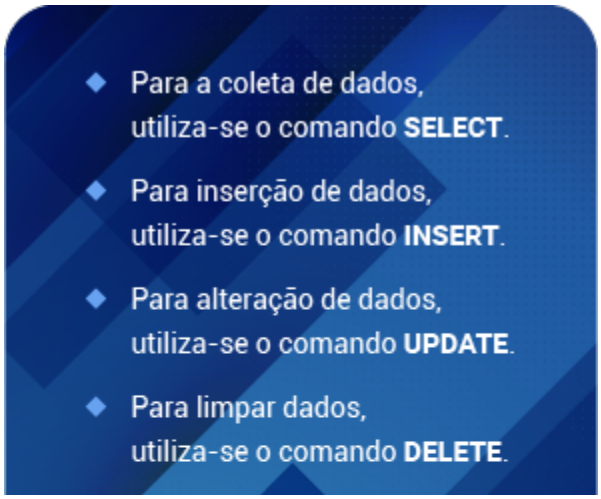
Técnico em Desenvolvimento de Sistemas

Manipulação de dados: inclusão, atualização e exclusão de dados

Manipulação de dados é o processo de coletar, adicionar, limpar ou alterar dados para torná-los mais legíveis e organizados. Em banco de dados, a manipulação de dados é feita por meio da linguagem SQL (*structured query language*, ou, em português, “linguagem de consulta estruturada”), utilizada para recuperar, incluir, remover ou modificar as informações armazenadas. Esse grupo de comandos é conhecido como **DML** (***data manipulation language***, ou linguagem de manipulação de dados).

Cada uma dessas operações é feita por meio de um comando construído com uma série de sintaxes para chegar ao resultado esperado. Para cada operação existe um comando específico:



- Para a coleta de dados, utiliza-se o comando SELECT.
- Para inserção de dados, utiliza-se o comando INSERT.
- Para alteração de dados, utiliza-se o comando UPDATE.
- Para limpar dados, utiliza-se o comando DELETE.

Esse conteúdo focará no uso dos comandos **INSERT**, **UPDATE** e **DELETE**. Porém, em alguns momentos, será preciso realizar a coleta dos dados para garantir que a inserção, remoção ou atualização foram realizadas com sucesso. Para isso, será utilizado o seguinte comando:

```
SELECT * FROM nome_da_tabela;
```

No conteúdo desta unidade sobre a consulta de dados, o comando **SELECT** será explorado com mais detalhes. Por enquanto, será utilizada essa combinação de sintaxes para consultar todos os dados de uma tabela específica. Caso deseje fazer consultas mais complexas ou específicas, confira o conteúdo citado.

Preparando ambiente no MySQL Workbench

Antes continuar, saiba que haverá muitos exemplos no decorrer deste conteúdo, os quais você poderá testar diretamente no seu computador por meio do MySQL Workbench.

Caso ainda não tenha configurado seu ambiente de banco de dados, confira o conteúdo específico sobre banco de dados desta mesma unidade curricular. Caso você já tenha tudo pronto, então este será o momento de você criar a base de dados e as tabelas que manipulará em seguida. Para isso, basta abrir o MySQL Workbench, conectar-se ao banco de dados e executar o seguinte *script*:

```
DROP DATABASE IF EXISTS senac_ead;

CREATE DATABASE senac_ead;
USE senac_ead;

CREATE TABLE curso (
id int AUTO_INCREMENT,
nome varchar(150),
descricao text,
PRIMARY KEY (id)
);

CREATE TABLE aluno (
id int AUTO_INCREMENT,
nome varchar(30),
sobrenome varchar(30),
data_nascimento date,
curso_id int,
PRIMARY KEY (id),
FOREIGN KEY (curso_id) REFERENCES curso(id)
```

);

Isso criará a base de dados **senac_ead** com as tabelas **curso** e **aluno**, cujos dados você manipulará durante os exemplos.

Inclusão de dados

A instrução **INSERT** é um dos comandos mais populares em SQL e um dos primeiros comandos que você aprenderá a usar. A instrução **INSERT**, ou **INSERT INTO**, é utilizada para inserir (ou adicionar) dados em uma tabela e permite fazer isso de várias maneiras, pois esse comando tem muitas configurações e variáveis.

Para que se possa executar esse comando, é essencial que a tabela já exista. Caso você tente inserir um registro em uma tabela ou coluna inexistente, a SQL retornará um erro.

Comando INSERT

A sintaxe mais básica do comando **INSERT** é:

INSERT INTO nome_da_tabela VALUES (valores);

Comece o comando com **INSERT INTO**, que são palavras-chave da SQL e, em seguida, mencione a tabela na qual você quer inserir os dados. Após isso, adicione a palavra-chave **VALUES**. Em seguida, abra os parênteses e especifique cada um dos valores que você quer inserir. O primeiro valor corresponde à primeira coluna da tabela; o segundo valor corresponde à segunda coluna, e assim por diante. Os valores devem corresponder ao tipo de dado apontado na criação da tabela. Sendo assim, os dados serão passados da seguinte maneira:

Números

Não precisam estar entre aspas.

