

**CURSOS
TÉCNICOS**

**DESENVOLVIMENTO DE
SISTEMAS WEB II**

Eixo Informática para Internet

Unidade 5

SUMÁRIO

UNIDADE 5.....	3
1.Introdução a SQL e modelagem de Banco de Dados.....	3
1.1 Conceito de SQL.....	3
1.2 Configurando DBMS para escrever código SQL.....	3
2.Comandos SQL	7
2.1 Criando uma tabela.....	7
2.2 Inserindo dados em uma tabela.....	10
2.3 Recuperando registros de uma tabela	11
3.Comandos de funções Agregadas SQL.....	13
3.1 Busca condicional de strings com coringas SQL.....	14
3.2 Atualizando campos de uma tabela.....	15
3.3 Excluindo registros de uma tabela.....	17
3.4 Alterando tabelas.....	17
4.Relacionamento e integridade de tabelas.....	19
5.Diagramas de entidade e relacionamento (ER).....	21
6.Fixando conhecimento: Desafios de SQL.....	22
7. Referências.....	25

UNIDADE 5

1. Introdução a SQL e modelagem de Banco de Dados

Nesta unidade, o foco central é basicamente a modelagem de banco de dados.

1.1 Conceito de SQL

SQL (Structured Query Language) é uma linguagem de programação usada para gerenciar bancos de dados relacionais. Foi desenvolvida para permitir que os usuários consultem, atualizem, gerenciem e manipulem dados em sistemas de gerenciamento de banco de dados (DBMS) relacionais, como MySQL, PostgreSQL, Microsoft SQL Server, Oracle, e muitos outros.

Através do SQL podemos executar uma série de operações em um banco de dados, incluindo:

- Definição de esquemas e relacionamento de tabelas.
- Inserção de dados;
- Consulta de dados;
- Atualização de valores de dados;
- Exclusão de dados;

Vamos analisar o funcionamento de cada uma dessas operações, mais adiante.

1.2 Configurando DBMS para escrever código SQL

Primeiro precisamos escolher qual Sistema de gerenciamento de banco de dados vamos utilizar. No nosso caso, vamos utilizar o MySQL por ter um workbench gratuito e de fácil instalação.

Instalação passo-a-passo:

1. Acesse o artigo de instalação da Alura:

→ [Alura - MySQL: da instalação até a configuração](#)

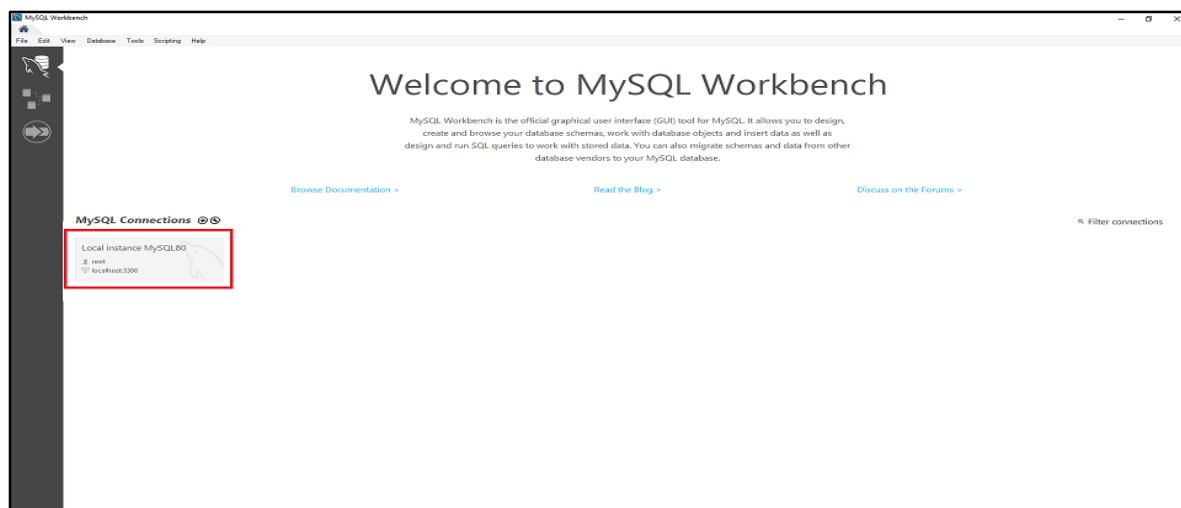
*1.1 Caso você prefira, há um vídeo da hostinger sobre a mesma instalação também

→ [Como Instalar MYSQL 📌 WorkBench e Server 📌 \(2023\)](#)

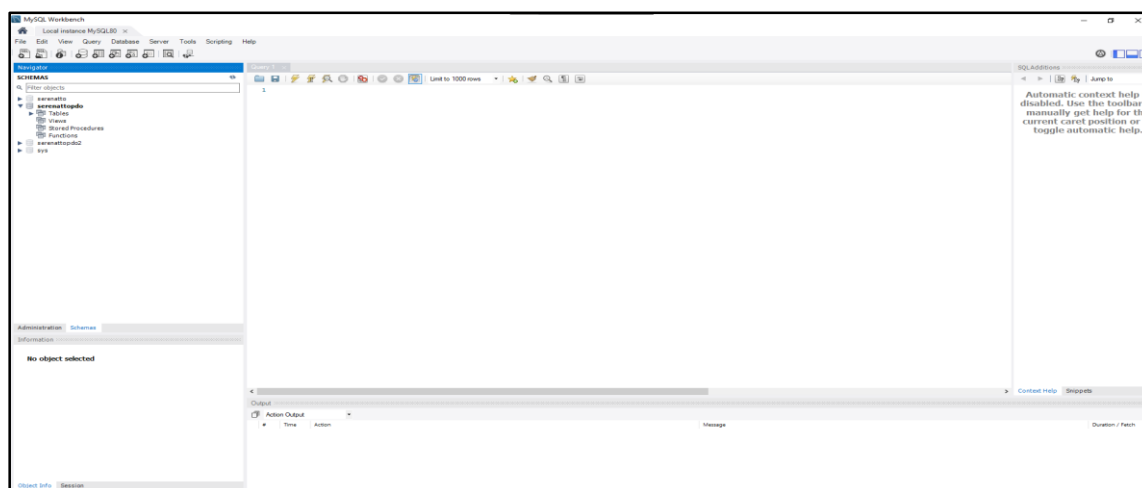
Siga o passo-a-passo de acordo com seu sistema operacional.

* Você estará baixando basicamente uma interface gráfica gratuita do MySQL Workbench para escrever códigos SQL.

