BANCO DE DADOS

AULA 5

Prof. Ricardo Sonaglio Albano
Profª Silvie Guedes Albano

CONVERSA INICIAL

Nesta etapa, estudaremos sobre subconsultas, usadas com outros comandos aninhados para otimizar os resultados de comandos *Data Manipulation Language* (DML) e *Data Query Language* (DQL).

Apresentaremos também as principais funções de formatação para dados textuais, numéricos e temporais, incluindo aspectos de agregação ou extração de dados que fornecem detalhes, como resumos, totalizadores e dados estatísticos. Aprenderemos através de exemplos práticos e situações simuladas para esclarecer conceitos.

Finalmente, discutiremos sobre a importância da aplicação de mecanismos e comandos que assegurem a integridade e a segurança dos dados por meio de comandos e mecanismos adequados, assim como os aspectos que diferenciam esses dois conceitos tão fundamentais quando se trata de Banco de Dados.

TEMA 1 – *SUBQUERIES*

Subconsultas ou *subqueries* são linhas de comando *select* que estão incluídas (aninhadas) dentro da estrutura de outros comandos (*select*, *insert*, *update* ou *delete*), inclusive em outras *subqueries*, ou seja, a declaração de uma *query* dentro de outra *query*. Esse encadeamento de comandos torna a *query* mais abrangente e otimizada, pois o resultado de uma *subquery* impactará nas demais *subqueries* em um efeito em cadeia (de dentro para fora).

Em alguns casos, as subconsultas podem ser substituídas por opções alternativas de declaração, como o uso do comando *join*, sendo que em termos de desempenho, geralmente, o *join* é a melhor escolha.

Regras da *subquery*:

- O *select* de uma *subquery* é sempre declarado entre parênteses;
- Pode estar inserida em uma lista de *select*, ou seja, um encadeamento de *selects*, utilizando as cláusulas *where* ou *having* da *query* externa;
- Pode estar inserida na lista de *select* para ampliar o conjunto de valores que serão analisados através do filtro *where* ou *having* (*query* externa);
- Em aninhamentos normais, a *subquery* mais interna possui precedência na execução, ou seja, será executada primeiro.

A *subquery* pode ser aplicada em três situações específicas:

1. Como um filtro, sendo declarada dentro de uma cláusula *where*.
2. Atuando como uma nova coluna, sendo declarada dentro de um *select*.
3. Funcionando como uma nova tabela, sendo declarada dentro da cláusula *from* ou do comando *join*.

Figura 1 – *Select* nas das tabelas Cidade, Cliente e Vendas

Tabela Cidade

id	nomecidade	Uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	PI
4	Videira	SC
5	Imperatriz	MA
6	Belo Horizonte	MG
7	São Paulo	SP
NULL	NULL	NULL

Tabela Cliente

id	nome	genero	datanascimento	salario	cidadeid
1	Helena Magalhaes	F	2000-01-01	12500.99	2
2	Nicolas	M	2002-12-10	8500.00	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	1
5	Bia Meireles	F	2002-03-14	9450.00	2
6	Pedro Filho	M	1998-05-22	6800.00	5
7	Helena Magalhaes	F	1994-08-10	8600.00	4
NULL	NULL	NULL	NULL	NULL	NULL

Tabela Vendas

numeroVenda	data	clienteID	ValorCompras
1	2022-02-10	4	1000.00
2	2022-02-10	2	500.00
3	2022-03-11	3	100.00
4	2022-03-11	2	400.00
5	2022-03-11	3	200.00
NULL	NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

De uma forma simples, iremos exemplificar, para que você possa compreender, o funcionamento básico de uma *subquery*.

Uma empresa deseja descobrir todos os clientes que residem na cidade de Curitiba. Ao consultarmos a tabela Cliente, percebemos que tem apenas o ‘id’ da cidade (chave estrangeira). O que podemos fazer? O caminho mais fácil para resolver essa situação é desmembrar em pequenas tarefas, as quais são:

Descobrir qual o ‘id’ que corresponde a cidade de Curitiba na tabela Cidade.

```
select id from cidade where descricao = 'Curitiba';
```

Figura 2 – ‘Id’ da cidade de Curitiba

id
1
NULL

Fonte: Albano e Albano, 2023.

1. Na tabela Cliente, filtrar todas as linhas que correspondem ao ‘id’ da cidade de Curitiba.

```
select nome, cidadeId from cliente where cidadeId = 1;
```

Figura 3 – Clientes que residentes em Curitiba

nome	cidadeID
Ana Rosa Silva	1
Tales Heitor Souza	1

Fonte: Albano e Albano, 2023.

Dessa forma, conseguimos resolver em duas etapas, mas podemos realizar a mesma tarefa em um único processo, utilizando uma *subquery*. Esse procedimento irá produzir um resultado equivalente a consulta anterior, sendo:

```
select nome, cidadeId from cliente
    where cidadeId =
    (select id from cidade where nomecidade = 'Curitiba');
```

Perceba que ao aplicarmos o uso de uma *subquery*, simplesmente, incluímos a primeira consulta (entre parênteses) dentro da consulta principal.

Nota

Aqui vale uma ressalva quanto à limitação do uso do símbolo da igualdade (=), que permite apenas a comparação de um único valor.

Existem situações que podemos substituir uma *subquery* pelo comando *join*. Por exemplo, na consulta de clientes que residem na cidade de Curitiba, podemos também resolver com o comando *join*.

Script usando o *join*:

```
select nome, cidadeId from cliente inner join cidade
    on cliente.cidadeid = cidade.id
    where nomecidade = 'Curitiba';
```

As cláusulas para tratamento de subconsultas são os operadores relacionais [*not*] *in*, *any*, *all* e [*not*] *exists*.

1.1 Operadores relacionais (=, >, <, >= e <=)

O uso dos operadores relacionais só poderá ser aplicado nos casos em que a *subquery* retornar um valor.

Sintaxe: **`where expressao = (subconsulta);`**

Exemplo: qual é o cliente mais jovem? Para podermos obter esse resultado, teremos que descobrir a data de nascimento mais recente e depois comparar com a data de nascimento de todos os clientes.

```
select nome, genero, dataNascimento from cliente

        where dataNascimento = (select max(dataNascimento)

        from cliente);
```

Figura 4 – Cliente mais jovem

	nome	genero	DataNascimento
▶	Nicolas	M	2002-12-10

Fonte: Albano e Albano, 2023.

1.2 *In*

Verifica se um ou mais valores, presentes em uma coluna, possuem correspondência em uma lista ou em um resultado de uma subconsulta, sendo definida como parâmetro em uma cláusula *where*.

Sintaxe: **`where expressao [not] in (subconsulta);`**

Exemplo: quais clientes residem nas cidades de Curitiba (1) e Imperatriz (5)?

Consulta com valores predefinidos:

```
select nome,cidadeId from cliente where cidadeId in (1,5);
```

Consulta dinâmica:

```
select nome, cidadeId from cliente where cidadeId
```

```
in (select id from cidade where nomeCidade = 'Curitiba'

or nomeCidade = 'Imperatriz');
```

Figura 5 – *Subquery* com o uso do in

nome	cidadeID
Ana Rosa Silva	1
Tales Heitor Souza	1
Pedro Filho	5

Fonte: Albano e Albano, 2023.

Perceba que é uma consulta com dois valores. Assim, não podemos usar o operador de comparação (=), que compara somente um único valor. Dessa forma, usamos o *in*, que possibilita a comparação de vários valores.

Também é possível negar a consulta usando o operador *not*, ou seja, quais os clientes não residem em Curitiba ou Imperatriz.

```
select nome, cidadeId from cliente

where cidadeId not in (1,5);
```

Figura 6 – Subquery com o uso do not

nome	cidadeID
Helena Magalhaes	2
Nicolas	3
Bia Meireles	2
Helena Magalhaes	4

Fonte: Albano e Albano, 2023.

1.3 *Exists*

Esse argumento em uma subconsulta (também conhecida como *select* arbitrária) é responsável pela realização de um teste de existência, retornando um valor booleano. Se a subconsulta retornar, no mínimo, uma linha, isso significa que o *exists* é verdadeiro (*true*), caso contrário, o argumento *exists* é falso (*false*). Também é possível usar nesse argumento a negação *not exists*.

Outro detalhe interessante é que a palavra-chave *exists* não é precedida por um nome de coluna, constante ou qualquer outra expressão.

Sintaxe: `where [not] exists (subconsulta);`

Exemplo: verificar se existem clientes com renda superior a R$ 11.000,00? Se sim, quais clientes têm renda inferior a R$ 7.000,00?

```
select nome, genero, salario from cliente

        where salario < 7000 and

        exists (select * from cliente where salario > 11000);
```

Figura 7 – *Subquery* com *exists*

nome	genero	salario
Pedro Filho	M	6800.00

Fonte: Albano e Albano, 2023.

