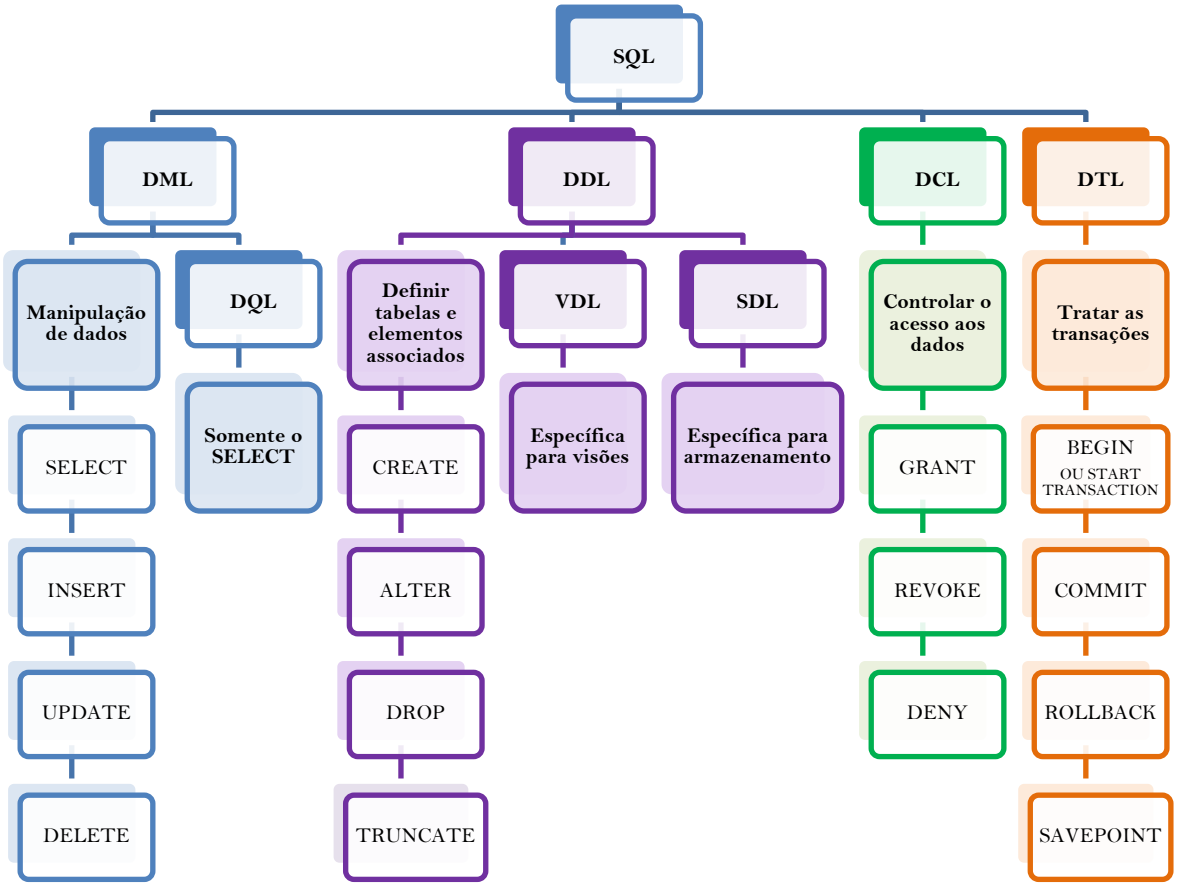


Logo, os resultados são desconhecidos em II e III.