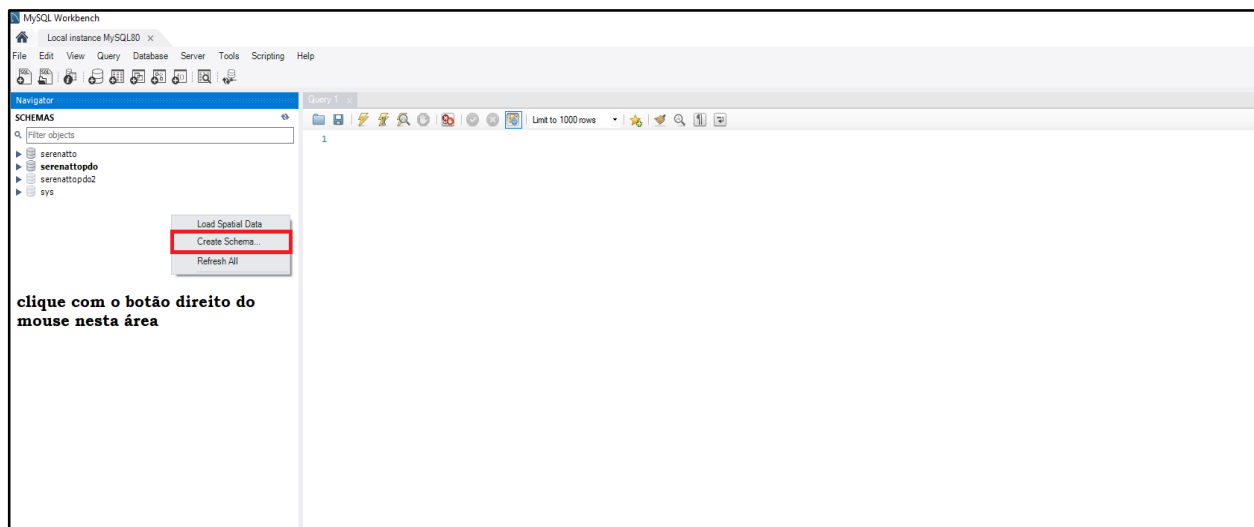
2. Uma vez que o processo de instalação funcionou você poderá abrir o workbench e já terá um server no localhost (ou seja, um servidor local) e uma porta para testar o seu banco de dados.



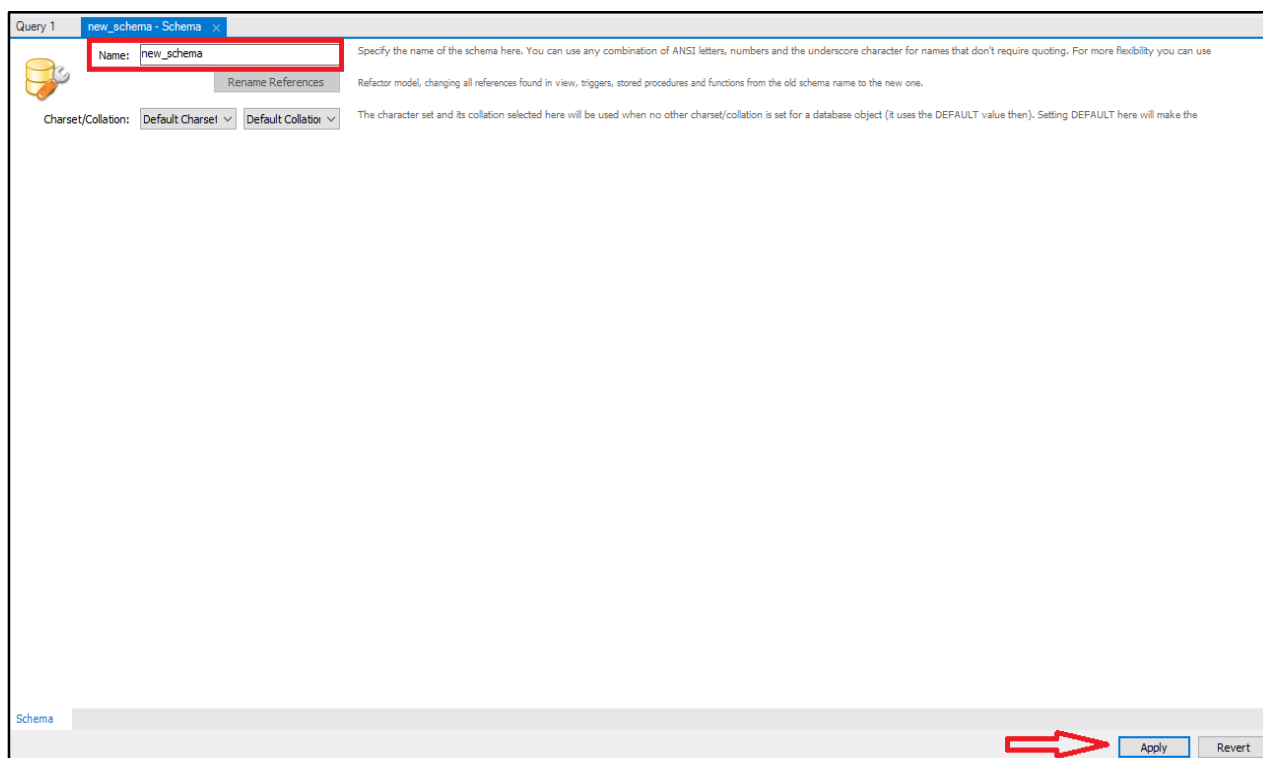
3. Clique na sua conexão, digitando a sua senha.
4. Uma vez que as credenciais forem validadas, você entra na conexão e consegue visualizar seus *schemas*.



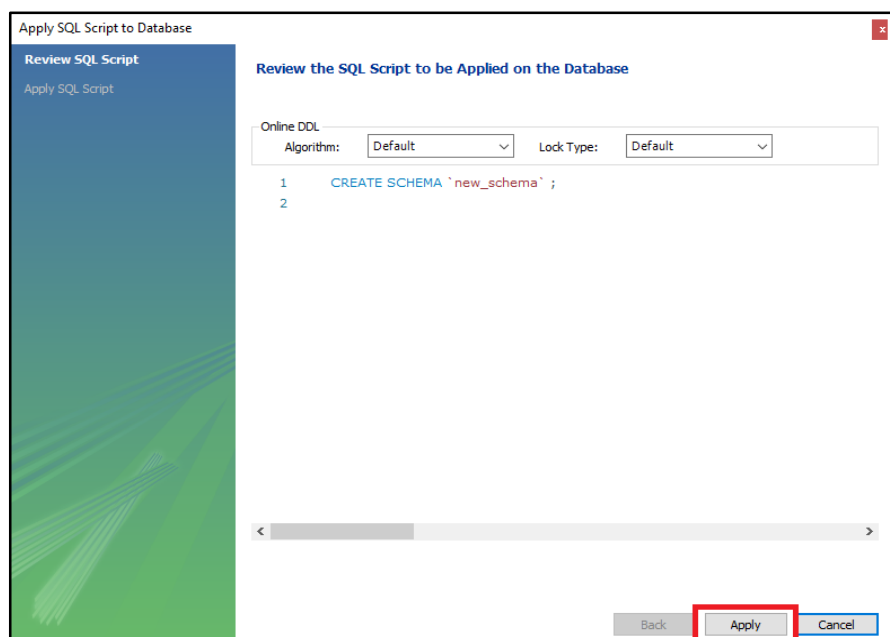
5. Crie um novo **schema** para armazenar as suas tabelas. Você deve clicar com o botão direito do mouse na aba dos schemas, em seguida, clicar em **create schema**.



