BANCO DE DADOS

AULA 4

Prof. Ricardo Sonaglio Albano e Profª. Silvie Guedes Albano

CONVERSA INICIAL

Manipulação de estruturas de dados

Nesta etapa, aprenderemos como realizar a manipulação dos registros armazenados no Banco de Dados, percorrendo as operações de inclusão, modificação e exclusão de registros, desvendando a sintaxe de cada comando e demonstrando de forma prática como aplicar os comandos da *Data Manipulation Language* (DML).

Também discutiremos sobre as diversas formas de desenvolvimento de requisições de consultas, incluindo o uso de restrições e construindo *queries* mais complexas, envolvendo a busca e processamento de informações contidas em diversas tabelas do Banco de Dados.

Continue a nos acompanhar em nosso estudo de caso.

Bons estudos e sucesso!

TEMA 1 – INCLUSÃO DE REGISTROS

O processo de inclusão ou armazenamento de registros em um arquivo físico do Banco de Dados deve ocorrer de forma rápida e segura sob o estrito controle de um Sistema Gerenciador de Banco de Dados (SGBD).

Para realizarmos a tarefa de inserção de dados em uma tabela, utilizamos o comando *insert* da *Structured Query Language* (SQL). Tal comando permite especificar uma linha (registro) a ser inserida ou, por meio de uma consulta, gerar um conjunto de linhas (registros) a serem inseridas na tabela.

Vale lembrar que continua valendo o mesmo princípio discutido na aplicação de outros comandos, isto é, a inserção de linhas segue a ordem predefinida na declaração da tabela, juntamente com os respectivos domínios de cada coluna.

Na etapa anterior, apresentamos uma visão introdutória sobre esse comando e, nesse material, trataremos de forma mais detalhada, lembrando que:

- Campo do tipo caractere deve ser declarado entre aspas simples;
- Declaração de valores numéricos não acompanha o uso de aspas;
- Datas são declaradas entre aspas simples e no formato AAAA-MM-DD;

- O *insert* permite uma sintaxe mais simples, quando todos os valores forem declarados na mesma sequência definida na criação das colunas da tabela. Porém, não é possível usar a sintaxe mais simples em tabelas com a declaração da cláusula *auto_increment*;
- O comando *insert* também permite a inserção de dados em colunas específicas de uma tabela, desde que sejam respeitadas as restrições de cada coluna, como por exemplo, uma chave primária com a declaração da cláusula *auto_increment* não deve estar incluída no *insert*.

Uma curiosidade sobre o comando *insert* é que você pode fazer uso do valor *null* para colunas que não possuem uma restrição de preenchimento obrigatório, independentemente do tipo de dado. Veja o exemplo:

```
insert into cidade (id, nome, uf) values(4, 'Videira', null);
```

Figura 1 – *Insert* utilizando *null*

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
4	Videira	NULL
NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Como já comentado, podemos inserir valores em colunas específicas, porém, devemos levar em consideração a presença de colunas obrigatórias.

Exemplo: Inserir valores em colunas específicas.

```
insert into cidade (id, nome) values (3, 'Parnaíba');
```

Figura 2 – *Insert* especificando as colunas

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	NULL

Fonte: Albano; Albano, [S.d.].

1.1 Trabalhando com chave estrangeira

Para incluirmos novas linhas em uma tabela que possui chave(s) estrangeira(s), é necessário que o valor exista em sua tabela de origem. Para esclarecer, de forma simples, o uso de chave estrangeira em um *insert*, vamos exemplificar utilizando o dicionário de dados da tabela Cliente.

Figura 3 – *Describe* Cliente e *select* Cidade

Dicionário de dados da tabela Cliente

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
nome	varchar(100)	YES		NULL	
genero	char(1)	YES		NULL	
dataNascimento	date	YES		NULL	
salario	decimal(10,2)	YES		NULL	
email	varchar(120)	YES	UNI	NULL	
cidadeId	int(11)	NO	MUL	NULL	

Tabela Cidade

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	NULL
4	Videira	NULL
5	São Paulo	SP
6	Belo Horizonte	MG
NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Avaliando o dicionário de dados, podemos perceber que a tabela Cliente possui uma chave estrangeira “cidadeId” que pertence a tabela Cidade (origem).

Query do *insert*:

```
insert into cliente (id, nome, genero, dataNascimento,
  salario, email, cidadeId) values (1, 'Helena', 'F',
  '2000-01-01', 12500.99, 'helena@email.com', 2);
```

Figura 4 – *Insert* com FK na tabela Cliente

id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena	F	2000-01-01	12500.99	helena@email.com	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Essa mesma *query* poderia ser convertida para uma forma mais dinâmica, utilizando o conceito de *subquery* (comandos aninhados) que estudaremos em outro momento. Aqui, apenas demonstraremos um exemplo para que você possa começar a avaliar as possibilidades do uso de *subqueries*.

```
insert into cliente values (3, 'Nair Verne', 'F',
   '1986-04-21', 6400.66, 'nair.verne@email.com',
   (select id from cidade where nome = 'São Paulo'));
```

Figura 5 – *Insert* Cliente com *subquery* Cidade

id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena	F	2000-01-01	12500.99	helena@email.com	2
3	Nair Verne	F	1986-04-21	6400.66	nair.verne@email.com	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Perceba que tornamos o *insert* mais otimizado, não existindo a necessidade de consultar a tabela Cidade, pois essa nova linha será inserida automaticamente com o 'id' correspondente a cidade de São Paulo (contido na tabela Cidade) por meio da declaração do *select* aninhado ao comando *insert*. Recordando que trataremos sobre o tema *subquery* com maior riqueza de detalhes em outro momento.

Seguindo nosso passeio pelas possibilidades do comando *insert*, vamos demonstrar com um único *insert* a inclusão de várias linhas em uma tabela. Para realizar tal operação só é necessário a inclusão de parênteses para cada linha separados por vírgula. Acompanhe o exemplo:

Figura 6 – *Describe* na tabela Funcionário

Field	Type	Null	Key	Default
matricula	int(11)	NO	PRI	NULL
nomeFunc	varchar(100)	YES		NULL
sexoFunc	char(1)	YES		NULL
NascFunc	date	YES		NULL
salarioFunc	decimal(10,2)	YES		NULL
departamento	int(11)	YES		NULL
cargo	int(11)	YES		NULL
emailFunc	varchar(120)	YES		NULL
cidadeId	int(11)	NO	MUL	NULL

Fonte: Albano; Albano, [S.d.].

Query do *insert*:

```
insert into funcionario values
   (1, 'Ana Rosa', 'F', '1996-12-31', 8500, 1, 1,
   'ana.rosa@email.com', 1),
   (2, 'Tales Heitor', 'M', '2000-10-01', 7689, 1, 2,
   'tales.heitor@email.com', 1),
   (3, 'Bia Meireles', 'F', '2002-03-14', 9450, 1, 2,
   'bia.meireles@email.com', 1),
   (4, 'Pedro Filho', 'M', '1998-05-22', 6800, 3, 3,
   'pedro.filho@email.com', 1);
```

Figura 7 – *Select* na tabela Funcionário

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	ana.rosa@email.com	1
2	Tales Heitor Souza	M	2000-10-01	7689.00	1	2	tales.heitor@email.com	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	bia.meireles@email.com	1
4	Pedro Filho	M	1998-05-22	6800.00	3	3	pedro.filho@email.com	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Fazendo novamente referência a *subquery* e aplicando esse conceito para demonstrar como tornar o *insert* inicial mais dinâmico. Para tanto, usaremos um *select* aninhado (embutido) ao *insert* para incluir as linhas na tabela. Nesse caso, você irá perceber que não existem parênteses ou a aplicação de filtros no comando *select*, apenas a declaração das colunas finalizando com a identificação da tabela origem (Funcionário).

```
insert into cliente (nome, genero, dataNascimento,
   salario, email, cidadeId)
   select nomeFunc, sexoFunc, nascFunc, salarioFunc,
   emailFunc, cidadeId from funcionario;
```

Figura 8 – *Insert* Cliente com *subquery* Funcionário

id	nome	genero	dataNascimento	salario	email	cidadeId
3	Nair Verne	F	1986-04-21	6400.66	nair.verne@email.com	5
4	Ana Rosa	F	1996-12-31	8500.00	ana.rosa@email.com	1
5	Tales Heitor	M	2000-10-01	7689.00	tales.heitor@email.com	1
6	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	1
7	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Por meio do *select* serão lidas todas as linhas da tabela Funcionário, incluindo automaticamente cada linha lida na tabela Cliente. Vale salientar que as colunas da tabela Funcionário, obrigatoriamente, devem estar em concordância com o tipo de dado existente em cada coluna da tabela Cliente.

1.2 Ocorrências de erros na inclusão de linhas em uma tabela

É importante ressaltar que ao incluirmos uma nova linha em uma tabela, precisamos levar em consideração todos os fatores já discutidos. Porém, erros na declaração de um *insert* podem ocorrer facilmente. Por isso, simularemos algumas situações para que você possa reconhecer e solucionar erros comuns.

Para exemplificar as simulações de erros, utilizaremos o dicionário de dados da tabela Cidade.

Figura 9 – *Describe* da tabela Cidade

Field	Type	Null	Key	Default
id	int(11)	NO	PRI	NULL
nome	varchar(100)	NO		NULL
uf	char(2)	YES		NULL

Fonte: Albano; Albano, [S.d.].

Simulação 1: tentativa de inclusão de uma linha sem declarar uma coluna que possui a restrição *not null*.

```
insert into cidade (id, uf) values (5, 'MG');
```

Figura 10 – Erro no *insert* e restrição *not null*

	Action	Response
1	insert into cidade (id, uf) values (5, 'MG')	Error Code: 1364. Field 'nome' doesn't have a default value

Fonte: Albano; Albano, [S.d.].