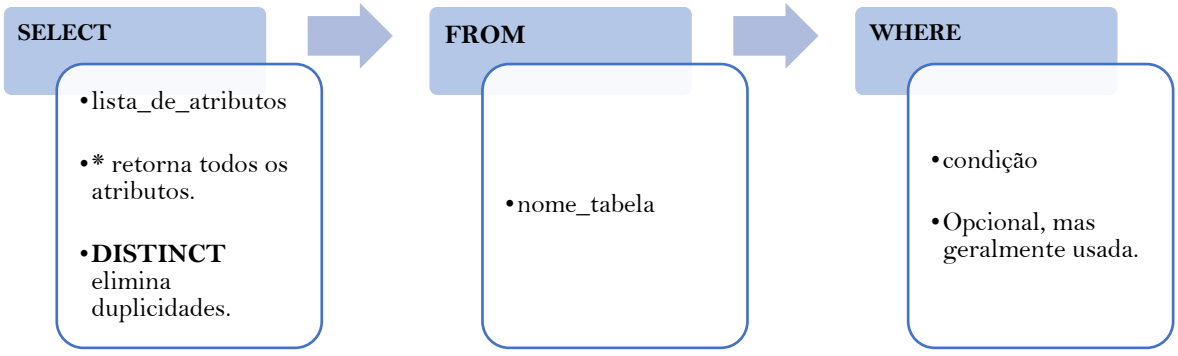
Gabarito: **Letra D.**

4. ESQUEMAS DE AULA

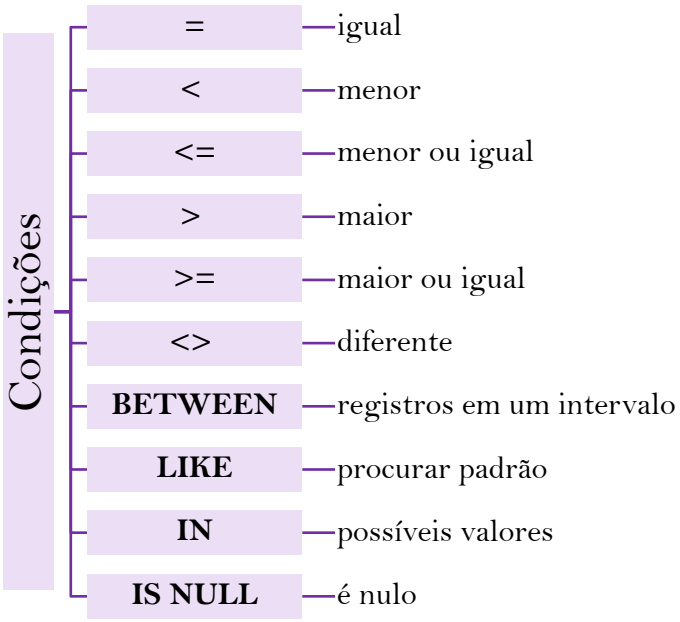
Linguagem SQL e subdivisões



Sintaxe básica da instrução SELECT



Condições na cláusula WHERE



Operador LIKE e exemplos

Operador LIKE	Procurar padrão em uma coluna
%	Substitui um número qualquer de 0 ou mais caracteres.
_	Substitui um único caractere.
LIKE 'A%'	Qualquer string que inicie com A.
LIKE '%A'	Qualquer string que termine com A.
LIKE '%A%'	Qualquer string que tenha A em qualquer posição.
LIKE 'A_'	String de dois caracteres que tenha a primeira letra A e o segundo caractere seja qualquer outro.
LIKE '_A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja a letra A.
LIKE '_A_'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
LIKE '%A_'	Qualquer string que tenha a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '_A%'	Qualquer string que tenha a letra A na segunda posição e o primeiro caractere seja qualquer outro caractere.
LIKE '_ _ _'	Qualquer string com exatamente três caracteres.
LIKE '_ _ _%'	Qualquer string com pelo menos três caracteres.
LIKE '%"%'	Qualquer string que tenha o caractere " em qualquer posição.

Cláusulas para definir mais de uma condição e negação de condição

AND

- Registros em que todas as condições são verdadeiras.
- **SELECT** coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição1 **AND** condição2 **AND** condição3 ...;