Para entender o resultado dessa consulta, considere o salário de cada cliente. Pergunte-se se esse valor faz a subconsulta retornar pelo menos uma linha. Em outras palavras, a consulta faz com que o teste de existência seja avaliado como *true*?

Observe que as subconsultas que utilizam o *exists* apresentam algumas diferenças entre as demais subconsultas. São elas:

- Retorna verdadeiro ou falso;
- Possui como argumento um *select*;
- A *subquery* não devolve nenhuma tabela resultante, pois não existe razão para listar nomes de colunas, uma vez que estamos apenas testando se existem linhas que satisfazem as condições especificadas na subconsulta, ou seja, os valores do resultado são irrelevantes, importando apenas saber se há ou não resultado na subconsulta;
- *Exists* pode ser negado por um *not*.

1.4 *Any*

Esse predicado permite recuperar registros da consulta principal que satisfaçam a condição de comparação com qualquer outro registro recuperado na subconsulta.

Sintaxe: `where any (subconsulta);`

Exemplo: projetar o código, o nome e o gênero dos clientes que possuem o ‘id’ maior que o ‘id’ de qualquer cliente que já realizou alguma compra.

```
select id, nome, genero from cliente

        where id > any (select distinct clienteId from
    vendas);
```

A subconsulta (entre parênteses) retorna os ‘ids’ dos clientes (2, 3 e 4) que realizaram no mínimo uma compra. O resultado contendo os ‘ids’ dos compradores serão comparados com os ‘ids’ dos clientes e, somente os ‘ids’ maiores que 2, serão apresentados na consulta. Por esse motivo, os ‘ids’ 1 e 2 não aparecem no resultado.

Figura 8 – *Subquery* com uso do *any*

id	nome	genero
3	Ana Rosa Silva	F
4	Tales Heitor Souza	M
5	Bia Meireles	F
6	Pedro Filho	M
7	Helena Magalhaes	F
NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

1.5 *All*

Esse predicado recupera unicamente os registros da consulta principal que satisfazem a comparação com todos os registros recuperados na subconsulta. A condição deverá ser satisfeita para todos os elementos de um conjunto (=, >, < ou <>).

Exemplo: projetar o código, o nome e o gênero dos clientes que possuem o ‘id’ maior que o ‘id’ de todos os clientes que já realizaram alguma compra.

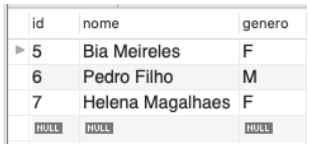
```
select id, nome, genero from cliente

        where id > all (select distinct clienteId from
    vendas);
```

A subconsulta (entre parênteses) retorna os ‘ids’ dos clientes (2, 3 e 4) que realizaram no mínimo uma compra. Os ‘ids’ da relação de compradores resultante serão comparados com os ‘ids’ dos clientes, e somente os ‘ids’

maiores que 2 serão apresentados na consulta. Por esse motivo, os números menores do que 5 não aparecem no resultado.

Figura 9 – *Subquery* com uso do *all*



id	nome	genero
5	Bia Meireles	F
6	Pedro Filho	M
7	Helena Magalhaes	F
NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

1.6 Subquery correlacionada

Esse predicado recupera unicamente os registros da consulta principal que satisfazem a comparação com todos os registros recuperados na subconsulta. A condição deverá ser satisfeita para todos os elementos de um conjunto.

Exemplo: projetar o código, o nome, o gênero e o salário dos clientes que ganham um salário superior a outros clientes do mesmo gênero.

```
select id, nome, genero, salario from cliente c
        where salario > any (select salario from cliente
        where genero = c.genero);
```

Figura 10 – *Subquery* correlacionada

id	nome	genero	salario
1	Helena Magalhaes	F	12500.99
2	Nicolas	M	8500.00
4	Tales Heitor Souza	M	7689.00
5	Bia Meireles	F	9450.00
7	Helena Magalhaes	F	8600.00
NULL	NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

Esse tipo de subconsulta é um pouco diferente das demais. Para cada linha da consulta externa, será executada uma subconsulta. Note que a tabela da consulta externa possui um *alias* (c), o qual é usado na subconsulta. Isso é uma regra da subconsulta correlacionada.

Nesse exemplo, estamos comparando o salário de um cliente com todos os salários dos clientes do mesmo gênero. Por isso, a cliente Ana Rosa Silva não aparece no resultado, pois ela possui um salário inferior às demais clientes do gênero feminino. O mesmo acontece com o cliente Pedro Filho.

É importante ressaltar que no SQL padrão existem duas cláusulas denominadas *except* e *intersect*, as quais atuam, respectivamente, na diferença e na intersecção (valores em comum) entre duas consultas. Contudo, o MySQL não implementa essas cláusulas.

De maneira alternativa, podemos resolver as limitações do MySQL simulando a função do *except* com o *in* e o *intersect* utilizando o *in* ou o *exists*.

Simulando o *intersect*: quais cidades possuem clientes?

```
select distinct cidadeId from cliente
        where cidadeId in (select id from cidade);
```

Figura 11 – Simulando o *intersect*

cidadeid
1
2
3
4
5

Fonte: Albano e Albano, 2023.

Simulando o *except*: quais cidades não possuem clientes cadastrados?

```
select distinct id from cidade
        where id not in (select cidadeId from cliente);
```

Figura 12 – Simulando o *except*

id
6
7
NULL

Fonte: Albano e Albano, 2023.

TEMA 2 – FORMATAÇÃO DE DADOS TEXTUAIS

O MySQL tem diversas funções para tratamento e manipulação de caracteres, ou seja, dos dados armazenados. Em sua maioria, envolvem operações de conversão, teste, localização, substituição, concatenação e remoção. Nessa seção, abordaremos as funções mais utilizadas.

Figura 13 – *Select* da tabela Cliente

id	nome	genero	dataNascimento	placaVeiculo	salario	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	ABC-1234	12500.00	helena#email.com	2
2	Nicolas	M	2002-12-10	bbc-4567	9800.00	nicolas#email.com	3
3	Ana Rosa Silva	F	1996-12-31	XPT-99541	11453.23	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	PFA-0836	NULL	tales.heitor@email.com	1
5	Bia Meireles	F	2002-03-14	qer-3809	0.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	MLQ-5408	6500.00	pedro.filho@email.com	5

Fonte: Albano e Albano, 2023.

2.1 *Length*()

A função *length* retorna o tamanho da coluna, sendo muito utilizada em situações em que existe a necessidade de obter o tamanho da cadeia de caracteres para realizar uma ação.

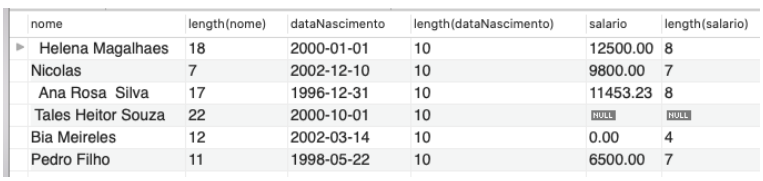
Sintaxe: **`select length(coluna) from <tabela>`**

`where <condição>;`

Exemplo:

```
select nome, length(nome), dataNascimento,
length(dataNascimento), salario, length(salario)
from cliente;
```

Figura 14 – *Select* com o uso do *length*



nome	length(nome)	dataNascimento	length(dataNascimento)	salario	length(salario)
Helena Magalhaes	18	2000-01-01	10	12500.00	8
Nicolas	7	2002-12-10	10	9800.00	7
Ana Rosa Silva	17	1996-12-31	10	11453.23	8
Tales Heitor Souza	22	2000-10-01	10	NULL	NULL
Bia Meireles	12	2002-03-14	10	0.00	4
Pedro Filho	11	1998-05-22	10	6500.00	7

Fonte: Albano e Albano, 2023.

2.2 *Upper*() e *Lower*()

A função *upper* tem como objetivo converter para letras maiúsculas um valor de um tipo *char* ou *varchar*, ou seja, um texto. Já a função *lower* faz o inverso, isto é, converte uma cadeia de caracteres para letras minúsculas.