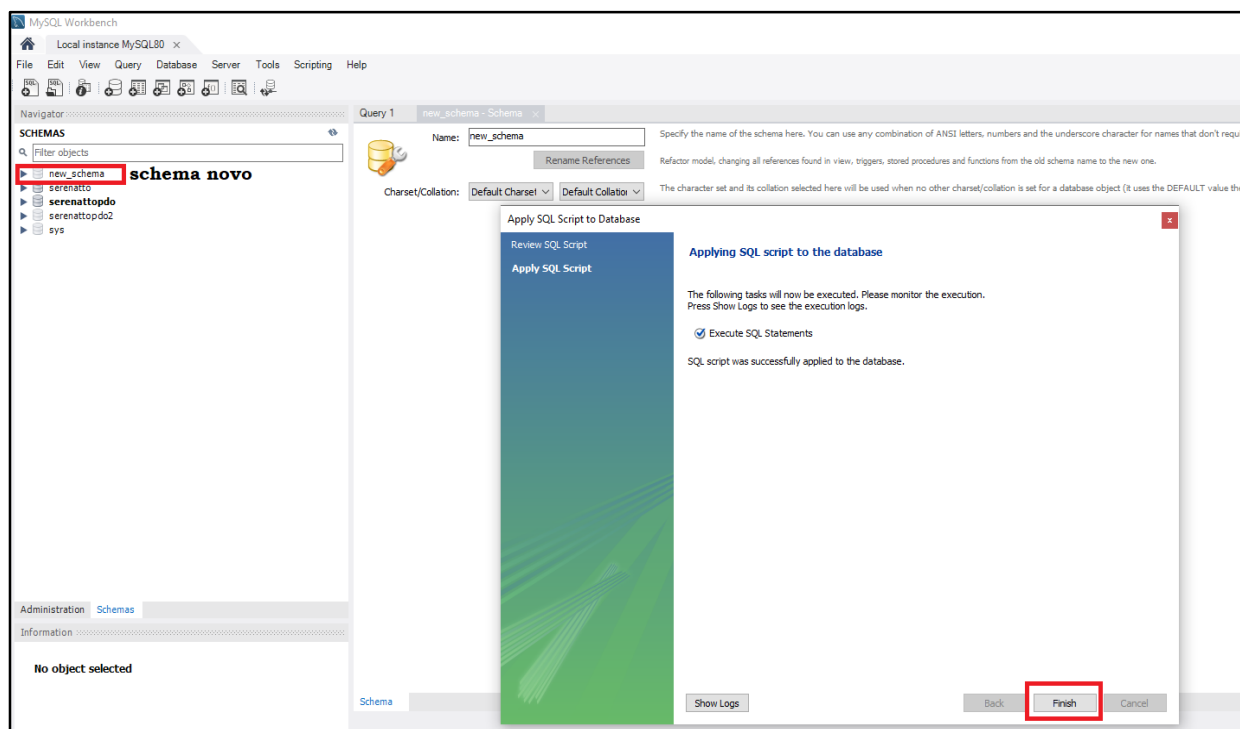
6. Abrirá uma aba, onde você pode dar um nome para schema, e clicar em ***apply***.



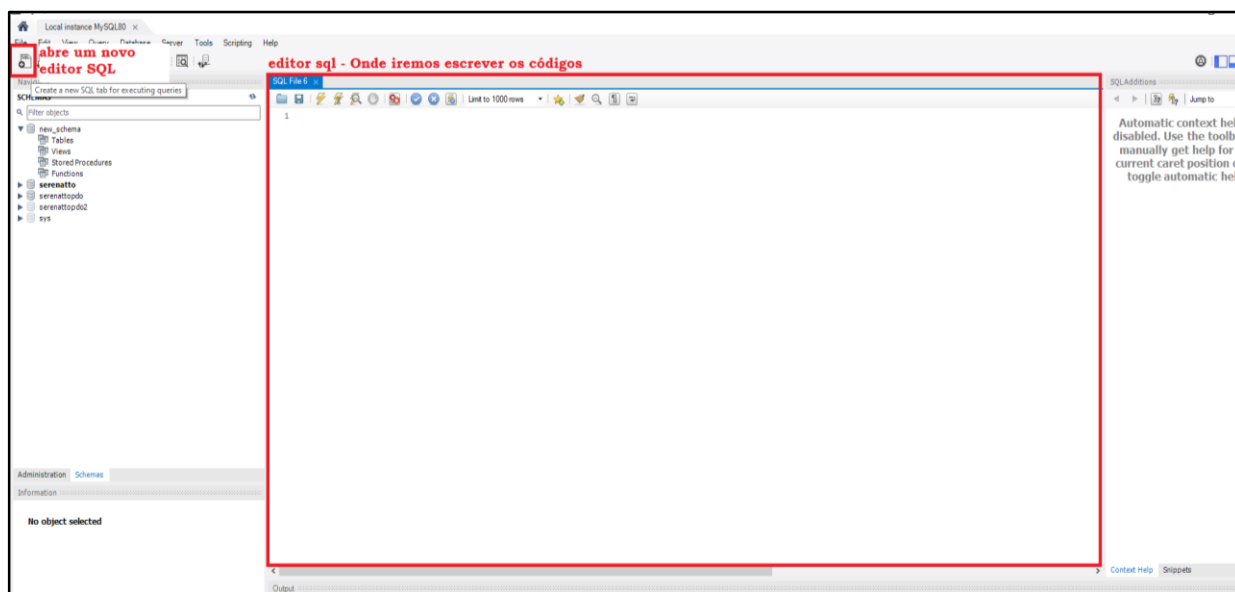
7. Após isso, a ferramenta gera um SQL para criação de um novo schema. Você deve conferir se o nome certo, e após isso, clicar em ***apply***.



8. Com o sucesso nas etapas anteriores, deve gerar um modal de sucesso na criação do schema, e seu schema novo deve estar listado. Clique em ***finish***.