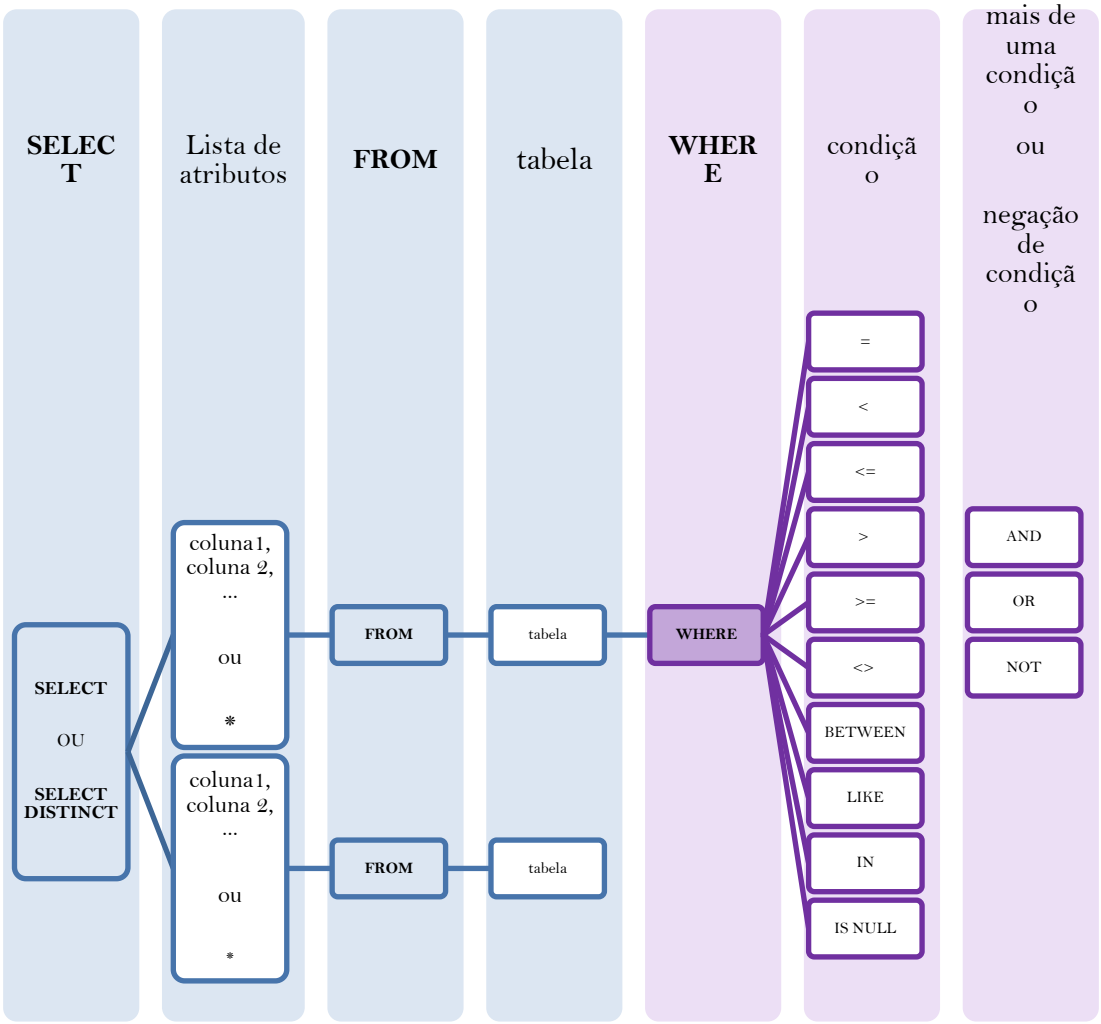
OR

- Registros em que pelo menos uma das condições é verdadeira.
- **SELECT** coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE** condição1 **OR** condição2 **OR** condição3 ...;

NOT

- Registros que não satisfazem uma condição.
- **SELECT** coluna1, coluna2, ... **FROM** nome_da_tabela **WHERE NOT** condição;

Instrução SELECT

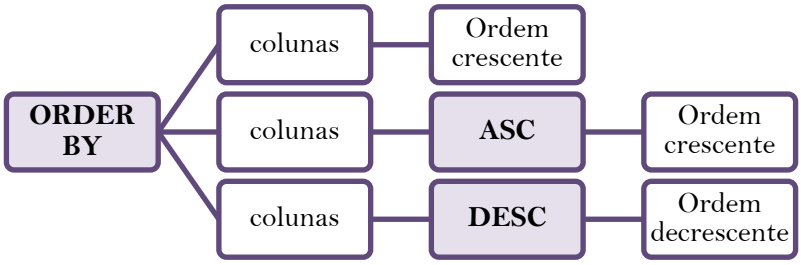


Atribuição de alias

Atribuição de alias

- Nome temporário a uma tabela ou coluna
- Tornar os nomes das colunas mais legíveis
- Existe apenas para a duração da consulta
- Cláusula AS (pode ser omitida)

