

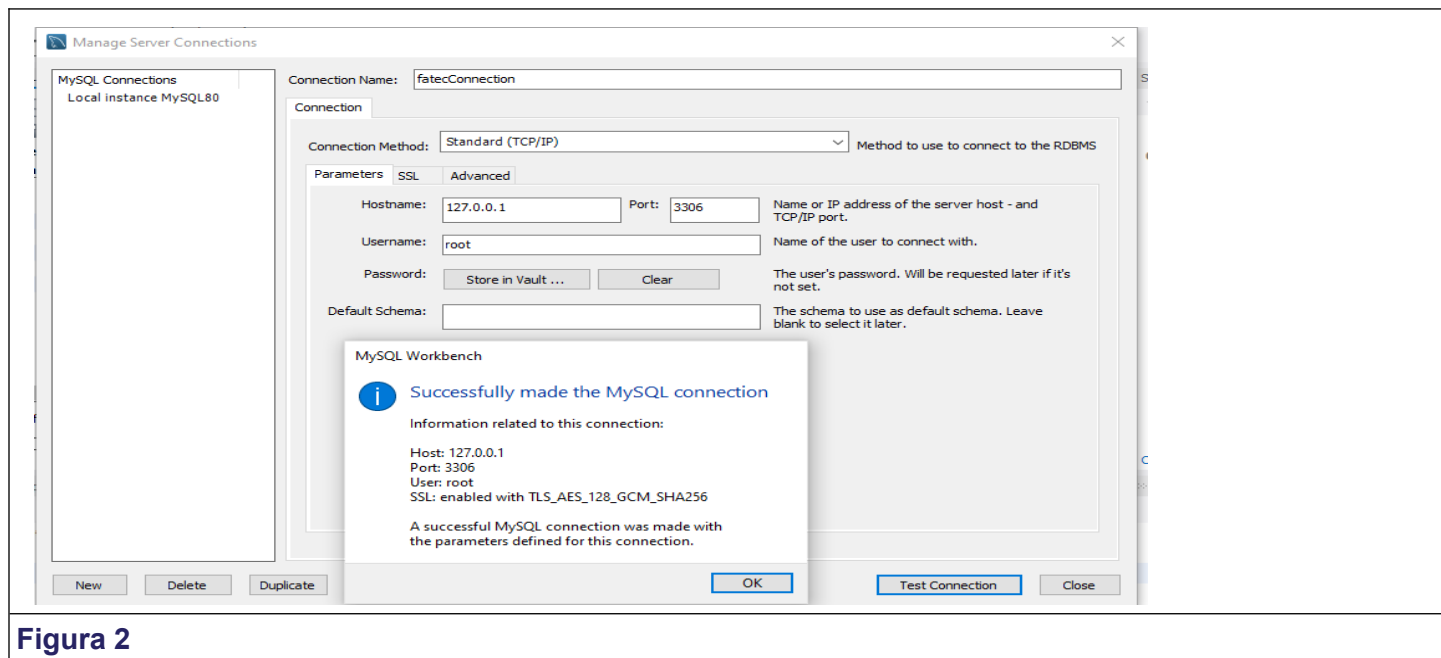
Aula 03 – Primeiro exemplo com MySQL

Criando conexão, banco de dados e tabelas com MySQL Workbench

Abra o MySQL Workbench e crie uma nova conexão com o nome fatecConnection.



User: root
senha em branco



Interface do programa MySQL

Essa é a tela inicial do programa, os menus básicos na parte superior não vamos utilizar por agora, mas você pode sempre explorar essas ferramentas.

Na aba escritor Query 1 é onde você vai poder escrever o código em SQL.

Agora já podemos dar início aos primeiros passos no SQL.

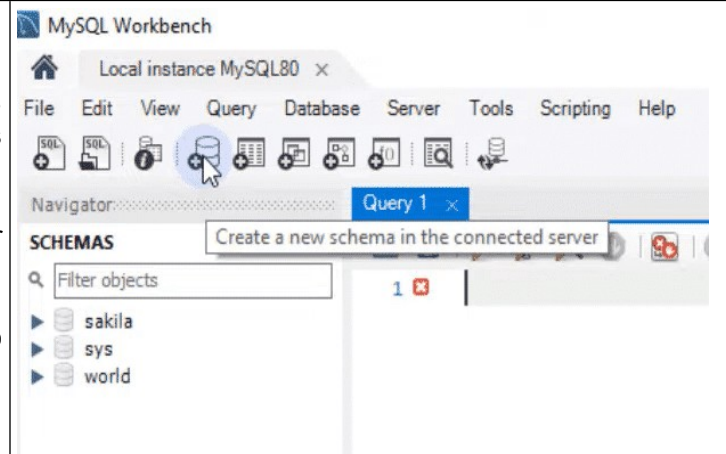


Figura 3

Criando um novo esquema

Nessa parte você vai dar um nome a esse esquema e vai prosseguir normalmente.

Depois de criar esse esquema, ele já vai aparecer na parte esquerda da sua tela e você vai notar que temos alguns conteúdos dentro dele.

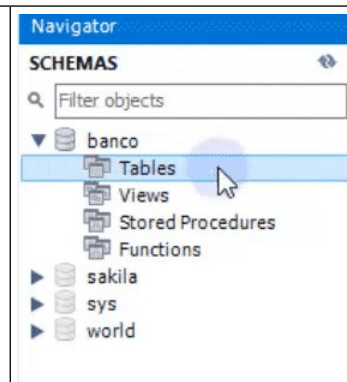


Figura 4

Passos para criar um BD

Passo 1: CREATE DATABASE

Passo 2: SHOW DATABASES

Passo 3: USE

Passo 3: CREATE TABLE

Passo 4: INSERT INTO

Passo 5: SELECT FROM

Passo 6: ORDER BY

Passo 7: WHERE

Passo 8: UPDATE

Passo 9: DELETE FROM

Selecionar faterConnection, dois cliques para abrir.