Sintaxes:

```
select upper(coluna) from <tabela> where <condição>;

select lower(coluna) from <tabela> where <condição>;
```

Exemplo:

```
select nome, upper(nome) as 'maiúscula',

   lower(nome) as 'minuscula' from cliente;
```

Figura 15 – *Select* com o uso do *upper* e *lower*

nome	maiúscula	minuscula
Helena Magalhaes	HELENA MAGALHAES	helena magalhaes
Nicolas	NICOLAS	nicolas
Ana Rosa Silva	ANA ROSA SILVA	ana rosa silva
Tales Heitor Souza	TALES HEITOR SOUZA	tales heitor souza
Bia Meireles	BIA MEIRELES	bia meireles
Pedro Filho	PEDRO FILHO	pedro filho

Fonte: Albano e Albano, 2023.

2.3 *Ltrim*(), *Rtrim*() e *Trim*()

São funções utilizadas para a remoção de espaços vazios presentes em colunas de cadeia de caracteres, sendo:

1. *Ltrim* – remove os espaços vazios que estão no início da cadeia de caracteres. O ‘L’ equivale a *left* (esquerda).
2. *Rtrim* – remove os espaços à direita da cadeia de caracteres. O ‘R’ equivale a *right* (direita).
3. *Trim* – remove os espaços que estão à esquerda e/ou à direita da cadeia de caracteres. Podemos usar os parâmetros *both* (retira os espaços das duas extremidades), *leading* (retira somente os espaços da esquerda) e *trailing* (retira somente os espaços da direita, isto é, do final).

Sintaxes:

```
select ltrim(coluna) from <tabela> where <condição>;

select rtrim(coluna) from <tabela> where <condição>;

select trim([parametro]coluna)
   from <tabela> where <condição>;
```

Exemplo com *ltrim*:

```
select nome, length(nome), ltrim(nome),
   length(ltrim(nome)) from cliente;
```

Figura 16 – *Select* com uso do *ltrim*

nome	length(nome)	ltrim(nome)	length(ltrim(nome))
Helena Magalhaes	18	Helena Magalhaes	16
Nicolas	7	Nicolas	7
Ana Rosa Silva	17	Ana Rosa Silva	15
Tales Heitor Souza	22	Tales Heitor Souza	21
Bia Meireles	12	Bia Meireles	12
Pedro Filho	11	Pedro Filho	11

Fonte: Albano e Albano, 2023.

Perceba que, usando o *ltrim* na coluna nome, foram eliminados os espaços vazios no início dos nomes e, dessa forma, o tamanho ocupado pela coluna também mudou. Essa mesma situação ocorre na aplicação das funções *rtrim* e *trim*.

Exemplo com *rtrim*:

```
select nome, length(nome), rtrim(nome),
   length(rtrim(nome)) from cliente;
```

Figura 17 – *Select* com uso do *rtrim*

nome	length(nome)	Rtrim(nome)	length(rtrim(nome))
Helena Magalhaes	18	Helena Magalhaes	18
Nicolas	7	Nicolas	7
Ana Rosa Silva	17	Ana Rosa Silva	17
Tales Heitor Souza	22	Tales Heitor Souza	19
Bia Meireles	12	Bia Meireles	12
Pedro Filho	11	Pedro Filho	11

Fonte: Albano e Albano, 2023.

Exemplo com *trim*:

```
select trim(nome), trim(both from nome),
   trim(leading from nome),
   trim(trailing from nome) from cliente;
```

Figura 18 – *Select* com uso do *trim*

trim(nome)	trim(both from nome)	trim(leading from nome)	trim(trailing from nome)
Helena Magalhaes	Helena Magalhaes	Helena Magalhaes	Helena Magalhaes
Nicolas	Nicolas	Nicolas	Nicolas
Ana Rosa Silva	Ana Rosa Silva	Ana Rosa Silva	Ana Rosa Silva
Tales Heitor Souza	Tales Heitor Souza	Tales Heitor Souza	Tales Heitor Souza
Bia Meireles	Bia Meireles	Bia Meireles	Bia Meireles
Pedro Filho	Pedro Filho	Pedro Filho	Pedro Filho

Fonte: Albano e Albano, 2023.

2.4 *Substring*()

Recurso bastante interessante, pois permite a pesquisa e a seleção de parte de uma *string*.

Sintaxe:
```
substring(expressão, início, tamanho);

substring(expressão from início for tamanho);
```

Onde:

- expressão: indica a *string* a ser pesquisada;
- início: índice inicial da pesquisa na expressão fornecida;
- tamanho: comprimento da *string* do resultado.

Exemplo: extrair somente as três primeiras letras das placas dos veículos dos clientes.

Existem duas formas de resolver esse exemplo, sendo que as duas retornaram o mesmo resultado.

Primeira forma:
```
select placaVeiculo,
   substring(placaVeiculo, 1, 3) from cliente;
```

Segunda forma:
```
select placaVeiculo,
   substring(placaVeiculo from 1 for 3)
```

```
from cliente;
```

Figura 19 – *Substring* dos dois *scripts*

placaVeiculo	substring(placaVeiculo, 1, 3)
ABC-1234	ABC
bbc-4567	bbc
XPT-99541	XPT
PFA-0836	PFA
qer-3809	qer
MLQ-5408	MLQ

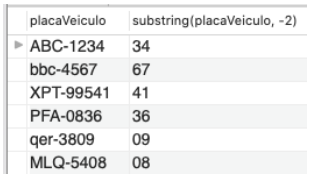
placaVeiculo	substring(placaVeiculo from 1 for 3)
ABC-1234	ABC
bbc-4567	bbc
XPT-99541	XPT
PFA-0836	PFA
qer-3809	qer
MLQ-5408	MLQ

Fonte: Albano e Albano, 2023.

Ao usarmos um número negativo, significa que a extração começará pela direita, ou seja, pelo final da *string*.

Exemplo:
```
select placaVeiculo,
substring(placaVeiculo, -2) from cliente;
```

Figura 20 – *Substring* com número negativo



placaVeiculo	substring(placaVeiculo, -2)
ABC-1234	34
bbc-4567	67
XPT-99541	41
PFA-0836	36
qer-3809	09
MLQ-5408	08

Fonte: Albano e Albano, 2023.

2.5 *Replace*()

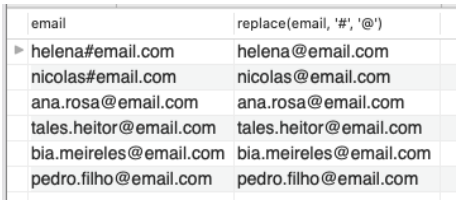
Baseado em uma expressão definida, a função *replace* realiza uma busca em uma determinada *string* e, ao encontrar, substitui o valor da *string* pelo valor especificado no argumento.

Sintaxe:
```
replace('expressao' ou coluna,
'string a consultar', 'substituir');
```

Exemplo: substituir o caractere # (sustenido) pelo caractere @ (arroba) no e-mail.

```
select email, replace(email, '#', '@') from cliente;
```

Figura 21 – *Select* com uso do *replace*



email	replace(email, '#', '@')
helena#email.com	helena@email.com
nicolas#email.com	nicolas@email.com
ana.rosa@email.com	ana.rosa@email.com
tales.heitor@email.com	tales.heitor@email.com
bia.meireles@email.com	bia.meireles@email.com
pedro.filho@email.com	pedro.filho@email.com

Fonte: Albano e Albano, 2023.

2.6 *Cast*()

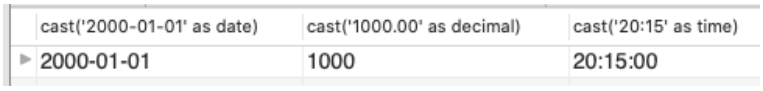
Permite a conversão de um tipo de dado em outro tipo de dado.

Sintaxe: **`cast(value as datatype);`**

Exemplo:

```
select cast('2000-01-01' as date),
    cast('1000.00' as decimal), cast('20:15' as time);
```

Figura 22 – *Select* com uso do *cast*



cast('2000-01-01' as date)	cast('1000.00' as decimal)	cast('20:15' as time)
2000-01-01	1000	20:15:00

Fonte: Albano e Albano, 2023.

TEMA 3 – FORMATAÇÃO DE DADOS NUMÉRICOS E TEMPORAIS

Nesta seção, trataremos das funções fornecidas pelo MySQL para obtenção, conversão e manipulação de dados dos tipos numérico, data e hora.

É importante ressaltar que trabalhar com valores parciais ou completos do tipo temporal sempre requer um pouco mais de atenção, pois eles têm características específicas.

3.1 *Round*()

Arredondamento de valores numéricos com casas decimais.

