**GOVERNO DO DISTRITO FEDERAL**

**SECRETARIA DE ESTADO DE ECONOMIA**

**COMPANHIA DE PLANEJAMENTO DO DISTRITO FEDERAL – CODEPLAN**

**GERAÇÃO DE DADOS UTILIZANDO LINGUAGEM SQL**

Brasília (DF), fevereiro de 2021

CONTATO:

Telefones: 3342-2272 e 3342-2264

luiz.araujo@codeplan.df.gov.br

Sumário

[1 Introdução 1](#_Toc63243312)

[1.1 Conceito de Banco de Dados 1](#_Toc63243313)

[1.2 A linguagem SQL 1](#_Toc63243314)

[1.3 Vantagens Linguagem SQL 2](#_Toc63243315)

[1.4 Comandos SQL 3](#_Toc63243316)

[2 explorando o comando *select* 3](#_Toc63243317)

[2.1 Consultas básicas 3](#_Toc63243318)

[2.2 Usando *ALIAS* 4](#_Toc63243319)

[2.3 Cláusula ORDER BY 5](#_Toc63243320)

[2.4 Cláusula *DISTINCT* 5](#_Toc63243321)

[2.5 Criando grupos com a cláusula *CASE* 5](#_Toc63243322)

[2.6 Aplicando filtros a seleção 6](#_Toc63243323)

[2.7 Operadores e Precedência 7](#_Toc63243324)

[2.8 Agrupando Dados 8](#_Toc63243325)

[2.8.1 A instrução *GROUP BY* 9](#_Toc63243326)

[2.9 A instrução *HAVING* 9](#_Toc63243327)

[2.10 Junções de Tabelas ou Consultas 9](#_Toc63243328)

[2.11 Utilização de algumas funções para manipulação de dados 13](#_Toc63243329)

[3 PROCEDIMENTOS PARA INSTALAÇÃO DE FERRAMENTAS PARA CONSULTAS EM BANCO DE DADOS 13](#_Toc63243330)

[3.1 Como instalar o *DBeaver* 13](#_Toc63243331)

[3.2 Como configurar o *DBeaver* 14](#_Toc63243332)

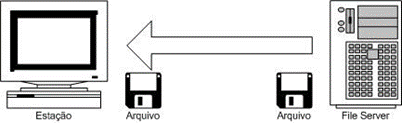
1. Introdução
   1. Conceito de Banco de Dados

Antes do surgimento dos bancos de dados, as informações eram gravadas em arquivos de dados com formato texto (*flat file*) ou binário proprietário, como por exemplo, o ***Clipper*** com ***dbf***, ***Cobol*** com ***dat***.

Estes arquivos eram copiados em um servidor e disponibilizados aos usuários por acesso em rede, ou seja, era dado o acesso ao diretório onde os arquivos estavam, para que o usuário pudesse acessá-los e alterá-los.

O principal problema gerado por esta forma de trabalho com arquivos de dados era a falta de controle do que era feito com eles, já que era possível controlar apenas quem acessava, mas não o que era alterado.

Outra característica é que os dados eram processados localmente, ou seja, ao acessar o arquivo de dados o usuário carregava e processava as informações na memória do computador em que estava trabalhando, como mostra a figura abaixo.



Para arquivos de dados pequenos e com acesso restrito a poucas pessoas este modelo funcionou bem, mas com o crescimento de usuários de dados e de volume de informações, este modelo passou a gerar uma série de problemas, como lentidão de rede e falta de controle de alteração de arquivos.

Para solucionar este problema foram criados na década de 70 os primeiros servidores de banco de dados, chamada de ***client-server***, onde os dados residem em um servidor e o usuário, chamado de cliente, envia ao servidor um conjunto de comandos que são interpretados e processados no próprio servidor, e recebe de volta o resultado final do processamento.

Após o surgimento dos servidores foi criado um padrão chamado ***Database management system*** (***DBMS***), ou em bom português brasileiro ***Sistema Gerenciador de*** ***Banco de Dados Relacional*** (***SGBD***).

* 1. A linguagem SQL

No início cada SGBD tinha o seu próprio conjunto de comandos e instruções. Mas a IBM, ainda na década de 70, criou um conjunto de comandos e instruções que se tornaram padrão e são utilizados até hoje em todos os SGBD. Este conjunto de comandos e instruções passou a ser chamado ***Structured Query Language***, ou em bom português brasileiro ***Linguagem de Consulta Estruturada***, ou simplesmente ***SQL***.

A linguagem SQL pode ter vários enfoques:

Linguagem interativa de consulta (***query AdHoc***): Através de comandos SQL os usuários podem montar consultas poderosas, sem a necessidade da criação de um programa, podendo utilizar ferramentas ***front end[[1]](#footnote-1)*** para a montagem de relatórios;

Linguagem de programação para acesso às bases de dados: Comandos SQL embutidos em programas de aplicação (escritos em C, C++, Java e Visual Basic, entre outros) acessam os dados armazenados em um SGBD.

Linguagem de administração de banco de dados: O responsável pela administração do banco de dados (DBA) utiliza comandos SQL para realizar tarefas relacionadas com a manutenção dos bancos de dados do SGBD.

Linguagem de consulta em ambiente cliente/servidor: Os programas sendo processados nos computadores dos clientes (***front ends***) usam comandos SQL para se comunicarem, através de uma rede, com um SGBD, para processar informações em uma máquina servidora (***back end***);

Linguagem para bancos de dados distribuídos: A linguagem SQL é também a linguagem padrão para a manipulação de dados em uma base de dados distribuída.

Linguagem de definição de dados (DDL): Permite ao usuário a definição da estrutura e organização dos dados armazenados, e das relações existentes entre eles.

Linguagem de manipulação de dados (DML): Permite a um usuário, ou a um programa de aplicação, a execução de operações de inclusão, remoção, seleção ou atualização de dados previamente armazenados na base de dados.

Controle de acesso: Protege os dados de manipulações não autorizadas.

Integridade dos dados: Auxilia no processo de definição da integridade dos dados, protegendo contra corrupções e inconsistências geradas por falhas do sistema de computação, ou por erros nos programas de aplicação.

* 1. Vantagens Linguagem SQL
* Independência de fabricante;
* A linguagem é adotada por praticamente todos os SGBD existentes no mercado, além de ser uma linguagem padronizada. Com isso, pelo menos em tese[[2]](#footnote-2), é possível mudar de SGBD sem se preocupar em alterar os programas de aplicação;
* Pode ser utilizada tanto em máquinas Intel rodando Windows, passando por workstations RISC rodando UNIX, até mainframes rodando sistemas operacionais proprietários;
* Portabilidade entre plataformas de software (ou seja, é possível migrar de Windows para Linux, ou Oracle para PostgreSQL);
* Redução dos custos com treinamento;
* Com base no item anterior, as aplicações podem se movimentar de um ambiente para o outro sem que seja necessária uma reciclagem da equipe de desenvolvimento;
* Usa inglês estruturado de alto nível;
* É formado por um conjunto bem simples de sentenças em inglês, oferecendo um rápido e fácil entendimento;
* Permite consultas interativas;
* Permite aos usuários acesso fácil e rápido aos dados a partir de um ***front end*** que permita a edição e a submissão de comandos SQL;
* Múltiplas visões dos dados;
* Permite ao criador do banco de dados levar diferentes visões dos dados aos diferentes usuários;
* Definição dinâmica dos dados;
* Através da linguagem SQL pode-se alterar, expandir ou incluir dinamicamente as estruturas dos dados armazenados, com máxima flexibilidade.
  1. Comandos SQL

Os comandos SQL são separados em três famílias:

**Data Definition Language**

**(DDL)**

Utilizada para criar, deletar e alterar objetos como *views*, *databases*, *stored procedures*, etc.