Exemplo:

INSERT INTO tabela VALUES (1, 2, 3);

Strings

Precisam estar entre aspas simples.

Exemplo:

INSERT INTO tabela VALUES ('Valor 1', 'Valor 2', 'Valor 3');

Null

Significa que o valor nulo será inserido.

Exemplo:

INSERT INTO tabela VALUES (null, null, null);

Por fim, feche os parênteses e termine com um ponto e vírgula.

Apesar de não ser obrigatório quando se executa apenas um comando, é importante utilizar o ponto e vírgula no final de cada instrução para indicar o fim da *query*. Caso haja outra instrução a seguir, a falta do ponto e vírgula indicará à SQL uma continuidade da instrução anterior, o que pode resultar em erros.

Observe que, para utilizar o comando **INSERT**, é essencial que você tenha conhecimento de como é a estrutura da tabela que será manipulada. Uma tabela pode conter uma ou mais colunas e cada coluna tem seu tipo de dado especificado. Não conhecer essas informações resultará em erros na execução dos seus comandos. Sendo assim, primeiramente, conheça a estrutura da tabela na qual você está trabalhando.

De acordo com o *script* disponibilizado no tópico “Preparando o ambiente...”, haverá a seguinte estrutura:

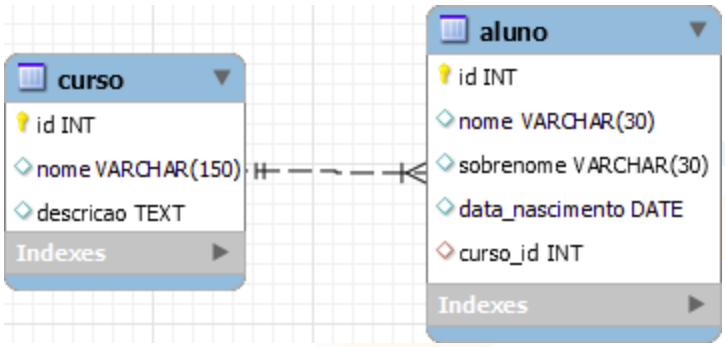


Figura 1 – Modelo ER da base de dados “senac_ead”

Na imagem constam dois retângulos. O retângulo da esquerda está sendo identificado com o texto “curso” no topo e, logo abaixo, os atributos “id INT”, “nome VARCHAR(150)” e “descrição TEXT”. Já o retângulo da direita contém o texto “aluno” no topo e os atributos “id INT”, “nome VARCHAR(30)”, “sobrenome VARCHAR(30)”, “data_nascimento DATE” e “curso_id INT” logo abaixo.
Ambos os retângulos estão conectados por uma linha que tem a ponta em formato de tridente no lado direito e duas pequenas linhas paralelas na ponta esquerda.

Por meio desse modelo ER, é possível identificar quais colunas compõem cada uma das tabelas e seus respectivos tipos de dados.

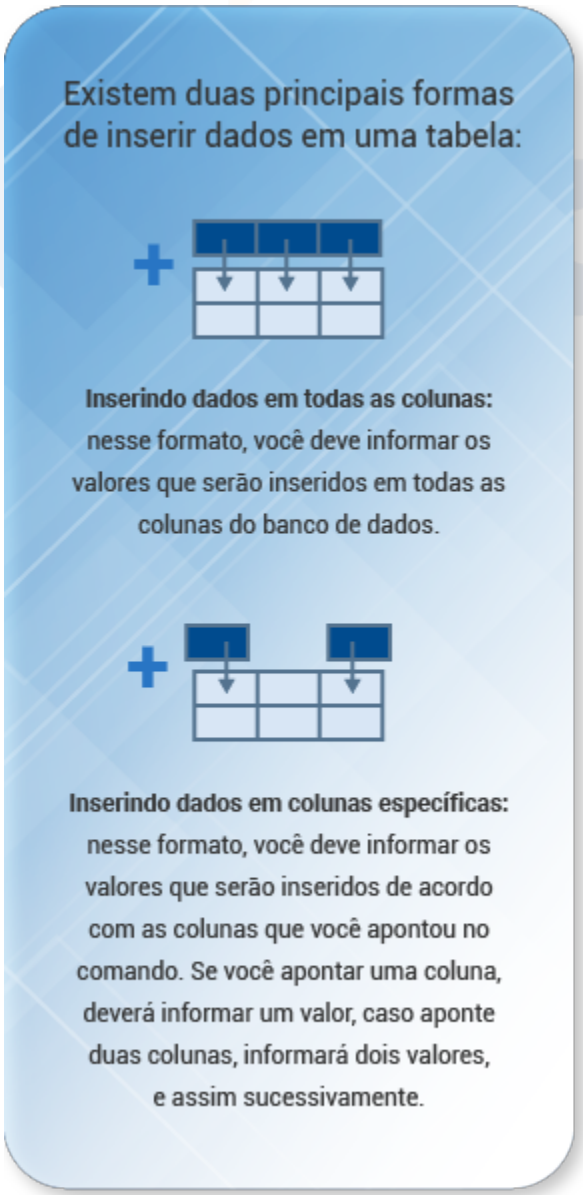
Para utilizar a instrução **INSERT**, você precisará dos seguintes dados:

- O nome da tabela na qual quer inserir os dados
- Os valores a serem inseridos na tabela
- As colunas nas quais serão inseridos os valores (opcional)

Apesar de não serem necessários os nomes das colunas, é uma boa prática especificá-los. Se não o fizer, haverá uma grande probabilidade de as instruções **INSERT** falharem, caso a estrutura da tabela mude. Portanto, o ideal é que você sempre aponte as colunas com seus respectivos dados no comando para garantir que não ocorram erros. Nesse cenário, a estrutura do comando ficará da seguinte maneira:

INSERT INTO nome_da_tabela(coluna1, coluna2, coluna3) VALUES (valor1, valor2, valor3);

A diferença, em comparação ao comando anterior, é que aqui você abrirá os parênteses e, dentro deles, especificará o nome de cada coluna, separando-as por vírgulas. Os valores inseridos se alinham com as colunas especificadas anteriormente. Caso não sejam especificadas as colunas, você deverá informar os valores de todas as colunas e a ordem dos valores deverá seguir a estrutura da tabela no banco de dados.



Existem duas principais formas de inserir dados em uma tabela:

- Inserindo dados em todas as colunas: nesse formato, você deve informar os valores que serão inseridos em todas as colunas do banco de dados.
- Inserindo dados em colunas específicas: nesse formato, você deve informar os valores que serão inseridos de acordo com as colunas que você apontou no comando. Se você apontar uma coluna, deverá informar um valor, caso aponte duas colunas, informará dois valores, e assim sucessivamente.

Em ambos os casos, fique atento à quantidade de colunas da tabela e à quantidade de valores que você está passando. Se, por exemplo, você apontar cinco colunas e passar quatro valores, o MySQL pode apresentar o seguinte erro:

Error Code: 1136. Column count doesn't match value count at row 1

Esse erro indica que a contagem de colunas não combina com a quantidade de valores passados. Portanto, a *query* nem chegará a ser executada. O mesmo ocorrerá com o comando executado sem que se especifique as colunas, pois, se a tabela contém cinco colunas, por exemplo, deve existir cinco valores dentro dos parênteses do **VALUES**.

Agora que você já conhece a estrutura do banco de dados e como funciona o comando **INSERT**, veja alguns exemplos práticos de como inserir dados nas tabelas.

Clique ou toque para visualizar o conteúdo.

Instrução INSERT sem colunas

O primeiro exemplo demonstrará como se executa um **INSERT** sem especificar as

colunas no comando. Comece inserindo um registro na tabela “curso”.

```
INSERT INTO curso VALUES (
1,
'Técnico em Desenvolvimento de Sistemas',
'Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.'
);
SELECT * FROM curso;
```

Após a execução do comando **INSERT**, foi feita uma consulta na tabela “curso” para buscar todos os registros que se tem com o comando **SELECT**. Se tudo for executado corretamente, haverá o seguinte resultado:

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.

Caso você queira inserir um outro registro, lembre-se de que já há um registro com a id 1. Portanto, o próximo registro deverá conter o id 2. Essa gestão manual dos valores de uma coluna com chave primária (*primary key*) torna-se bastante difícil conforme novos registros são adicionados. Por essa razão, estas colunas “id” contêm a configuração **AUTO_INCREMENT**, que fará a gestão automaticamente para você.

Porém, ao utilizar o **INSERT** sem especificar as colunas, você não usufruirá do **AUTO_INCREMENT**, pois será obrigado a **especificar os valores de todos os campos**. Com isso, o uso do comando **INSERT** especificando as colunas torna-se mais interessante.

Instrução INSERT com colunas

No próximo exemplo, haverá a inserção de um registro na tabela “curso”, informando apenas o nome. O curso que será cadastrado agora é o Técnico em Informática para Internet.

INSERT INTO curso(nome) VALUES ('Técnico em Informática para Internet'); SELECT * FROM curso;

Após a execução do comando, o registro será adicionado à tabela “curso” e, apesar de só ter sido passado o valor de nome, haverá o “id 2”, já que a coluna “id” está configurada com **AUTO_INCREMENT**. **Os campos que não forem especificados, como descrição, terão seu valor preenchido como NULL**.

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	NULL

A coluna só preencherá o valor do campo como **NULL** caso a coluna não tenha a configuração **NOT NULL**. Se uma coluna estiver configurada como **NOT NULL**, isso significa que o campo deve obrigatoriamente ser preenchido com algum valor.

Se você quiser adicionar mais registros, basta escrever novamente o comando **INSERT INTO**, passar os valores que desejamos e executá-lo. Porém, essa tarefa pode se tornar bastante repetitiva e cansativa caso você tenha muitos registros para inserir. Nesse cenário, é melhor realizar a inserção dos dados com apenas uma instrução **INSERT**.