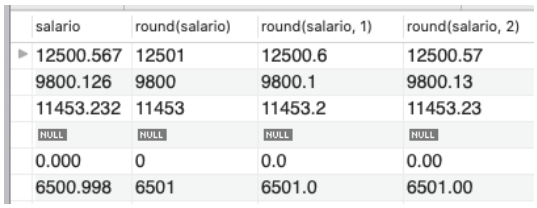
Sintaxe: **`select round(coluna, casas decimais)`**

```
from <tabela> where <condição>;
```

Exemplo:

```
select salario, round(salario), round(salario, 1),
round(salario, 2) from cliente;
```

Figura 23 – *Select* com uso do *round*



salario	round(salario)	round(salario, 1)	round(salario, 2)
12500.567	12501	12500.6	12500.57
9800.126	9800	9800.1	9800.13
11453.232	11453	11453.2	11453.23
NULL	NULL	NULL	NULL
0.000	0	0.0	0.00
6500.998	6501	6501.0	6501.00

Fonte: Albano e Albano, 2023.

Avaliando a tabela, podemos perceber que os valores que apresentaram a última casa decimal igual ou superior a 5 tiveram o valor arredondado para mais, caso contrário, o valor foi arredondado para menos.

Podem existir situações em que o arredondamento não seja a melhor alternativa para trabalharmos com casas decimais. Para esses casos, podemos usar o *truncate*.

3.2 *Truncate*()

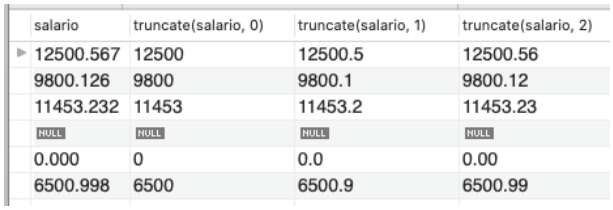
Retorna o número truncado em *N* casas decimais. Se a definição do número de casas decimais for igual a zero (0), não haverá ponto decimal.

Sintaxe:
```
select truncate(coluna, casas decimais)
from <tabela> where <condição>;
```

Exemplo:

```
select salario, truncate(salario,0), truncate(salario,1),
truncate(salario, 2) from cliente;
```

Figura 24 – *Select* com uso do *truncate*



salario	truncate(salario, 0)	truncate(salario, 1)	truncate(salario, 2)
12500.567	12500	12500.5	12500.56
9800.126	9800	9800.1	9800.12
11453.232	11453	11453.2	11453.23
NULL	NULL	NULL	NULL
0.000	0	0.0	0.00
6500.998	6500	6500.9	6500.99

Fonte: Albano e Albano, 2023.

Perceba que no *truncate* não existe arredondamento dos valores, apenas a quebra ou limitação de casas para a apresentação do valor.

3.3 *Mod*() e *Div*()

A função *mod* retorna o resto da divisão, sendo muito útil quando precisamos saber se um número é par (resto igual a zero) ou ímpar (resto igual a um). Já a função *div* retorna o quociente (parte inteira) da divisão.

Sintaxes: `select mod(dividendo, divisor);`

`select dividendo div divisor;`

Exemplo:

```
select mod(4, 2) as 'resto divisão',
   mod(5,2) as 'resto divisão',
   4 div 2 as 'quociente',
   5 div 2 as 'quociente';
```

Figura 25 – *Select* com uso do mod e do *div*

resto divisão	resto divisão	quociente	quociente
0	1	2	2

Fonte: Albano e Albano, 2023.

3.4 *Curdate*(), *Curtime*() e *Now*()

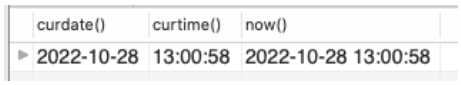
A função *curdate* retorna a data corrente, sendo a função *current_date*() sinônimo da *curdate*().

A função *curtime* retorna a hora corrente, sendo a função *current_time*() sinônimo na função *curtime*().

Já a função *now* retorna a data e a hora corrente.

Exemplo: `select curdate(), curtime(), now();`

Figura 26 – Funções *curdate*, *curtime* e *now*



curdate()	curtime()	now()
2022-10-28	13:00:58	2022-10-28 13:00:58

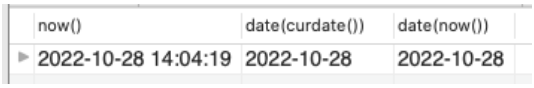
Fonte: Albano e Albano, 2023.

3.5 *Date*()

A função `date(expressão)` realiza a extração de uma data de um tipo *date* ou *datetime*.

Exemplo: `select now(), date(curdate()), date(now());`

Figura 27 – Uso da função *date*



now()	date(curdate())	date(now())
2022-10-28 14:04:19	2022-10-28	2022-10-28

Fonte: Albano e Albano, 2023.

3.6 *Day*() e suas variações

A função *day* possui algumas variações, as quais são:

- `day(date)`: retorna o dia de uma data, sendo sinônimo da função *dayofmonth*();
- `dayname(date)`: retorna o nome do dia da semana;
- `dayofweek(date)`: retorna o dia da semana;
- `dayofyear(date)`: retorna o dia do ano;
- `last_day(date)`: retorna o último dia do mês.

Exemplo:

```
select curdate(), day(curdate()),
dayname(curdate()),
dayofweek(curdate()), dayofyear(curdate()),
last_day(curdate());
```

Figura 28 – Função *day* e suas variações

curdate()	day(curdate())	dayname(curdate())	dayofweek(curdate())	dayofyear(curdate())	last_day(curdate())
2022-10-28	28	Friday	6	301	2022-10-31

Fonte: Albano e Albano, 2023.

3.7 *Week*() e *Weekday*()

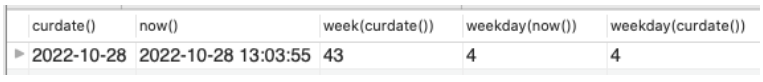
Funções *week* e *weekday*:

- **`week(date[,mode])`**: retorna a semana do ano de um tipo *date*, sendo a função **`weekofyear(date)`** semelhante à função *week*. O *model* indica se a semana começa no domingo (0) ou na segunda (1).
- **`weekday(date)`**: retorna o dia da semana, sendo: 0 = *monday*, 1 = *tuesday*, …, 6 = *sunday*.

Exemplo:

```
select curdate(), now(), week(curdate()),
weekday(now()), weekday(curdate());
```

Figura 29 – Funções *week* e *weekday*



curdate()	now()	week(curdate())	weekday(now())	weekday(curdate())
2022-10-28	2022-10-28 13:03:55	43	4	4

Fonte: Albano e Albano, 2023.

3.8 *Month*(), *Monthname*() e *Year*()

A função **`month(date)`** retorna o número do mês correspondente a uma data. A função **`monthname(date)`**, por sua vez, retorna o nome do mês de uma data. Já a função **`year(date)`** retorna o ano de uma data.

Exemplo:

```
select month(curdate()), monthname(curdate()),
    year(curdate());
```

Figura 30 – Funções *month*, *monthname* e *year*

month(curdate())	monthname(curdate())	year(curdate())
10	October	2022

Fonte: Albano e Albano, 2023.

3.9 *Adddate*()

Adiciona a uma data um intervalo predefinido, gerando uma nova data.

Sintaxe: **`adddate(data, intervalo_data expressão unidade);`**

Exemplo:

```
select curdate(), adddate(curdate(), interval 31 day),
    adddate(curdate(), interval 1 month);
```

Figura 31 – Uso da função *adddate*

curdate()	adddate(curdate(), interval 31 day)	adddate(curdate(), interval 1 month)
2022-10-28	2022-11-28	2022-11-28

Fonte: Albano e Albano, 2023.

3.10 *Datediff*()

A função **`datediff(data1, data2)`** calcula a diferença em dias (inteiro) do intervalo entre duas datas informadas.

Exemplo:

```
select curdate(), datediff('2022-01-01', curdate()),
  datediff(curdate(), '2022-01-01');
```

Figura 32 – Uso da função *datediff*

	curdate()	datediff('2022-01-01',curdate())	datediff(curdate(), '2022-01-01')
▶	2022-10-28	-300	300

Fonte: Albano e Albano, 2023.

3.11 *Date_Format*()

A função **`date_format(date, format)`** formata a data de acordo com o formato especificado.

Exemplo:

```
select curdate(), date_format(curdate(), '%w %m %y'),
  date_format('2022-01-01 20:15:00', '%h:%i:%s');
```

Figura 33 – Uso da função *date_format*

	curdate()	date_format(curdate(), '%w %m %y')	date_format('2022-01-01 20:15:00', '%h:%i:%...
▶	2022-10-28	5 10 22	08:15:00

Fonte: Albano e Albano, 2023.

Onde:

- %w – equivale ao dia da semana, iniciando no número 0 (domingo) e indo até o número 6 (sábado);
- %m – equivale ao mês;
- %y – equivale ao ano;
- %h – equivale à hora;
- %i – equivale ao minuto;
- %s – equivale ao segundo.