Perceba que no comando *insert* não foi declarado o valor da coluna “nome”, mas se você consultar o dicionário de dados notará que essa coluna necessita de um valor, pois possui a restrição *not null*. Além disso, o SGBD não pode fazer uso do recurso de valor *default* (padrão), já que ele também não foi previamente definido.

Simulação 2: tentativa de inclusão de linha sem a declaração do valor da chave primária.

```
insert into cidade (nome, uf) values ('Imperatriz', 'MA');
```

Figura 11 – Erro no *insert* sem chave primária

	Action	Response
1	insert into cidade (nome, uf) values ('Imperatriz', 'MA')	Error Code: 1364. Field 'id' doesn't have a default value

Fonte: Albano; Albano, [S.d.].

Apresenta o mesmo erro da simulação 1. Isso ocorre porque em chaves primárias a restrição *not null* é padrão (automática).

Simulação 3: tentativa de inclusão do mesmo valor da chave primária de forma repetida.

```
insert into cidade values (5, 'São Paulo', 'SP');
insert into cidade values (5, 'Belo Horizonte', 'MG');
```

Figura 12 – Erro no *insert* de PK duplicada

	Action	Response
1	insert into cidade values (5, 'São Paulo', 'SP')	Error Code: 1062. Duplicate entry '5' for key 'PRIMARY'

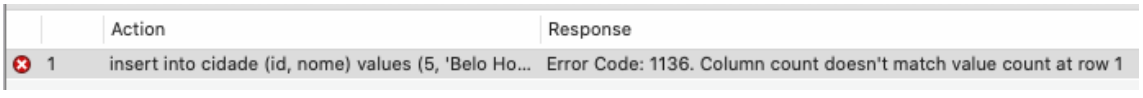
Fonte: Albano; Albano, [S.d.].

O SGBD executa com sucesso o primeiro *insert*, porém, falha na execução do segundo *insert*, isso porque o valor informado para a chave primária já existe, ocorrendo uma duplicação de chave primária.

Simulação 4: tentativa de inclusão de valor em coluna não especificada (sem correspondência).

```
insert into cidade (id, nome)
   values (5, 'Belo Horizonte', 'MG');
```

Figura 13 – Erro no *insert*, valor sem correspondência



	Action	Response
1	insert into cidade (id, nome) values (5, 'Belo Ho...	Error Code: 1136. Column count doesn't match value count at row 1

Fonte: Albano; Albano, [S.d.].

Nesse caso, foram declaradas duas colunas, mas atribuídos três valores, ficando o último sem correspondência com uma coluna. Assim, o SGBD falha na execução da requisição, informando que a contagem de colunas não corresponde ao total de valores informados.

Simulação 5: tentativa de inclusão de valor inexistente em chave estrangeira, baseado nas tabelas Cliente e Cidade.

Figura 14 – *Describe* em Cliente e *select* em Cidade

Dicionário de dados da tabela Cliente

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
nome	varchar(100)	YES		NULL	
genero	char(1)	YES		NULL	
dataNascimento	date	YES		NULL	
salario	decimal(10,2)	YES		NULL	
email	varchar(120)	YES	UNI	NULL	
cidadeId	int(11)	NO	MUL	NULL	

Tabela Cidade

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	NULL
4	Videira	NULL
5	São Paulo	SP
6	Belo Horizonte	MG
NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

```
insert into cliente (id, nome, genero, dataNascimento,
   email, cidadeId) values (2, 'Nicolas', 'M',
   '2002-12-10', 'nicolas@email.com', 10);
```

Figura 15 – Erro no *insert* com FK na tabela Cliente

	Action	Response
1	insert into cliente (id, nome, genero, dataNasci...	Error Code: 1452. Cannot add or update a child row: a foreign key constr...

Fonte: Albano; Albano, [S.d.].

O SGBD, ao tentar realizar a inclusão na tabela Cliente, verifica que a chave estrangeira (FK) informada não possui correspondência na tabela Cidade, gerando uma mensagem de erro, a qual informa que não é possível adicionar ou alterar uma linha “filha”, pois o valor da chave estrangeira não existe na tabela Cidade (tabela “pai” ou origem).

Agora que você aprendeu sobre a inclusão de linhas em uma tabela, faça a inclusão de algumas linhas nas tabelas do nosso estudo de caso. Dessa forma, você colocará em prática o comando aprendido.

Saiba mais

Nota: em cada seção dedicada a um novo comando, as tabelas que estão sendo utilizadas como base para a execução dos exemplos serão sempre atualizadas com os mesmos dados originais, com o intuito de evitar limitações nos exemplos.

TEMA 2 – EXCLUSÃO E MODIFICAÇÃO DE REGISTROS

Nesta seção, trataremos das operações de exclusão e alteração das linhas (registros) armazenadas em cada tabela do Banco de Dados.

2.1 Removendo linhas da tabela

O comando SQL utilizado para a exclusão de linhas de uma tabela chama-se ***delete***. Por motivos óbvios, esse é um dos comandos que exige muita atenção em sua aplicação. Ninguém deseja eliminar todas as linhas de uma tabela por descuido e precisar correr atrás do backup de dados. Então, fique esperto!

Sintaxe:	`delete from ntabela where condição;`
Onde:	
delete	Comando de exclusão.
from	Cláusula do *delete*, sendo o elo de ligação (“de”).
ntabela	Nome da tabela que sofrerá a ação de *delete*.
where	Filtro indicando o início da condição. Pode não ser declarado, nas esse procedimento não é aconselhável.
condição	Condição a ser satisfeita para a realização da exclusão.
;	Ponto e vírgula, determina o final do comando.

A melhor prática no uso do comando *delete* é a inclusão da cláusula *where*, ou seja, aplicando um filtro para limitarmos o impacto sofrido pelos registros dentro de uma base de dados.

Para exemplificar, simularemos algumas situações utilizando as tabelas Cidade, Cliente e Funcionário.

Figura 16 – *Select* das tabelas Cidade, Cliente e Funcionário

Tabela Cidade

id	nome	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	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	tales.heitor@email.com	1
5	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela Funcionario

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	ana.rosa@email.com	1
2	Tales Heitor Souza	M	2000-10-01	7689.00	1	2	tales.heitor@email.com	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	10450.00	2	3	camila.fialho@email.com	6
6	Ulisses Frota	M	1997-06-30	9800.00	1	4	ulisses.frota@email.com	3
7	Leonardo Timbo	M	2001-07-02	7850.00	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	6320.00	2	2	sophia.arcanjo@email.com	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Simulação 1: remoção do cliente com ‘id’ número 5.

```
delete from cliente where id = 5;
```

Figura 17 – *Select* resultante após a exclusão da linha

id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	tales.heitor@email.com	1
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

É importante ressaltar que, mesmo com o uso da cláusula *Where*, devemos tomar alguns cuidados, como, por exemplo, avaliar se existe mais de uma linha que satisfaça a condição declarada e se todas essas linhas devem ser realmente excluídas.

Simulação 2: exclusão da cliente “Helena Magalhaes”.

```
delete from cliente where nome = 'Helena Magalhaes';
```

Figura 18 – *Select* resultante após a exclusão de dados coincidentes

id	nome	genero	dataNascimento	salario	email	cidadeId
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	tales.heitor@email.com	1
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Perceba que existe um pequeno detalhe ao executar o comando *delete*. Na tabela havia duas “Helena Magalhaes” e, dessa forma, ao aplicar o filtro as duas linhas satisfizeram a condição declarada e foram eliminadas.

A aplicação de *subqueries*, em conjunto com o comando *delete*, também pode ser utilizada com o intuito de otimizar o processo.

Situação 3: remover todos os clientes que residem na cidade de Curitiba.

```
delete from cliente where cidadeId in
   (select id from cidade where nome = 'Curitiba');
```

Figura 19 – *Select* após a exclusão com *subquery*

id	nome	genero	dataNascimento	salario	email	cidadeId
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

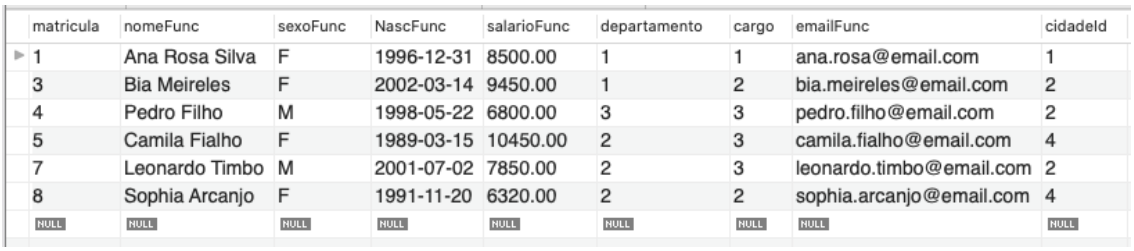
Fonte: Albano; Albano, [S.d.].

Fazendo o uso dos operadores lógicos, em conjunto com a cláusula *where*, podemos ampliar o uso de filtros com a adição de mais uma condição.

Situação 4: eliminar todos os funcionários do gênero masculino que pertencem ao departamento 1.

```
delete from funcionario
   where departamento = 1 and sexoFunc = 'M';
```

Figura 20 – *Select* após a exclusão usando operadores lógicos



matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	ana.rosa@email.com	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	10450.00	2	3	camila.fialho@email.com	4
7	Leonardo Timbo	M	2001-07-02	7850.00	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	6320.00	2	2	sophia.arcanjo@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

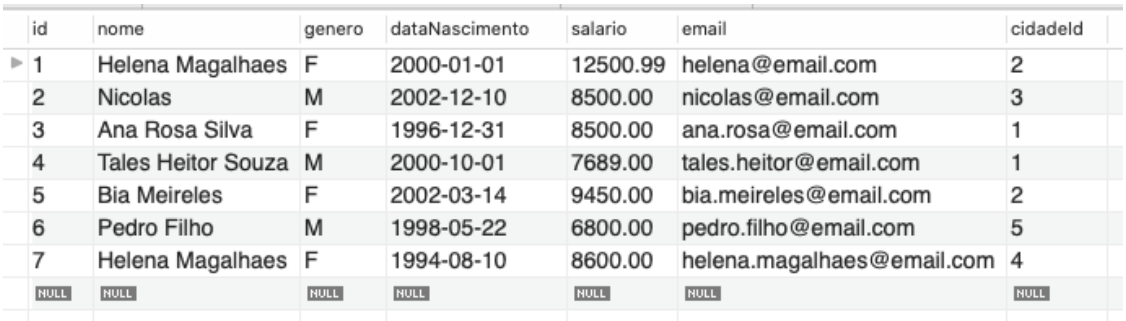
Fonte: Albano; Albano, [S.d.].