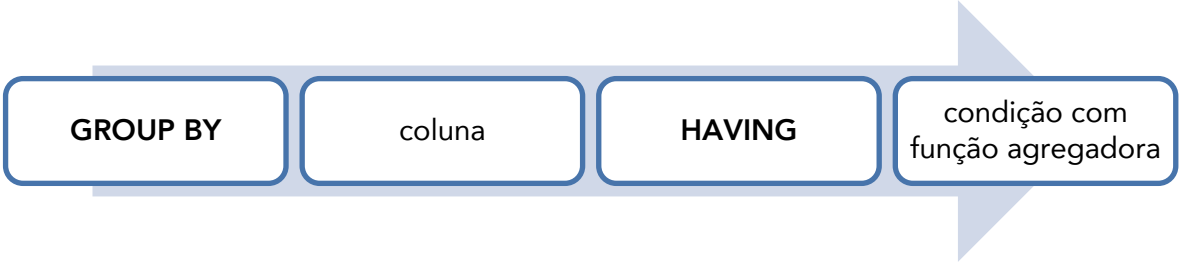
Cláusula ORDER BY



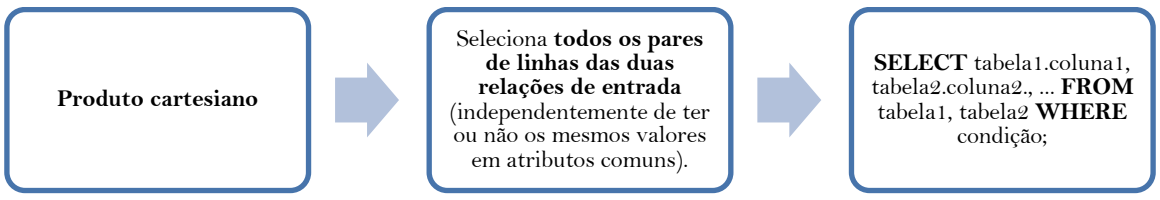
Funções de agregação

FUNÇÃO	RETORNO
MIN	Menor valor de uma coluna.
MAX	Maior valor de uma coluna.
COUNT	Número de linhas que atendem a um critério.
AVG	Média dos valores de uma coluna numérica.
SUM	Soma dos valores de uma coluna numérica.

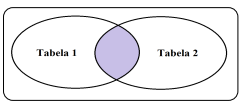
Cláusula GROUP BY e HAVING



Produto Cartesiano

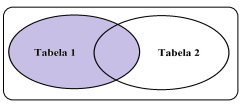


Tipos de Join



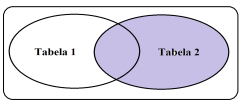
INNER JOIN (ou simplemente JOIN)

- Retorna somente os registros que possuem valores relacionados em ambas as tabelas, isto é, as intersecções.



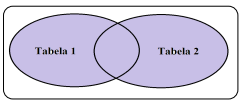
LEFT JOIN (ou LEFT OUTER JOIN)

- Retorna todos os registros da tabela da esquerda, e os registros relacionados da tabela da direita.
- Preenche campos não relacionados na tabela da direita com NULL.



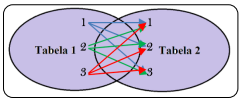
RIGHT JOIN (ou RIGHT OUTER JOIN)

- Retorna todos os registros da tabela da direita, e os registros relacionados da tabela da esquerda.
- Preenche campos não relacionados na tabela da esquerda com NULL



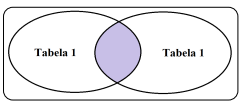
FULL OUTER JOIN

- Retorna todos os registros, independente de relação.
- Preenche campos não relacionados em qualquer das tabelas com NULL.



CROSS JOIN

- Retorna todos os registros da primeira relacionados com todos os registros da segunda.
- É o produto cartesiano.



SELF JOIN

- União de uma tabela com ela mesma.

Operadores de conjuntos

OPERADOR	RETORNO
UNION	Todas as linhas pertencentes as consultas envolvidas, sem as repetições.
UNION ALL	Todas as linhas pertencentes as consultas envolvidas, incluindo as repetições.
INTERSECT	Linhas que estão na primeira e na segunda consulta. Intersecção, sem repetições.
EXCEPT	Linhas que estão na primeira, mas não estão na segunda, sem repetições.

Consultas aninhadas e EXISTS (e NOT EXISTS)

Consulta aninhada

- Uma subconsulta, consulta interna ou seleção interna é uma consulta que está aninhada dentro de uma instrução SELECT, INSERT, UPDATE ou DELETE ou em outra subconsulta.

- As subconsultas podem ser comparadas com a consulta externa com o uso de operadores IN (ou NOT IN), ANY, ALL ou EXISTS (ou NOT EXISTS), além dos operadores básicos =, <, <=, >, >=, <>.

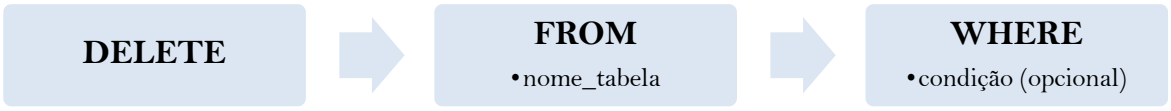
Cláusulas Especiais

- **ANY**: retorna TRUE se qualquer um dos valores da subconsulta atender a condição.
- **ALL**: retorna TRUE se todos os valores da subconsulta atenderem a condição.
- **EXISTS**: retorna TRUE se a subconsulta retornar um ou mais registros.
- **NOT EXISTS**: retorna TRUE se a subsconsulta não retornar nenhum registro.