Inserir vários registros em uma única instrução

Para inserir vários registros com apenas um comando **INSERT**, basta escrever a instrução normalmente e, no final, após inserir os valores dentro dos parênteses, adicionar uma virgula após o fechamento dos parênteses, abrindo um novo parêntese para passar os outros valores. Repita o processo até passar todos os valores desejados. No final, você terá o seguinte resultado:

```
INSERT INTO curso (nome) VALUES
('Técnico em Administração'),
('Técnico em Contabilidade'),
('Técnico em Design de Interiores');

SELECT * FROM curso;
```

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	NULL
3	Técnico em Administração	NULL
4	Técnico em Contabilidade	NULL
5	Técnico em Design de Interiores	NULL

Essa abordagem é bastante útil quando se precisa inserir vários registros de uma única vez.

Agora que já foram cadastrados vários cursos, veja como adicionar alguns registros na tabela “aluno”.

Inserir valores de data e hora

Uma das principais diferenças entre as tabelas “curso” e “aluno” está no número de dados e nos tipos utilizados. Na tabela “curso” estão tipos de dados mais comuns, como numéricos e textos, sendo aplicados. Já na tabela “aluno”, além destes, tem-se o tipo **Date** (data, em inglês) sendo aplicado na coluna “data_nascimento”.

Para inserir uma data ou data e hora no MySQL, você deve especificar os valores dentro de uma *string* em um formato específico:

- AAAA-MM-DD (ou AA-MM-DD) para a data
- AAAA-MM-DD HH: MM: SS (ou AA-MM-DD HH: MM: SS) para data e hora

Observe essas situações na prática, usando a tabela “aluno” como exemplo.

Como primeiro exemplo, será feito um cadastro informando a data no formato AAAA-MM-DD.

INSERT INTO aluno (nome, sobrenome, data_nascimento) VALUES ('Alice', 'Gonçalves', '1998-10-18');

SELECT * FROM aluno;

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL

No segundo exemplo, será utilizado o formato AA-MM-DD para informar o ano.

INSERT INTO aluno (nome, sobrenome, data_nascimento) VALUES ('Eduardo', 'Machado', '98-07-16');

SELECT * FROM aluno;

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	NULL

No terceiro exemplo, será utilizado o formato AAAA-MM-DD novamente, mas, dessa vez, deverá ser informado um valor menor que dez em MM que não esteja acompanhado do zero.

INSERT INTO aluno (nome, sobrenome, data_nascimento) VALUES ('Vitória', 'Prestes', '1997-2-15');

SELECT * FROM aluno;

Nesse exemplo, empregou-se o valor 2 para o mês e, como se pode ver, esse é um formato aceito pelo SQL. No final, o valor será salvo como 02.

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	NULL
3	Vitória	Prestes	1997-02-15	NULL

Inserção de registros com chave estrangeira

Nos exemplos anteriores, foram cadastrados alguns alunos na tabela, mas não

foram inseridos valores nos campos “curso_id”. Isso porque o campo “curso_id” é um campo que referencia uma chave primária (*primary key*) de outra tabela. Ou seja, trata-se de um campo com **chave estrangeira** (*foreign key*).

Tabelas com a chave estrangeira são chamadas de “tabelas filhas”, enquanto a tabela com a chave primária é chamada de “tabela pai”. Neste cenário, a “tabela filha” é a tabela “aluno”, pois ela contém a coluna “curso_id” com uma chave estrangeira que referencia a chave primária da tabela “curso”.

O uso de chave estrangeira é muito útil para se evitar dados repetidos dentro do banco de dados. Isso ocorre porque os valores da tabela “curso” são referenciados pela sua chave primária.

Se você executar o comando **SELECT * FROM curso**, verá que a tabela “curso” se encontra com os seguintes registros:

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	NULL
3	Técnico em Administração	NULL
4	Técnico em Contabilidade	NULL
5	Técnico em Design de Interiores	NULL

Portanto, são cinco registros e cada um deles tem um código identificador único, que é o “id”. Então, se quiser apontar para o curso Técnico em Desenvolvimento de Sistemas, informe o “id 1”. Se, por exemplo, o nome desse curso estivesse com algum erro no nome e precisasse ser atualizado, não seria necessário fazer esse ajuste para cada aluno vinculado a esse curso. Portanto, o uso de **FOREIGN KEY** permite que os registros de tabelas sejam atualizados independentemente, garantindo a integridade dos dados.

Como chaves estrangeiras apontam para chaves primárias, que, em sua maioria, são valores numéricos, então deve-se preencher esse campo com o número do registro que é necessário vincular.

Isso, na prática, terá a seguinte sintaxe:

INSERT INTO aluno (nome, curso_id) VALUES ('Alex', 1); SELECT * FROM aluno;

Com esse comando, o aluno “Alex” foi inserido na tabela “aluno” e seu registro vinculado ao curso que contém “id 1” na tabela “curso”.

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	NULL
3	Vitória	Prestes	1997-02-15	NULL
4	Alex	NULL	NULL	1

Agora que você já aprendeu como inserir registro em tabelas do banco de dados, veja como alterar valores já inseridos.

Antes de seguir para o conteúdo sobre atualização, insira novos registros nas tabelas “curso” e “aluno”. Para isso, em ambas as tabelas, você deve:

- Inserir cinco registros sem especificar as colunas das tabelas
- Inserir cinco registros determinando colunas específicas das tabelas. Para cada inserção, altere as colunas especificadas.