Na utilização do comando *delete* é importante ficarmos atentos, pois com uma única linha de comando podemos remover todas as linhas armazenadas em uma tabela. Uma alternativa é utilizar o comando *truncate*, que exige menos recursos do sistema e, em comparação ao *delete*, é considerado mais rápido.

Contudo, apesar de ser mais rápido, o *truncate* possui diversas limitações em comparação ao *delete*, como por exemplo, não permite o uso de *triggers*, nem de *cascade* e, também, não dá suporte a replicações no Banco de Dados.

Para exemplificar o uso dos dois comandos, utilizaremos a tabela Cliente contendo os dados originais.

Figura 21 – *Select* da tabela Cliente



id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	tales.heitor@email.com	1
5	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

As duas declarações têm a mesma função, diferenciando apenas no tempo de execução.

```
delete from cliente;
```

ou

```
truncate table cliente;
```

Figura 22 – *Select* resultante após a exclusão de todas as linhas da tabela Cliente

id	nome	genero	dataNascimento	salario	email	cidadeId
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Geralmente, a definição do comportamento do comando *delete* está presente no momento de criação das tabelas que possuem chaves estrangeiras, com a inclusão da declaração "*on delete no action*", referenciando cada coluna que corresponde a chave estrangeira na tabela. Esse é um procedimento padrão considerado necessário para assegurar a integridade dos dados relacionados.

A ação "*on delete*" é definida por padrão, assim como "*no action*", não permitindo a exclusão de linhas na tabela pai que estejam associadas a outra tabela (filha).

Para tornar mais claro o funcionamento dessa declaração, exemplificaremos criando três tabelas que possuem relações entre si.

```
create table cidade (  -- Tabela pai
   id int,
   nome varchar(100) not null
   uf char(02),
   constraint pkCidade primary key (id));

create table cliente (  -- Tabela filha
   id int auto_increment,
   -- Demais colunas foram suprimidas
   cidadeId int not null,
   constraint pkcliente primary key (id),
   constraint fkClienteCidade
   foreign key (cidadeId)references cidade(id)
        on delete no action on update no action);

create table funcionario (  -- Tabela filha
   matricula int not null,
   -- Demais colunas foram suprimidas
```

```
    cidadeId int not null,
    constraint pkfuncionario primary key (matricula),
    constraint fkFuncCidade
    foreign key (cidadeId) references cidade(id)
        on delete cascade on update no action);
```

Perceba que na tabela Cliente existe a declaração padrão da cláusula "*on delete no action*". Essa declaração evita a execução de qualquer tentativa de exclusão de uma cidade que possua um cliente cadastrado, ou seja, uma linha registrada na tabela Cliente que referencie a cidade armazenada na tabela Cidade.

Vamos ao exemplo:

Figura 23 – *Select* das tabelas Cidade, Cliente e Funcionário

Tabela Cidade

id	nome	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	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas	M	2002-12-10	8500.00	nicolas@email.com	3
3	Ana Rosa Silva	F	1996-12-31	8500.00	ana.rosa@email.com	1
4	Tales Heitor Souza	M	2000-10-01	7689.00	tales.heitor@email.com	1
5	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela Funcionario

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	ana.rosa@email.com	1
2	Tales Heitor Souza	M	2000-10-01	7689.00	1	2	tales.heitor@email.com	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	10450.00	2	3	camila.fialho@email.com	6
6	Ulisses Frota	M	1997-06-30	9800.00	1	4	ulisses.frota@email.com	3
7	Leonardo Timbo	M	2001-07-02	7850.00	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	6320.00	2	2	sophia.arcanjo@email.com	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Query:

```
delete from cidade where id = 1;
```

Figura 24 – Erro de integridade referencial

	Action	Response
1	delete from ci...	Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails

Fonte: Albano; Albano, [S.d.].

Nesse caso, ocorreu um erro porque não é possível eliminar a cidade com ‘id’ igual à 1 (que corresponde a Curitiba), pois sua chave primária está vinculada à chave estrangeira declarada nas tabelas filhas (Cliente e Funcionário), ou seja, existem linhas registradas nas duas tabelas que referenciam o código dessa cidade. Isso é o que chamamos de *restrição de chave estrangeira*, que garante a integridade referencial do Banco de Dados.

Em alguns casos, a exclusão desse tipo de situação se faz necessária, dessa forma, podemos fazer uso da declaração “*on delete cascade*”, que é um recurso que permite a exclusão de linhas em tabelas filhas e pai, ou seja, excluindo da tabela filha para a tabela pai (origem). Essa declaração deve constar no *script* de criação da tabela, que também possui uma chave estrangeira.

Vamos simular seu funcionamento utilizando a tabela Funcionário. Verifique que a tabela possui oito linhas armazenadas e todas fazem referência a tabela Cidade (pai). Vamos realizar uma tentativa de exclusão da cidade com o ‘id’ igual à 6 na tabela Cidade.

Como na tabela Funcionário foi declarado o *delete* com a opção *cascade*, o SGBD identifica todas as tabelas que possuem relação com a tabela Cidade e que possuam linhas vinculadas a cidade com ‘id’ igual a 6, executando a exclusão.

Query:

```
delete from cidade where id = 6;
```

Figura 25 – *Select* nas tabelas Cidade e Funcionário

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	PI
4	Videira	SC
5	Imperatriz	MA
7	São Paulo	SP
NULL	NULL	NULL

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	8500.00	1	1	ana.rosa@email.com	1
2	Tales Heitor Souza	M	2000-10-01	7689.00	1	2	tales.heitor@email.com	1
3	Bia Meireles	F	2002-03-14	9450.00	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	pedro.filho@email.com	2
6	Ulisses Frota	M	1997-06-30	9800.00	1	4	ulisses.frota@email.com	3
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; Albano, [S.d.].

Analisando a tabela Funcionário, verifica-se que ela, originalmente, possuía apenas duas linhas vinculadas a cidade com ‘id’ ao invés de igual a 6, as quais foram eliminadas com sucesso. Porém, é importante ressaltar que apesar de apenas uma das tabelas filhas (Cliente) possuir a declaração “*on delete no action*”, não ocorreu nenhum erro na eliminação das linhas contidas

nas tabelas Cidade e Funcionário. Isso aconteceu porque não existe nenhuma linha na tabela Cliente que seja relacionada ao 'id' igual a 6 da tabela Cidade.

2.2 Alterando as linhas da tabela

No SQL, o comando *update* é o responsável por manter a base de dados atualizada, realizando alterações no conteúdo (existente) de uma ou mais colunas ou mesmo em múltiplas linhas armazenadas em uma tabela. Esse comando é sempre acompanhado pelo argumento *set* e, geralmente, pela cláusula *where*.

É importante ressaltarmos o uso recomendável da cláusula *where*, a qual aplica filtros em colunas e linhas, limitando a alteração a condições preestabelecidas. Apesar de ser de caráter opcional (pode ser omitida na *query*), a opção de não fazer uso desse recurso afetará todas as linhas da tabela, ou seja, todos os dados das colunas especificadas serão alterados.

O *update* também faz parte dos comandos que devem ser utilizados com cautela. Imagine acabarmos atualizando uma tabela com dados totalmente equivocados? E, pior, não percebermos a falha até começarmos a receber reclamações?

Sintaxe:

```
update ntabela
  set coluna1 = valor1, ..., colunan = valorn
  where ncoluna = valorfiltro;
```

Onde:

update	Comando de alteração.
ntabela	Nome da tabela que irá sofrer a alteração.
set	Especifica qual(is) a(s) coluna(s) que receberá(ão) a atualização.
coluna1	Nome da coluna que irá sofrer a atualização.
valor1	Novo conteúdo.
where	Define um filtro que limitará os dados que serão atualizados.
ncoluna	Coluna que armazena o valor a ser filtrado.
valorfiltro	Valor a ser pesquisado para realizar a alteração.
;	Ponto e vírgula, determina o final do comando.

Para exemplificar o uso do *update*, usaremos os mesmos *scripts* (criação das tabelas Cidade, Cliente e Funcionário) utilizados no comando *delete*. Porém,

aqui faremos uma alteração na tabela Funcionário na declaração de chave estrangeira da cidade. A cláusula será substituída por "*on update cascade*".

```
create table funcionario (
  matricula int not null,
  nomeFunc varchar(100),
  sexoFunc char(01),
  NascFunc date,
  salarioFunc decimal(10,2),
  departamento int,
  cargo int,
  emailFunc varchar(120),
  cidadeId int not null,
  constraint pkfuncionario primary key (matricula),
  constraint fkFuncCidade foreign key (cidadeId)
    references cidade(id)
    on delete cascade on update cascade);
```

Simulação 1: a empresa decidiu reajustar em 15% os salários de todos seus funcionários.

```
update funcionario set salarioFunc = salarioFunc * 1.15;
```

Perceba que não estamos fazendo uso da cláusula *where*. Dessa forma, a *query* será aplicada uma vez em cada linha da tabela Funcionário.