9. Para abrir um editor SQL você deve **clicar no ícone indicado**. Ao clicar no ícone, é aberto um novo editor SQL em branco para que você possa escrever códigos SQL.



* É possível salvar script SQL, abrir arquivos de scripts existentes ou criar novos como fizemos agora.

2. Comandos SQL

Assim como as linguagens de programação, o SQL segue um padrão (sintaxe) para a escrita de códigos para a manipulação dos bancos de dados. Vamos explorar alguns comandos de manipulação de tabelas, captação, alteração e deleção de registros.

2.1 Criando uma tabela

Para criar uma nova tabela usamos a cláusula create table no seguinte formato:

```
CREATE TABLE <schema.>nome_tabela(
    nome_coluna1 <tipo_do_valor_coluna1> <constraints>,
    nome_coluna2 <tipo_do_valor_coluna2> <constraints>
);
```

É importante destacar que, cada coluna deve ter seu **tipo de dados especificado** e também **conjunto de regras (constraints)**, se necessário.

Tipos de dados de colunas:

- **Integer** → Armazena números inteiros, como 1, 2, -3, 0, etc.
- **Float** → Armazena números de ponto flutuante (decimais) com precisão limitada.
- **Numeric(p, s)** → Armazena números decimais com precisão arbitrária, onde "p" é o número total de dígitos e "s" é o número de casas decimais.

- **CHAR(n)** → Armazena strings de tamanho fixo, onde "n" é o número de caracteres.
- **Varchar(n)** → Armazena strings de tamanho variável, onde "n" é o número máximo de caracteres.
- **Date** → Armazena uma data no formato 'AAAA-MM-DD'.
- **Timestamp** → Armazena data e hora no formato 'AAAA-MM-DD HH:MM:SS'. Muito usado para armazenar horários de registro de atividades para gerar logs.
- **Boolean** → Armazena valores booleanos, como TRUE ou FALSE.
- **BLOB ou BINARY LARGE OBJECT** → Armazena dados binários, como imagens ou arquivos.

* Você deve selecionar **o tipo de dado mais adequado de acordo com o valor esperado na sua coluna**. Existem outros tipos disponíveis, estes são alguns deles para criação de tabelas.

Tipos de constraints de colunas:

- *** Primary key** → Uma coluna que identifica exclusivamente cada registro em uma tabela. **Mais informações nos próximos índices.**
- *** Foreign key** → Uma coluna que estabelece uma relação com a coluna de chave primária de outra tabela, garantindo integridade referencial. **Mais informações nos próximos índices.**
- **Not null** → Garante que os valores em uma coluna não podem ser nulos.
- **Unique** → Garante que os valores em uma coluna sejam únicos, mas permite valores nulos.
- **Default** → Especifica um valor padrão para uma coluna se nenhum valor for fornecido.
- **Check** → Define uma condição que os valores em uma coluna devem atender. Se a condição não for satisfeita, a inserção ou atualização dos dados é impedida.

Exemplo - Criação da tabela tb_pessoa com colunas de dados simples de uma pessoa:

```
SQL: File 6* x
1 • CREATE TABLE new_schema.tb_pessoa(
2     id_pessoa integer not null primary key,
3     nome_pessoa varchar(50) not null,
4     data_nascimento DATE not null
5 );
6
7
```


→ **Como executar os comandos SQL no MySql workbench?**

SQL File 6 - 2 - Clique no ícone de raio para executar

```

1 • CREATE TABLE new_schema.tb_pessoa(
2     id_pessoa integer not null primary key,
3     nome_pessoa varchar(50) not null,
4     data_nascimento DATE null
5 );
6
7

```

1 - selecione o comando a ser executado

Output

#	Time	Action	Message
1	13:38:16	CREATE TABLE new_schema.tb_pessoa(id_pessoa integer not null primary key, nome_pessoa varchar(50) not null, data_nasci...	0 row(s) affected

3 - Confira se o comando teve sucesso ou não. Nesse caso, teve.

→ **É possível verificar pela interface gráfica também, se a tabela foi criada.**

Navigator

SCHMAS

- new_schema
 - tb_pessoa <-- clique

SQL de criação da tabela tb_pessoa

```

1 • CREATE TABLE new_schema.tb_pessoa(
2     id_pessoa integer not null primary key,
3     nome_pessoa varchar(50) not null,
4     data_nascimento DATE not null
5 );
6
7

```

Informação da tabela

Table: tb_pessoa

Columns	
id_pessoa	int PK
nome_pessoa	varchar(50)
data_nascimento	date

2.2 Inserindo dados em uma tabela

Para inserir registros em uma determinada tabela utilizamos o comando

INSERT INTO.

-- Insere todos campos de acordo com a ordem da tabela.

INSERT INTO <schema.>nome_da_tabela **VALUES**(valor1, valor2, valor3);

-- Insere somente valores nas colunas especificadas entre parênteses.

INSERT INTO <schema.>nome_tabela (coluna1, coluna2) **VALUES**(valor1, valor2);

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Schemas' tree shows 'new_schema' expanded, with 'tb_pessoa' selected under 'Tables'. A red box highlights this selection, with the text 'clique' next to it. In the center, the SQL script editor shows the following code:

```
1 CREATE TABLE new_schema.tb_pessoa(
2     id_pessoa integer not null primary key,
3     nome_pessoa varchar(50) not null,
4     data_nascimento DATE not null
5 );
6
7
```

A red box highlights the SQL code, with a red annotation to its right: 'SQL de criação da tabela tb_pessoa'. At the bottom, the 'Information' tab for 'Table: tb_pessoa' is shown, listing the columns: 'id_pessoa' (int PK), 'nome_pessoa' (varchar(50)), and 'data_nascimento' (date). A red box highlights this information.

Exemplo - Inserção de dados na tabela tb_pessoa.

```

1 • INSERT INTO new_schema.tb_pessoa VALUES(
2     1,
3     'João',
4     '2000-01-02'
5 );

```

Output

Action Output **mensagem de sucesso**

#	Time	Action	Message
1	13:47:57	INSERT INTO new_schema.tb_pessoa VALUES(1, 'João', '2000-01-02')	1 row(s) affected

Exemplo - Inserção de dados nulos em coluna que não pode ser nula.

```

1 • INSERT INTO new_schema.tb_pessoa VALUES(
2     2,
3     null, ————— coluna nm_pessoa recebendo valor nulo
4     '2000-01-02'
5 );

```

Output

Action Output **mensagem de erro**

#	Time	Action	Message
1	13:50:23	INSERT INTO new_schema.tb_pessoa VALUES(2, null, '2000-01-02')	Error Code: 1048. Column 'nm_pessoa' cannot be null

Neste caso, está sendo atribuído um **valor nulo** na coluna **nm_pessoa** da tabela **tb_pessoa**. A mensagem de erro exibida diz que: **“A coluna ‘nm_pessoa’ não pode ser nula”**.