Passo 1: CREATE DATABASE

Criar um Novo banco de dados com o nome db_Empresa01.

```
CREATE DATABASE db_empresa01;
```

Passo 2: SHOW DATABASES

Para ver todos os bancos de dados existentes no servidor:

```
SHOW DATABASES;
```

Passo 3: USE

```
USE db_empresa01
```

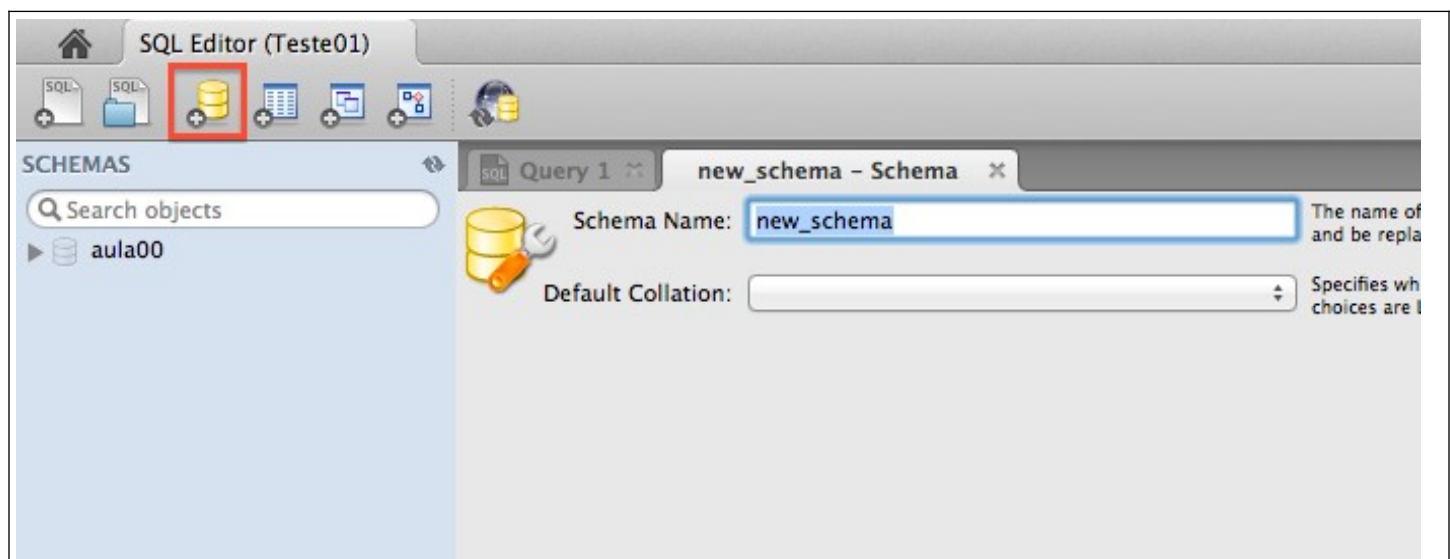


Figura 5

Para habilitar o BD digite a seguinte linha.

```
USE db_empresa1;
```

Ou

Clicar com botão direito e selecionar Set as Default Schema.