Figura 26 – *Select* na tabela Funcionário com reajuste de salário

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva	F	1996-12-31	9775.00	1	1	ana.rosa@email.com	1
2	Tales Heitor Souza	M	2000-10-01	8842.35	1	2	tales.heitor@email.com	1
3	Bia Meireles	F	2002-03-14	10867.50	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	7820.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	12017.50	2	3	camila.fialho@email.com	4
6	Ulisses Frota	M	1997-06-30	11270.00	1	4	ulisses.frota@email.com	3
7	Leonardo Timbo	M	2001-07-02	9027.50	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	7268.00	2	2	sophia.arcanjo@email.com	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Simulação 2: a funcionária Ana Rosa Silva (matrícula 1) casou-se e optou por usar o nome do cônjuge, alterando, inclusive, seu contato de e-mail. Dessa forma, é necessário atualizar seu cadastro na empresa.

```
update funcionario set nomeFunc = 'Ana Rosa Silva Vieira',
    emailFunc = 'anavieira@email.com' where matricula = 1;
```

Figura 27 – *Select* na tabela Funcionário com os dados da funcionária atualizados

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva Vieira	F	1996-12-31	9775.00	1	1	anavieira@email.com	1
2	Tales Heitor Souza	M	2000-10-01	8842.35	1	2	tales.heitor@email.com	1

Fonte: Albano; Albano, [S.d.].

Em algumas situações aplicar apenas um filtro não será suficiente para atingir todas as linhas que desejamos atualizar, sendo necessário aplicar mais de uma condição (filtro) no *where*. Para esses casos, geralmente, fazemos uso dos operadores lógicos (serão apresentados no tema restrições de consultas).

Acompanhe o exemplo: atualizar o salário em 5% de todos os funcionários do gênero masculino pertencentes ao departamento 1.

```
update funcionario set salarioFunc = salarioFunc * 1.05
    where sexoFunc = 'M' and departamento = 1;
```

Figura 28 – *Select* na tabela Funcionário após o uso de operadores lógicos

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
2	Tales Heitor Souza	M	2000-10-01	9284.47	1	2	tales.heitor@email.com	1
6	Ulisses Frota	M	1997-06-30	11833.50	1	4	ulisses.frota@email.com	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Também é possível mesclar ao *update* o comando *select*, a fim de facilitar a atualização e tornar a operação mais otimizada, adicionando, inclusive, funções.

Simulação 3: um funcionário da empresa solicitou transferência para a cidade de Imperatriz, no Maranhão. Dessa forma, é necessário registrar essa atualização.

```
update funcionario set cidadeId =
    (select id from cidade where nome = 'Imperatriz')
    where matricula = 2;
```

Analisando a *query* é possível perceber que, primeiramente, será verificado qual o ‘id’ correspondente a cidade de Imperatriz. Obtido o ‘id’, o

mesmo será atualizado (alterado) na coluna “cidadeId” do funcionário que possui a matrícula igual à 2.

Figura 29 – Resultado do *update* após a transferência do funcionário

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva Vieira	F	1996-12-31	9775.00	1	1	anavieira@email.com	1
2	Tales Heitor Souza	M	2000-10-01	9284.47	1	2	tales.heitor@email.com	5
3	Bia Meireles	F	2002-03-14	10867.50	1	2	bia.meireles@email.com	2

Fonte: Albano; Albano, [S.d.].

Da mesma forma como ocorre no comando *delete*, por padrão, o *update* também possui a limitação de chave estrangeira “*on update no action*”, ou seja, não é possível alterar o valor de uma chave estrangeira que está sendo utilizada em outra tabela (integridade referencial). Nessa situação, o comando *update* também permite o uso do *cascade*.

Acompanhe o exemplo: Vamos alterar o ‘id’ igual à 7 da tabela Cidade para o ‘id’ igual a 11. Perceba que os funcionários que tinham o ‘id’ igual a 7 na coluna “cidadeId” passou a ser 11.

Figura 30 – *Select* nas tabelas Cidade e Funcionário para teste do *update cascade*

id	nome	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

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva Vieira	F	1996-12-31	9775.00	1	1	anavieira@email.com	1
2	Tales Heitor Souza	M	2000-10-01	9284.47	1	2	tales.heitor@email.com	5
3	Bia Meireles	F	2002-03-14	10867.50	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	7820.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	12017.50	2	3	camila.fialho@email.com	4
6	Ulisses Frota	M	1997-06-30	11833.50	1	4	ulisses.frota@email.com	7
7	Leonardo Timbo	M	2001-07-02	9027.50	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	7268.00	2	2	sophia.arcanjo@email.com	7

Fonte: Albano; Albano, [S.d.].

Recorde que a tabela Cliente possui a declaração “*on update no action*”. Porém, na tabela Funcionário foi declarado “*on update cascade*” (consulte os *scripts*).

Query:

```
update cidade set id = 11 where id = 7;
```

Figura 31 – *Select* nas tabelas Cliente e Funcionário com execução *update cascade*

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

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	emailFunc	cidadeId
1	Ana Rosa Silva Vieira	F	1996-12-31	9775.00	1	1	anavieira@email.com	1
2	Tales Heitor Souza	M	2000-10-01	9284.47	1	2	tales.heitor@email.com	5
3	Bia Meireles	F	2002-03-14	10867.50	1	2	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	7820.00	3	3	pedro.filho@email.com	2
5	Camila Fialho	F	1989-03-15	12017.50	2	3	camila.fialho@email.com	4
6	Ulisses Frota	M	1997-06-30	11833.50	1	4	ulisses.frota@email.com	11
7	Leonardo Timbo	M	2001-07-02	9027.50	2	3	leonardo.timbo@email.com	2
8	Sophia Arcanjo	F	1991-11-20	7268.00	2	2	sophia.arcanjo@email.com	11
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

A *query* foi executada com sucesso, pois na tabela Cliente (que seria o único impedimento para a execução devido ao uso do "*on update no action*") não existe correspondência que impeça essa modificação.

Sugerimos que você aplique os conhecimentos adquiridos sobre os comandos de atualização e exclusão de linhas no estudo de caso. Lembre-se, a prática é fundamental para a fixação dos conhecimentos.

TEMA 3 – RESTRIÇÃO DE CONSULTAS

Para ampliar o potencial dos filtros utilizados em consultas SQL, podemos fazer uso dos operadores lógicos, relacionais (de comparação) e aritméticos.

3.1 Operadores lógicos

Operador	Explicação
or	Teste lógico entre dois operandos. Retorna verdadeiro (*true*) caso um dos operandos seja verdadeiro.
and	Teste lógico entre dois operandos. Retorna verdadeiro (*true*) se todos os operandos são verdadeiros.
not	Testa se o operando é verdadeiro (*true*) e retorna falso (*false*).

3.2 Operadores relacionais

Ao fazer uso dos operadores relacionais devemos levar em consideração que sempre retornam verdadeiro.

Operador	Explicação
= (igual a)	Operandos são iguais.
<> ou != (diferente)	Operandos são diferentes.
> (maior que)	Operando da esquerda é maior.
< (menor que)	Operando da esquerda é menor.
>= (maior ou igual a)	Operando da esquerda é maior ou igual.

<= (menor ou igual a)	Operando da esquerda é menor ou igual.
in (pertence a)	Operando da esquerda pertence ao da direita.
not in (não pertence)	Operando não pertencer ao operador da direita.
like (compatível)	Operando da esquerda for compatível com o da direita.
between / and (intervalo)	Operando da esquerda estiver entre o segundo e o terceiro operando.
is null (é nulo)	Operando for nulo.
exists (existe)	Subconsulta retornando no mínimo uma linha.
any (qualquer)	Comparação com qualquer elemento do conjunto.
all (todos)	Comparação com todos os elementos do conjunto.

3.3 Operadores aritméticos

Em qualquer uma das operações aritméticas, sempre que um dos operandos possui o valor *null*, o resultado será *null*.

Operador	Explicação
+ (soma)	Retorna a soma dos operandos.
– (subtração)	Retorna a subtração do operando da esquerda pelo operando da direita.
* (multiplicação)	Esse operador retorna à multiplicação dos dois operandos.
/ (divisão)	Retorna a divisão do operando da esquerda pelo da direita.

3.4 Usando os operadores na prática

Preparamos alguns exemplos para que você possa compreender melhor a utilização dos operadores. Contudo, informamos que os operadores *exists*, *any* e *all* serão demonstrados no tema sobre subconsulta (*subquery*).

Para realizarmos as simulações, utilizaremos as tabelas Cidade e Cliente.

Figura 32 – *Select* nas tabelas Cidade e Cliente

Tabela Cidade

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
3	Parnaíba	NULL
4	Videira	SC
5	Imperatriz	MA
6	Belo Horizonte	NULL
7	São Paulo	SP
NULL	NULL	NULL

Tabela Cliente

id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas Silva	M	2002-12-10	8500.00	nicolas.silva@email.com	3
3	Silva Junior	F	1996-12-31	8500.00	silva.junor@email.com	1
4	Tales Silva Souza	M	2000-10-01	7689.00	tales.souza@email.com	1
5	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
8	Sophia Arcanjo	F	1991-11-20	6320.00	sophia.arcanjo@email.com	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Simulação 1: projetar o nome, o salário e o gênero de todos os clientes que possuem uma renda entre R$ 5.000,00 e R$ 8.000,00.

```
select nome, genero, salario from cliente
   where salario >= 5000 and salario <= 8000;
```

Figura 33 – *Select* após os operadores >=, *and* e <=

nome	genero	salario
Tales Heitor Souza	M	7689.00
Pedro Filho	M	6800.00
Sophia Arcanjo	F	6320.00

Fonte: Albano; Albano, [S.d.].