Nota

Consulte a relação de formatos. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>. Acesso em: 2 fev. 2023.

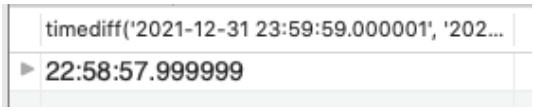
3.12 *Timediff*()

A função **`timediff (expressão 1, expressão 2)`** retorna a diferença entre dois tempos definidos.

Exemplo:

```
select timediff('2021-12-31 23:59:59.000001',
    '2021-12-30 01:01:01.000002');
```

Figura 34 – Uso da função *timediff*



Fonte: Albano e Albano, 2023.

3.13 *Time*() e suas Subdivisões

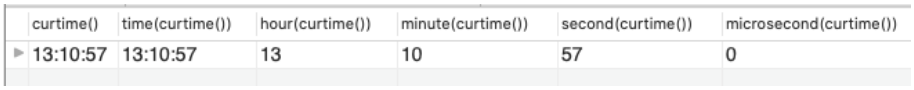
Função *time* e suas subdivisões:

- **`time(expressão)`**: extrai a hora de um tipo *datetime*;
- **`hour(time)`**: extrai somente a hora de um tempo;
- **`minute(time)`**: extrai somente os minutos de um tempo;
- **`second(time)`**: extrai somente os segundos de um tempo;
- **`microsecond(expressão)`**: extrai somente os microssegundos de um tempo.

Exemplo:

```
select curtime(), time(curtime()), hour(curtime()),
    minute(curtime()), second(curtime()),
    microsecond(curtime());
```

Figura 35 – Uso da função *time* e suas subdivisões



curtime()	time(curtime())	hour(curtime())	minute(curtime())	second(curtime())	microsecond(curtime())
13:10:57	13:10:57	13	10	57	0

Fonte: Albano e Albano, 2023.

3.14 *Addtime*() e *Timestamp*()

A função **`addtime(expressão 1, expressão 2)`** adiciona um intervalo de tempo a um tipo *date* ou *datetime*. Já as funções **`timestamp(expressão)`** e **`timestamp(expressão 1, expressão 2)`** adicionam a um tipo *date* ou *datetime* a quantidade de tempo informada.

Exemplos:

```
select addtime('01:00:00.999999', '02:00:00.999998');

select timestamp('2003-12-31');

select timestamp('2003-12-31 12:00:00', '12:00:00');
```

Figura 36 – Funções *addtime* e *timestamp*

addtime('01:00:00.999999', '02:00:00.9999...
03:00:01.999997

timestamp('2003-12-31')	timestamp('2003-12-31 12:00:00','12:00:00')
2003-12-31 00:00:00	2004-01-01 00:00:00

Fonte: Albano e Albano, 2023.

3.15 *Timestampadd*()

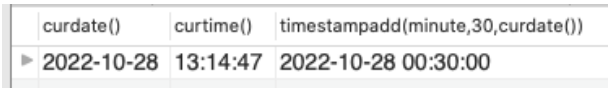
Adiciona uma fração de tempo a um tipo *datetime*.

Sintaxe: **`timestampadd(unit, interval, datetime_expr);`**

Exemplo:

```
select curdate(), curtime(),
        timestampadd(minute, 30, curdate()),
        timestampadd(week, 1, 'curtime()');
```

Figura 37 – Uso da função *timestampadd*



curdate()	curtime()	timestampadd(minute,30,curdate())
2022-10-28	13:14:47	2022-10-28 00:30:00

Fonte: Albano e Albano, 2023.

3.16 *Time_Format*()

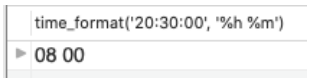
Formata um valor do tipo *time*.

Sintaxe: `time_format(time, format);`

Exemplo:

```
select time_format('100:00:00', '%h %k %h %i %l');
```

Figura 38 – Uso da função *time_format*



time_format('20:30:00', '%h %m')
08 00

Fonte: Albano e Albano, 2023.

TEMA 4 – AGREGAÇÃO / EXTRAÇÃO DE DADOS

Em um primeiro momento, pode parecer que um banco de dados sirva apenas para o armazenamento de dados, mas na verdade é o processo de recuperação, análise e transformação em informações significativas (com valor agregado). Dessa forma, as funções que auxiliam na sintetização ou sumarização de informações têm uma grande importância no banco de dados.

Vale salientar que essas funções descartam os valores nulos presentes nas colunas, ou seja, valores *null* não são contabilizados.

Figura 39 – *Select* na tabela Funcionário

matricula	nomeFunc	generoFunc	NascFunc	salarioFunc	departamento	cargo	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	1
2	Tales Heitor Souza	M	2000-10-01	7689.00	1	2	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	2
4	Pedro Filho	M	1998-05-22	12340.00	3	3	2
5	Camila Fialho	F	1989-03-15	10450.00	2	3	4
6	Ulisses Frota	M	1997-06-30	12340.00	1	4	7
7	Leonardo Timbo	NULL	2001-07-02	7850.00	2	3	2
8	Lucas Goes	M	2002-03-02	8834.00	3	4	5
9	Sofia Lima	NULL	1999-12-23	9578.00	4	4	5
10	Nicolas figueira	M	1997-06-01	12340.00	3	2	3
11	Helena Arcanjo	F	1998-11-20	6320.00	2	2	7
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

4.1 *Count*()

Retorna a quantidade de registros correspondentes a uma pesquisa.

Sintaxe: **`select count(*) from <tabela> where <coluna>;`**

Exemplo 1: Quantos funcionários existem na tabela Funcionário?

```
select count(*), count(generoFunc) from funcionario;
```

Figura 40 – Uso da função *count*

count(*)	count(generoFunc)
11	9

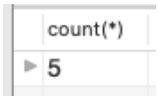
Fonte: Albano e Albano, 2023.

Perceba que usamos duas vezes a função *count* e o resultado foi diferente. No primeiro *count*, usamos o asterisco (*), o que indica ao Sistema Gerenciador de Banco de Dados (SGBD) que todas as linhas da tabela devem ser computadas. No segundo caso, quando usamos o nome de uma coluna, indicamos ao SGBD que apenas as linhas em que a coluna apresente valor devem ser computadas. Por isso, o resultado é dois funcionários a menos do que o total de funcionários, pois na tabela existem dois funcionários sem a informação de gênero (*null*).

Exemplo 2: há quantos funcionários do sexo masculino?

```
select count(*) from funcionario where generoFunc = 'm';
```

Figura 41 – Uso da função *count*



count(*)
5

Fonte: Albano e Albano, 2023.

4.2 *Sum*()

Retorna o somatório dos valores de uma coluna.

Sintaxe: **`select sum(coluna) from <tabela> [where <coluna>];`**

Exemplo: mostrar a soma dos salários de todos os funcionários.

```
select sum(salarioFunc) from funcionario;
```

Figura 42 – Uso da função *sum*

sum(salarioFunc)
105691.00

Fonte: Albano e Albano, 2023.

4.3 *Min*()

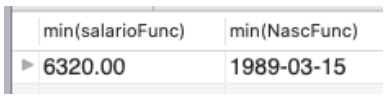
Retorna o menor valor encontrado em uma coluna.

Sintaxe: **`select min(coluna) from <tabela> [where <coluna>];`**

Exemplo: qual o menor salário entre todos os funcionários? E qual o funcionário mais velho?

```
select min(salarioFunc), min(NascFunc) from funcionario;
```

Figura 43 – Uso da função *min*



min(salarioFunc)	min(NascFunc)
6320.00	1989-03-15

Fonte: Albano e Albano, 2023.

4.4 *Max*()

Retorna o maior valor encontrado em uma coluna.

Sintaxe: **`select max(coluna) from <tabela> [where <coluna>];`**

Exemplo: qual o maior salário entre todos os funcionários? E qual o funcionário mais novo?

```
select max(salarioFunc), max(NascFunc) from funcionario;
```

Figura 44 – Uso da função *max*

max(salarioFunc)	max(NascFunc)
12340.00	2002-03-14

Fonte: Albano e Albano, 2023.

4.5 *Avg*()

Realiza a média dos valores de uma coluna.

Sintaxe: **`select avg(coluna) from <tabela> [where < coluna >];`**

Exemplo: qual a média salarial dos funcionários?

```
select avg(salario) from funcionario;
```

Figura 45 – Uso da função *avg*

avg(salarioFunc)
9608.272727

Fonte: Albano e Albano, 2023.

4.6 *Group By*()

Agrupa todas as linhas do retorno de uma pesquisa que possuem o valor de suas colunas iguais.