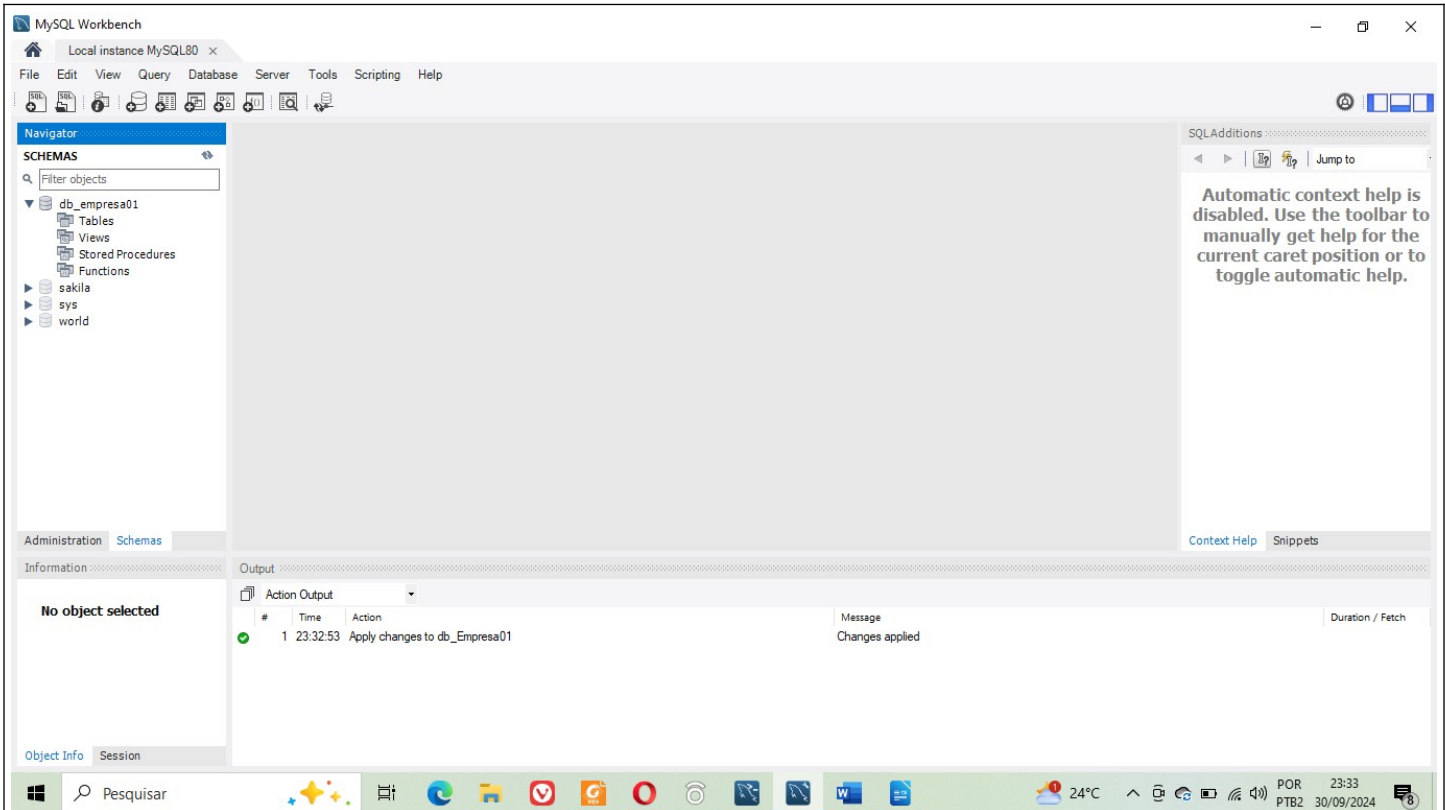


Figura 6

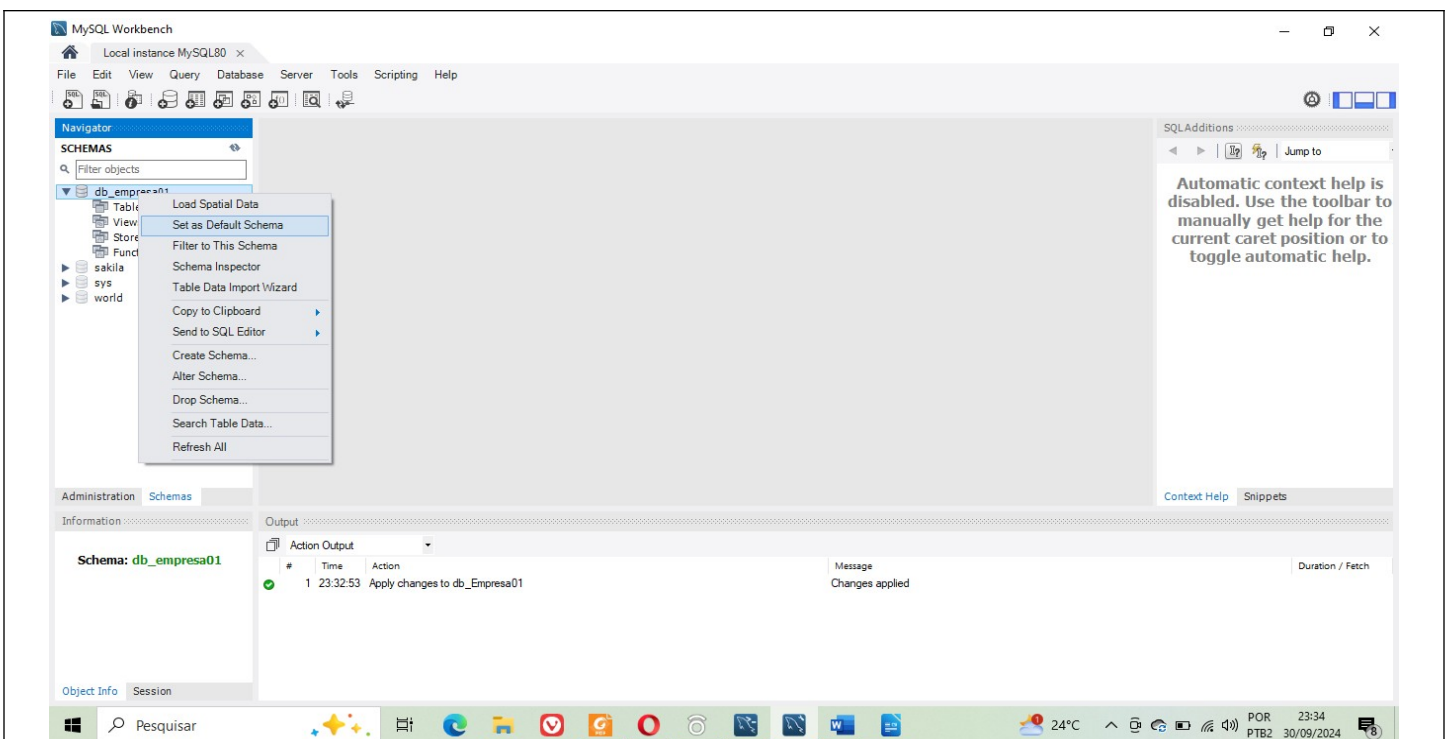


Figura 7

Passo 3: CREATE TABLE

Para criar a Tabela funcionário (tb_funcionario)