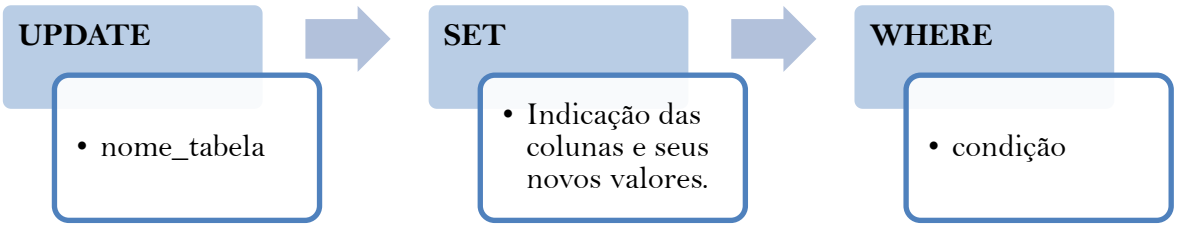
Cláusulas Especiais

OPERADOR	RETORNO
CASE	Percorre condições e retorna um valor para a 1ª condição atendida.
TOP, LIMIT ou FETCH FIRST	Especifica o número de registros a serem retornados.
OFFSET	Pula um número de registros antes de começar a exibir.
SELECT INTO	Copia dados de uma tabela para uma nova tabela