*** Neste caso, o valor não foi inserido na tabela, pois o comando SQL falhou por conta de uma violação em uma regra da coluna.**

2.3 Recuperando registros de uma tabela

Em muitos casos, precisamos **recuperar dados de uma tabela** como:

- quais registros existem?
- quantos registros existem?
- quantas pessoas na tabela de pessoas são maiores de idade? (com condições específicas de busca).

Para isso, utilizamos o comando: **SELECT**.

-- Busca pelas colunas identificadas na tabela atribuída.

SELECT <colunas> **FROM** <nome_tabela>;

-- Busca por TODAS colunas identificadas na tabela atribuída. * = todos.

SELECT * **FROM** <nome_tabela>;

-- Busca condicional: **SELECT FROM + WHERE** com uma condição.

SELECT * **FROM** <nome_tabela> **WHERE** <condição>;

Exemplo - Buscando por todos os registros cadastrados na tb_pessoa.

```

1 • INSERT INTO new_schema.tb_pessoa VALUES(
2     2,
3     'João',
4     '2000-01-02'
5 );
6
7 • INSERT INTO new_schema.tb_pessoa VALUES(
8     2,
9     'Pedro',
10    '2000-01-02'
11 );
12
13 -- Selecciona todas colunas
14 • select * from new_schema.tb_pessoa;

```

comentário em sql

Query SQL/Comando de busca

id_pessoa	nome_pessoa	data_nascimento
1	João	2000-01-02
2	Pedro	2000-01-02

Resultados encontrados na tabela

O comando retorna todas as colunas e todos os registros inseridos na tabela tb_pessoa.

Utilizamos **schema.nome_tabela**, pois neste servidor, há mais de uma tabela cadastrada, então neste caso, precisamos identificar de qual *schema* está sendo feita a busca.

Exemplo - Buscar somente os nomes das pessoas na tb_pessoa.

```

12
13 -- Selecciona somente a coluna nome dos registros
14 • select nome_pessoa from new_schema.tb_pessoa;
15

```


nome_pessoa
João
Pedro

Exemplo - Buscar somente pela pessoa com id = 1 na tb_pessoa.

```

7  -- Seleciona somente a pessoa com id = 1.
8 • SELECT nome_pessoa FROM new_schema.tb_pessoa WHERE id_pessoa = 1;
9

```



nome_pessoa
João

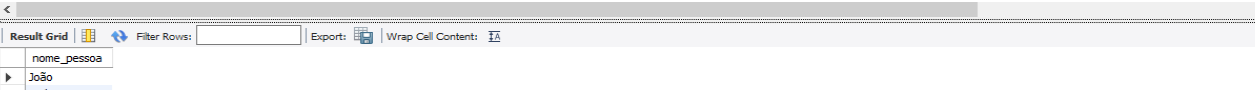
Note que, após a declaração da cláusula **WHERE** atribuímos a condição: valor na coluna id_pessoa deve ser igual a 1, ou seja id_pessoa = 1. Com isso, somente os registros que corresponderem a esta condição serão retornados.

Exemplo - Retorna o nome de todas pessoas maiores de idade (idade >= 18) na tb_pessoa considerando o **ano atual E ano da data de nascimento** da pessoa.

```

1  -- Inserção de menor de idade para teste.
2 • INSERT INTO new_schema.tb_pessoa VALUES(
3      3,
4      'Lucas',
5      '2013-01-02'
6  );
7
8  -- Seleciona somente o nome das pessoas cujo cálculo de idade seja maior ou igual a 18.
9 • SELECT nome_pessoa FROM new_schema.tb_pessoa
10 WHERE (EXTRACT(year from CURRENT_DATE) - EXTRACT(year from data_nascimento)) >= 18;
11

```



nome_pessoa
João
Pedro

Utilizamos o comando **EXTRACT**, pois iríamos considerar somente o ano de uma data. O comando basicamente pega uma data e extrai somente a parte especificada:

Ex: '2013-01-02' → '2013'

No comando acima somente Lucas não foi retornado, pois tanto João quanto Pedro possuem mais de 18 anos. Considerando somente o ano Lucas teria 10 anos apenas, pois: 2023 - 2013 = 10.

No exemplo acima, **CURRENT_DATE** é uma das funções agregadas do SQL que retorna a data atual.

3. Comandos de funções Agregadas SQL

Funções agregadas, também conhecidas como funções de agregação, são funções em SQL que operam em conjuntos de dados para retornar um único valor resumido.

- **Count()** → Retorna o número de linhas

– Se “*” fornecido em count conta todos os registros.

SELECT count(campo ou *) FROM tb_pessoa; --Retorna o número de registros encontrados.

- **MAX()** → Retorna o maior valor da coluna

SELECT max(id_pessoa) FROM tb_pessoa; --Retorna o maior valor do campo.

- **MIN()** → Retorna o menor valor da coluna

SELECT min(id_pessoa) FROM tb_pessoa; --Retorna o menor valor do campo.

- **AVG()** → Retorna o valor médio dos valores da coluna

SELECT avg(id_pessoa) FROM tb_pessoa; --Retorna o valor médio dos campos.

- **LOWER()** → Converte o valor de um campo para minúsculo.

SELECT lower(nome_pessoa) FROM tb_pessoa; --Retorna o campo com seus caracteres todos em minúsculo.

- **UPPER()** → Converte o valor de um campo para maiúsculo.

SELECT upper(nome_pessoa) FROM tb_pessoa; --Retorna o campo com seus caracteres todos em maiúsculo.

Existem outras funções. Estas são só alguns que você pode utilizar em muitos casos.

3.1 Busca condicional de strings com coringas SQL

Em SQL, você pode realizar buscas condicionais de strings com coringas usando a cláusula LIKE e os caracteres curinga “%” e “_”. Aqui está como você pode usar esses coringas:

NOTA: Utilize **sempre ASPAS SIMPLES** (‘ ’) em seus comandos SQL. Utilizar aspas duplas resultará em um ERRO DE SINTAXE.

Coringas:

→ **% (porcentagem):** Representa zero ou mais caracteres em qualquer posição.

Por exemplo: “%ola” corresponderá a “bola”, “escola”, “sacola”, etc.

-- Busca pelo nome das pessoas que iniciem com J e tenha qualquer caractere após.

SELECT nome_pessoa **FROM** tb_pessoa **WHERE** nome_pessoa **LIKE** 'J%';

→ **_(underline)**: Representa um único caractere em qualquer posição.

Por exemplo: 'T_m' corresponderá a "Tom", "Tim", etc. Ou seja, o underline representa que a busca será satisfeita independente do caractere adicionado na segunda posição desta string.