Sintaxe

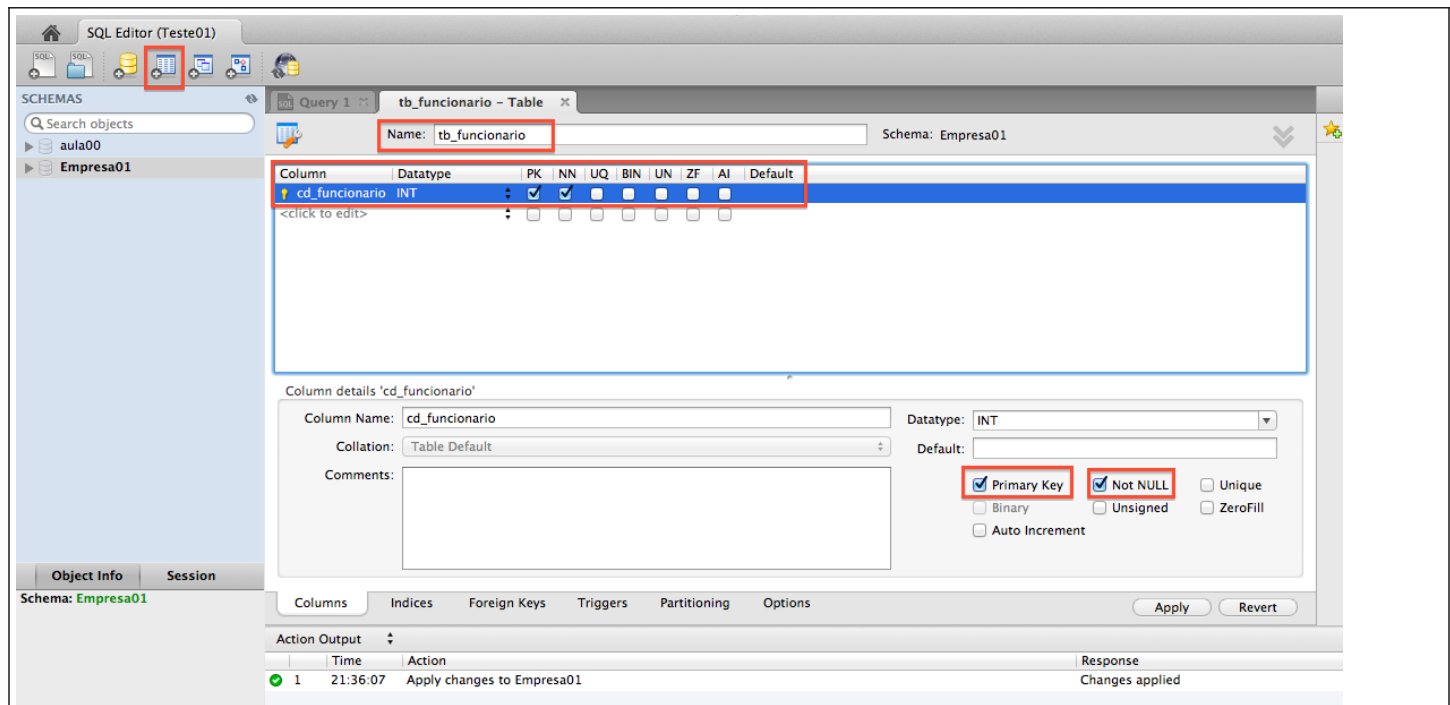


Figura 8

Tabela funcionário

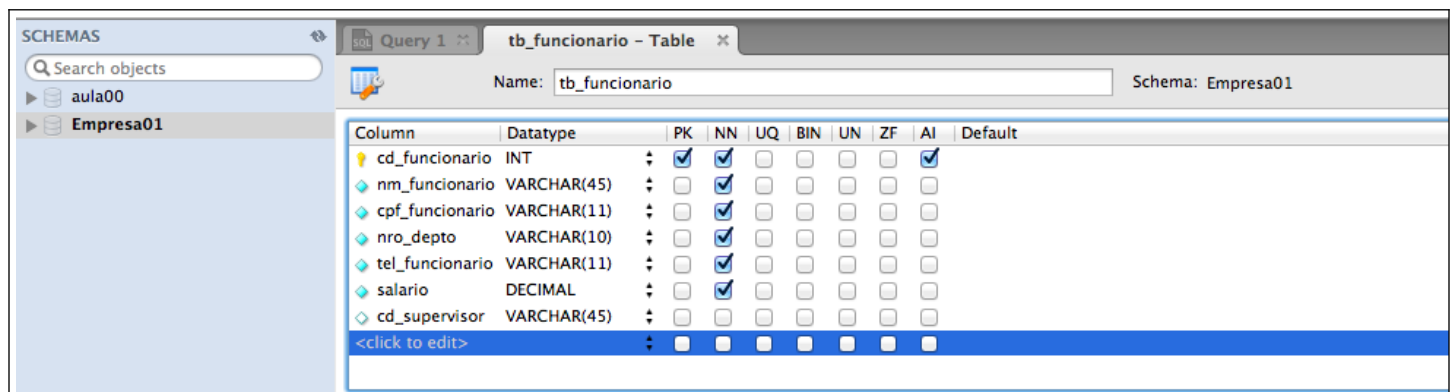


Figura 9

```
CREATE TABLE `tb_funcionario` (
  `cd_funcionario` int(11) NOT NULL AUTO_INCREMENT,
  `nm_funcionario` varchar(45) NOT NULL,
  `cpf_funcionario` varchar(11) NOT NULL,
  `nro_depto` varchar(45) NOT NULL,
  `tel_funcionario` varchar(10) NOT NULL,
  `salario` float NOT NULL,
  `cd_supervisor` int(11) DEFAULT NULL,
  PRIMARY KEY (`cd_funcionario`));
```

Passo 4: INSERT INTO

Inserindo valores na tabela (Popular as tabelas)

4.a) Via script

```
INSERT INTO `tb_funcionario` VALUES (1111,'Carlos','1122334455','3','1333334455',800.5,4444),  
(1112,'Ana','3333444555','2','1332325544',950,4444),  
(1113,'Eduardo','4446677888','1','1331317744',1500,NULL),  
(1114,'Andre','5553334448','2','1334343778',1200,4444);
```

4.b) Diretamente, utilizando a interface gráfica

Popular as tabelas – Botão direito no nome da tabela – Select Rows – Adicionar os valores

Passo 5: SELECT FROM (Consulta/Recuperação em bancos de dados)

Uma consulta é uma investigação no banco de dados feita através da instrução SELECT. Uma consulta é usada para extrair dados do banco de dados em um formato legível de acordo com a solicitação do usuário. Por exemplo, se tivermos uma tabela funcionário, poderemos emitir uma instrução SQL que informe qual funcionário tem o maior salário.

Essa solicitação de informações úteis sobre funcionários feita ao banco de dados é uma típica consulta que pode ser executada em um banco de dados relacional.

Pode ser utilizado a ferramenta no modo texto-linha de comando ou no modo gráfico(MYSQL WORKBENCH).

Introdução à instrução SELECT

A instrução SELECT é a instrução usada para construir consultas de banco de dados. Ela não é uma instrução independente, o que quer dizer que são necessárias uma ou mais cláusulas adicionais (elementos) para uma consulta sintaticamente correta. Além das cláusulas obrigatórias, existem também as opcionais, que aumentam a funcionalidade total da instrução SELECT, que é, de longe, uma das instruções mais usadas e mais avançadas na SQL.

A cláusula FROM é a cláusula obrigatória e deve ser usada junto com a instrução SELECT.

Existem quatro palavras-chaves, ou cláusulas, que são partes importantes de uma instrução SELECT. São elas:

SELECT (o que?????)
FROM (vem?)
WHERE
ORDER BY

A instrução SELECT

A instrução SELECT é usada junto com a cláusula FROM para extrair dados do banco de dados em um formato legível e organizado. A parte SELECT da consulta tem por finalidade selecionar os dados que desejamos ver de acordo com as colunas em que eles estão armazenados em uma tabela.

Observe a sintaxe abaixo para uma consulta simples, onde todas as informações (linhas) da tabela FUNCIONARIO serão retornadas.

```
SELECT * FROM tb_funcionario;
```

A palavra-chave SELECT em uma consulta é seguida de uma lista de colunas que desejamos exibir como parte da saída da consulta. A palavra-chave FROM é seguida de uma lista de uma ou mais tabelas de onde desejamos selecionar os dados.

O asterisco (*) é usado como um curinga, **indicando que todas as colunas(atributos)** em uma ou mais tabelas devem ser exibidas no resultado da consulta. Perceba o uso de vírgula para separar argumentos em uma lista de instruções SQL.

Algumas listas comuns incluem listas de colunas em uma consulta, listas de tabelas para serem selecionadas em uma consulta, valores para serem inseridos em uma tabela e valores agrupados como uma condição em uma cláusula WHERE da consulta.

Vamos criar uma consulta (consulta01), onde queremos todas as informações contidas nesta tabela. Veja a seguir:

```
SELECT * FROM tb_funcionario;
```

A consulta acima é originária da seguinte função em álgebra relacional:

```
CONSULTA 1=  $\pi_{cd\_func, nm\_funcionario, cd\_cpf, nro\_depto, tele, salario, cd\_super}$  (tb_funcionario);
```

23 • `select * from tb_funcionario;`

24

<

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	cd_funcionario	nm_funcionario	cpf_funcionario	nro_depto	tel_funcionario	salario	cd_supervisor
▶	1111	Carlos	1122334455	3	1333334455	800.5	4444
	1112	Ana	3333444555	2	1332325544	950	4444
	1113	Eduardo	4446677888	1	1331317744	1500	NULL
	1114	Andre	5553334448	2	1334343778	1200	4444
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela 01: Após a Consulta01

Agora, uma consulta (consulta02) onde queremos apenas o nome do funcionário e o seu cpf. Veja abaixo:

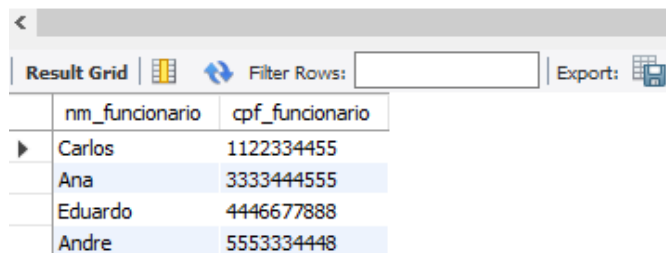
```
SELECT nm_funcionario, cpf_funcionario
FROM tb_funcionario;
```

A consulta acima é originária da seguinte função em álgebra relacional:

CONSULTA 2= $\pi_{nm_func, cd_cpf}(tb_funcionario);$

O retorno será:

```
25 • SELECT nm_funcionario, cpf_funcionario
26 FROM tb_funcionario;
```



	nm_funcionario	cpf_funcionario
▶	Carlos	1122334455
	Ana	3333444555
	Eduardo	4446677888
	Andre	5553334448