Simulação 2: selecionar todas as cidades cuja unidade federativa seja diferente do valor nulo.

```
select * from cidade where uf is not null;
```

Figura 34 – *Select* após a combinação dos operadores *not* e *null*

id	nome	uf
1	Curitiba	PR
2	Bagé	RS
4	Videira	SC
5	Imperatriz	MA
7	São Paulo	SP
NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Simulação 3: selecionar os clientes que possuem o termo Silva no nome.

```
select * from cliente where nome like '%Silva%';
```

O *like* possui alguns detalhes que devem ser esclarecidos. O operador % tem como função substituir uma cadeia de caracteres, logo, se colocarmos o % somente no início (‘%Silva’), indicaremos que queremos qualquer nome que seja finalizado com Silva, porém, se adicionarmos o % (‘Silva%’) apenas no final, indicaremos que estamos procurando o nome que contenha “Silva” no início. No caso da simulação 3, perceba que o % foi inserido tanto no início quanto no final, indicando que a pesquisa deve procurar o nome Silva em qualquer parte da cadeia de caracteres da coluna nome (tabela Cliente).

Figura 35 – *Select* após o operador *like*

id	nome	genero	dataNascimento	salario	email	cidadeId
2	Nicolas Silva	M	2002-12-10	8500.00	nicolas.silva@email.com	3
3	Silva Junior	M	1996-12-31	8500.00	silva.junor@email.com	1
4	Tales Silva Souza	M	2000-10-01	7689.00	tales.souza@email.com	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

Simulação 4: projetar o nome, o salário, o gênero e o código (‘id’) da cidade cujos clientes residam nas cidades 1, 2 ou 4.

```
select nome, salario, genero, cidadeId
   from cliente where cidadeId in (1, 2, 4);
```

ou

```
select nome, salario, genero, cidadeId from cliente
   where cidadeId = 1 or cidadeid = 2 or cidadeid = 4;
```

Perceba que existem duas formas de realizarmos a mesma pesquisa com resultado idêntico, utilizando o operador *in* (pertence) ou o operador *or*.

Vale ressaltar que o operador *in* torna a instrução mais simples e enxuta.

Figura 36 – *Select* após os operadores *in* ou *or*

nome	salario	genero	cidadeId
Silva Junior	8500.00	M	1
Tales Silva Souza	7689.00	M	1
Helena Magalhaes	12500.99	F	2
Bia Meireles	9450.00	F	2
Helena Magalhaes	8600.00	F	4

Fonte: Albano; Albano, [S.d.].

Simulação 5: projetar o nome, o salário, o gênero e o código (‘id’) da cidade cujos clientes residam nas cidades de 1 a 4 (intervalo).

```
select nome, salario, genero, cidadeId
   from cliente where cidadeId between 1 and 4;
```

ou

```
select nome, salario, genero, cidadeId
   from cliente where cidadeId >= 1 and cidadeid <= 4;
```

Assim como no exemplo anterior, existem duas formas de realizarmos a mesma consulta, usando o operador *between* (intervalo) ou o operador *and*.

Figura 37 – *Select* após os operadores *between* ou *and*

nome	salario	genero	cidadeId
Silva Junior	8500.00	M	1
Tales Silva Souza	7689.00	M	1
Helena Magalhaes	12500.99	F	2
Bia Meireles	9450.00	F	2
Nicolas Silva	8500.00	M	3
Helena Magalhaes	8600.00	F	4

Fonte: Albano; Albano, [S.d.].

3.5 Order By

A cláusula *order by* tem por objetivo ordenar o resultado de um *select*. Tal ordenação poderá ser apresentada de forma ascendente (*asc*) ou descendente (*desc*), sendo que a declaração *asc* é padrão, podendo ser omitida. Além disso, o *order by* será sempre a última cláusula a ser declarada.

Simulação 1: projetar o nome e o salário dos clientes do gênero feminino por ordem crescente de nome.

```
select nome, salario from cliente
   where genero = 'F' order by nome;
```

Figura 38 – *Select* após o *order by* padrão

nome	salario
Bia Meireles	9450.00
Helena Magalhaes	12500.99
Helena Magalhaes	8600.00
Sophia Arcanjo	6320.00

Fonte: Albano; Albano, [S.d.].

Também é possível realizar a ordenação por mais de uma coluna ao mesmo tempo, simplesmente separando as colunas por vírgula e definindo a ordem desejada (*asc* ou *desc*).

Simulação 2: projetar o nome e o salário dos clientes do gênero feminino, ordenando o nome de forma decrescente e o salário de forma crescente.

```
select nome, salario from cliente
   where genero = 'F' order by nome desc, salario asc;
```

Figura 39 – *Order by* usando duas colunas

nome	salario
Sophia Arcanjo	6320.00
Helena Magalhaes	8600.00
Helena Magalhaes	12500.99
Bia Meireles	9450.00

Fonte: Albano; Albano, [S.d.].

Avaliando o resultado da consulta, percebemos que está ordenado pelo nome em ordem decrescente, mas nas duas ocorrências de homônimos (nomes iguais), o salário foi organizado em ordem crescente.

O *order by* ainda permite que possamos ordenar a consulta sem declararmos o nome da coluna, usando apenas a sua posição no *select*.

```
select nome, salario, email from cliente order by 3 asc;
```

Na *query* foi indicado a ordenação baseada na terceira coluna (“email”).

Figura 40 – *Order by* pela posição da coluna

nome	salario	email
Bia Meireles	9450.00	bia.meireles@email.com
Helena Magalhaes	8600.00	helena.magalhaes@email.com
Helena Magalhaes	12500.99	helena@email.com
Nicolas Silva	8500.00	nicolas.silva@email.com
Pedro Filho	6800.00	pedro.filho@email.com
Silva Junior	8500.00	silva.junor@email.com
Sophia Arcanjo	6320.00	sophia.arcanjo@email.com
Tales Silva Souza	7689.00	tales.souza@email.com

Fonte: Albano; Albano, [S.d.].

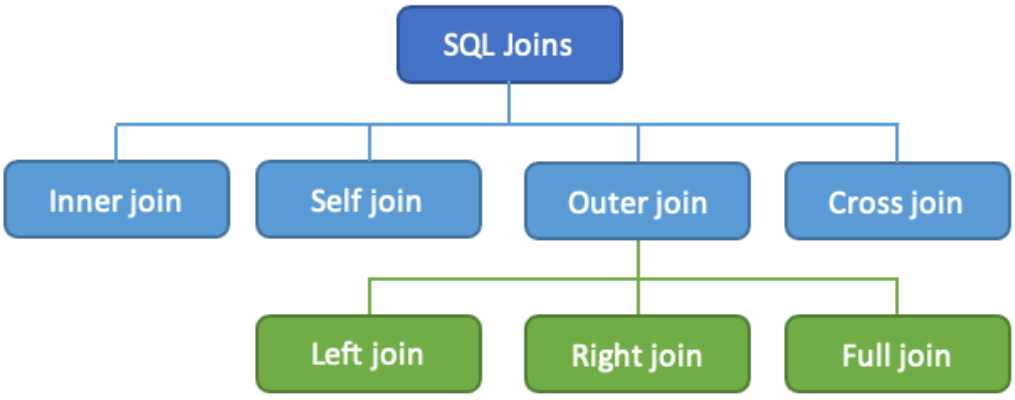
Até esse ponto, você possui uma grande quantidade de novos comandos a serem praticados (e vem mais por aí!). Então, avalie as possibilidades e se aventure aplicando seus novos conhecimentos em nosso estudo de caso.

TEMA 4 – JUNÇÃO DE TABELAS

Geralmente, as consultas em um Banco de Dados envolvem mais do que uma única tabela, sendo necessário fazer buscas mais complexas envolvendo diversas tabelas ao mesmo tempo e, inclusive, em uma autorrelação. Essa operação é conhecida como *join* (junção) e, para que seja possível essa relação entre tabelas, é obrigatória a presença de uma coluna em comum, que exercerá o papel de elo entre as tabelas.

O *join* pode ser dividido em quatro tipos:

Figura 41 – SQL *joins*

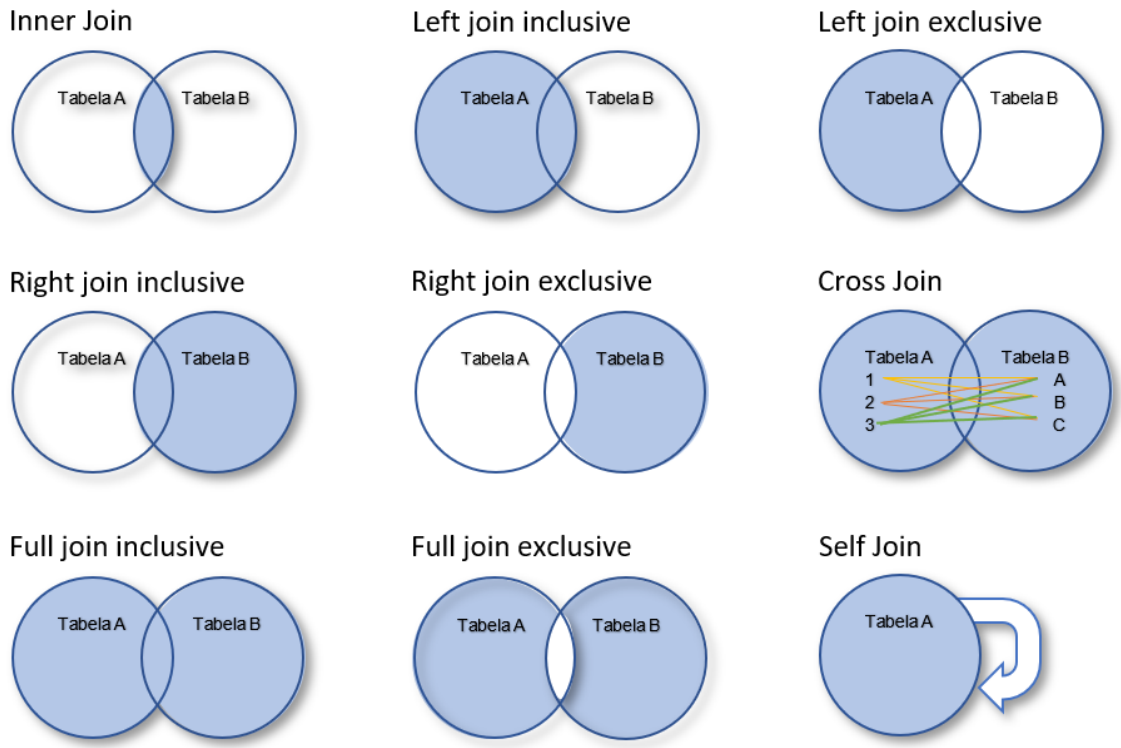


Fonte: Albano; Albano, [S.d.].