-- Busca pelo nome das pessoas que iniciem T, qualquer caractere na 2a posição e termine com m.

SELECT nome_pessoa **FROM** tb_pessoa **WHERE** nome_pessoa **LIKE** 'T_m';

3.2 Atualizando campos de uma tabela

Em alguns casos precisamos acessar registros existentes dentro de uma tabela e atualizar seus valores. Seja por ter ocorrido algum erro de inserção ou até mesmo uma atualização nas regras de negócio da empresa.

Para atualizar valores de tabelas utilizamos o comando **UPDATE**.

-- Atualiza todos os campos dos registros existentes na tabela com o novo valor.

UPDATE <nome_tabela> **SET** <coluna> = <novo_valor>;

-- Atualiza mais de uma coluna com todos os registros existentes na tabela com os novos valores atribuídos.

UPDATE <nome_tabela> **SET** <coluna1> = <novo_valor1>, <coluna2> = <novo>;

Exemplo - Atualizando todos os nomes na tb_pessoa para maiúsculo. (emite erro)

9 -- Retorna um erro por segurança, porém funciona.

10 • **UPDATE** new_schema.tb_pessoa **SET** nome_pessoa = UPPER(nome_pessoa);

Output
Action Output
Time Action Message
1 15:46:40 UPDATE new_schema.tb_pessoa SET nome_pessoa = UPPER(nome_pessoa) Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column.


Este comando emite um erro, pois estamos tentando atualizar o valor de TODOS os campos de uma tabela sem nenhuma condição. Neste caso o MySQL Workbench sugeriu que utilizássemos a cláusula WHERE para executar o comando.

Exemplo - Atualizando todos os nomes na tb_pessoa para maiúsculo. Com WHERE.

```

9  -- Definimos um intervalo com a clausula WHERE para corrigir.
10 • UPDATE new_schema.tb_pessoa SET nome_pessoa = UPPER(nome_pessoa)
11   WHERE id_pessoa >= 1 AND id_pessoa <= 100;
12
13 • SELECT nome_pessoa from new_schema.tb_pessoa; — lista nomes
14

```



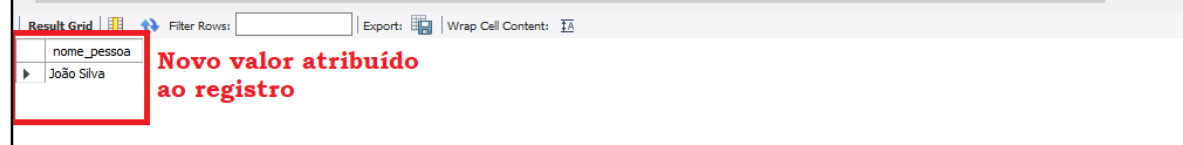
Todos os nomes foram atualizados para maiúsculo agora

Exemplo - Atualizando nome “João” para “João Silva” através do id em tb_pessoa.

```

9  -- Definimos um novo valor para o nome_pessoa do registro com id = 1.
10 • UPDATE new_schema.tb_pessoa SET nome_pessoa = 'João Silva'
11   WHERE id_pessoa = 1;
12
13 • SELECT nome_pessoa from new_schema.tb_pessoa WHERE id_pessoa = 1; lista o registro
14

```



Novo valor atribuído ao registro

NOTA: Comandos que atualizam ou deletam campos de uma tabela devem obter uma atenção maior por parte de quem está desenvolvendo os comandos SQL, pois sem um backup prévio não é possível reverter a ação destes comandos em uma tabela. Isso pode causar prejuízos para uma empresa/cliente caso não seja executado de forma correta.

3.3 Excluindo registros de uma tabela

Eventualmente precisamos remover registros de uma tabela. Para isso utilizamos sempre identificadores únicos dos registros como **ID's**. O comando utilizado para deletar um registro por completo é o **DELETE**.

```
-- Deleta todos os registros de uma tabela.
DELETE FROM <nome_tabela>;

-- Deleta um ou mais registros que atendam a condição.
DELETE FROM <nome_tabela> WHERE <condição>;
```

The screenshot shows a database management interface. At the top, there's a SQL editor with the following code:

```
9
10 -- Deletamos os registro de id = 3.
11 • DELETE FROM new_schema.tb_pessoa WHERE id_pessoa = 3;
12
13 • SELECT * from new_schema.tb_pessoa;
14
```

Below the editor, there's a 'Result Grid' showing the results of the SQL commands. It has two columns: 'id_pessoa' and 'nome_pessoa'. The data is as follows:

id_pessoa	nome_pessoa
1	João Silva
2	PEDRO

A red box highlights the 'Result Grid' and the text 'Registros da tabela agora sem o "Lucas", cujo id = 3.'

At the bottom, there's a 'Comandos executados' (Executed Commands) section. It shows two commands:

Comando	Resultado
7 16:06:57 DELETE FROM new_schema.tb_pessoa WHERE id_pessoa = 3	1 row(s) affected
8 16:06:57 SELECT * from new_schema.tb_pessoa LIMIT 0, 1000	2 row(s) returned

3.4 Alterando tabelas

De acordo com o tempo é possível que seja necessário realizar alguma alteração em uma tabela do nosso banco de dados. Para isso temos inúmeros comandos para manipular estas tabelas e vão desde alterar o nome, adicionar constraints, até adicionar novos campos. O comando principal é: **ALTER TABLE**.

- **Alterando nome da tabela:**

```
ALTER TABLE <nome_tabela> RENAME TO <novo_nome_tabela>;
```

- Adicionando campo novo a uma tabela:

ALTER TABLE <nome_tabela> **ADD** <nome_campo> <tipo_campo>;

Exemplo: Adicionando campo id_profissao na tb_pessoa.

```
15 • ALTER TABLE new_schema.tb_pessoa ADD id_funcao integer;
16 • SELECT * from new_schema.tb_pessoa;
17
```

id_pessoa	nome_pessoa	data_nascimento	id_funcao
1	João Silva	2000-01-02	NULL
2	PEDRO	2000-01-02	NULL
...

Coluna adicionada.

- Remover uma coluna de tabela:

ALTER TABLE <nome_tabela> **DROP COLUMN** <nome_coluna>;

- Mudar o tipo da coluna:

ALTER TABLE <nome_tabela> **ALTER COLUMN** <nome_coluna> **TYPE** <novo_tipo>;

- Adicionar uma constraint a uma tabela

ALTER TABLE <nome_tabela> **ADD CONSTRAINT** <nome_constraint> <constraint>

→ Exemplo de constraint para adicionar PRIMARY KEY

ALTER TABLE <nome_tabela> **ADD CONSTRAINT** <nome_constraint> **PRIMARY KEY**(coluna);

→ Exemplo de constraint para adicionar FOREIGN KEY

ALTER TABLE <nome_tabela> **ADD CONSTRAINT** <nome_constraint> **FOREIGN KEY** (campo) **REFERENCES** nome_tabela(campo);

Existem muitas outras variações do comando **ALTER TABLE**. Estas são apenas algumas que podem ser úteis para quem está iniciando com SQL.