Tabela 02: Após a consulta02

Agora faremos uma consulta (consulta03), retornando apenas o nome e o telefone do funcionário. Veja abaixo:

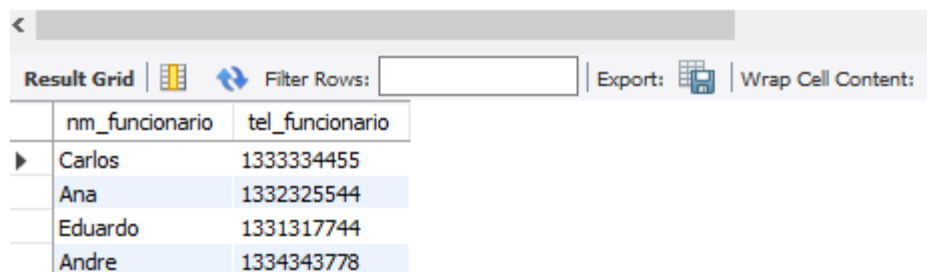
**SELECT nm_funcionario, tel_funcionario
FROM tb_funcionario;**

A consulta acima é originária da seguinte função em álgebra relacional:

CONSULTA 3= $\pi_{nm_func, telefone}(tb_funcionario);$

O retorno será:

```
28 • SELECT nm_funcionario, tel_funcionario
29 FROM tb_funcionario;
```

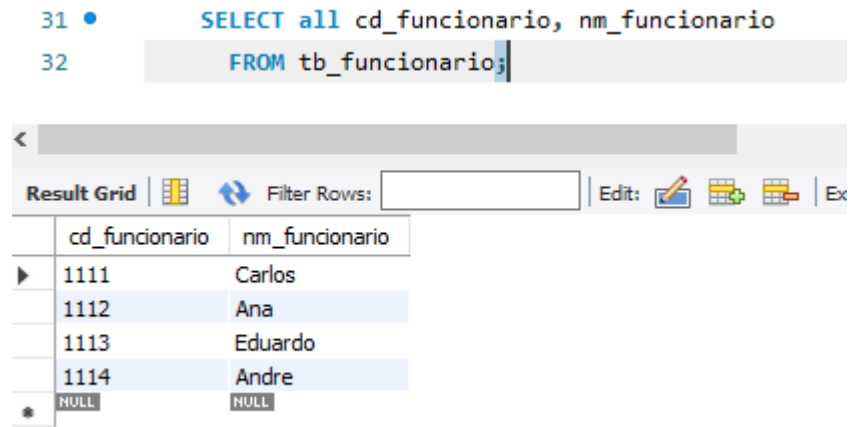


	nm_funcionario	tel_funcionario
▶	Carlos	1333334455
	Ana	1332325544
	Eduardo	1331317744
	Andre	1334343778

Tabela 03: Após a consulta03

A opção ALL é usada para exibir todos os valores de uma coluna, inclusive os duplicados.

```
SELECT ALL cd_funcionario, nm_funcionario
FROM tb_funcionario;
```



```
31 • SELECT all cd_funcionario, nm_funcionario
32 FROM tb_funcionario;
```

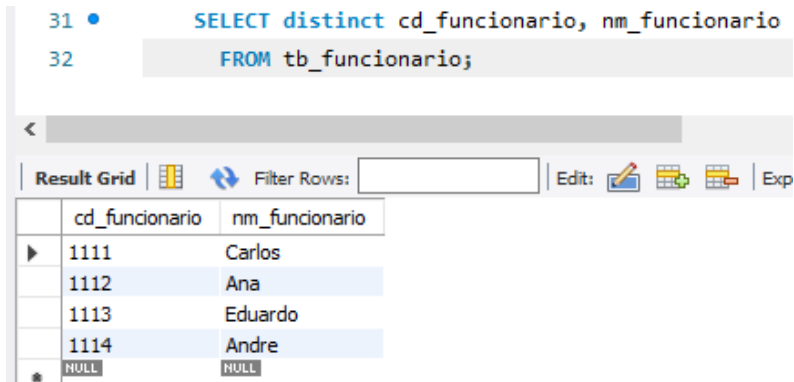
cd_funcionario	nm_funcionario
1111	Carlos
1112	Ana
1113	Eduardo
1114	Andre
NULL	NULL

Tabela 04:

A opção DISTINCT é usada para evitar que linhas repetidas sejam exibidas na saída. O padrão é ALL, que não precisa ser especificado.

Veja o próximo exemplo para a instrução SELECT.

```
SELECT DISTINCT cd_func, nm_funcionario
FROM tb_funcionario;
```



```
31 • SELECT distinct cd_funcionario, nm_funcionario
32 FROM tb_funcionario;
```

cd_funcionario	nm_funcionario
1111	Carlos
1112	Ana
1113	Eduardo
1114	Andre
NULL	NULL

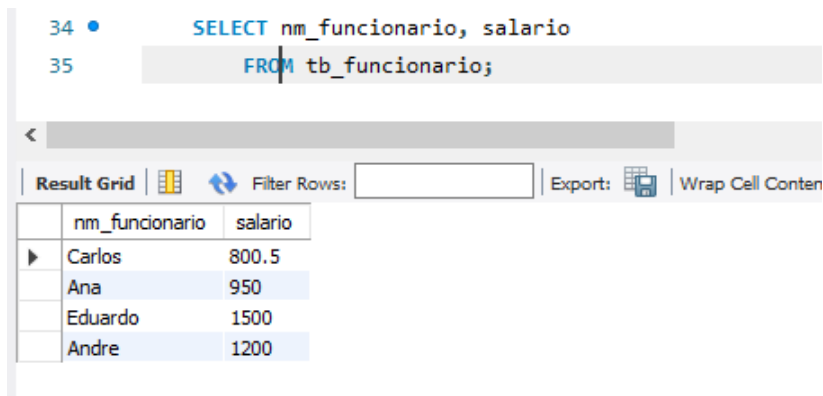
Tabela 05:

Passo 6: FROM (A cláusula FROM)

A cláusula FROM deve ser usada junto com a instrução SELECT, sendo um elemento obrigatório em qualquer consulta. Sua finalidade é informar ao banco de dados que tabelas devem ser acessadas para recuperar os dados desejados da consulta. A cláusula FROM pode conter uma ou mais tabelas e deve sempre listar pelo menos uma tabela.