Os comandos iniciam com:

CREATE

ALTER

DROP

**Data Control Language**

**(DCL)**

Permite controlar a segurança de dados, definindo quem pode fazer o quê no banco de dados.

Os comandos iniciam com:

GRANT

REVOKE

DENY

**Data Manipulation Language**

**(DML)**

Permite recuperar, incluir, remover ou modificar informações no banco de dados.

Os comandos iniciam com:

SELECT

INSERT

UPDATE

DELETE

Nesta apostila abordaremos o DML ***SELECT*** e suas principais funções.

1. explorando o comando *select*
   1. Consultas básicas

A principal estrutura do comando ***SELECT*** é a seguinte:

Os resultados gerados a partir de uma ***SELECT*** são gravados em uma tabela temporária salva no banco de dados, que é excluída após a sua cópia para a máquina local ou o fechamento da conexão local com o SGBD.

A maioria dos SGBD não faz distinção entre letras maiúsculas ou minúsculas ou colocar tudo em um único parágrafo ou em mais de um. Mas para fins de documentação a padronização sugerida é digitar cláusulas e funções com letras maiúsculas, mantendo cada um em parágrafos separados, e os demais, como nomes de tabelas e colunas, com letras minúsculas e em um único parágrafo.

**SELECT**

col1, col2, ..., coln

**FROM**

tabela1, ..., tabelan;

É possível também inserir comentários, iniciando o parágrafo com dois sinais de menos (--).

--Estrutura de um select;

**SELECT**

col1, col2, ..., coln

**FROM**

tabela1, ..., tabelan;

Usualmente cada bloco de programação é finalizado por ponto-e-vírgula (;). Mas há SGBD que aceitam outras formas. Por exemplo o SGBD ***SQL Server*** da Microsoft aceita a expressão ***go*** para identificação dos blocos de programação.

Por exemplo, tomando como base a tabela ***dom2018\_33ras***, no esquema ***pdad***, execute os seguintes comandos:

**SELECT**

A01ra

**FROM**

pdad.dom2018\_33ras;**SELECT**

A01ra,

B01

**FROM**

pdad.dom2018\_33ras;**SELECT**

d.A01ra,

d.B01

**FROM**

pdad.dom2018\_33ras;

Na tabela **dom2018\_33ras** existem 130 colunas, onde na programação à esquerda é solicitado a apresentação dos dados da coluna na posição 1, no caso a coluna **A01ra**. Na programação ao centro é solicitado a apresentação dos dados das colunas nas posições 1 e 6, no caso as colunas **A01ra** e **B01**, respectivamente. Por fim na programação à direita solicitamos a apresentação de todos os dados de todas as colunas.

* 1. Usando *ALIAS*

***ALIAS*** são literalmente apelidos, pois algumas colunas em um **SELECT** podem ser resultado da combinação de duas ou mais colunas. Por exemplo, para ver a soma da quantidade de cômodos que estão servindo permanentemente de dormitório com a quantidade de banheiros e/ou sanitários o domicílio possui, basta executar a seguinte programação:

**SELECT**

B12 + B13

**FROM**

pdad.dom2018\_33ras;

Se não for especificado o nome da coluna o sistema atribuirá um, que com certeza não servirá. Para o usuário atribuir um nome, ele pode proceder de duas formas:

--Forma 1

**SELECT**

B12 + B13 **as** qtd\_dormi\_banh

**FROM**

pdad.dom2018\_33ras;--Forma 2

**SELECT**

B12 + B13 **as** "Qtd. de Dormitórios e banheiros"

**FROM**

pdad.dom2018\_33ras;

***ALIAS*** também é utilizado para identificar e distinguir diversas tabelas em uma consulta, como será visto em 2.9.

* 1. Cláusula ORDER BY

Quando os dados solicitados aparecem na tela, eles são apresentados na maioria das vezes na ordem em que encontram-se gravados na tabela consultada.

Para ordená-los, basta inserir a cláusula ORDER BY e escrever o nome da(s) coluna(s) que deseja classificar, como mostra os comandos abaixo.

--Forma 1

**SELECT**

A01ra,

B01 + B02 **as** qtd\_dormi\_banh

**FROM**

pdad.dom2018\_33ras

**ORDER** **BY**

A01ra;--Forma 2

**SELECT**

A01ra,

B01 + B02 **as** "Qtd. de Dormitórios e banheiros"

**FROM**

pdad.dom2018\_33ras

**ORDER** **BY**

A01ra;

* 1. Cláusula *DISTINCT*

Nos comandos executados acima, são mostradas todas as informações contidas na tabela consultada, independente se há ou não multiplicidade de linhas (repetição).

Com a cláusula ***DISTINCT*** é possível ver o conteúdo de uma ou mais colunas, eliminando possíveis multiplicidades.

Por exemplo, para ver quais foram os setores pesquisados na PDAD 2011, basta executar o comando abaixo:

**SELECT**

**DISTINCT cd\_dom\_ra**

**FROM**

**pdad.dom2011**

**ORDER BY cd\_dom\_ra;**

Para ver quais são as RA em que cada um dos setores está vinculado, basta executar o comando abaixo.

**SELECT**

**DISTINCT ra, cd\_dom\_ra**

**FROM**

**pdad.dom2011**

**ORDER BY ra, cd\_dom\_ra;**

Observe que os setores 201, 202 e 203 estão vinculados a RA de número 20 (Águas Claras).

* 1. Criando grupos com a cláusula *CASE*

A cláusula CASE permite que os valores retornados pelas consultas possam ser modificados caso sejam compatíveis com determinadas condições. Como exemplo podemos criar uma nova coluna contendo a seguinte regra:

* Se o campo **tp\_mor\_cor\_raca** (Cor ou Raça) da tabela **mor2011** estiver preenchido com os valores 1 ou 3, atribua o valor 1;
* Se o campo **tp\_mor\_cor\_raca** (Cor ou Raça) da tabela **mor2011** estiver preenchido com os valores 2, 4 ou 5, atribua o valor 2;
* Para todos os outros valores identificados na coluna **tp\_mor\_cor\_raca** (Cor ou Raça) da tabela **mor2011**, atribua valor 3;
* Salve os resultados em uma coluna com o nome *negro\_naonegro*.