Pontos importantes:

- Usar a cláusula *on* para fazer a ligação entre as tabelas e deixar as demais condições para a cláusula *where*;
- O uso do *join* é uma junção explícita e o uso do *where* é uma junção implícita. Isso significa que podemos definir dentro do *where* uma condição de junção, omitindo o comando *join* (implícito);
- Os tipos *full join* e *cross join* geram impacto no desempenho da consulta, dessa forma, devem ser usados somente quando necessário.

Figura 42 – Tipos de *joins*



Fonte: Albano; Albano, [S.d.].

Para que possamos simular todos os exemplos de *join*, criamos um modelo de dados que não utiliza as restrições de chave estrangeira (*constraint*). As tabelas possuem a coluna de chave estrangeira, mas não estão definidas como uma FK.

Figura 43 – *Select* nas tabelas Estado, Cidade, Pergunta e Funcionário

Tabela Estado

id	nomeEstado	sigla
1	Paraná	PR
2	São Paulo	SP
3	Pernambuco	PE
4	Pará	PA
5	Rio Grande do Sul	RS
NULL	NULL	NULL

Tabela Cidade

id	nomecidade	EstadoID
1	Bagé	5
2	Curitiba	1
3	Recife	3
4	São Paulo	NULL
5	Porto Alegre	NULL
6	Olinda	3
NULL	NULL	NULL

Tabela Pergunta

id	pergunta
1	Qual a sua função?
2	Avalie a sua gerência:
NULL	NULL

Tabela Funcionario

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	gerente	emailFunc	cidadeId
1	Ana Rosa	F	1996-12-31	8500.00	1	1	NULL	ana.rosa@email.com	1
2	Tales Heitor	M	2000-10-01	7689.00	1	2	1	tales.heitor@email.com	NULL
3	Bia Meireles	F	2002-03-14	9450.00	1	2	1	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	2	pedro.filho@email.com	4
5	Helena Magalhaes	F	2000-01-01	12500.99	4	5	2	helena@email.com	6
6	Nicolas pinto	M	2002-12-10	8500.00	6	3	NULL	nicolas.pinto@email.com	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

4.1 *Inner Join*

O *inner join* ou junção natural, por meio de operadores de comparação em uma condição, realiza a combinação de linhas de duas tabelas e retorna apenas as que satisfazem a condição definida no *join*.

Chamada de *equi Join*, quando as colunas são comparadas usando o operador igual (=), ou *non equi join*, para os demais operadores.

Sintaxe:

```
select coluna1, coluna2
  from tabela_A inner join tabela_B
  on tabela_A.tabela_B_id = tabela_B.id;
```

Onde:

select	Comando de seleção.
coluna1 ... 2	Colunas a serem consultadas.
from	Cláusula de ligação do *select*, indica a tabela.
tabela_A	Nome da primeira tabela que fará parte da junção.
inner join	Comando de junção das duas tabelas.
tabela_B	Nome da segunda tabela que fará parte da junção.
on	Cláusula onde é definida a condição de ligação.
tabela_A.tabela_B_id	Nome da tabela e da coluna que possui dados em comum com a segunda tabela (B).
=	Operador de igualdade.
tabela_B.id	Tabela e nome coluna que possui dados em comum

	com a coluna da primeira tabela (A).
;	Ponto e vírgula, determina o final do comando.

Simulação: a empresa precisa de uma relação de cidades com o nome por extenso do estado e a unidade federativa (UF) a que pertence.

```
select nomecidade, nomeEstado, sigla from cidade
  inner join estado on cidade.estadoID = estado.id;
```

Figura 44 – Resultado do *inner join*

nomecidade	nomeEstado	sigla
Bagé	Rio Grande do Sul	RS
Curitiba	Paraná	PR
Recife	Pernambuco	PE
Olinda	Pernambuco	PE

Fonte: Albano; Albano, [S.d.].

Recorde que o *inner join* retorna somente as linhas que satisfaçam a condição do *on* e, para conectarmos duas tabelas, deve existir uma ou mais colunas em comum. Na simulação, a consulta retornou as linhas das cidades que possuem estados cadastrados, não retornando as cidades de São Paulo e Porto Alegre, pois não possuem o 'id' do estado.

Vale salientar que podemos utilizar o *join* implícito dentro da condição *where*. O resultado será o mesmo, sendo a *query* correspondente:

```
select nomecidade, nomeEstado, sigla from cidade, estado
  where cidade.estadoID = estado.id;
```

É recomendado usar o *join* para fazer a convergência de tabelas e o *where* para aplicar filtros, mas não existe impedimento em usar o *where* para realizar a função de junção de tabelas. Contudo, optando pelo *join*, a instrução será mais clara e mais fácil de ser lida.

4.2 *Outer Join*

Apresenta todas as linhas de uma tabela, mesmo quando elas não satisfazem a condição definida. Logo, permite obter a totalidade das linhas de uma tabela, mesmo sem existir uma linha correspondente na outra tabela. O

outer join subdivide-se em: *left join*, *right join* e *full join*. O termo *outer* é comumente omitido nas *queries*.

4.2.1 *Left Outer Join*

Satisfazendo ou não a condição de junção entre as tabelas, todas as linhas da tabela à esquerda do *join* serão incluídas no resultado.

Sintaxe do *left join inclusive*:

```
select coluna1, coluna2 from tabela_A left join tabela_B
   on tabela_A.tabela_B_id = tabela_B.id;
```

Sintaxe do *left join exclusive*:

```
select coluna1, coluna2 from tabela_A left join tabela_B
   on tabela_A.tabela_B_id = tabela_B.id
   where tabela_B.id is null;
```

Simulação 1 (*left join*): projetar o nome da cidade, o estado por extenso e a unidade federativa das cidades que possuem ou não estado informado.

```
select nomecidade, nomeEstado, sigla from cidade
   left join estado on cidade.EstadoID = estado.id;
```

Figura 45 – Resultado do *left join*

nomecidade	nomeEstado	sigla
Bagé	Rio Grande do Sul	RS
Curitiba	Paraná	PR
Recife	Pernambuco	PE
São Paulo	NULL	NULL
Porto Alegre	NULL	NULL
Olinda	Pernambuco	PE

Fonte: Albano; Albano, [S.d.].

Perceba que o resultado é diferente do retornado no *inner join*. Enquanto o *inner join* retorna apenas as cidades que possuem um estado relacionado, no *left join* são retornadas todas as cidades (tabela da esquerda), independente da cidade possuir ou não um estado (tabela da direita) relacionado.

Simulação 2 (*left join exclusive*): projetar o nome da cidade, o estado por extenso e a sigla do estado das cidades que não possuem um estado informado.

```
select nomecidade, nomeEstado, sigla from cidade
   left join estado on cidade.EstadoID = estado.id
   where estado.id is null;
```

Figura 46 – Resultado do *left join exclusive*

nomecidade	nomeEstado	sigla
São Paulo	NULL	NULL
Porto Alegre	NULL	NULL

Fonte: Albano; Albano, [S.d.].

4.2.2 *Right Outer Join*

O *right join* retorna todas as linhas da tabela da direita e apenas as linhas correspondentes da tabela da esquerda.

Sintaxe do *right join inclusive*:

```
select coluna1, coluna2 from tabela_A right join tabela_B
   on tabela_A.tabela_B_id = tabela_B.id;
```

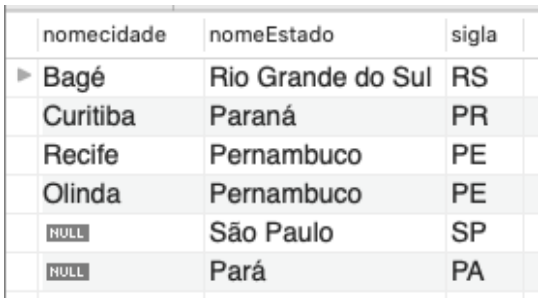
Sintaxe do *right join exclusive*:

```
select coluna1, coluna2 from tabela_A right join tabela_B
   on tabela_A.tabela_B_id = tabela_B.id
   where tabela_A.id is null;
```

Simulação (*right join inclusive*): projetar o nome da cidade, o nome do estado e a sigla do estado que possui ou não cidades relacionadas.

```
select nomecidade, nomeEstado, sigla from cidade
   right join estado on cidade.EstadoID = estado.id;
```

Figura 47 – Resultado do *right join inclusive*



nomecidade	nomeEstado	sigla
Bagé	Rio Grande do Sul	RS
Curitiba	Paraná	PR
Recife	Pernambuco	PE
Olinda	Pernambuco	PE
NULL	São Paulo	SP
NULL	Pará	PA

Fonte: Albano; Albano, [S.d.].

Nesse exemplo, foram retornados todos os estados, independente de possuírem ou não cidades relacionadas, ou seja, o inverso do *left join*.

4.2.3 *Full Outer Join*

Apresenta todas as linhas de ambas as tabelas envolvidas no *join*, mesmo as que não estão relacionadas com a outra tabela.