Atualização de dados

O comando **UPDATE** é responsável por atualizar dados já armazenados em uma tabela do banco de dados. Com ele, é possível atualizar um único registro e alterar múltiplas informações de uma vez.

Esse comando tem uma das funções mais importantes da linguagem SQL: manter os dados atualizados. Porém, por manipular diretamente dados já existentes no banco de dados, o uso incorreto desse comando pode resultar em perdas de dados significativos. Por isso, é preciso muito cuidado na execução desse comando.

Comando UPDATE

A instrução **UPDATE** contém a seguinte sintaxe:

```
UPDATE nome_da_tabela;

SET
coluna1 = valor1,
coluna2 = valor2,
...
WHERE condição;
```

A princípio, a sintaxe do comando pode parecer complexa, por isso, observe o detalhamento de cada instrução presente no comando.

Comece com o comando **UPDATE** e, em seguida, informe o nome da tabela que deseja atualizar. O argumento **SET** é responsável por especificar as colunas da tabela que você quer alterar, assim como os novos valores que serão inseridos nas colunas indicadas. Não é necessário (e nem se deve) repetir o **SET** para cada coluna nova

apontada. Portanto, após o argumento **SET**, informe a coluna que deseja atualizar seguida do valor. Caso queira atualizar mais valores, basta utilizar a vírgula e repetir esse comportamento até inserir todas as colunas e valores que serão atualizados. Por fim, utilize a cláusula **WHERE** para indicar qual linha deseja atualizar. Essa ação é opcional. **Se não for fornecida nenhuma condição, todos os registros da tabela serão atualizados**. Também é possível utilizar os operadores **AND** e **OR** junto à cláusula **WHERE** para combinar outras condições e realizar uma filtragem mais refinada na tabela.

Veja na prática como utilizar o comando **UPDATE**. Comece com a tabela “aluno”, que contém atualmente os seguintes registros:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	NULL
3	Vitória	Vitória	1997-02-15	NULL
4	Alex	NULL	NULL	1

Atualizar uma única coluna

Nesse primeiro exemplo, será atualizada uma única coluna de um único registro.

```
UPDATE aluno SET curso_id = 2 WHERE id = 2;

SELECT * FROM aluno;
```

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Vitória	1997-02-15	NULL
4	Alex	NULL	NULL	1

Note a presença do item **WHERE** na cláusula **UPDATE**. Neste caso, ele indica que a atualização ocorrerá apenas no registro da tabela “aluno”, que contém “id” igual a 2 – ou seja, apenas o registro com “id 2” e nome “Eduardo” terá sua coluna “curso_id” alterada para o valor 2. O mais usual é, de fato, usar o **WHERE** no **UPDATE** para definir o “id” da linha a ser alterada; no entanto, essas condições podem ser variadas – seria possível, por exemplo, solicitar alteração em todos os registros cujo nome iniciem com a letra “A”, ou em todos os alunos nascidos após 1998. As cláusulas ficariam como a seguir, respectivamente:

```
UPDATE aluno
SET curso_id = 2
WHERE nome LIKE 'A%';
```

```
UPDATE aluno
SET curso_id = 2
WHERE data_nascimento >= '1998-01-01';
```

Mais detalhes sobre **WHERE** você pode conferir no conteúdo sobra a consulta de dados, desta unidade curricular

Atualizar várias colunas

Você pode atualizar várias colunas do mesmo registro na mesma tabela com uma única instrução **UPDATE**. Para isso, separe cada **coluna = valor** com uma vírgula.

```
UPDATE aluno SET
sobrenome = 'Ferreira',
data_nascimento = '1983-09-15'
WHERE id = 4;

SELECT * FROM aluno;
```

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
1	Alice	Gonçalves	1998-10-28	NULL
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Vitória	1997-02-15	NULL
4	Alex	Ferreira	1983-09-15	1

Atualizar todos os registros

Como a cláusula **WHERE** é opcional, você pode executar uma instrução **UPDATE** sem ela. Mas isso atualizará todos os registros da tabela. Dificilmente você encontrará uma situação real em que realmente queira fazer isso, então tome cuidado com esse comando, pois ele pode causar danos irreversíveis à integridade dos dados.

Para esse exemplo, será utilizada a tabela “curso” para atualizar a descrição de todos os cursos cadastrados.

```
UPDATE curso SET descricao = 'Em breve...';

SELECT * FROM curso;
```

Se você tentar executar esse comando, o MySQL Workbench retornará o seguinte erro:

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

Traduzindo para o português, a mensagem declara: “Você está usando um modo de atualização segura e você tentou atualizar a tabela sem um WHERE que use a coluna chave. Para desabilitar o modo seguro, altere a opção em Preferências -> SQL Editor e reconecte”.

Essa mensagem aparece porque o MySQL Workbench instalado já veio pré-configurado com um modo de segurança que impede a execução de comandos **UPDATE** sem a cláusula **WHERE** para evitar danos ao banco de dados. Se você quiser, poderá desabilitar esse modo de segurança em **Edit > Preferences > SQL Editor**, desmarcar a caixa **Safe Updates** e reconectar no Workbench.