Implementando as regras acima em SQL, a programação é a seguinte:

**SELECT**

**CASE WHEN tp\_mor\_cor\_raca = 1 THEN 1**

**WHEN tp\_mor\_cor\_raca = 3 THEN 1**

**WHEN tp\_mor\_cor\_raca = 2 THEN 2**

**WHEN tp\_mor\_cor\_raca = 4 THEN 2**

**WHEN tp\_mor\_cor\_raca = 5 THEN 2**

**ELSE 3**

**END AS negro\_naonegro**

**FROM** pdad.mor2011;

Também é possível a utilização de operadores e funções específicas para descrever as condições. Utilizando o operador ***IN***, a programação acima pode ser reescrita da seguinte forma:

**SELECT**

**CASE WHEN tp\_mor\_cor\_raca IN (1,3) THEN 1**

**WHEN tp\_mor\_cor\_raca IN (2,4,5) THEN 2**

**ELSE 3**

**END AS negro\_naonegro**

**FROM** pdad.mor2011;

Segue abaixo uma lista com os operadores e funções mais usadas e suas descrições.

|  |  |
| --- | --- |
| Operador | Descrição |
| = | Valor na coluna igual a um valor especificado |
| <> | Valor na coluna diferente de um valor especificado |
| > ou >= | Valor na coluna Maior ou Maior ou igual a um valor específico |
| < ou <= | Valor na coluna Menor ou Menor ou igual a um valor específico |
| Between | Valor na coluna Entre um intervalo de valores específicos |
| Not between | Valor na coluna Fora do intervalo de valores específicos |
| Is NULL e is not null | Valor na coluna É ou Não nulo |
| And | Valor na coluna atende a duas ou mais condições específicas |
| Or | Valor na coluna atende a al menos uma das condições |

* 1. Aplicando filtros a seleção

É possível reaxlizar consultas aplicando filtros específicos através da cláusula WHERE. Sua estrutura é a seguinte:

**SELECT**

col1, col2, ..., coln

**FROM**

tabela1, ..., tabelan

**WHERE**

condição1, ..., condiçãon;

Por exemplo, para ver as informações de todas as pessoas entrevistadas com idade superior a 18 anos na PDAD de 2013, basta executar a seguinte programação:

**SELECT**

\*

**FROM**

pdad.mor2013

**WHERE**

qt\_mor\_idade > 18;

Os mesmos operadores utilizados na cláusula ***CASE*** também podem ser aplicados em filtros, como mostram os exemplos abaixo.

|  |  |
| --- | --- |
| Operador | Exemplo |
| = | --Selecionar pessoas com idade igual a 18 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade = 18; |
| <> | --Selecionar pessoas com idade diferente a 18 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade <> 18; |
| > e >= | --Selecionar pessoas com idade maior ou igual a 18 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade >= 18; |
| < e <= | --Selecionar pessoas com rendimento menor ou igual a R$100,00;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** vl\_mor\_princ\_rend\_bruto +  vl\_mor\_outros\_rend\_bruto +  vl\_mor\_benef\_sociais <= 100; |
| Between | --Selecionar pessoas com idade entre 18 e 24 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade **BETWEEN** 18 **AND** 24; |
| Not between | --Selecionar pessoas que não tenham idade entre 18 e 24 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade **NOT** **BETWEEN** 18 **AND** 24; |
| Is NULL e is not null | --Selecionar pessoas que tenham o campo vl\_mor\_princ\_rend\_bruto vazio;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** vl\_mor\_princ\_rend\_bruto **is** **null**; |
| And | --Selecionar pessoas com idade menor que 18 anos e com algum rendimento menor ou igual a R$100,00;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade < 18  **AND** vl\_mor\_princ\_rend\_bruto +  vl\_mor\_outros\_rend\_bruto +  vl\_mor\_benef\_sociais<=100; |
| Or | --Selecionar pessoas com idade menor que 18 anos ou maior que 50 anos;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade < 18  **OR** qt\_mor\_idade > 50; |
| IN | --Selecionar pessoas com as seguintes idades: 5, 50 59 e 100;  **SELECT** \*  **FROM** pdad.mor2013  **WHERE** qt\_mor\_idade **in** (5,50,59,100); |

* 1. Operadores e Precedência

Assim como em cálculos matemáticos, existe uma precedência em processamento de colunas e valores. Por exemplo na equação 1+2\*3/4, o resultado será dois e meio uma vez que multiplicação e divisão são executadas antes da adição e subtração.

Como referência basta seguir a seguinte regra: **\*, /, %, +, -, AND e OR**.

Para definir uma precedência basta utilizar parênteses nos comandos. Por exemplo, qual a diferença entre as expressões abaixo?

Expressão1: A OR B AND C

Expressão 2: (A OR B) AND C?

No primeiro B e C tem que ser igual e A não faz diferença. No segundo exemplo A e B não fazem diferença, mas qualquer um deles deve ser igual com C.

* 1. Agrupando Dados

As funções listadas abaixo são utilizadas para totalizar, somar, gerar relatórios, estatísticas e outras informações de maneira resumida.

|  |  |
| --- | --- |
| Operador | Função |
| AVG | Média aritmética |
| COUNT | Conta o número de ocorrências de linhas |
| MAX | Maior valor na coluna |
| MIN | Menor valor na coluna |
| SUM | Soma todos os valores da coluna |
| STDEV | Desvio padrão estatístico na coluna |
| STDEVP | Desvio padrão populacional na coluna |
| VAR | Variação estatística dos valores da coluna |
| VARP | Variação populacional dos valores da coluna |

Para utilizar as funções de agregação, utilizamos a(s) coluna(s) a ser(em) agregada(s) dentro da função de agregação. Por exemplo:

**SELECT**

**MAX**(vl\_mor\_princ\_rend\_bruto),

**MIN**(vl\_mor\_princ\_rend\_bruto),

**AVG**(vl\_mor\_princ\_rend\_bruto),

**COUNT**(\*),

**COUNT**(**DISTINCT** vl\_mor\_princ\_rend\_bruto)

**FROM** pdad.mor2013;

Ao utilizar as funções de agregação os dados são agregados em uma única linha.

Deve-se tomar cuidado com o resultado das funções de agregação, com exceção do **COUNT**, pois elas desconsideram os valores nulos. O comando abaixo traz um exemplo.

**SELECT**

**AVG**(vl\_mor\_princ\_rend\_bruto) **as** media,

**SUM**(vl\_mor\_princ\_rend\_bruto) **as** soma,

**COUNT**(\*) **as** contagem,

**SUM**(**CASE** **WHEN** vl\_mor\_princ\_rend\_bruto **is** **null** **THEN** 1 **ELSE** 0 **END**) **as** contagem\_nulo

**FROM** pdad.mor2013;

Observe que se:

Mas:

Caso seja necessária a utilização de duas ou mais agregações, elas precisam ser especificadas por uma função de agregação, como veremos a seguir.

* + 1. A instrução *GROUP BY*

Nos casos em que colunas não agregadas serão utilizadas para criarem grupos, utilizamos a instrução ***GROUP BY***. Para utilizar esta instrução, é preciso definir qual coluna da consulta será utilizada para fazer a quebra, ou subgrupo das agregações solicitadas.