Vamos relembrar uma consulta sem a cláusula where, isto é, sem condições. Fazemos uma consulta na tabela FUNCIONÁRIO, buscando o nome do funcionário e seu salário.

```
SELECT nm_funcionario, salario
FROM tb_funcionario;
```



	nm_funcionario	salario
▶	Carlos	800.5
	Ana	950
	Eduardo	1500
	Andre	1200

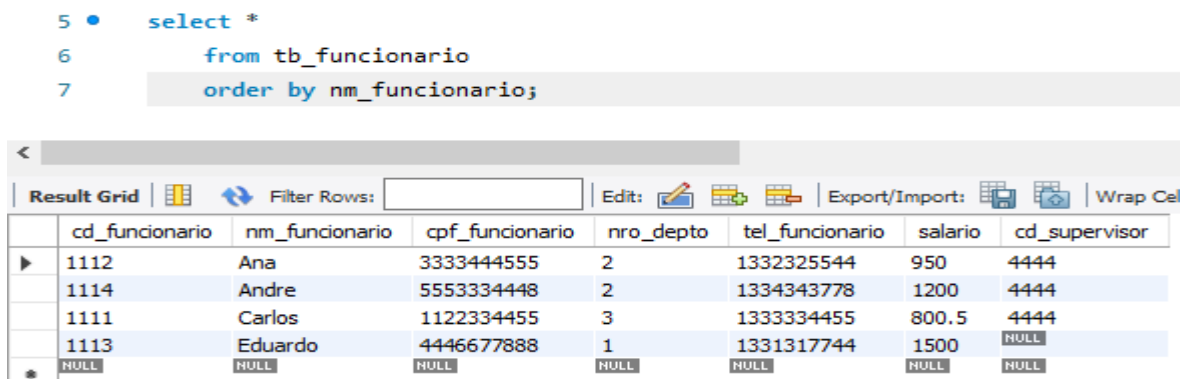
Tabela 06: O resultado:

Passo 6: ORDER BY

Normalmente, queremos que o resultado da consulta siga algum tipo de ordem e, para tanto, podemos usar a cláusula **ORDER BY** para ordenar os dados. Esta cláusula organiza os resultados de uma consulta em um formato de listagem que especificamos.

A ordenação padrão para a cláusula **ORDER BY** é a crescente; por isso, se estivermos ordenando nomes de saída em ordem alfabética, a ordenação é exibida de A a Z.

```
select *
from tb_funcionario
order by nm_funcionario;
```



	cd_funcionario	nm_funcionario	cpf_funcionario	nro_depto	tel_funcionario	salario	cd_supervisor
▶	1112	Ana	3333444555	2	1332325544	950	4444
	1114	Andre	5553334448	2	1334343778	1200	4444
	1111	Carlos	1122334455	3	1333334455	800.5	4444
	1113	Eduardo	4446677888	1	1331317744	1500	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela 07:

```
select *
from tb_funcionario
order by nro_depto;
```

```
9 • select *
10 from tb_funcionario
11 order by nro_depto;
```

cd_funcionario	nm_funcionario	cpf_funcionario	nro_depto	tel_funcionario	salario	cd_supervisor
1113	Eduardo	4446677888	1	1331317744	1500	NULL
1112	Ana	333344	4446677888	1332325544	950	4444
1114	Andre	5553334448	2	1334343778	1200	4444
1111	Carlos	1122334455	3	1333334455	800.5	4444
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela 08:

Veja um exemplo de consulta (consulta07) com valores ordenados. Consultaremos novamente o nome do funcionário e salário maior do que R\$ 1.200,00, mas desta vez ordenaremos os nomes, que nos exemplos anteriores, estavam em desordem.

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario >= 1200
ORDER BY nm_funcionario;
```

O resultado:

```
40 • SELECT nm_funcionario, salario
41 FROM tb_funcionario
42 WHERE salario >= 1200
43 ORDER BY nm_funcionario;
44
```

nm_funcionario	salario
Andre	1200
Eduardo	1500

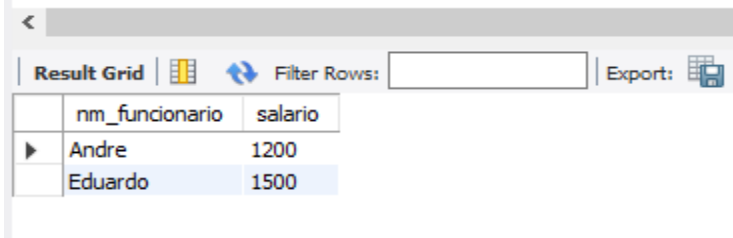
Tabela 09:

Agora outro exemplo na mesma consulta (consulta08), ordenando os valores de salário, do menor para o maior:

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario >= 1200
ORDER BY salario;
```

O resultado:

```
45 • SELECT nm_funcionario, salario
46     FROM tb_funcionario
47     WHERE salario >= 1200
48     ORDER BY salario;
49
```



The screenshot shows a database query interface. At the top, the SQL query is displayed with line numbers 45 to 49. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', and 'Export'. The results are shown in a table grid with two columns: 'nm_funcionario' and 'salario'. The table contains two rows: 'Andre' with a salary of 1200 and 'Eduardo' with a salary of 1500.

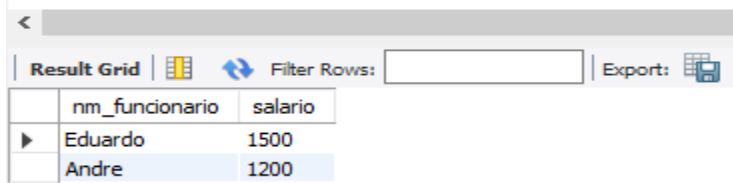
	nm_funcionario	salario
▶	Andre	1200
	Eduardo	1500

Tabela 10:

A ordenação, por padrão, se dá na ordem crescente, de A a Z e de 1 a 9. Podemos alterar a ordenação, passando a ordenar do maior valor para o menor. Para isso, usamos o parâmetro DESC após a cláusula de ordenação. Veja o exemplo (consulta09):

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario >= 1200
ORDER BY salario DESC;
```

```
50 • SELECT nm_funcionario, salario
51     FROM tb_funcionario
52     WHERE salario >= 1200
53     ORDER BY salario DESC;
54
```



The screenshot shows a database query interface. At the top, the SQL query is displayed with line numbers 50 to 54. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', and 'Export'. The results are shown in a table grid with two columns: 'nm_funcionario' and 'salario'. The table contains two rows: 'Eduardo' with a salary of 1500 and 'Andre' with a salary of 1200.

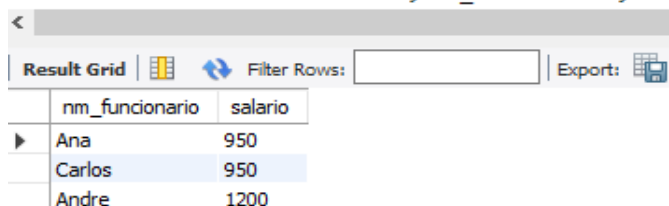
	nm_funcionario	salario
▶	Eduardo	1500
	Andre	1200

Tabela 11:

Agora os salários estão do maior valor para o menor valor. Quem ganha mais está no topo da lista. Podemos também ordenar por mais de um campo, isto é, ordenamos primeiro por um campo, depois por outro. Veja o exemplo(consulta10):

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario <= 1200
ORDER BY salario , nm_funcionario;
```

```
55 • SELECT nm_funcionario, salario
56     FROM tb_funcionario
57     WHERE salario <= 1200
58     ORDER BY salario , nm_funcionario;
```



nm_funcionario	salario
Ana	950
Carlos	950
Andre	1200

Tabela 12:

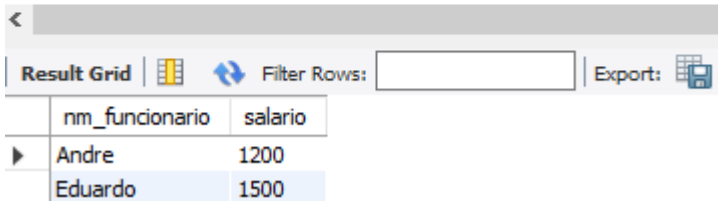
Neste exemplo, ordenamos primeiro pelo valor do salário em ordem crescente, depois ordenamos pelo nome dos funcionários. Se houver funcionário com o mesmo valor de salário, o nome será exibido ordenadamente.

A SQL oferece alguns atalhos. Uma coluna listada na cláusula **ORDER BY** pode ser abreviada com um número inteiro. Esse número é uma substituição do verdadeiro nome da coluna, ou seja, ele é um alias para ser usado na operação de ordenação e identifica a posição da coluna depois da palavra-chave **SELECT**.

Observe o exemplo a seguir, baseado nas consultas(consulta11) que temos feito acima. Usaremos como identificador um número inteiro na cláusula **ORDER BY**.

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario >= 1200
ORDER BY 2;
```

```
60 • SELECT nm_funcionario, salario
61     FROM tb_funcionario
62     WHERE salario >= 1200
63     ORDER BY 2;
64
```



nm_funcionario	salario
Andre	1200
Eduardo	1500

Tabela 13:

Perceba que foi usado o valor 2, indicando que a ordenação deve ser feita pelo segundo argumento da cláusula SELECT, o campo salário.

Passo 7: WHERE (A cláusula WHERE - Usando condições para distinguir dados)

Uma condição é parte de uma consulta que é usada para exibir informações seletivas, conforme especificadas pelo usuário. O valor de uma condição pode ser TRUE ou FALSE, limitando assim, os dados recebidos da consulta.

A cláusula WHERE é usada para impor condições a uma consulta, eliminando linhas que normalmente seriam retornadas por uma consulta que não tivesse condições definidas.

A cláusula WHERE pode conter mais de uma condição e, nesse caso, as condições são unidas por operadores AND e OR. Veremos o uso dos operadores AND e OR e também de operadores condicionais mais adiante.

Agora, a mesma consulta, retornando apenas os funcionários que recebem salários acima de R\$ 1.800,00.

```
Select nm_funcionario, salario
      from tb_funcionario
      where salario > 1200;
```

A consulta acima é originária da seguinte função em álgebra relacional:

CONSULTA 5= $\pi_{nm_func, salario} (\sigma_{salario > 1200,00} (tb_funcionario))$;

O resultado:

```
37 • Select nm_funcionario, salario
38       from tb_funcionario
39       where salario > 1200;
```

Result Grid
Filter Rows:
Export:
Wrap Cell Content:

	nm_funcionario	salario
▶	Carlos	800.5
	Ana	950
	Eduardo	1500
	Andre	1200

Tabela 14:

Agora, a mesma consulta, retornando apenas os funcionários que recebem salários acima ou igual a R\$ 1.200,00.

```

41 •      Select nm_funcionario, salario
42      from tb_funcionario
43      where salario >= 1200;
    
```

Result Grid	
nm_funcionario	salario
Eduardo	1500
Andre	1200

Tabela 15:

Agora queremos os nomes e salários dos funcionários que recebem menos de R\$ 1.800,00.

```

SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario <= 1200.00;
    
```

```

45 •      Select nm_funcionario, salario
46      from tb_funcionario
47      where salario <= 1200;
    
```

Result Grid	
nm_funcionario	salario
Carlos	800.5
Ana	950
Andre	1200

Tabela 16:

A consulta acima é originária da seguinte função em álgebra relacional:

```

CONSULTA 6=  $\pi_{nm\_func, salario} (\sigma_{salario \leq 1200,00} (tb\_funcionario))$ ;
    
```

Aliases de colunas

Os *aliases* (nomes alternativos de colunas) são usados para renomear colunas de uma tabela para adequá-las a determinada consulta. Observe como podemos utilizar estes *aliases* no exemplo abaixo:

```

Select nome_coluna as "Novo Nome"
From nome_tabela;
    
```

Neste exemplo, nome_coluna é o nome da coluna que queremos consultar, nome_alternativo é o nome que queremos que seja apresentado nesta coluna, no resultado da consulta, e o nome_tabela é o nome da tabela que queremos usar para esta consulta. **Este alias, se formado por mais de uma palavra, como “Nome do funcionário”, deve obrigatoriamente estar entre aspas, simples ou duplas.**

Vamos a um exemplo na tabela funcionário, sem o uso de alias:

Consulta (consulta12) sem alias

```
Select nm_funcionario
From tb_funcionario;
```

65 • **SELECT** *

66 **FROM** tb_funcionario;

<

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

	cd_funcionario	nm_funcionario	cpf_funcionario	nro_depto	tel_funcionario	salario	cd_supervisor
▶	1111	Carlos	1122334455	3	1333334455	950	4444
	1112	Ana	3333444555	2	1332325544	950	4444
	1113	Eduardo	4446677888	1	1331317744	1500	NULL
	1114	Andre	5553334448	2	1334343778	1200	4444
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela 17:

Este recurso (aliases) é bem interessante quando precisamos emitir algum tipo de relatório e gostaríamos de apresentar um nome mais amigável para o dado, ao invés dos nomes utilizados para a criação dos dados nas tabelas - Código do funcionário em vez de cd_funcionario, Nome do funcionário ao invés de nm_funcionario. Agora a mesma consulta(consulta13), com alias:

```
Select nm_funcionario as 'Primeiro Nome'
From tb_funcionario as F;
```

68 • **Select** nm_funcionario as 'Primeiro Nome'

69 **From** tb_funcionario as F;

<

Result Grid | Filter Rows: | Export: | V


	Primeiro Nome
▶	Carlos
	Ana
	Eduardo
	Andre

Tabela 18:


```

17
18 • SELECT nm_funcionario AS 'Nome do Funcionario'
19     FROM tb_funcionario

```



Nome do Funcionario
Carlos
Ana
Andre

Tabela 19:

```

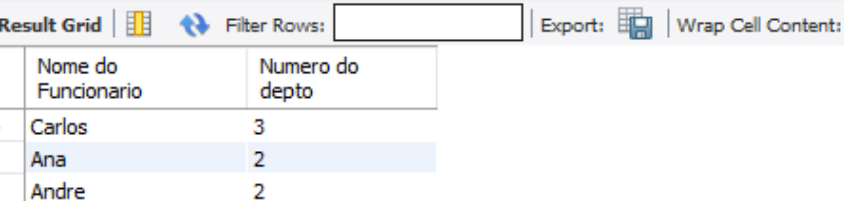
SELECT nm_funcionario AS 'Nome do Funcionario',
       nro_depto AS 'Numero do depto'
FROM tb_funcionario
WHERE cd_supervisor = 4444;

```

```

13 • SELECT nm_funcionario AS 'Nome do Funcionario',
14         nro_depto AS 'Numero do depto'
15     FROM tb_funcionario
16     WHERE cd_supervisor = 4444;
17

```



Nome do Funcionario	Numero do depto
Carlos	3
Ana	2
Andre	2

Tabela 20:

Nenhuma alteração é apresentada no resultado, sendo este igual aos resultados apresentados por comandos sem aliases de tabelas. Uma justificativa para o uso de aliases para nomes de tabelas é que este recurso permite uma digitação “mais curta” quando estamos relacionando mais de uma tabela em nossas consultas.

Para finalizar, os aliases também podem ser utilizados para dar nomes alternativos para as tabelas.

```

Select nm_funcionario
      From tb_funcionario as F;

```

```

SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario not BETWEEN 1200.00 AND 1500.00;

```

22	•	SELECT nm_funcionario, salario
23		FROM tb_funcionario
24		WHERE salario not BETWEEN 1200.00 AND 1500.00;

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	nm_funcionario	salario		
▶	Carlos	800.5		
	Ana	950		

Tabela 21:

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario IN (1100.00, 1500.00, 1800.00);
```

27	•	SELECT nm_funcionario, salario
28		FROM tb_funcionario
29		WHERE salario IN (1100.00, 1500.00, 1800.00);

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	nm_funcionario	salario		
▶	Eduardo	1500		

Tabela 22:

```
SELECT nm_funcionario, salario
FROM tb_funcionario
WHERE salario <= 1200.00
ORDER BY salario desc;
```

31	•	SELECT nm_funcionario, salario
32		FROM tb_funcionario
33		WHERE salario <= 1200.00
34		ORDER BY salario desc;

Result Grid		Filter Rows:	Export:
	nm_funcionario	salario	
▶	Andre	1200	
	Ana	950	
	Carlos	800.5	

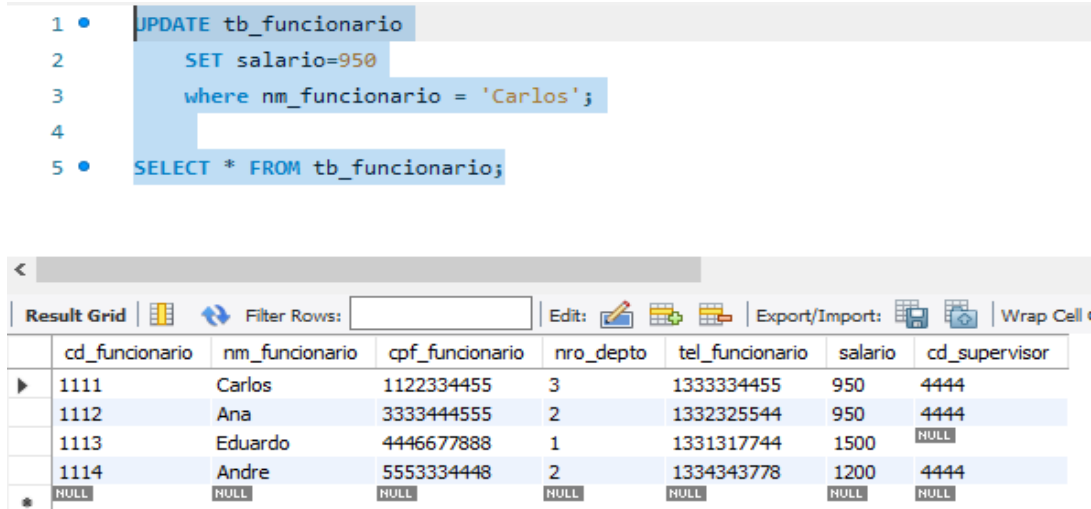
Tabela 23:

Passo 8: UPDATE

```
UPDATE tb_funcionario
SET salario=950
where nm_funcionario = 'Carlos';
```

Para verificar

```
SELECT * FROM tb_funcionario;
```



The screenshot shows a MySQL IDE interface. The top part displays a list of SQL commands: 1. UPDATE tb_funcionario, 2. SET salario=950, 3. where nm_funcionario = 'Carlos';, 4. (empty line), 5. SELECT * FROM tb_funcionario;. Below the commands is a 'Result Grid' showing the output of the SELECT statement. The grid has columns: cd_funcionario, nm_funcionario, cpf_funcionario, nro_depto, tel_funcionario, salario, and cd_supervisor. The data rows are: 1111 Carlos 1122334455 3 1333334455 950 4444, 1112 Ana 3333444555 2 1332325544 950 4444, 1113 Eduardo 4446677888 1 1331317744 1500 NULL, 1114 Andre 5553334448 2 1334343778 1200 4444, and a summary row with NULL values.

cd_funcionario	nm_funcionario	cpf_funcionario	nro_depto	tel_funcionario	salario	cd_supervisor
1111	Carlos	1122334455	3	1333334455	950	4444
1112	Ana	3333444555	2	1332325544	950	4444
1113	Eduardo	4446677888	1	1331317744	1500	NULL
1114	Andre	5553334448	2	1334343778	1200	4444
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela 24:

Mão na Massa! (Vamos Trabalhar?!?!)

- 1) Crie o banco Empresa01.
- 2) Faça a tabela, conforme a tabela 1 da apostila.
- 3) Faça uma consulta onde o número do departamento seja igual a 2, retornando somente o código do funcionário e o departamento.
- 4) Liste todos os funcionários que ganham menos que R\$ 1000,00, retornando somente o nome do funcionário.
- 5) Faça uma consulta que retorne todos os atributos dos funcionários que trabalham no departamento 1.

Todas as consultas devem ser escritas em álgebra relacional também.

Referências Bibliográficas

ELMASRI, Ramez & NAVATHE, Shamkant, Sistema de Banco de dados; Editora Pearson; 2005;

KAMAKRISHNAN & GEHRKE, Sistema de Gerenciamento de Banco de Dados; Editora Mc Graw Hill; 2008;

DATE, C. J. – Introdução a Sistemas de Bancos de Dados, Campus, 8ª ed., 2004.

HEUSER, Carlos Alberto – Projeto de Banco de Dados. 1ª ed 2009. Editora Bookman

<https://www.hashtagtreinamentos.com/curso-basico-de-sql>

<https://www.devmedia.com.br/instalando-e-configurando-a-nova-versao-do-mysql/25813>