4. *Relacionamento e integridade de tabelas*

Relacionamentos e integridade de tabelas são conceitos importantes em bancos de dados relacionais. Eles ajudam a garantir a consistência dos dados e permitem que informações de diferentes tabelas estejam interconectadas. Confira os conceitos relacionados abaixo:

Primary Key - PK (Chave primária):

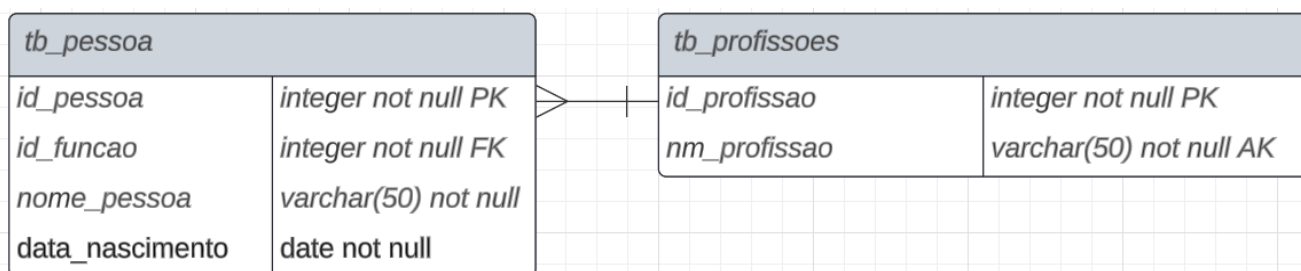
- Uma chave primária é uma coluna ou conjunto de colunas que identifica exclusivamente cada registro em uma tabela.
- Ela deve conter valores únicos e não nulos.
- Usada para indexar a tabela e garantir a unicidade dos registros.
- Cada tabela deve ter uma chave primária.

Foreign Key - FK (Chave estrangeira):

- Uma chave estrangeira é uma coluna ou conjunto de colunas em uma tabela que estabelece um relacionamento com a chave primária de outra tabela.
- Ela é usada para garantir a integridade referencial, garantindo que os valores na coluna correspondam aos valores na tabela referenciada.
- Os relacionamentos entre tabelas são estabelecidos por meio das chaves estrangeiras.

Em suma, os relacionamentos entre tabelas são estabelecidos usando chaves primárias e chaves estrangeiras. Eles podem ser de um para um, um para muitos ou muitos para muitos. Estes relacionamentos são fundamentais para a organização eficiente dos dados em várias tabelas, reduzindo a duplicação de informações.

Exemplo - Garantindo integridade entre a declaração de uma profissão para uma nova pessoa em **tb_pessoa** e profissões existentes na tabela **tb_profissoes**.



Vamos utilizar o diagrama como base para construirmos e declararmos os relacionamentos entre tabelas.

1. Criação e relacionamento das tabelas

```

1 • CREATE TABLE new_schema.tb_profissoes(
2     id_profissao integer not null,
3     nm_profissao varchar(50) not null unique,
4     primary key(id_profissao)
5 );
6
7 • CREATE TABLE new_schema.tb_pessoa(
8     id_pessoa integer not null,
9     id_funcao integer not null,
10    nome_pessoa varchar(50) not null,
11    data_nascimento date not null,
12    primary key(id_pessoa),
13    foreign key(id_funcao) REFERENCES tb_profissoes(id_profissao)
14 );

```

O nome da profissão não deve se repetir em outros registros

Definição da PK

Define chave estrangeira que referencia o campo id_profissao da tabela tb_profissoes

2. População das tabelas com dados para teste.

→ **tb_profissoes**

```

16 -- Dados tb_profissoes
17 • INSERT INTO tb_profissoes VALUES(1, 'programador');
18 • INSERT INTO tb_profissoes VALUES(2, 'médico');
19 • INSERT INTO tb_profissoes VALUES(3, 'auxiliar');
20
21 • SELECT * from tb_profissoes;
22

```

id_profissao	nm_profissao
3	auxiliar
2	médico
1	programador

→ **tb_pessoa**

```

23 -- Dados tb_pessoa
24 • INSERT INTO tb_pessoa VALUES(1, 1, 'Cláudio', '1980-04-01');
25 • INSERT INTO tb_pessoa VALUES(2, 1, 'Marcos', '1982-07-01');
26 • INSERT INTO tb_pessoa VALUES(3, 2, 'Cláudio', '1978-06-09');
27
28 • SELECT * from tb_pessoa;

```

id_pessoa, id_funcao, nome_pessoa, data_nascimento --> Os campos são inseridos nesta ordem.

id_pessoa	id_funcao	nome_pessoa	data_nascimento
1	1	Cláudio	1980-04-01
2	1	Marcos	1982-07-01
3	2	Cláudio	1978-06-09

Quais validações estamos realizando com a criação da estrutura acima?

1. Não é possível atribuir uma profissão a uma pessoa que não esteja registrada na tabela de profissões.

```
28 -- Retorna erro, pois o id dessa profissão não existe na tabela de profissões
29 • INSERT INTO tb_pessoa VALUES(4, 10, 'Claúdio', '1978-06-09');
```

#	Time	Action	Message	Duration / Fetch
1	17:36:17	INSERT INTO tb_pessoa VALUES(4, 10, 'Claúdio', '1978-06-09')	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('new_schema', 'tb_pessoa', CONSTRAINT 'tb_pessoa_ibfk_1' FOREIGN KEY (id_funcao) REFERENCES 'tb_profissoes' ('id_profissao'))	0.000 sec

A mensagem de erro retornado diz que: “Não é possível adicionar ou atualizar um registro filho, pois uma regra de chave estrangeira falhou”.

2. Tanto a tb_pessoa quanto a tb_profissoes exigem um identificador único de cada registro, por conta da Primary Key definida.

```
28 -- Ambos inserts retornam erro, pois estes ID's já existem em ambas tabelas.
29 • INSERT INTO tb_pessoa VALUES(1, 1, 'Sérgio', '1999-06-09');
30 • INSERT INTO tb_profissoes VALUES(1, 'Arquiteto');
```

#	Time	Action	Message
1	17:41:38	INSERT INTO tb_pessoa VALUES(1, 1, 'Sérgio', '1999-06-09')	Error Code: 1062. Duplicate entry '1' for key 'tb_pessoa.PRIMARY'
2	17:41:41	INSERT INTO tb_profissoes VALUES(1, 'Arquiteto')	Error Code: 1062. Duplicate entry '1' for key 'tb_profissoes.PRIMARY'

O erro foi: “Entrada duplicada pela chave <nome_tabela>.PRIMARY”.