**SELECT**

tp\_mor\_cor\_raca,

**MAX**(vl\_mor\_princ\_rend\_bruto),

**MIN**(vl\_mor\_princ\_rend\_bruto),

**AVG**(vl\_mor\_princ\_rend\_bruto),

**COUNT**(\*),

**COUNT**(**DISTINCT** vl\_mor\_princ\_rend\_bruto)

**FROM** pdad.mor2013

**GROUP** **BY** tp\_mor\_cor\_raca;

É possível observar que a coluna **tp\_mor\_cor\_raca** foi incluída na consulta sem nenhuma função de agregação, portanto ela foi utilizada como grupo, gerando os dados anteriores agora agrupadas segundo as codificações de cor ou raça.

* 1. A instrução *HAVING*

Assim como podemos filtrar linhas de uma tabela com a instrução **WHERE**, podemos fazer o mesmo quando os dados foram agrupados utilizando **HAVING**.

A diferença básica entre o **where** e o **having** é que primeiro faz o filtro ao selecionar os registros para serem somados, sobre as linhas originais. O segundo faz a filtragem quando as agregações já foram efetuadas, portanto, sobre o valor agrupado. Observe os dois exemplos abaixo:

--Exemplo 01;

**SELECT**

cd\_dom\_ra,

**SUM**(vl\_mor\_princ\_rend\_bruto)

**FROM** pdad.mor2013

**WHERE** vl\_mor\_princ\_rend\_bruto > 100000000

**GROUP** **BY** cd\_dom\_ra;--Exemplo 02;

**SELECT**

cd\_dom\_ra,

**SUM**(vl\_mor\_princ\_rend\_bruto)

**FROM** pdad.mor2013

**GROUP** **BY** cd\_dom\_ra

**HAVING** **SUM**(vl\_mor\_princ\_rend\_bruto) > 100000000;

O primeiro exemplo não apresenta informação pois não há nenhuma pessoa pesquisada que tenha R$100.000.000 na coluna **vl\_mor\_princ\_rend\_bruto**.

No segundo observa-se duas RA’s porque a soma dos valores da coluna **vl\_mor\_princ\_rend\_bruto** ultrapassa o valor de R$100.000.000.

* 1. Junções de Tabelas ou Consultas

O cruzamento entre tabelas é necessário para que se possa extrair informações a partir do cruzamento de dados.

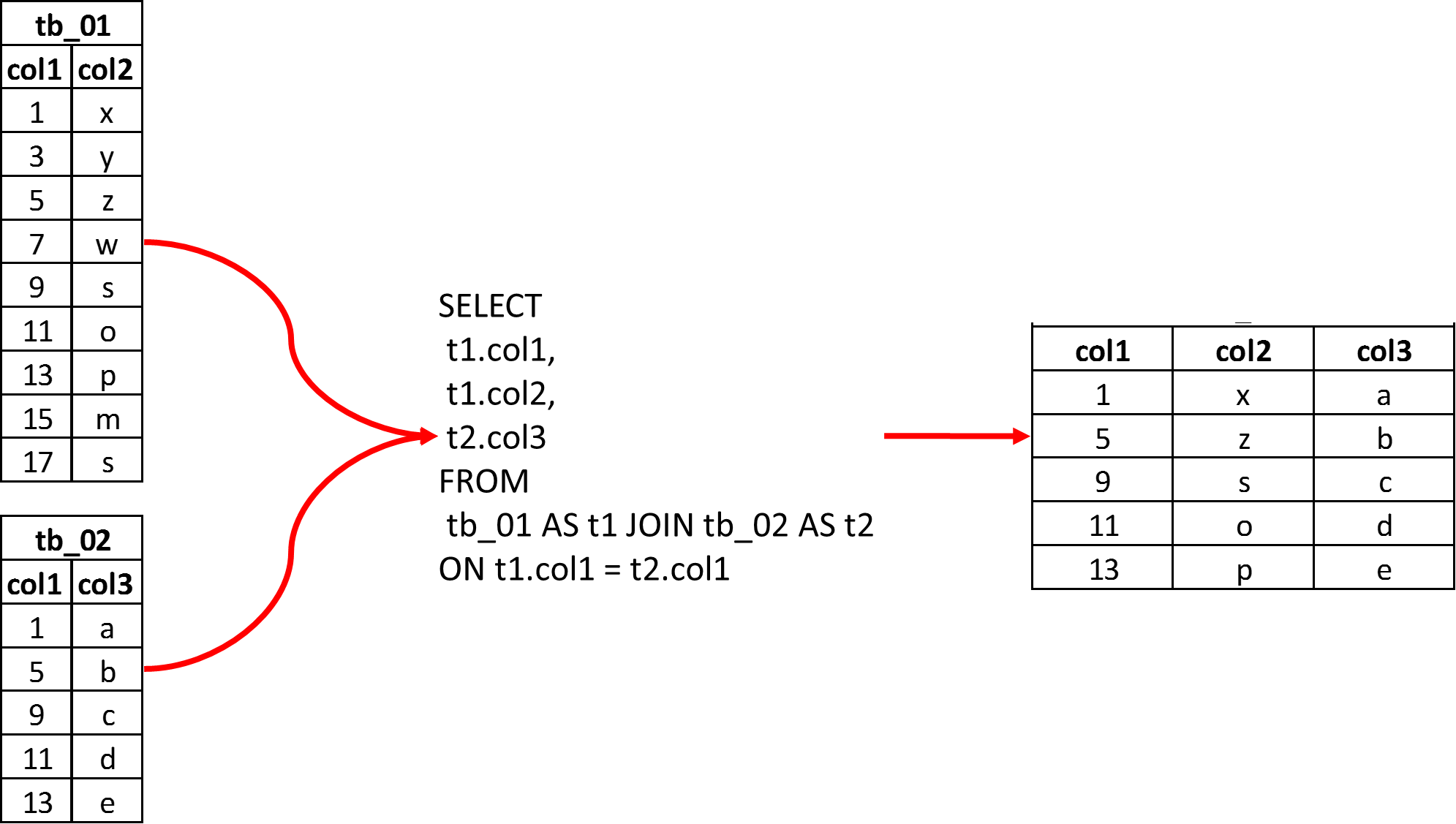
Por exemplo, para saber a quantidade de pessoas por sexo (feminino ou masculino) nas Regiões Administrativas com base na PDAD 2018, é necessário cruzar as informações da coluna **A01ra**, gravada na tabela **dom2018\_33ras** com as informações da coluna **E03** da tabela **mor2018\_33ras**.

É importante ressaltar a necessidade da utilização de uma ou mais colunas como chave para o cruzamento. Estas colunas devem ter o mesmo formato (colunas numéricas só podem ser cruzadas com colunas numéricas, por exemplo) e conter informações comuns entre si.

Dependendo do SGBD a não indicação da(s) coluna(s) chave(s) para o cruzamento pode resultar apenas em erro, ou pode resultar no travamento do banco de dados, uma vez que ao não indicar a(s) coluna(s) chave(s), o banco de dados pode cruzar todas as colunas das tabelas indicadas, gerando uma nova tabela tão grande que pode não ser suportado pelo SGBD.

A junção das tabelas é feita de três maneiras:

* **INNER JOIN**: ou simplesmente **JOIN**, traz os registros das tabelas consultadas, desde que os dados contidos na(s) coluna(s) chave sejam os mesmos.