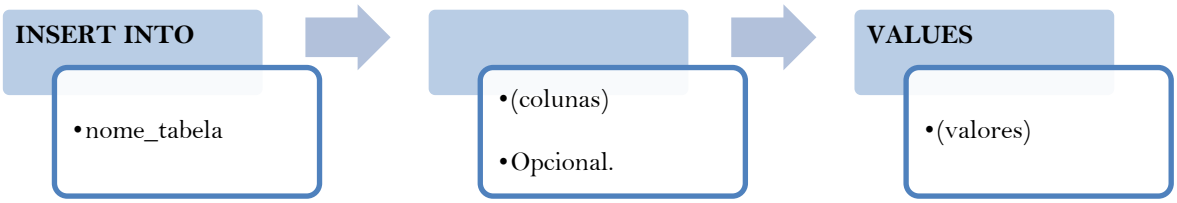
Sintaxe básica da instrução DELETE



Sintaxe básica da instrução UPDATE



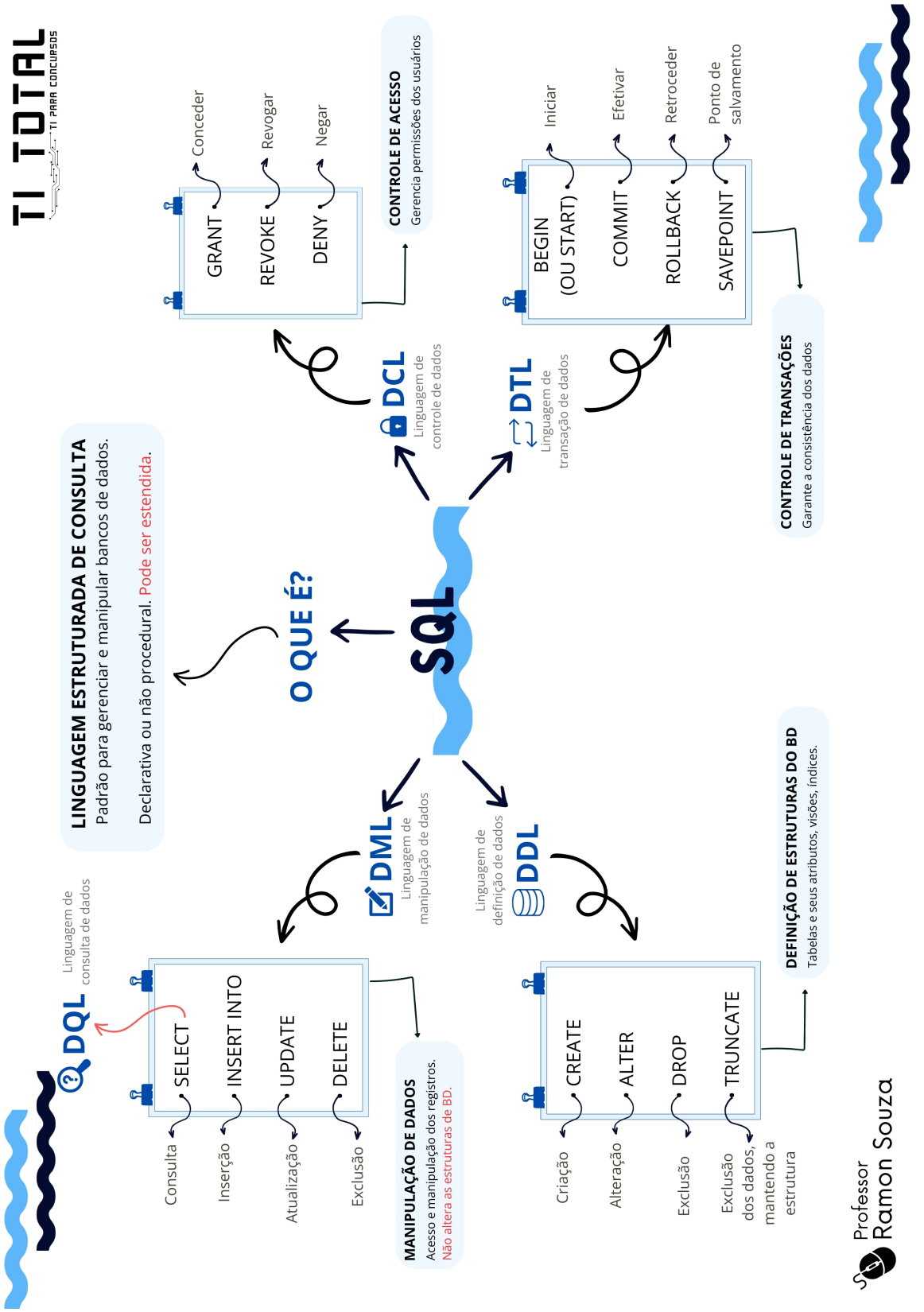
Sintaxe básica da instrução INSERT INTO



Lógica de três estados

NOT A	A	B	A AND B	A OR B
F	T	T	T	T
F	T	F	F	T
F	T	?	?	T
T	F	T	F	T
T	F	F	F	F
T	F	?	F	?
?	?	T	?	T
?	?	F	F	?
?	?	?	?	?

5. MAPA MENTAL



6. CHEAT SHEET (FOLHA DE CÓDIGO)

SQL (DML) - CHEAT SHEET

TI TOTAL TI PARA CONCURSOS

DADOS DE AMOSTRA

pais

id	nome	populacao	area
1	França	66600000	640680
2	Alemanha	80700000	357000
...	...	...	...

cidade

id	nome	id_pais	populacao	nota
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

CONSULTA EM ÚNICA TABELA

Buscar todas as colunas da tabela pais:
```
SELECT *
FROM pais;
```

Buscar colunas id e nome da tabela cidade:
```
SELECT id, nome
FROM cidade;
```

Buscar notas distintas da tabela cidade:
```
SELECT DISTINCT(nota)
FROM cidade;
```

FILTRANDO OS RESULTADOS

OPERADORES DE COMPARAÇÃO

=	<	<=
<> ou !=	>	>

Buscar nome de cidades com nota maior que 3:
```
SELECT nome
FROM cidade
WHERE nota > 3;
```

Buscar nomes de cidades que não tem nota 5:
```
SELECT nome
FROM cidade
WHERE nota <> 5;
```

PADRÃO TEXTUAL

LIKE busca um padrão textual em uma coluna.
- **%** substitui um nº qualquer de caracteres.
- **_** substitui um único caractere.

Buscar nomes de cidades que começam com 'P':
```
SELECT nome
FROM cidade
WHERE nome LIKE 'P%';
```

Buscar nomes de cidades que começam com qualquer letra seguidos de 'ublin':
```
SELECT nome
FROM cidade
WHERE nome LIKE '_ublin';
```

OUTROS OPERADORES

BETWEEN busca registros em intervalo.
Buscar nomes de cidades que possuam nota no intervalo de 3 a 5:
```
SELECT nome
FROM cidade
WHERE nota BETWEEN 3 AND 5;
```

IN busca registros em uma lista de valores.
Buscar nomes de cidades que possuam nota entre 4 ou 5:
```
SELECT nome
FROM cidade
WHERE nota IN (4, 5);
```

IS NULL testa se um valor é nulo.
Buscar nomes de cidades que não possuam valor em nota:
```
SELECT nome
FROM cidade
WHERE nota IS NULL;
```

NEGAÇÃO DE CONDIÇÃO

Buscar nomes de cidades que não possuam notas 1 e 2:
```
SELECT nome
FROM cidade
WHERE nota NOT IN (1, 2);
```