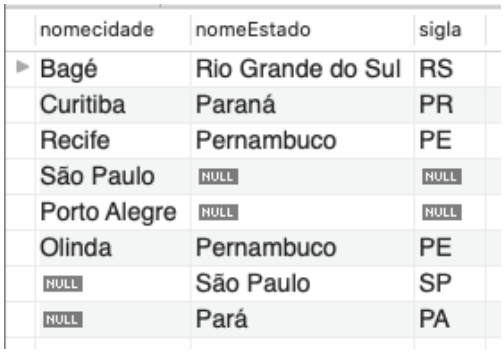
Sintaxe:

```
select coluna1, coluna2 from tabela_A full join tabela_B
   on tabela_A.tabela_B_id = tabela_B.id;
```

Simulação: projetar o nome da cidade, o estado por extenso e a sigla dos estados que possuem ou não cidades relacionadas e as cidades que não possuem estados relacionados.

```
select nomecidade, nomeEstado, sigla from cidade
   full join estado on cidade.EstadoID = estado.id;
```

Figura 48 – Resultado do *full join*



nomecidade	nomeEstado	sigla
Bagé	Rio Grande do Sul	RS
Curitiba	Paraná	PR
Recife	Pernambuco	PE
São Paulo	NULL	NULL
Porto Alegre	NULL	NULL
Olinda	Pernambuco	PE
NULL	São Paulo	SP
NULL	Pará	PA

Fonte: Albano; Albano, [S.d.].

Importante: o MySQL não possui suporte ao *full join*. Como alternativa, pode-se realizar um *left join* combinado com o *right join exclusive*.

```
select nomecidade, nomeEstado, sigla from cidade
   left join estado on cidade.EstadoID = estado.id
   union select nomecidade, nomeEstado, sigla from cidade
   right join estado on cidade.EstadoID = estado.id
   where cidade.EstadoID is null;
```

4.3 *Cross Join*

Usado apenas em situações específicas. O *cross join* gera um resultado formado por todas as combinações possíveis de uma linha da primeira tabela com uma linha da segunda tabela. Nesse caso, não existe uma condição de junção, sendo o resultado um produto cartesiano das duas tabelas, ou seja, a multiplicação da quantidade de linhas de uma tabela pela quantidade de linhas da outra tabela.

Sintaxe:

```
select coluna1, coluna2 from tabela_A
   cross join tabela_B;
```

Simulação: a empresa criou um questionário contendo diversas perguntas para serem respondidas por todos os funcionários. Baseado nas tabelas Pergunta e Funcionário é necessário cruzarmos essas duas tabelas, realizando combinações e, assim, gerando para cada funcionário todas as perguntas contidas na tabela Pergunta.

```
select nomefunc, pergunta from pergunta
   cross join funcionario;
```

Figura 49 – Resultado do *cross join*

nomefunc	pergunta
Ana Rosa	Qual a sua função?
Ana Rosa	Avalie a sua gerência:
Tales Heitor	Qual a sua função?
Tales Heitor	Avalie a sua gerência:
Bia Meireles	Qual a sua função?
Bia Meireles	Avalie a sua gerência:

Fonte: Albano; Albano, [S.d.].

4.4 *Self Join*

Apesar de não ser frequente, em algumas situações, será necessário realizar uma autorrelação, ou seja, a tabela se relacionado com ela mesma. Nesses casos, gera-se uma cópia da tabela e, obrigatoriamente, usamos um *alias* (apelido) de tabela para distinguir uma tabela da outra.

Sintaxe:

```
select coluna1, coluna2 from tabela_A tabA, tabela_A tabB
   where tabA.colunaY = tabB.colunaX;
```

Simulação: cada funcionário possui um gerente e, obviamente, o gerente exerce o papel de funcionário, ou seja, está registrado na tabela Funcionário. Como poderíamos projetar o nome do funcionário e o nome do gerente correspondente em uma mesma consulta?

```
select funcionario.nomefunc, gerente.nomeFunc 'gerente'
   from funcionario inner join funcionario as gerente
   on funcionario.gerente = gerente.matricula
   order by funcionario.nomefunc;
```

Figura 50 – Resultado do *self join*

nomefunc	gerente
Bia Meireles	Ana Rosa
Helena Magalhaes	Tales Heitor
Pedro Filho	Tales Heitor
Tales Heitor	Ana Rosa

Fonte: Albano; Albano, [S.d.].

Perceba que na *query* utilizamos duas vezes a mesma tabela, Funcionário, em que uma fez o papel de funcionário e a outra o de gerente. Somente dessa forma é possível aplicar o *join* entre elas. Como não podemos usar o nome da tabela de forma duplicada, fazemos uso do recurso *alias* (apelido), que será tratado na próxima etapa.

Outro aspecto a ser considerado é que apenas os funcionários que possuem gerentes foram apresentados. Isso ocorreu porque optamos pelo uso do *inner join*, se tivéssemos usado o *left join*, retornariam todos os funcionários.

4.5 Junção com várias tabelas

Em uma consulta podemos relacionar quantas tabelas forem necessárias para a realização da operação desejada, sempre reforçando que é obrigatório existir uma coluna em comum para atuar como ligação entre as tabelas.

Simulação: projetar o nome do funcionário, o nome da cidade e o estado por extenso onde esse funcionário reside.

```
select nomeFunc, nomecidade, nomeEstado from funcionario
   inner join cidade on funcionario.cidadeID = cidade.id
   inner join estado on cidade.EstadoID = estado.id
   order by nomeFunc;
```

Figura 51 – *Join* com várias tabelas

nomeFunc	nomecidade	nomeEstado
Ana Rosa	Bagé	Rio Grande do Sul
Bia Meireles	Curitiba	Paraná
Helena Magalhaes	Olinda	Pernambuco

Fonte: Albano; Albano, [S.d.].

Analisando as tabelas envolvidas, percebermos que não há uma ligação direta entre as tabelas Funcionário e Estado, mas é possível fazer a ligação de forma indireta envolvendo Funcionário → Cidade → Estado. Assim, demonstramos que todas as tabelas envolvidas na consulta não necessariamente precisam possuir uma ligação direta.

Também é necessário ressaltar que as cidades com ‘id’ igual a 4 e 5 não possuem valor na coluna “estadoId” e, dessa forma, não atendem a condição do *inner join*.

Em nosso cotidiano, realizamos diversas consultas que envolvem muitas tabelas. Então, lembre-se de aplicar esses comandos sobre *join* em nosso estudo de caso. Assim, você fixará melhor esse importante assunto.

TEMA 5 – OUTROS COMANDOS E RECOMENDAÇÕES

Para exemplificar os comandos utilizaremos as tabelas Cidade, Cliente e Funcionário.

Figura 52 – *Select* nas tabelas Cidade, Cliente e Funcionário

Tabela Cidade

id	nomecidade	EstadoID
1	Bagé	5
2	Curitiba	1
3	Recife	3
4	São Paulo	NULL
5	Porto Alegre	NULL
6	Olinda	3
NULL	NULL	NULL

Tabela Cliente

id	nome	genero	dataNascimento	salario	email	cidadeId
1	Helena Magalhaes	F	2000-01-01	12500.99	helena@email.com	2
2	Nicolas Silva	M	2002-12-10	8500.00	nicolas.silva@email.com	3
3	Silva Junior	M	1996-12-31	8500.00	silva.junor@email.com	1
4	Tales Silva Souza	M	2000-10-01	7689.00	tales.souza@email.com	1
5	Bia Meireles	F	2002-03-14	9450.00	bia.meireles@email.com	2
6	Pedro Filho	M	1998-05-22	6800.00	pedro.filho@email.com	5
7	Helena Magalhaes	F	1994-08-10	8600.00	helena.magalhaes@email.com	4
8	Sophia Arcanjo	F	1991-11-20	6320.00	sophia.arcanjo@email.com	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabela Funcionario

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	gerente	emailFunc	cidadeId
1	Ana Rosa	F	1996-12-31	8500.00	1	1	NULL	ana.rosa@email.com	1
2	Tales Heitor	M	2000-10-01	7689.00	1	2	1	tales.heitor@email.com	NULL
3	Bia Meireles	F	2002-03-14	9450.00	1	2	1	bia.meireles@email.com	2
4	Pedro Filho	M	1998-05-22	6800.00	3	3	2	pedro.filho@email.com	4
5	Helena Magalhaes	F	2000-01-01	12500.99	4	5	2	helena@email.com	6
6	Nicolas pinto	M	2002-12-10	8500.00	6	3	NULL	nicolas.pinto@email.com	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

5.1 *Alias* (apelido das tabelas e colunas)

Recurso que permite renomear uma tabela ou uma coluna. Utiliza-se *alias* nas tabelas para simplificar o nome das tabelas e/ou reduzir a escrita dos comandos. Algumas vezes, definir um nome mais amigável a uma tabela ou coluna torna mais fácil e intuitivo a identificação dos dados. Inclusive, podemos criar um *alias* com palavras compostas apenas usando as aspas simples.

Vale salientar que o *alias* possui a cláusula *as*, que pode ser omitida. Além disso, existem poucas situações em que o uso do *alias* é obrigatório, como, por exemplo, no autorrelacionamento (*self join*) e em subconsultas correlacionadas.

Vamos comparar o *select* com e sem o uso do *alias* (aplicação em colunas):

```
-- Sem alias
select nomeFunc, salarioFunc, salarioFunc * 1.10
   from funcionario;
-- Com alias
select nomeFunc 'Nome do Funcionário',
   salarioFunc as 'Salário atual',
   salarioFunc * 1.10 as 'Novo salário' from funcionario;
```

Figura 53 – Comparativo usando o *alias* em colunas

Sem Alias

nomeFunc	salarioFunc	salarioFunc * 1.10
Ana Rosa	8500.00	9350.0000
Tales Heitor	7689.00	8457.9000
Bia Meireles	9450.00	10395.0000
Pedro Filho	6800.00	7480.0000
Helena Magalhaes	12500.99	13751.0890
Nicolas pinto	8500.00	9350.0000

Com Alias

Nome do Funcionário	Salário atual	Novo salário
Ana Rosa	8500.00	9350.0000
Tales Heitor	7689.00	8457.9000
Bia Meireles	9450.00	10395.0000
Pedro Filho	6800.00	7480.0000
Helena Magalhaes	12500.99	13751.0890
Nicolas pinto	8500.00	9350.0000

Fonte: Albano; Albano, [S.d.].