****

A programação que pode ser utilizada no exemplo citado (quantidade de pessoas por sexo nas Regiões Administrativas com base na PDAD 2018), a programação pode ser feita da seguinte forma:

**SELECT**

d.A01ra,

**SUM**(**CASE** **WHEN** m.E03 <> 1 **then** 1 **else** 0 **end**) qtd\_fem,

**SUM**(**CASE** **WHEN** m.E03 = 1 **then** 1 **else** 0 **end**) qtd\_masc

**FROM** pdad.dom2018\_33ras **as** d

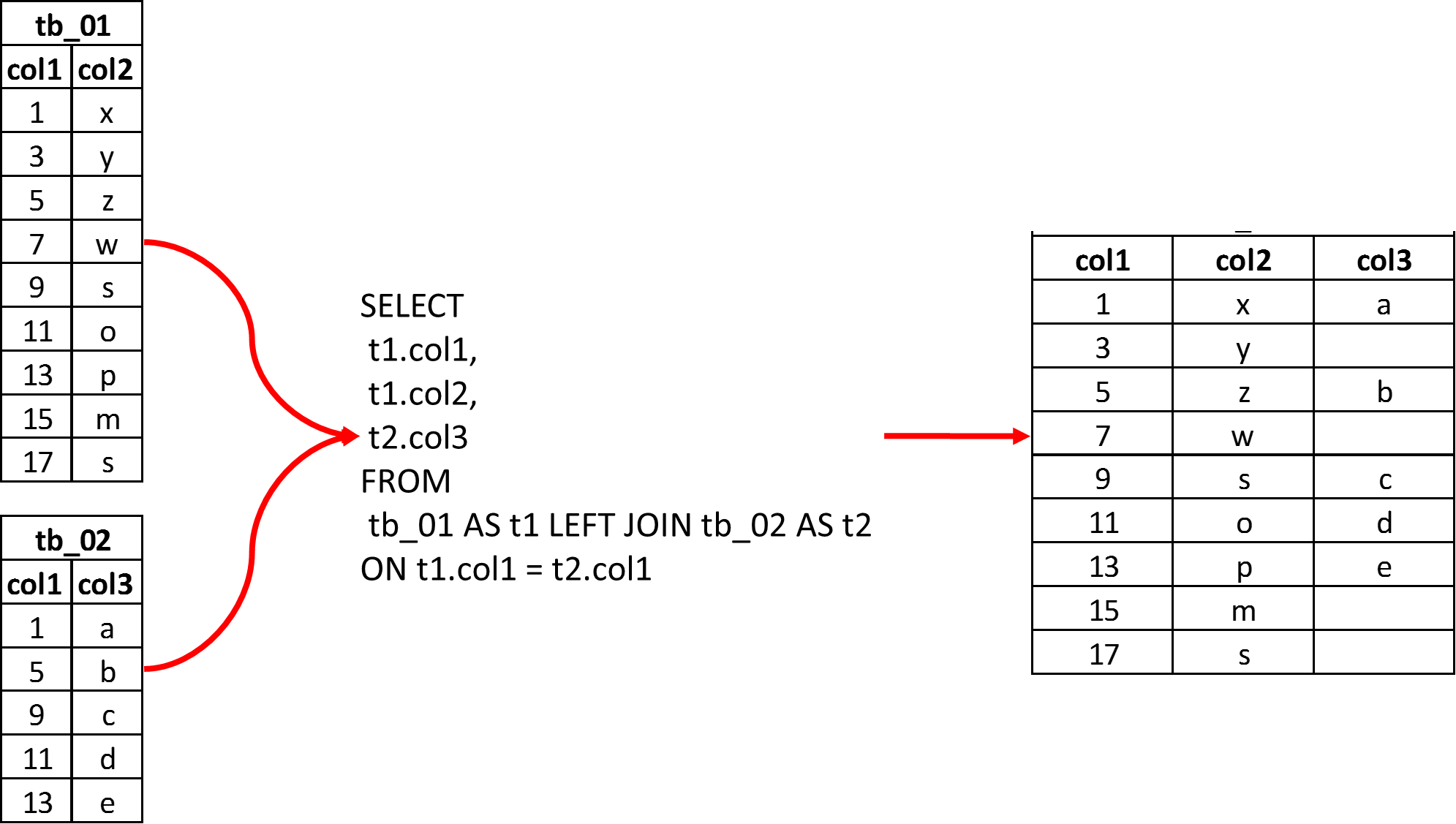
**JOIN** pdad.mor2018\_33ras **as** m

**ON** d.A01nFicha = m.A01nficha

**GROUP** **BY** d.A01ra

**ORDER** **BY** d.A01ra;

* **LEFT OUTER JOIN**: somente os registros da tabela da esquerda (left) serão retornados, tendo ou não registros relacionados na tabela da direita. Neste cruzamento, a tabela à esquerda do operador de junção exibirá cada um dos seus registros, enquanto que a da direita exibirá somente seus registros que tenham correspondentes aos da tabela da esquerda. Para os registros da direita que não tenham correspondentes na esquerda serão colocados valores nulos.



A programação que pode ser utilizada no exemplo citado (quantidade de pessoas por sexo nas Regiões Administrativas com base na PDAD 2018), a programação pode ser feita da seguinte forma:

**SELECT**

d.A01ra,

**SUM**(**CASE** **WHEN** m.E03 <> 1 **then** 1 **else** 0 **end**) qtd\_fem,

**SUM**(**CASE** **WHEN** m.E03 = 1 **then** 1 **else** 0 **end**) qtd\_masc

**FROM** pdad.dom2018\_33ras **as** d

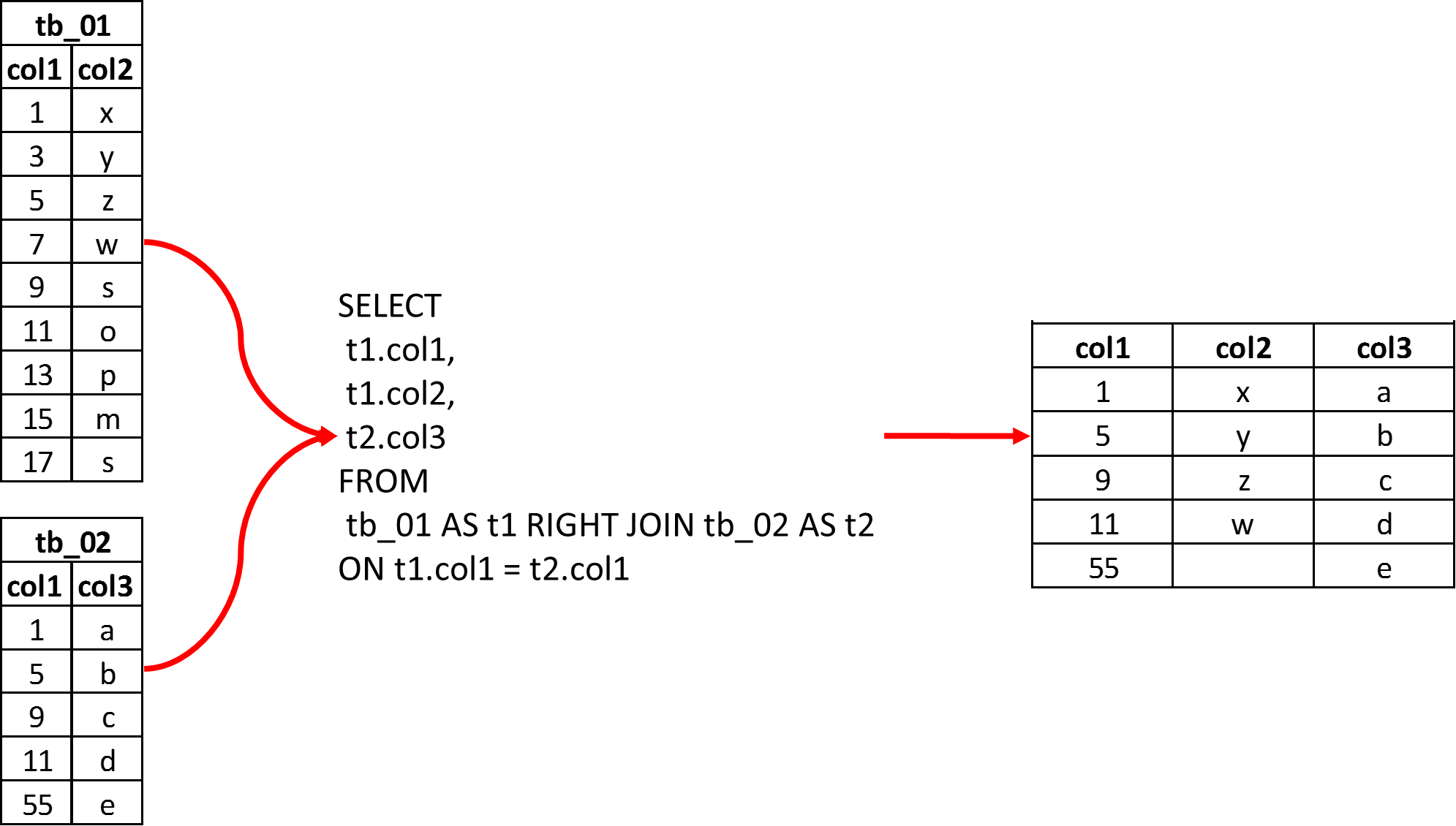
**LEFT JOIN** pdad.mor2018\_33ras **as** m

**ON** d.A01nFicha = m.A01nficha

**GROUP** **BY** d.A01ra

**ORDER** **BY** d.A01ra;

* **RIGHT OUTER JOIN**: ou simplesmente **RIGHT** **JOIN**, é o inverso do **LEFT OUTER JOIN**;



A programação que pode ser utilizada no exemplo citado (quantidade de pessoas por sexo nas Regiões Administrativas com base na PDAD 2018), a programação pode ser feita da seguinte forma:

**SELECT**

d.A01ra,

**SUM**(**CASE** **WHEN** m.E03 <> 1 **then** 1 **else** 0 **end**) qtd\_fem,

**SUM**(**CASE** **WHEN** m.E03 = 1 **then** 1 **else** 0 **end**) qtd\_masc

**FROM** pdad.dom2018\_33ras **as** d

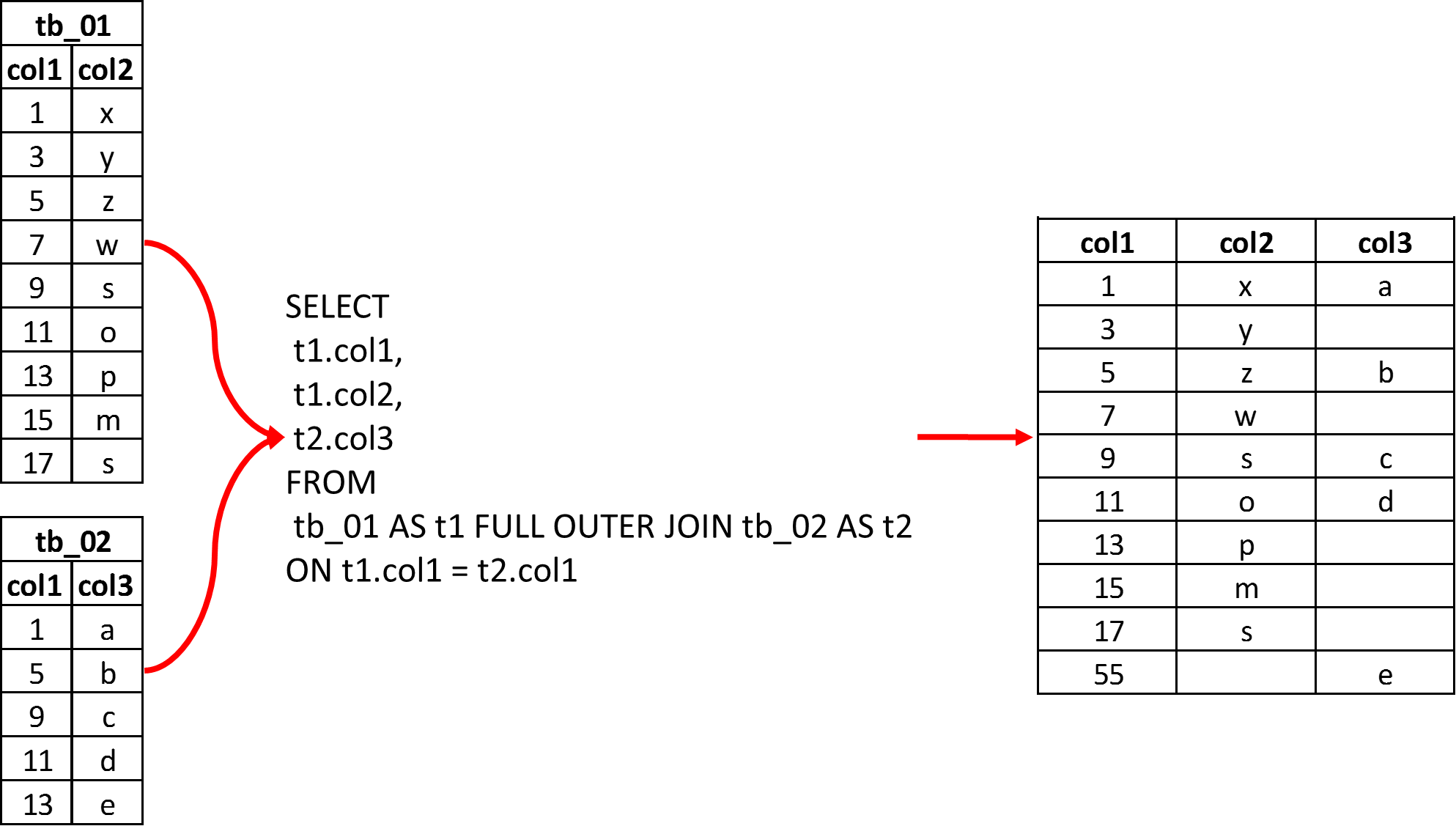
**RIGHT JOIN** pdad.mor2018\_33ras **as** m

**ON** d.A01nFicha = m.A01nficha

**GROUP** **BY** d.A01ra

**ORDER** **BY** d.A01ra;

* **FULL OUTER JOIN**: todos os registros de todas as tabelas serão retornados, tendo ou não registros relacionados. Neste cruzamento as tabelas, tanto à esquerda quanto a direita do operador de junção.