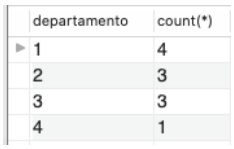
Sintaxe:

```
select (<coluna_agrupamento> e/ou <função_agrupamento>)
   from <tabela> [where <condição>]
   group by <coluna_agrupamento> [having <condição>};
```

Exemplo 1: quantos funcionários existem por departamento?

```
select departamento, count(*) from funcionario
   group by departamento order by departamento;
```

Figura 46 – Uso da função *group by* na coluna 'departamento'



departamento	count(*)
1	4
2	3
3	3
4	1

Fonte: Albano e Albano, 2023.

Exemplo 2: qual a média salarial por gênero?

```
select generoFunc, avg(salarioFunc) from funcionario
   group by generoFunc;
```

Figura 47 – Uso da função *group by* na coluna 'generoFunc'

generoFunc	avg(salarioFunc)
F	8680.000000
M	10708.600000
NULL	8714.000000

Fonte: Albano e Albano, 2023.

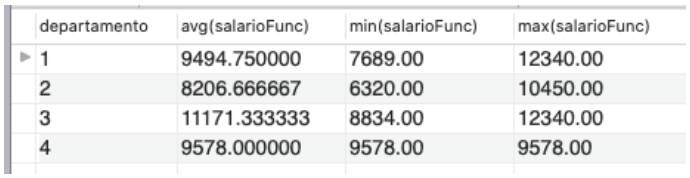
Perceba que no retorno aparece a média dos funcionários que não possuem o gênero cadastrado.

Exemplo 3: qual a média salarial por gênero?

```
   select departamento, avg(salarioFunc),
min(salarioFunc),
   max(salarioFunc) from funcionario group by departamento
```

```
order by departamento;
```

Figura 48 – Uso das funções *avg*, *min*, *max* e *group by*



departamento	avg(salarioFunc)	min(salarioFunc)	max(salarioFunc)
1	9494.750000	7689.00	12340.00
2	8206.666667	6320.00	10450.00
3	11171.333333	8834.00	12340.00
4	9578.000000	9578.00	9578.00

Fonte: Albano e Albano, 2023.

Exemplo 4: qual funcionário recebe o maior salário?

```
select nomeFunc, max(salarioFunc) from funcionario;
```

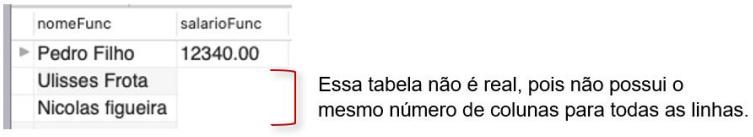
Figura 49 – Erro com o uso da função *group by*

	Time	Action	Response
1	15:00:42	select nomeFunc, max(sala...	Error Code: 1140. In aggregated query without GROUP BY, expression #1 of SELECT list contain...

Fonte: Albano e Albano, 2023.

Note que ocorreu um erro, pois não podemos aplicar uma função de agregação (*min*, *max*, *sum*, *avg* e *count*) em conjunto com outras colunas sem a declaração do *group by*. Recorde que as funções de agregação retornam apenas um único valor e, dessa forma, mesmo que houvesse vários funcionários recebendo o maior salário, a função só retornaria um valor. Além disso, nesse exemplo, a função está associada a um *select* que sempre retornará uma tabela (devendo possuir o mesmo número de linhas para todas as colunas), gerando, assim, um erro.

Figura 50 – Uso indevido da função de agregação



nomeFunc	salarioFunc
Pedro Filho	12340.00
Ulisses Frota	
Nicolas figueira	

Essa tabela não é real, pois não possui o mesmo número de colunas para todas as linhas.

Fonte: Albano e Albano, 2023.

Para responder à questão do exemplo 4, podemos usar a subconsulta:

```
select nomeFunc, salarioFunc from funcionario
   where salarioFunc = (select max(salarioFunc) from
   funcionario);
```

Figura 51 – *Subquery* com o maior salário

nomeFunc	salarioFunc
Pedro Filho	12340.00
Ulisses Frota	12340.00
Nicolas figueira	12340.00

Fonte: Albano e Albano, 2023.

4.7 *Having*()

Realiza um filtro sobre o *group by*.

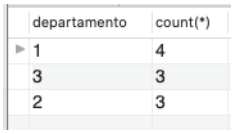
Sintaxe:

```
select (<coluna_agrupamento> e/ou <função_agrupamento>)
   from <tabela> [where <condição>]
   group by <coluna_agrupamento> having <condição>;
```

Exemplo: quantos departamentos possuem mais de dois funcionários?

```
select departamento, count(*) from funcionario
        group by departamento having count(departamento)
   > 2;
```

Figura 52 – Uso das funções *group by* e *having*



departamento	count(*)
1	4
3	3
2	3

Fonte: Albano e Albano, 2023.

TEMA 5 – INTEGRIDADE E SEGURANÇA DE DADOS

Como já comentado ao longo deste estudo, o Sistema Gerenciador de Banco de Dados (SGBD) garante que diferentes usuários possam acessar e enviar requisições ao banco de dados de forma simultânea e, o mais importante, utilizando técnicas de controle de concorrência, em que o tráfego de requisições ocorre de maneira controlada, garantindo que operações de consultas, inclusões, alterações e exclusões de dados sejam executadas sem prejuízo à segurança e à integridade dos dados armazenados pelo SGBD.

Contudo, o tema é ainda mais complexo no segmento empresarial, pois um banco de dados eficiente e de alto desempenho é um componente essencial e totalmente estratégico para a análise, definição de estimativas e tomada de decisões. De fato, o banco de dados de uma empresa é um dos ativos mais cobiçados não só pelos concorrentes, como também por criminosos. Logo, garantir a segurança dos dados da empresa, dos clientes e dos parceiros de negócios é uma tarefa que deve ser desempenhada de forma eficaz pelo Administrador de Banco de Dados (DBA).

Para que possamos compreender melhor todas essas questões listadas, precisamos, primeiramente, entender a diferença entre o que é assegurar a integridade dos dados e o que significa garantir a segurança dos dados em um SGBD.

5.1 Segurança de dados

Refere-se a estabelecer mecanismos de controle para prevenir o acesso não autorizado à base de dados. Isso significa que a proteção dos dados é seu principal objetivo, devendo abranger tanto os aspectos de segurança lógica quanto os aspectos de segurança física.

5.2 Segurança lógica

A finalidade da segurança lógica é assegurar que cada operação executada no banco de dados seja efetuada somente por usuários que possuem privilégios para tal fim, ou seja, que previamente tenham recebido permissões específicas para interagir de uma determinada forma com o banco de dados.

Para tanto, podemos fazer uso de técnicas de autenticação ou autorização (contas de usuários, senhas, permissões, *log* de monitoramento, entre outros) e mascaramento dos dados no carregamento, além de armazenar os dados criptografados na base de dados.

Quanto às permissões ou aos privilégios definidos no controle de acesso, estes podem ser atribuídos tanto para usuários específicos como para os objetos do banco de dados, ou seja, definindo permissões de uso de um objeto para todos os usuários ou outorgando uma permissão de uso geral ou específico de um objeto para o usuário.

Tais privilégios abrangem operações de conexão com o banco de dados, criação, seleção, inserção, alteração, exclusão e execução (*connect*, *create*, *select*, *insert*, *update*, *delete*, *drop* e *execute*). Para realizar esses procedimentos, fazemos uso dos comandos *grant* e *revoke*, os quais pertencem a *Data Control Language* (DCL), sendo:

- *Grant*: concede privilégios para o usuário e/ou objetos na execução de uma determinada tarefa;
- *Revoke*: remove a permissão concedida.

Sintaxe do *grant*:

```
grant privilégios on [banco_dados].tabela
   to usuário_1 [identified by 'senha_1'],
   usuário_n [identified by 'senha_n'] with grant option;
```

Sintaxe do *revoke*:

```
revoke privilégios on objeto
   from usuário_1, usuário_2, ...;
```

Exemplos por usuário:

```
grant select to nome_usuario;
revoke create table to nome_usuario;
```

Exemplos por objeto:

```
grant select on estado to nome_usuario;

revoke select on estado to nome_usuario;
```

No MySQL, os privilégios são atribuídos em quatro níveis diferentes:

- *Global* – acesso a todas as tabelas de todos os Bancos de Dados;
- *Database* – acesso a todas as tabelas de um Banco de Dados específico;
- *Table* – acesso a uma tabela específica com todas as suas colunas;
- *Column* – acesso apenas às colunas especificadas de uma determinada tabela.

Os privilégios podem ser atribuídos ou retirados, sendo:

- Trabalhar com os dados – *Insert*, *update*, *delete*, *select* e *execute*;
- Estrutura do Banco de Dados – *Create*, *alter* e *drop*.