Como o comando falhou, é preciso adicionar uma cláusula **WHERE** para atualizar a descrição dos cursos que contêm o “id” 2, 3 e 4. Assim, todos os cursos que ainda não têm uma descrição passarão a ter o valor “Em breve...”. Nesse caso, utilize o operador **OR**, pois é necessária uma condição verdadeira para cada registro que será atualizado.

```
UPDATE curso SET descricao = 'Em breve...' WHERE id = 2 OR id = 3 OR id = 4;
SELECT * FROM curso;
```

Resultado:

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...
4	Técnico em Contabilidade	Em breve...
5	Técnico em Design de Interiores	NULL

Se você utilizasse o operador **AND**, os registros afetados seriam aqueles com as três condições verdadeiras, ou seja, nenhum, pois nenhum registro possui o “id” com valor 2, 3 e 4 simultaneamente.

Antes de seguir para o conteúdo sobre exclusão, você precisa atualizar os registros que inseriu no desafio anterior nas tabelas “curso” e “aluno”. Para isso, em ambas as tabelas, faça o seguinte:

- Atualize a coluna descrição de cada curso inserido com apenas um comando **INSERT**.
- Transfira todos os alunos do curso de Administração (id 3) para o curso de Desenvolvimento de Sistemas (id 1).
- Atualize o sobrenome de três alunos para “Ramos” e também atualize o nome de dois desses alunos para “Luis”.
- Verifique qual curso contém o maior número de alunos e qual contém o menor. Anote o “id” deles e depois transfira os alunos com “id” maior que 6 e o “curso_id” igual ao código do curso verificado para o curso que tiver a menor quantidade de alunos.

Exclusão de dados

Agora que você já estudou sobre os comandos de inserção e atualização de dados, conheça o comando responsável pela remoção de registros da base de dados: o **DELETE**. Assim como os comandos **INSERT** e **UPDATE**, esse contém suas características próprias na sintaxe, o que será melhor explicado em seguida.

Comando DELETE

A instrução **SQL DELETE**, ou consulta de exclusão, é uma instrução que você pode executar para excluir registros de uma tabela.

DELETE FROM nome_da_tabela WHERE condição;

A sintaxe do comando começa com **DELETE FROM**. Aqui, não há necessidade de especificar as colunas, já que a ação se refere à exclusão de linhas. Portanto, as colunas não importam nesse contexto.

Uma cláusula **WHERE** é usada para especificar os critérios, e quaisquer linhas que correspondam a esses critérios serão excluídas. Você pode empregar este comando para excluir um único registro, vários registros ou todos os registros de uma tabela.

A condição **WHERE** permite que você especifique quais linhas deseja excluir. Se você quiser excluir apenas algumas linhas de uma tabela, precisará, obrigatoriamente, da condição **WHERE**. Caso você deseje excluir todas as linhas de uma tabela, não é necessário usar a condição **WHERE**. Fique atento ao fato de que essa ação não pode ser desfeita, mas dificilmente você se encontrará em um contexto no qual seja necessário excluir todos os dados de uma tabela.

Observe alguns exemplos:

Clique ou toque para visualizar o conteúdo.

Exclusão simples

Este é um exemplo de exclusão de um único registro:

```
DELETE FROM curso WHERE id = 5;
SELECT * FROM curso;
```

Aqui, o registro que tem o “id” 5 é excluído e todas as outras linhas permanecem na tabela.

Resultado:

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...
4	Técnico em Contabilidade	Em breve...

Exclusão utilizando duas condições

Este é um exemplo que utiliza duas condições na cláusula **WHERE** com a cláusula **AND**.

DELETE FROM aluno WHERE nome = 'Alice' AND id = 1; SELECT * FROM aluno;

Nesse exemplo, haverá apenas uma linha excluída, pois apenas uma linha corresponde às duas condições da cláusula **WHERE**.

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Prestes	1997-02-15	NULL
4	Alex	Ferreira	1983-09-15	1

Este é um exemplo que usa duas condições na cláusula **WHERE** com a cláusula **OR**.

DELETE FROM aluno WHERE nome = 'Alice' OR id = 1;
SELECT * FROM aluno;

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Prestes	1997-02-15	NULL
4	Alex	Ferreira	1983-09-15	1

Nesse exemplo, perceba que, mesmo não havendo nenhum registro afetado, a execução ainda é realizada com sucesso. Esse cenário – o de executar comandos sem afetar os registros ou as tabelas – costuma acontecer com mais frequência em consultas com o comando **SELECT**.

Exclusão com chave estrangeira

Agora, as tabelas nas quais você está trabalhando têm os seguintes registros:

Tabela "curso":

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...
4	Técnico em Contabilidade	Em breve...