A programação que pode ser utilizada no exemplo citado (quantidade de pessoas por sexo nas Regiões Administrativas com base na PDAD 2018), a programação pode ser feita da seguinte forma:

**SELECT**

d.A01ra,

**SUM**(**CASE** **WHEN** m.E03 <> 1 **then** 1 **else** 0 **end**) qtd\_fem,

**SUM**(**CASE** **WHEN** m.E03 = 1 **then** 1 **else** 0 **end**) qtd\_masc

**FROM** pdad.dom2018\_33ras **as** d

**FULL OUTER JOIN** pdad.mor2018\_33ras **as** m

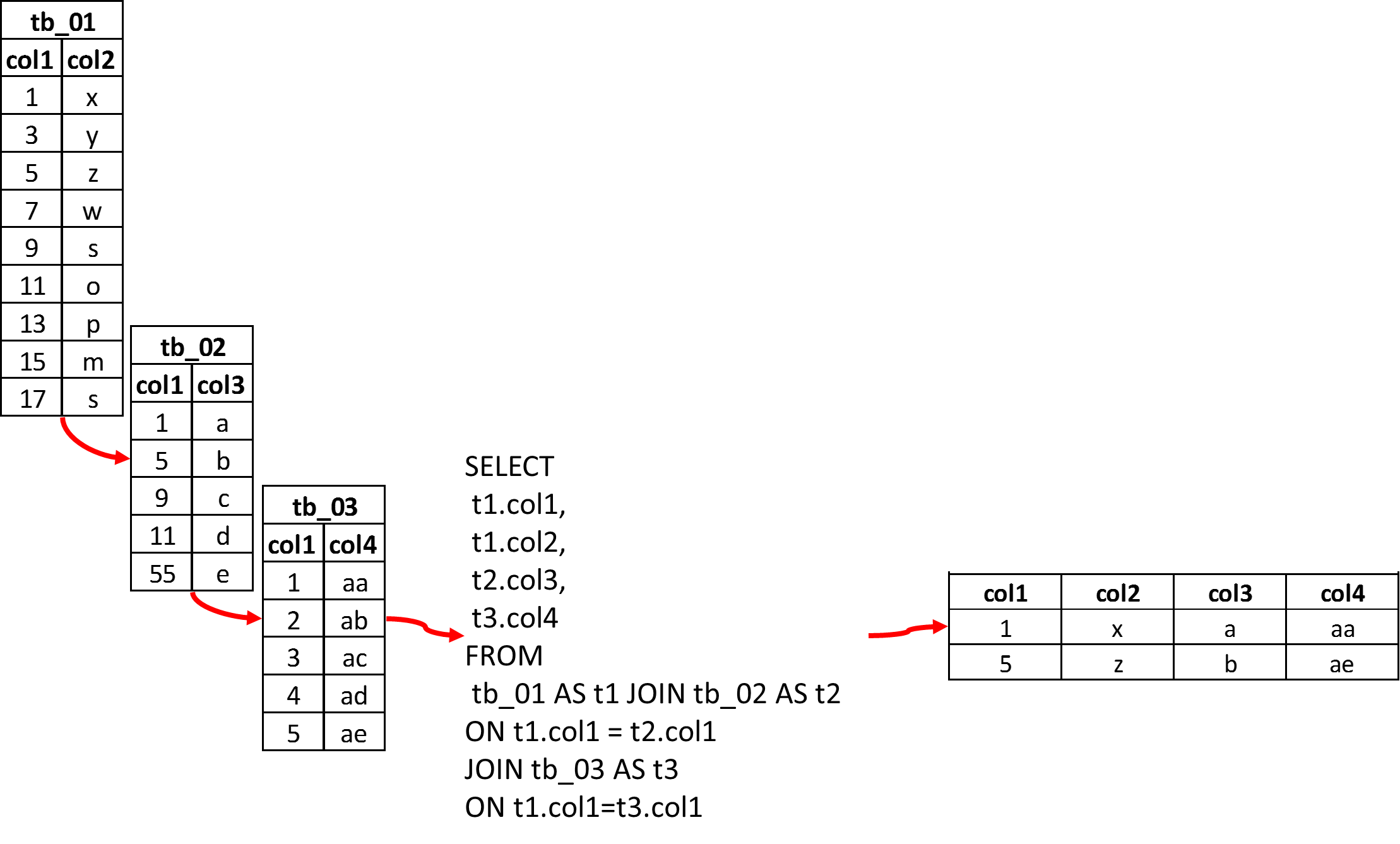
**ON** d.A01nFicha = m.A01nficha

**GROUP** **BY** d.A01ra

**ORDER** **BY** d.A01ra;

Algumas considerações importantes em relação a forma de programar:

1. O(s) nome(s) da(s) coluna(s) chave(s) pode(m) ou não ser(rem) iguais. O importante é que o(s) formato(s) seja(m) igual(is). Ou seja, se uma for texto e outra coluna número, o JOIN não é possível;
2. Quando é feito o cruzamento de duas ou mais tabelas, é necessário especificar de qual tabela vem cada coluna. Para isto é necessário escrever o nome da tabela seguido de ponto (.) mais o nome da coluna que deseja mostrar. Para não escrever sempre o nome da tabela, é possível atribuir um *ALIAS* para cada tabela especificadas após o **FROM**. No exemplo citado (quantidade de pessoas por sexo nas Regiões Administrativas com base na PDAD 2018), a tabela dom2018\_33ras foi chamada simplesmente de *d* (pdad.dom2018\_33ras **as** d) e a tabela mor2018\_33ras foi chamada de *m* (pdad.mor2018\_33ras **as** m);
3. Para utilizar o **JOIN** em três ou mais tabelas, basta realizar em blocos, como mostra o exemplo abaixo.



* 1. Utilização de algumas funções para manipulação de dados

Há inúmeras situações em que a manipulação de dados faz-se necessária, como converter uma coluna de texto para número e vice-versa, calcular uma idade ou até mesmo criar uma data.