Todo SGBD possui tabelas que são utilizadas para gerenciar as permissões. No caso do MySQL, temos as seguintes tabelas:

- *user* – armazena nomes e senhas dos usuários, assim como os privilégios globais empregados a todos os Bancos de Dados;
- *db* – armazena privilégios dos Bancos de Dados;
- *tables_priv* – armazena privilégios das tabelas;
- *columns_priv* – armazena privilégios de colunas;
- *procs_priv* – armazena privilégios de acesso às funções e *stored procedures*.

Principais comandos:

- *Create user* – cria um usuário.

Sintaxe resumida:

```
create user 'nomeUsuario'@'servidor'
    identified by 'senha' password expire;
```

Onde:

- nomeUsuario: nome de *login* do usuário.
- servidor: nome do servidor onde será criado o usuário.
- identified by: comando para identificação da senha do usuário.
- senha: senha de acesso do usuário.

- password expire: indica que no primeiro *login* será solicitado o cadastro de uma nova senha.
- *Drop user* – exclui um usuário do Banco de Dados.

Sintaxe: **`drop user [if exists] nomeusuario;`**

- *Flush privileges* – atualiza os dados de privilégios nas tabelas do MySQL.

Sintaxe: **`flush privileges;`**

- *Show grants* – visualiza os privilégios de um usuário.

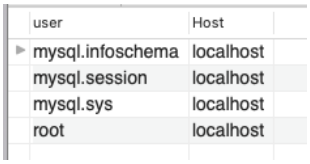
Sintaxe: **`show grants for usuario@localhost;`**

Baseado nos comandos apresentados, vamos desenvolver um exemplo abrangendo todos os passos, desde a criação do usuário, atribuição de permissões, revogação dessas permissões e, por fim, a exclusão do usuário.

1. Mostrando os usuários existentes no Banco de Dados.

   `select user, host from mysql.user;`

Figura 53 – Usuários existentes



user	Host
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Fonte: Albano e Albano, 2023.

2. Criando um usuário com nome “aluno”, senha “123” e sem a declaração da política de atualização de senha.

   `create user 'aluno'@'localhost' identified by '123';`

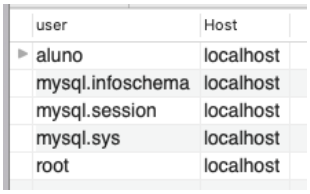
3. Atualizando os privilégios dos usuários.

   `flush privileges;`

4. Mostrando os usuários do Banco de Dados.

   `select user, host from mysql.user;`

Figura 54 – Usuários existentes após o *create user*



user	Host
aluno	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost
root	localhost

Fonte: Albano e Albano, 2023.

5. Mostrando os privilégios do usuário criado.

```
show grants for aluno@localhost;
```

Figura 55 – Privilégios do usuário ‘aluno’

Grants for aluno@localhost
GRANT USAGE ON *.* TO `aluno`@`localhost`

Fonte: Albano e Albano, 2023.

O resultado apresenta que o usuário ‘aluno’ tem o privilégio *usage*, significando que não há privilégios definidos para esse usuário. Nesse caso, o “*.*” determina esse privilégio para todas as tabelas e Bancos de Dados do SGBD.

6. Conectando o novo usuário ao SGBD.

Figura 56 – Tela principal do MySQL Workbench

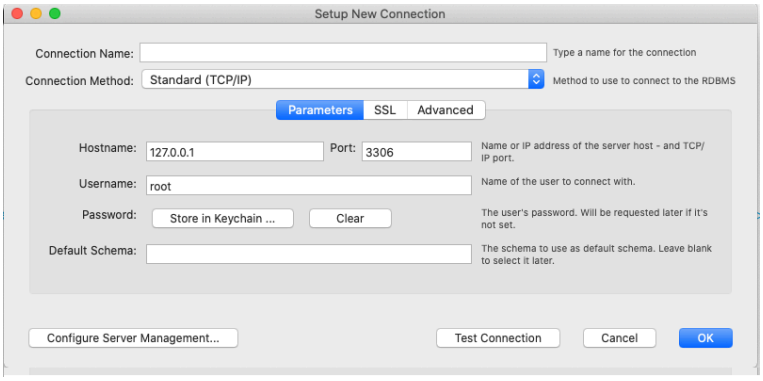
Fonte: Albano e Albano, 2023.

Perceba que na figura existe somente uma conexão do usuário ‘root’, que é o administrador do Banco de Dados. Vamos clicar em adicionar (+) para criarmos uma nova conexão com o usuário ‘aluno’. Se clicarmos com o botão direito na tela teremos o mesmo efeito (adicionar conexões).

Preenchendo os campos de registro do usuário:

- *Hostname* – nome do servidor, que, nesse caso, é *localhost* ou 127.0.0.1;
- *Username* – aluno;
- *Conector name* – nome para identificar a conexão criada (usuario_aluno).

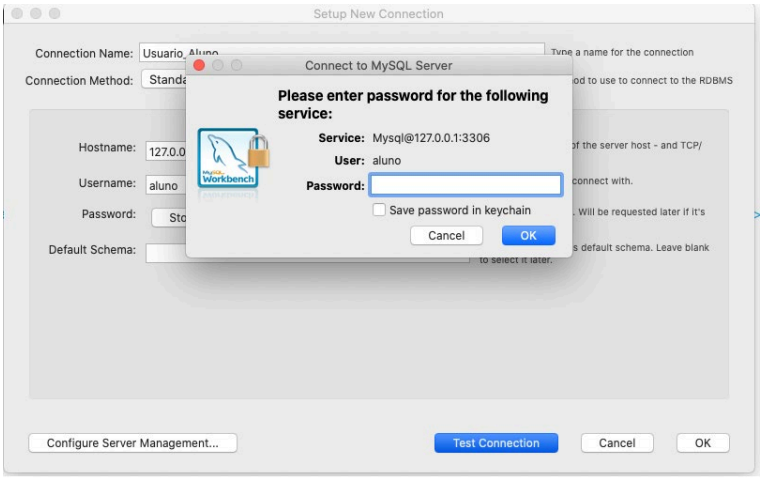
Figura 57 – Criando uma conexão



Fonte: Albano e Albano, 2023.

Clicando no botão *Test Connection* informaremos a senha ‘123’.

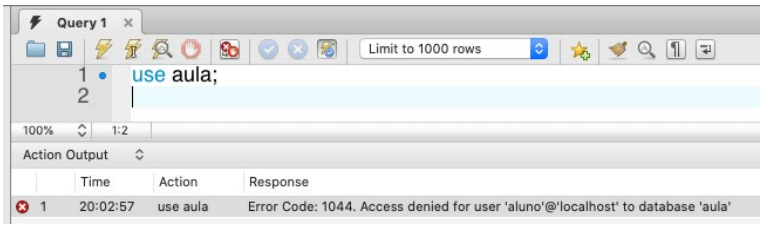
Figura 58 – Definindo a senha



Fonte: Albano e Albano, 2023.

Vale ressaltar que, após a conexão e o acesso, na sessão ‘aluno’, não aparecerá nenhum Banco de Dados para ser manipulado, uma vez que esse usuário não possui privilégio (*usage*), significado que não poderá fazer nenhuma operação. Para demonstrar essa situação, vamos tentar acessar o Banco de Dados ‘aula’.

Figura 59 – Erro de acesso ao Banco de Dados



Fonte: Albano e Albano, 2023.

A concessão de privilégios para um usuário deve ser definida pelo administrador do Banco de Dados (root) e será realizada nos próximos passos. Logo, vamos fechar a conexão ‘aluno’ e abrir a conexão do usuário ‘root’.

7. Atribuindo privilégios globais ao usuário (todas as permissões).

```
grant all privileges on *.* to aluno@localhost;

flush privileges;
```

Perceba que foi declarado o “*.*”, significando que foram concedidos os privilégios de acesso a todos os Bancos de Dados (*.) e todos os privilégios para todas as tabelas (.*). O comando *grant all privileges* permite que o usuário execute todas as tarefas que desejar.

Importante

Esse procedimento não deve ser aplicado no dia a dia, pois coloca em risco a segurança e a integridade das informações.

8. Mostrando os privilégios do usuário ‘aluno’.

```
show grants for aluno@localhost;
```

Figura 60 – Privilégios do usuário ‘aluno’

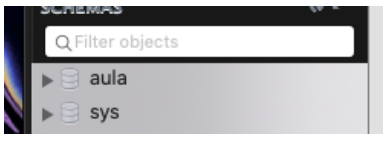
Fonte: Albano e Albano, 2023.