Tabela "aluno":

id	nome	sobrenome	data_nascimento	curso_id
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Prestes	1997-02-15	NULL
4	Alex	Ferreira	1983-09-15	1

Se você quiser remover um registro da tabela “curso”, que esteja sendo referenciado por um ou mais registros da tabela “aluno”, como, por exemplo, o curso com “id” 1, utilize o seguinte comando:

```
DELETE FROM curso WHERE id = 1;
```

Porém, ao tentar executar esse comando, a SQL retornará o seguinte erro:

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`senac_ead`.`aluno`, CONSTRAINT `aluno_ibfk_1` FOREIGN KEY (`curso_id`) REFERENCES `curso` (`id`))

Traduzindo para o português, a mensagem declara: “Não é possível excluir ou atualizar um registro pai: uma restrição de chave estrangeira falhou”.

Isso ocorre porque o SQL não consegue excluir registros relacionados. O mesmo erro ocorreria caso você tentasse atualizar o “id” do curso enquanto ele estiver sendo referenciado por outro registro. Esse tratamento garante a integridade dos dados, mas cria uma barreira na exclusão de dados.

Para que você possa realizar a exclusão nesse cenário, é necessário alterar a coluna “curso_id” da tabela “aluno” e remover a referência ao “id” da coluna “id” da tabela “curso”. Porém, uma vez quebrada essa referência, não será fácil ligar as colunas novamente sem ter que passar por cada um dos registros e alterar os que estiverem quebrando as regras do relacionado. Observe o exemplo do seguinte cenário:

- Você quebra o vínculo de “curso_id” da tabela “aluno” com “id” da tabela “curso”
- Cadastra novos alunos
- Alguns dos alunos cadastrados tiveram o valor 32 informado na coluna “curso_id”
- Você tenta ligar as colunas “curso_id” e “id” novamente
- A tabela curso não contém nenhum registro com “id 32”. Logo, o relacionamento não pode ser feito por inconsistência de dados entre uma tabela e outra.

Outra abordagem possível para excluir registros referenciados por chave estrangeira é a exclusão em cascata, na qual a exclusão de um registro pai levará à exclusão dos registros filhos. Em outras palavras, caso fosse excluído o curso de “id 1”, seriam excluídos todos os alunos vinculados a ele também. Por ser uma abordagem que põe em risco a exclusão de dados sensíveis, a exclusão em cascata precisa ser habilitada ao realizar a referência na chave estrangeira por meio da sintaxe **ON DELETE CASCADE**. Porém, isso só é possível na criação da coluna. Caso você queira criar uma tabela com essa configuração, deverá executar o seguinte comando:

```
CREATE TABLE aluno2 (
id int AUTO_INCREMENT,
nome varchar(30),
sobrenome varchar(30),
data_nascimento date,
curso_id int,
PRIMARY KEY (id),
FOREIGN KEY (curso_id) REFERENCES curso(id) ON DELETE CASCADE
);
```

Execute o comando apresentado para criar a tabela “aluno2” com a exclusão em cascata habilitada. Em seguida, insira um novo curso na tabela “curso” e adicione dois novos alunos na tabela “aluno2”:

INSERT INTO curso (id, nome) VALUES (12, 'Técnico em Informática');
INSERT INTO aluno2 (nome, curso_id) VALUES ('Daltron', 12), ('Wilson', 12);

SELECT * FROM curso;
SELECT * FROM aluno2;

Ao final, você terá as seguintes tabelas:

Tabela "curso:"

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...
4	Técnico em Contabilidade	Em breve...
12	Técnico em Informática	NULL

Tabela "aluno2":

id	nome	sobrenome	data_nascimento	curso_id
1	Daltron	NULL	NULL	12
2	Wilson	NULL	NULL	12

Agora, se você executar o comando **DELETE** para remover o curso com “id” 12, tanto o curso Técnico em Informática quanto os alunos “Daltron” e “Wilson” serão excluídos.

```
DELETE FROM curso WHERE id = 12;
SELECT * FROM curso;
SELECT * FROM aluno2;
```

Resultado:

Tabela "curso:"

id	nome	descricao
1	Técnico em Desenvolvimento de Sistemas	Prepara o aluno para exercer funções técnicas, como: administração do sistema de banco de dados, desenvolver software, programar rotinas de sistema utilizando linguagens e técnicas de programação, programação de sistemas para desktop e para web, entre outras atividades.
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...
4	Técnico em Contabilidade	Em breve...

Tabela "aluno2":

id	nome	sobrenome	data_nascimento	curso_id

Apesar de muito eficiente, o uso da exclusão em cascata pode comprometer a exclusão de dados muito sensíveis, já que todos os registros que estiverem referenciando com chave estrangeira o registro a ser excluído também serão apagados. Portanto, ainda falando da exclusão de registros referenciados por chaves estrangeiras, existem outras abordagens para esse cenário.

A primeira opção seria atualizar os valores de “curso_id” para **NULL** onde o “curso_id” for igual a 1. Com isso, você não teria mais nenhum registro vinculado ao registro de “id” 1 da tabela curso. Assim, você conseguiria excluir o registro utilizando o mesmo comando mencionado anteriormente. Na prática, você teria o seguinte.