Para tarefas como estas, as funções utilizadas são as seguintes:

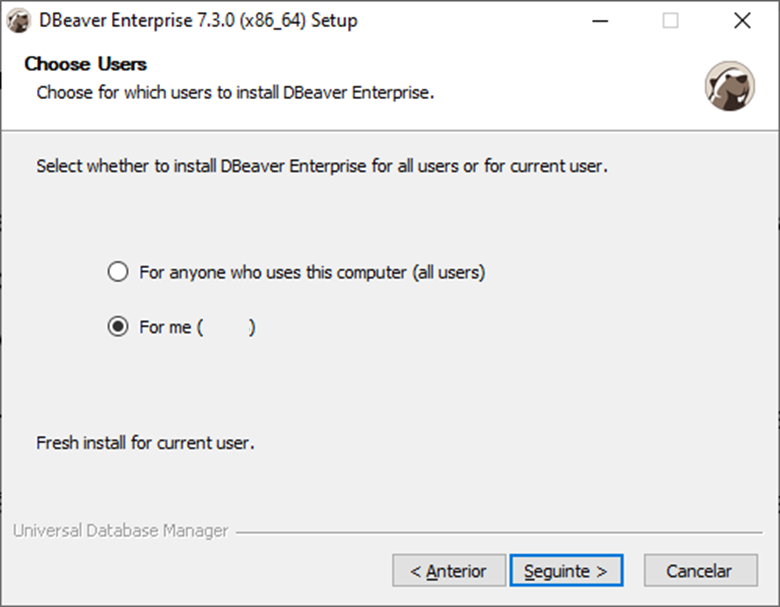
* **CONVERT(tipo, coluna ou expressão) AS apelido:** Esta é uma função tem o poder de converter diversos tipos de dados existentes em uma tabela. Por exemplo transformar uma coluna com informações do tipo inteiro para o formato *float*, usa-se a programação SELECT CONVERT(float, Nome da coluna no formato *integer*);
* **ROUND(coluna ou expressão, Quantidade de decimais) AS apelido:** Esta é uma função tem o poder de arredondar valores para uma quantidade específica de casas decimais;
* **CAST(coluna ou expressão AS tipo ) AS apelido:** Esta é uma função tem o poder de converter diversos tipos de dados a serem criados em uma tabela. Por exemplo para criar uma coluna com o número pi arredondado para duas decimais, usa-se a programação SELECT CAST((3.1415926535897) AS NUMERIC(7,2)). Pode-se converter caracteres para data, como por exemplo SELECT CAST('31 08 2015' AS DATE);
* **EXTRACT(YEAR FROM AGE(data2,data1)), para data2 > data1:** Com esta função é possível calcular a idade entre duas datas diferentes. Na realidade aqui são utilizadas duas funções, que são AGE e EXTRACT. Sugiro que seja feita uma pesquisa para ver como cada uma funciona.

1. PROCEDIMENTOS PARA INSTALAÇÃO DE FERRAMENTAS PARA CONSULTAS EM BANCO DE DADOS
   1. Como instalar o *DBeaver*

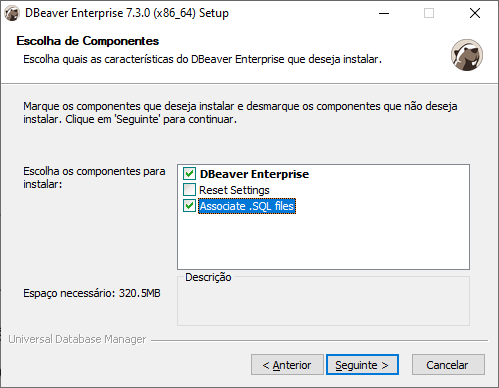
O *DBeaver* é um software livre utilizado para acesso, administração e consultas a diversos SGBD. Ele pode ser instalado em diversos sistemas operacionais. No caso do Windows 10, ele pode ser instalado e configurado sem a necessidade de perfil administrativo, como mostra os passos descritos abaixo.

1 – Acesse o endereço eletrônico [***https://dbeaver.com/***](https://dbeaver.com/), procure a área de download e baixe a versão .

2 – Execute o arquivo de instalação, clique no botão ***Seguinte*** e na tela seguinte clique no botão ***Aceito***. Na tela abaixo marque a opção ***For me (nº da matrícula)*** e clique no botão ***Seguinte***.



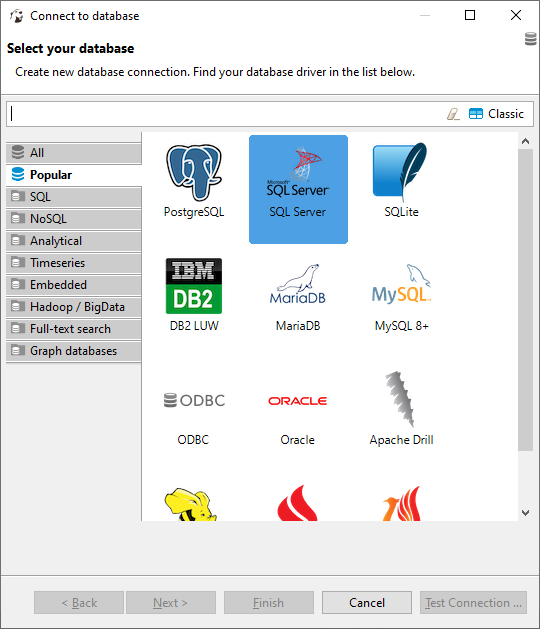
3 – Na tela abaixo, marque a opção ***Associate .SQL files*** e clique no botão ***Seguinte*** até chegar a tela com o botão ***Instalar***. Isto fará com que todo arquivo com extensão ***.sql*** seja associado ao ***DBeaver***.



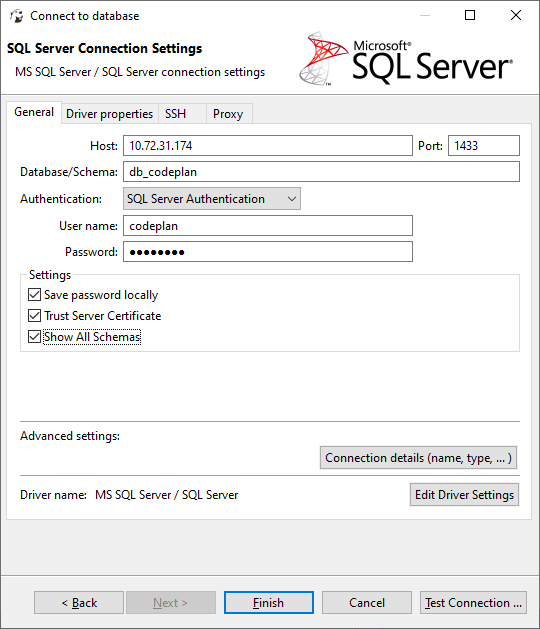
* 1. Como configurar o *DBeaver*

Uma vez instalado, basta executar o programa e seguir os passos a seguir para realizar a sua configuração.

1 – Ao executar o programa, clique em  e em seguida em  para acessar a janela abaixo. Selecione a opção ***SQL Server*** e clique no botão ***Next***.



2 – Preencha as informações da tela  como mostra a figura abaixo. O campo ***Passwor*** deve ser preenchido com ***codeplan***.



1. Em ciência da computação, ***front end*** e ***back end*** são termos generalizados que se referem às etapas inicial e final de um processo. De maneira geral podemos dizer que o ***front end*** é a forma como o usuário enxerga e gera as informações que são encaminhadas para o SGBD e o ***back end*** é a forma, não vista pelo usuário, como o SGBD processa e disponibiliza as informações para o usuário. [↑](#footnote-ref-1)
2. Há situações em que os fabricantes do SGBD criam funções específicas envolvendo um conjunto de programações em SQL, como uma função para o cálculo da diferença entre duas datas por exemplo, e patenteiam estas funções, tornando-se proprietária exclusiva de sua utilização. [↑](#footnote-ref-2)