Compare o cabeçalho das duas consultas. Perceba que os nomes das colunas da tabela da direita são mais informativos e fáceis de entender.

No caso do uso do *alias* em tabelas, não usamos aspas. Veja o exemplo:

```
select nomeFunc, nomeCidade from funcionario f
   inner join cidade c on f.cidadeID = c.id;
```

Nesse caso, a tabela Funcionário é chamada de ‘f’ apenas durante a consulta, enquanto a tabela Cidade é chamada de ‘c’.

5.2 Limit

A cláusula *limit* é utilizada para determinar o número de linhas que devem ser retornadas na consulta. Essa cláusula é bastante útil quando trabalhamos com tabelas muito grandes (com milhares/milhões de linhas), pois uma consulta desse volume de dados pode afetar o desempenho da consulta. Assim, consultamos apenas algumas linhas para verificação do resultado.

Sintaxe:

```
select colunas from nomeTabela
     limit deslocamento, nroLinhas;
```

Onde:

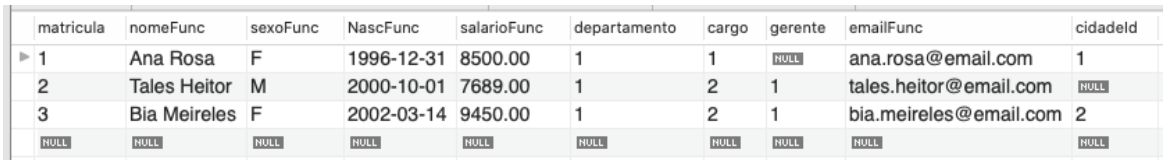
deslocamento — Indica o número de linhas que serão ignoradas. Se omitido, assume o valor 0.

nroLinhas — Quantidade de linhas que serão retornadas na consulta.

Exemplos:

```
select * from funcionario limit 3;
```

Figura 54 – Uso do *limit* padrão



matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	gerente	emailFunc	cidadeId
1	Ana Rosa	F	1996-12-31	8500.00	1	1	NULL	ana.rosa@email.com	1
2	Tales Heitor	M	2000-10-01	7689.00	1	2	1	tales.heitor@email.com	NULL
3	Bia Meireles	F	2002-03-14	9450.00	1	2	1	bia.meireles@email.com	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

```
select * from funcionario LIMIT 3, 2;
```

Figura 55 – Uso do *limit* com deslocamento

matricula	nomeFunc	sexoFunc	NascFunc	salarioFunc	departamento	cargo	gerente	emailFunc	cidadeId
4	Pedro Filho	M	1998-05-22	6800.00	3	3	2	pedro.filho@email.com	4
5	Helena Magalhaes	F	2000-01-01	12500.99	4	5	2	helena@email.com	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Albano; Albano, [S.d.].

A primeira consulta retornou as três primeiras linhas da tabela Funcionário (linhas de 1 a 3). Já a segunda consulta apresentou duas linhas, lendo a partir da quarta linha, pois foi declarado um deslocamento de 3.

5.3 *Distinct*

A cláusula *distinct*, quando usada no *select*, elimina as linhas repetidas, ou seja, se houver duas ou três linhas iguais, somente uma irá retornar no resultado. Essa cláusula deve aparecer logo após o *select*.

O *distinct* é muito útil em situações que desejamos contar, somar, fazer a média de valores únicos, entre outros.

Simulação 1: projetar o nome dos clientes por ordem alfabética, comparando o uso do *distinct*.

```
select nome from cliente order by nome;
select distinct nome from cliente order by nome;
```

Figura 56 – *Select* x *select com distinct*

Select

nome
Bia Meireles
Helena Magalhaes
Helena Magalhaes
Nicolas Silva

Select com Distint

nome
Bia Meireles
Helena Magalhaes
Nicolas Silva
Pedro Filho

Fonte: Albano; Albano, [S.d.].

Perceba que na primeira consulta aparecem duas ‘Helena Magalhaes’ e na segunda consulta aparece somente uma vez.

Simulação 2: projetar o nome e o e-mail dos clientes por ordem alfabética (comparativo com o uso do *distinct*).

```
select nome, email from cliente order by nome;
select distinct nome, email from cliente order by nome;
```

Figura 57 – *Select* x *select com distinct* (colunas)

Select

nome	email
Bia Meireles	bia.meireles@email.com
Helena Magalhaes	helena@email.com
Helena Magalhaes	helena.magalhaes@email.com
Nicolas Silva	nicolas.silva@email.com

Select com Distint

nome	email
Bia Meireles	bia.meireles@email.com
Helena Magalhaes	helena.magalhaes@email.com
Helena Magalhaes	helena@email.com
Nicolas Silva	nicolas.silva@email.com

Fonte: Albano; Albano, [S.d.].

Avaliando os novos *selects*, é possível perceber que aparecem duas ‘Helena Magalhaes’ nas duas consultas. Isso ocorre porque o *distinct* verifica a linha inteira do resultado e, nesse caso, apesar do nome ‘Helena Magalhaes’ ser igual, o e-mail é diferente. Logo, as duas linhas são apresentadas.

5.4 *Case*

Em consultas SQL podem ocorrer situações em que seja necessário realizarmos algum tipo de teste lógico. Logo, fazemos uso do *case*.

Sintaxe:

```
case
   when condicao1 then resultado1
   when condicaon then resultadon
   else resultado
end;
```

Exemplo:

```
select nomeFunc,
   case
        when sexoFunc = 'M' then 'Masculino'
        when sexoFunc = 'F' then 'Feminino'
```

```
        else 'Outros'
    end as 'Genero' from funcionario;
```

Figura 58 – Uso do *case*

nomeFunc	Genero
Ana Rosa	Feminino
Tales Heitor	Masculino
Bia Meireles	Feminino
Pedro Filho	Masculino
Helena Magalhaes	Feminino
Nicolas pinto	Masculino

Fonte: Albano; Albano, [S.d.].

No exemplo, estamos testando o gênero do funcionário, apresentando os termos ‘Masculino’, ‘Feminino’ ou ‘Outros’, em vez de mostrar apenas as letras ‘M’ ou ‘F’.

5.5 *Union* e *Union All*

Permite que a consulta realizada em vários *selects* seja exibida em um único resultado. Para isso, necessita atender os seguintes requisitos:

- As colunas devem ser do mesmo tipo, isto é, *varchar* com *varchar*, *int* com *int*, e assim sucessivamente;
- Todos os comandos *select* devem possuir o mesmo número de colunas.

Por padrão, o *union* elimina do resultado as linhas duplicadas. Usa-se a cláusula *all* para que elas sejam incluídas no resultado. Analise os exemplos.

Usando apenas o *union*:

```
select nome, dataNascimento from cliente
    union select nomeFunc, NascFunc from funcionario;
```

Figura 59 – Uso do *union*

nome	dataNascimento
Helena Magalhaes	2000-01-01
Nicolas Silva	2002-12-10
Silva Junior	1996-12-31
Tales Silva Souza	2000-10-01
Bia Meireles	2002-03-14

Fonte: Albano; Albano, [S.d.].

Usando a cláusula *all* (linhas duplicadas):

```
select nome, dataNascimento, 'cliente' from cliente
  union all select nomeFunc, NascFunc, 'funcionario'
  from funcionario order by 1;
```

Figura 60 – Uso do *union all* (linhas duplicadas)

nome	dataNascimento	cliente
Ana Rosa	1996-12-31	funcionario
Bia Meireles	2002-03-14	cliente
Bia Meireles	2002-03-14	funcionario
Helena Magalhaes	2000-01-01	cliente
Helena Magalhaes	1994-08-10	cliente
Helena Magalhaes	2000-01-01	funcionario

Fonte: Albano; Albano, [S.d.].

Perceba que no segundo exemplo aparecem nomes repetidos, algo que não aconteceu no primeiro exemplo. Isso ocorre em decorrência da cláusula *all*.

Outro detalhe é o *order by*. Sempre que você precisar usar o *order by*, deve declará-lo no último *select* do *union* e, obrigatoriamente, a ordenação será realizada pelo número da coluna.

5.6 Recomendações – boas práticas

Existem algumas recomendações sobre boas práticas que você deve levar em consideração na execução de *queries* em Banco de Dados, são elas:

- Não use o asterisco (*) no comando *select*, pois isso gera um impacto no desempenho da consulta;
- Sempre que possível use filtros (*where*) nos comandos *delete* e *update*;
- Evite criar um *insert* contendo muitas linhas, pois esse processo aumentar muito o tamanho do Banco de Dados.

FINALIZANDO

Nesta etapa, aprendemos sobre os comandos da *Data Manipulation Language* (DML), os quais permitem realizar operações de inserção, alteração e exclusão de dados nas tabelas. Também discutimos sobre a aplicação de diversos operadores em consultas, tornando os filtros mais avançados.

Por meio de exemplos práticos apresentamos os comandos de *join*, que possibilitam ampliar as consultas envolvendo mais de uma tabela e identificar os diversos tipos que produzem resultados diferentes. Por fim, conhecemos mais alguns comandos SQL que fazem a diferença em nossas tarefas diárias.

Novamente, recomendamos que você dedique um tempo de qualidade para assimilar os conhecimentos apresentados, revisando todos os conceitos cuidadosamente e colocando-os em prática, criando suas próprias *queries*.

Lembre-se, a palavra de ordem é praticar!