3. Não é possível inserir registros na tabela de profissões que façam o nome da profissão se repetir, mesmo com id's diferentes. Confira:

```
28 -- Retorna um erro, pois já existe uma profissão cadastrada de nome 'programador'.
29 • INSERT INTO tb_profissoes VALUES(9, 'programador');
```

#	Time	Action	Message
1	17:45:07	INSERT INTO tb_profissoes VALUES(9, 'programador')	Error Code: 1062. Duplicate entry 'programador' for key 'tb_profissoes.nm_profissao'

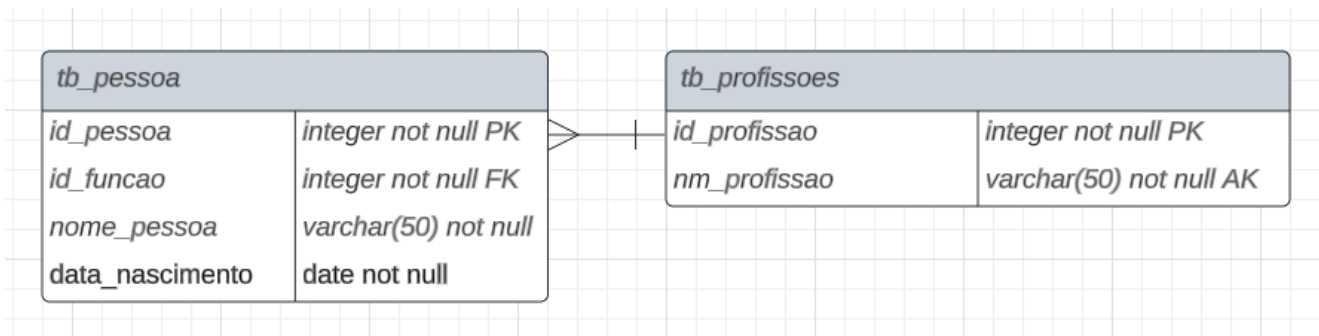
A constraint **UNIQUE** fez com que a entrada fosse barrada. O erro foi: “Entrada duplicada para a chave tb_profissoes.nm_profissao”.

Outras validações poderiam ser citadas. Somente com esta simples estrutura de tabelas, já é possível garantir uma grande segurança para a integridade dos dados.

5. Diagramas de entidade e relacionamento (ER)

Um Diagrama de Entidade e Relacionamento (DER), muitas vezes abreviado como ERD, é uma representação gráfica que descreve as entidades (tabelas) em um banco de dados e os relacionamentos entre essas entidades. Ele é usado para modelar a estrutura de um banco de dados e visualizar como as informações estão interconectadas.

Exemplo - Diagrama ER utilizado anteriormente



Somente com este pequeno diagrama já conseguimos montar tabelas com campos de determinados tipos e com determinadas regras, e também estabelecer uma relação entre estes campos.

A relação estabelecida entre estas tabelas é que: A **tb_profissoes pode conter múltiplas relações com a tb_pessoa**, porém para esta relação existir, **tb_pessoa deve possuir no mínimo uma relação com a tb_profissoes**. Esta relação pode ser lida através dos símbolos que ligam uma tabela na outra.

Você pode usar a plataforma **lucidchart** para criar seus diagramas, assim como este na imagem acima.

→ Para saber mais sobre diagramas ER: [Lucidchart - Diagrama entidade relacionamento](#).

6. Fixando conhecimento: Desafios de SQL

Você deve resolver a lista de exercícios passo a passo.

NOTA: Alguns nomes podem estar em uma linguagem mais coloquial (do dia a dia) e não adequada ao SQL. Você deve relacionar este tipo de linguagem ao comando SQL necessário para solucionar o que está sendo solicitado.

Exercícios

1. Criar tabela tb_aluno que contenha os seguintes campos:

Coluna	Instruções
cd_aluno	inteiro obrigatório (chave primária)
nm_aluno	caractere variável (255) obrigatório
nm_curso	carácter variável (255) obrigatório
nu_ano	inteiro obrigatório
nu_semestre	inteiro valor padrão = 1

2. Inserir os dados abaixo na tb_aluno. **Insira exatamente os dados como declarados abaixo.**

cd aluno	nm aluno	nm curso	nu ano	nu semestre
123	Felipe	ADS	2020	1
444	José	Letras	2018	3
891	Maria	MUSICA	2023	1
991	leANDrro	ADMINISTRAÇÃO	2021	3
002	Stefany	ADS	2019	5
888	Jéssica	Administração	2005	8
555	Paula	Administração	2008	4
754	Pedro	Administração	2009	

3. Liste todos os registros da tabela tb_aluno.
4. Liste apenas os alunos que fazem o curso de ADS.
5. Liste o número de registros existentes na tabela
6. Liste os cursos existentes sem repetições. **DICA:** [Veja select com distinct.](#)
7. Liste o nome dos alunos em ordem alfabética. **DICA:** [Veja ORDER BY](#)
8. Atualizar para maiúsculo todos os nomes dos alunos E cursos.
9. Atualizar o nome do aluno de código 991 para 'LEANDRO'.
10. Qual o registro mais recente de ano na tabela?
11. Quantos alunos fazem o curso de administração?
12. Qual o nome do aluno que está mais avançado no número de semestres?

DESENVOLVIMENTO DE SISTEMAS WEB II

13. Liste o nome do curso e a quantidade de matriculados. Ordene a listagem em ordem decrescente (do maior para o menor). **DICA:** [Veja GROUP BY](#).

14. Adicione uma regra na coluna nu_semestre para que o valor máximo inserido seja até 10. **DICA:** [Veja CHECK](#).

Caso queira, confira a solução do desafio para comparar com seus códigos:

→ **Github GIST:** [Códigos SQL - Solução desafio](#)

Referências

SILVEIRA, Paulo: **O que é SQL e para que serve?**, 2023. Disponível em:

<<https://www.alura.com.br/artigos/o-que-e-sql>>. Acesso em: 02 de outubro de 2023.

GABRIELA, Larissa: **MySQL: da instalação até a configuração**, 2022. Disponível em:

<<https://www.alura.com.br/artigos/mysql-instalacao-configuracao>>. Acesso em: 02 de outubro de 2023.

Múltiplos autores: **O que é um diagrama entidade relacionamento?**, ano não informado.

Disponível em: <<https://www.lucidchart.com/pages/pt/o-que-e-diagrama-entidade-relacionamento>>. Acesso em: 03 de outubro de 2023.