Depois de conceder os privilégios ao usuário ‘aluno’, podemos fechar a conexão do usuário ‘root’ e acessar novamente a conexão ‘aluno’.

9. Conectando-se ao usuário ‘aluno’.

Ao acessar a sessão do usuário ‘aluno’, verificamos que agora a sessão possui dois Bancos de Dados, o ‘sys’ (controle do SGBD) e o ‘aula’ (criado para implantar os exemplos discutidos em nosso estudo).

Figura 61 – Bancos de Dados disponíveis

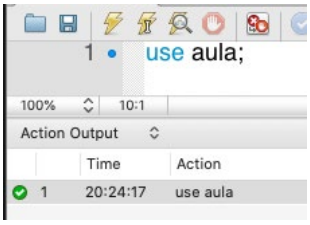


Fonte: Albano e Albano, 2023.

10. Acessando o Banco de Dados ‘aula’.

```
use aula;
```

Figura 62 – Acessando o Banco de Dados ‘aula’



Fonte: Albano e Albano, 2023.

Novamente, fecharemos a conexão ‘aluno’ para demonstrar como sucede a revogação de permissões para um usuário. Assim, acesse novamente a sessão do administrador, isto é, do usuário ‘root’.

11. Retirando todas as permissões do usuário ‘aluno’.

```
revoke all, grant option from aluno@localhost;

flush privileges;                    -- Atualizando

show grants for aluno@localhost;     -- Privilégios
```

Figura 63 – Privilégios do usuário ‘aluno’

Fonte: Albano e Albano, 2023.

12. Atribuindo privilégios específicos.

Agora vamos conceder privilégios para o Banco de Dados ‘aula’ e para o usuário ‘aluno’, permitindo o uso dos comandos *select*, *insert*, *update* e *delete*.

```
grant select, insert, update, delete

  on aula.* to aluno@localhost;

show grants for aluno;
```

13. Testando os privilégios.

Para entendermos melhor como funciona a concessão de privilégios, iremos implementar um exemplo usando o comando *update*.

Exemplo: atualizar o gênero do funcionário com matrícula igual a 7.

```
use aula;

update funcionario

  set generoFunc = 'M' where matricula = 7;

select * from funcionario where matricula = 7;
```

Figura 64 – *Select* do funcionário com matrícula 7

matricula	nomeFunc	generoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
7	Leonardo Timbo	M	2001-07-02	7850.00	2	3	leonardo.timbo@email.com	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano e Albano, 2023.

Para revogar qualquer permissão, precisamos acessar o usuário 'root'. Dessa forma, feche a conexão 'aluno' e acesse a sessão do usuário 'root', a fim de revogar o privilégio do comando *update* no Banco de Dados 'aula' ao usuário 'aluno'.

```
revoke update on aula.* from aluno@localhost;

flush privileges;

show grants for aluno@localhost;
```

Após a remoção da permissão, devemos retornar a sessão do usuário 'aluno' para testar o comando *update*.

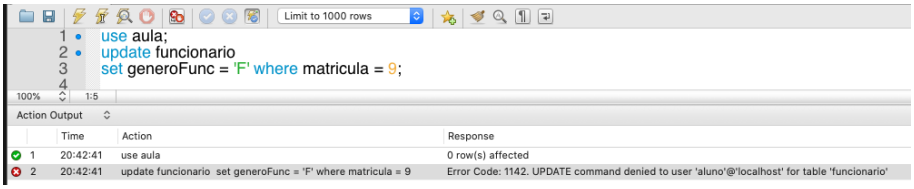
Na sessão 'aluno', faremos uma tentativa de modificação do conteúdo do gênero do funcionário com matrícula de número 9.

```
use aula;

update funcionario

set generoFunc = 'F' where matricula = 9;
```

Figura 65 – Erro no comando *update*



Fonte: Albano e Albano, 2023.

14. Excluindo o usuário ‘aluno’.

Para concluir, iremos excluir o usuário ‘aluno’, sendo que apenas o administrador (root) pode realizar essa eliminação. Logo, feche a conexão ‘aluno’ e acesse novamente com o usuário ‘root’.

```
drop user aluno@localhost;
```

Com isso, finalizamos os comandos de controle de acesso e permissões para os usuários de um Banco de Dados.

5.3 Segurança física

No que se refere à segurança física de dados, sua finalidade é proteger os dados de qualquer situação que possa provocar danos físicos, como, por exemplo, oscilações na rede elétrica, falhas de *hardware*, entre outros.

Vários procedimentos devem ser previamente definidos, como, por exemplo, a forma como ocorrerá o armazenamento físico, como as rotinas de *backup* serão realizadas, protocolo de acesso ao servidor, medidas contra o uso indevido de informações, perda acidental ou intencional, degradação e destruição dos dados, entre outros.

Todas essas ações têm como objetivo desenvolver um ambiente que forneça privacidade e proteção adequadas aos dados armazenados no Banco de Dados.

5.4 Integridade de Dados

O termo *integridade de dados* é a aplicação de regras e protocolos que visam garantir a consistência e a confiabilidade dos dados contidos no Banco de Dados. A integridade visa assegurar a qualidade e a validade dos dados armazenados, preservando e protegendo os dados de forma lógica.

Visando evitar o erro humano, quando os dados estão sendo incluídos ou manipulados, a integridade de dados possui diversos parâmetros de validação, restrições e controle de redundâncias, os quais são definidos no momento da criação de todos os elementos que compõem o Banco de Dados. Inclusive, as regras de negócio também auxiliam para que o processo de inclusão e atualização assegure a manutenção de dados confiáveis.

A integridade de dados pode ser implementada de duas formas: declarativa e procedural.

- **Integridade declarativa** – baseia-se nos comandos da *Data Definition Language* (DDL), aplicando restrições na chave primária, na chave estrangeira (integridade referencial) e no domínio, isto é:
  - Chave primária – que não contenha valores duplicados ou valor nulo;
  - Chave estrangeira ou integridade referencial – impedindo que uma chave estrangeira receba um valor que não possui correspondência (não existe) com a tabela de origem (pai) ou que uma linha da tabela de origem seja excluída sem levar em consideração as tabelas filhas (que contém a declaração de chave estrangeira), evitando, assim, registros “órfãos” sem referência com uma tabela “pai”;
  - Domínio – limitando o tipo de valor válido a ser recebido por uma coluna.

Vale ressaltar que já estudamos sobre as restrições que garantem a aplicação da integridade declarativa, sendo: *primary key*, *not null* (preenchimento obrigatório), *check* (limitação de domínio), *unique* (evita valores duplicados) e *foreign key*.

- **Integridade procedural** – fornece uma solução para as validações ou restrições que não são cobertas pela integridade declarativa. Nesse caso, a presença de um programador é fundamental para desenvolver essas funções. Aqui são utilizadas a declaração de estruturas que encapsulam as instruções em bloco, sendo:
  - *Trigger* – comandos que disparam automaticamente ações quando um outro comando SQL é executado. Tais comandos, geralmente, são disparados pelos comandos da *Data Manipulation Language* (DML) (*insert*, *update* e *delete*);

- *Stored procedures* – são programas propriamente ditos que executam tarefas e, inclusive, podem chamar funções definidas pelo usuário;
- Funções definidas pelo usuário – conjunto de instruções que retornam um valor. Tais funções podem ser executadas dentro de outras instruções SQL.

A aplicação adequada dos recursos auxilia e facilita o trabalho do Administrador de Banco de Dados (DBA) na execução de suas tarefas diárias para administrar o Banco de Dados.

FINALIZANDO

Nesta etapa, abordamos a utilização de *subqueries* através de simulações práticas. Discutimos também sobre as diversas funções disponíveis no SQL, apresentando seus usos e aplicações, demonstrando diversas situações adequadas para fazer uso desses recursos. Concluímos esta etapa tratando sobre a integridade e a segurança de dados, que são a base fundamental para manter o Banco de Dados confiável e em perfeito estado.

Novamente, recomendamos que você refaça todos os exemplos para assimilar o que estamos estudando, inclusive aplicando-os no estudo de caso. Não existe melhor forma de aprender do que aplicar de forma prática o que foi estudado. Então, revise, dedique-se e pratique.

REFERÊNCIAS

BEIGHLEY, L. **Use a Cabeça SQL**. Rio de Janeiro: Alta Books, 2010.

HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Bookman, 2009.

MYSQL DOCUMENTATION. Disponível em: <https://dev.mysql.com/doc>. Acesso em: 2 fev. 2022.

SETZER, V. W. **Banco de Dados**: conceitos, modelos, gerenciadores, projeto lógico, projeto físico. 3. ed. São Paulo: Edgard Blücher, 1986.