COMBINAÇÃO DE CONDIÇÕES

Buscar nomes de cidades que possuam nota 5 E população menor que 300 mil:
```
SELECT nome
FROM cidade
WHERE nota = 5
    AND populacao < 300000;
```

Buscar cidades que comecem com 'R' OU 'S':
```
SELECT nome
FROM cidade
WHERE nome LIKE 'R%'
      OR nome LIKE 'S%';
```

ALIASES

COLUNAS
```
SELECT nome AS nome_cidade FROM cidade;
```

TABELAS
```
SELECT p.nome, c.nome FROM cidade AS c
JOIN pais AS p ON c.id_pais = p.id;
```

ORDENAÇÃO

Buscar cidades ordenados pela nota na forma padrão crescente:
```
SELECT nome FROM cidade
ORDER BY nota [ASC];
```

Buscar cidades ordenados pela nota decrescente:
```
SELECT nome FROM cidade
ORDER BY nota DESC;
```

TESTAR CONDIÇÕES

Retornar o nome das cidades e uma classificação textual avaliada com base em condições de nota:
```
SELECT nome,
CASE
    WHEN nota > 3 THEN 'Avaliação boa'
    WHEN nota = 3 THEN 'Avaliacão Média'
    ELSE 'Avaliação ruim'
END AS Classificacao
FROM cidade;
```

AGREGAÇÃO E AGRUPAMENTO

FUNÇÕES

MIN	Menor valor de uma coluna
MAX	Maior valor de uma coluna
COUNT	Número de linhas
AVG	Média de valores de coluna numérica
SUM	Soma de valores de coluna numérica

Descobrir o número de cidades:
```
SELECT COUNT(*)
FROM cidade;
```

Descobrir o número de cidades sem notas nulas:
```
SELECT COUNT(nota)
FROM cidade;
```

Buscar a menor e a maior população dos países:
```
SELECT MIN(populacao), MAX(populacao)
FROM pais;
```

GRUPOS

GROUP BY agrupa registros que possuem os mesmos valores em colunas especificadas.
Buscar a população total de cidades de cada país:
```
SELECT id_pais, SUM(populacao)
FROM cidade
GROUP BY id_pais;
```

HAVING define uma condição para os grupos, operando com as funções agregadas.

Descobrir a classificação média das cidades nos países se a média estiver acima de 3:
```
SELECT id_pais, AVG(nota)
FROM cidade
GROUP BY id_pais
HAVING AVG(nota) > 3;
```

Professor Ramon Souza

Traduzido e adaptado de LearnSQL.com

SQL (DML) - CHEAT SHEET

TI TOTAL TI PARA CONCURSOS

CONSULTA EM VÁRIAS TABELAS

PRODUTO CARTESIANO (CROSS JOIN)

O **produto cartesiano** (ou o CROSS JOIN) faz o cruzamento de todos os pares de linhas das tabelas, independente de correspondência.
```
SELECT *
FROM cidade, pais;
```

```
SELECT *
FROM cidade
CROSS JOIN pais;
```

cidade			pais	
id	nome	id_pais	id	nome
1	Paris	1	1	França
1	Paris	1	2	Alemanha
2	Berlin	2	1	França
2	Berlin	2	2	Alemanha
...	...	...	...	...

INNER JOIN (JOIN)

O **INNER JOIN** retorna somente os registros que possuem valores relacionados em ambas as tabelas, isto é, as intersecções.
```
SELECT *
FROM cidade
INNER JOIN pais ON cidade.id_pais = pais.id;
```

cidade			pais	
id	nome	id_pais	id	nome
1	Paris	1	1	França
2	Berlin	2	2	Alemanha
3	Warsaw	4	3	Islândia

LEFT JOIN (LEFT OUTER JOIN)

O **LEFT JOIN** retorna todos os registros da tabela da esquerda e os correspondentes da direita. Se não houver correspondência, os campos da direita serão preenchidos com NULL.
```
SELECT *
FROM cidade
LEFT JOIN pais ON cidade.id_pais = pais.id;
```

cidade			pais	
id	nome	id_pais	id	nome
1	Paris	1	1	França
2	Berlin	2	2	Alemanha
3	Warsaw	4	NULL	NULL

RIGHT JOIN (RIGHT OUTER JOIN)