Primeiro, atualize o valor de “curso_id” na tabela aluno:

```
UPDATE aluno SET curso_id = NULL WHERE curso_id = 1;
SELECT * FROM aluno;
```

Resultado:

id	nome	sobrenome	data_nascimento	curso_id
2	Eduardo	Machado	1998-07-16	2
3	Vitória	Prestes	1997-02-15	NULL
4	Alex	Ferreira	1983-09-15	NULL

Em seguida, faça a exclusão do curso com “id” 1:

```
DELETE FROM curso WHERE id = 1;
SELECT * FROM curso;
```

Resultado:

id	nome	descricao
2	Técnico em Informática para Internet	Em breve...
3	Técnico em Administração	Em breve...

Apesar de mais trabalhosa, nessa abordagem, os dados dos alunos são preservados no banco de dados. Porém, os dados do curso Técnico em Desenvolvimento de Sistemas não existem mais e não há qualquer vestígio desses registros no banco de dados. De uma perspectiva didática, isso é muito bom, já que, com essas exclusões, é possível aprender as principais formas de remover registros de

uma tabela em diversos cenários diferentes.

Mas e se você estivesse em uma situação real no mercado de trabalho?

Com a alteração dos dados na coluna “curso_id” para **NULL**, não se sabe mais a qual curso os alunos estavam vinculados, e com a exclusão do curso com “id” 1, também não se sabe mais quais cursos foram oferecidos pela instituição de ensino. Sem essas informações, perde-se o histórico completo do curso Técnico em Desenvolvimento de Sistemas e dos alunos que já cursaram ou estavam cursando esse curso.

Este é apenas um exemplo hipotético, mas, em uma empresa real, a preservação dos dados pode ser a diferença entre o sucesso e a falência da companhia. Então, é esperado que se saiba preservar os registros de um banco de dados exatamente como eles são: um dos bens mais valiosos para uma empresa.

Agora, conheça uma alternativa à exclusão de registro de uma tabela. Para isso, será preciso fazer um ajuste na estrutura das suas tabelas. Portanto, não deixe de executar os comandos a seguir antes de continuar.

```
ALTER TABLE aluno ADD COLUMN status char(1) DEFAULT 'A';
ALTER TABLE curso ADD COLUMN status char(1) DEFAULT 'A';
```

Preservando registro no banco de dados

Para preservar os registros em banco de dados, deve-se evitar ao máximo a exclusão dos dados, o que significa evitar o uso do comando **DELETE**. Para isso, será comum você criar uma coluna “*status*” na estrutura das tabelas para indicar a situação

daquele registro. A coluna pode estar vinculada a uma outra tabela com todos os *status* cadastrados ou pode ser uma coluna com o *status* completo escrito por extenso (por exemplo: “Ativo”, “Inativo”). Para fins didáticos, utilize o tipo de dado **CHAR(1)** para ocupar apenas um caractere no banco de dados (o que poupará bastante espaço em disco conforme o número de registros aumentar) e definir seu valor padrão como “A” (sigla para “Ativo”). Quando quiser tornar um registro inativo, atualize o *status* para “I” (sigla para “Inativo”).

Se você já executou os comandos para adicionar a coluna “*status*”, então suas tabelas devem estar da seguinte maneira.

Tabela “curso”:

id	nome	descricao	status
2	Técnico em Informática para Internet	Em breve...	A
3	Técnico em Administração	Em breve...	A

Tabela aluno:

id	nome	sobrenome	data_nascimento	curso_id	status
2	Eduardo	Machado	1998-07-16	2	A
3	Vitória	Prestes	1997-02-15	NULL	A
4	Alex	Ferreira	1983-09-15	NULL	A

Perceba que, além de a coluna *“status”* ser adicionada, os valores padrão dessa coluna estão definidos como “A”. Dessa forma, sempre que adicionar um novo registro com o comando **INSERT**, o valor de *status* será definido automaticamente como “A”,

caso outro valor não seja inserido.

Agora, em vez de apagar um registro do banco de dados, atualize o *status* de “Ativo” para “Inativo”:

UPDATE aluno SET status = 'I' WHERE id = 3; SELECT * FROM aluno;

Resultado:

id	nome	sobrenome	data_nascimento	curso_id	status
2	Eduardo	Machado	1998-07-16	2	A
3	Vitória	Prestes	1997-02-15	NULL	I
4	Alex	Ferreira	1983-09-15	NULL	A

Com isso, os registros do banco de dados não serão apagados, e sim preservados para uma situação futura. Se você quiser recuperar apenas os registros ativos, basta adicionar uma cláusula **WHERE** na sua consulta com o **SELECT**.

Assim, você concluiu o conteúdo sobre o comando **DELETE**. Mas antes, de finalizar, que tal um último desafio?

Para o desafio final, coloque em prática tudo que aprendeu e:

- Remova um curso que não tenha nenhum aluno, informando o nome completo do curso.
- Remova o curso que esteja sendo cursado pela menor quantidade de alunos.
- Remova todos os alunos que não estejam vinculados a um curso, utilizando apenas um comando **DELETE**.

Revisando e aplicando

No vídeo a seguir, você poderá expandir seu conhecimento e sua experiência com novos exemplos de inserção, atualização e exclusão de dados com SQL. Acompanhe o passo a passo, pause ou retroceda sempre que necessário.