O **RIGHT JOIN** retorna todos os registros da tabela da direita e os correspondentes da esquerda. Se não houver correspondência, os campos da esquerda serão preenchidos com NULL.
```
SELECT *
FROM cidade
RIGHT JOIN pais ON cidade.id_pais = pais.id;
```

cidade			pais	
id	nome	id_pais	id	nome
1	Paris	1	1	França
2	Berlin	2	2	Alemanha
NULL	NULL	NULL	3	Islândia

FULL JOIN (FULL OUTER JOIN)

O **FULL JOIN** retorna todas as linhas de ambas as tabelas. Se não houver correspondência na outra tabela, os valores serão retornados como NULL.
```
SELECT *
FROM cidade
FULL JOIN pais ON cidade.id_pais = pais.id;
```

cidade			pais	
id	nome	id_pais	id	nome
1	Paris	1	1	França
2	Berlin	2	2	Alemanha
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Islândia

OPERAÇÕES DE CONJUNTOS

Combinam resultados de duas ou mais consultas.

ciclistas			skatistas		
id	nome	pais	id	nome	pais
1	João	BR	1	João	BR
2	Maria	BR	2	Júlio	BR
3	Peter	US	3	Zaz	FR

UNION

O **UNION** combina os resultados de dois conjuntos de dados e remove duplicatas. O **UNION ALL** não remove as duplicatas.
Buscar o nome de ciclistas e de skatistas do Brasil:
```
SELECT nome FROM ciclistas
WHERE pais = 'BR'
UNION / UNION ALL
SELECT nome  FROM skatistas
WHERE pais = 'BR'
```

INTERSECT

O **INTERSECT** retorna somente as linhas que aparecem em ambos os resultados.
Buscar o nome dos brasileiros que são ciclistas e skatistas:
```
SELECT nome FROM ciclistas
WHERE pais = 'BR'
INTERSECT
SELECT nome FROM skatistas
WHERE pais = 'BR'
```

EXCEPT

O **EXCEPT** retorna linhas que aparecem no primeiro resultado e não aparecem no segundo.
Buscar o nome dos brasileiros ciclistas que não são skatistas:
```
SELECT nome FROM ciclistas
WHERE pais = 'BR'
EXCEPT
SELECT nome FROM skatistas
WHERE pais = 'BR'
```

SUBCONSULTAS

Uma **subconsulta** é uma consulta aninhada dentro de outra consulta ou dentro de outra subconsulta.

VALORES SIMPLES

Retorna exatamente uma coluna e uma linha. Ela pode ser usada com operadores de comparação como =, <, <=, >, ou >=.
Buscar cidades com a mesma nota de Paris:
```
SELECT nome
FROM cidade
WHERE nota = (
        SELECT nota
        FROM cidade
        WHERE nome = 'Paris'
);
```

VALORES MÚLTIPLOS

Retorna várias colunas ou várias linhas. Esse tipo de subconsulta pode ser usado com os operadores **IN, EXISTS, ALL ou ANY**.
Buscar países que possuem ao menos uma cidade:
```
SELECT nome
FROM pais
WHERE EXISTS (
        SELECT *
        FROM cidade
        WHERE id_pais = pais.id
);
```

OUTROS COMANDOS DML

Excluir: DELETE FROM tabela WHERE...;

Atualizar UPDATE tabela SET coluna=valor WHERE...;

Inserir: INSERT INTO tabela VALUES (valor1, valo2, ...); ou INSERT INTO tabela (coluna1, coluna2) VALUES (valor1, valor2);

Professor Ramon Souza

Traduzido e adaptado de LearnSQL.com

7. REFERÊNCIAS

DEVMEDIA. **SQL: EXISTS**. Disponível em: <https://www.devmedia.com.br/sql-exists/41176>. Acesso em: 08 nov. 2021.

DEVMEDIA. **SQL: Utilizando o Operador UNION e UNION ALL**. Disponível em: <https://www.devmedia.com.br/sql-utilizando-o-operador-union-e-union-all/37457>. Acesso em: 08 nov. 2021.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de Banco de Dados**. 6ed. São Paulo: Pearson Addison Wesley, 2011.

SOFTBLUE. **Curso SQL Completo**. Disponível em: <http://www.softblue.com.br/site/curso/id/3/CURSO+DE+SQL+COMPLETO+BASICO+AO+AVANCADO+ON+LINE+BD03+GRATIS>. Acesso em: 22 out. 2018.

W3SCHOOLS. **SQL Tutorial**. Disponível em: <https://www.w3schools.com/sql/>. Acesso em: 19 out. 2018.

LEARNSQL. **SQL Basics Cheat Sheet.** Disponível em: <https://learnsql.com/blog/sql-basics-cheat-sheet/>. Acesso em: 09 fev. 2025.