

início
28/08/2023

Curso de SQL e MySQL

do básico ao avançado



O que é SQL?



- A **SQL** é uma linguagem de banco de dados;
- É um acrônimo de **Structured Query Language** (linguagem estruturada de consulta);
- É a linguagem para os **bancos de dados relacionais** (MySQL, PostgreSQL, SQL Server);
- Trabalhamos com ela criando **Queries** (INSERT, UPDATE, SELECT);
- Com esta linguagem podemos **criar e manipular DBs**;

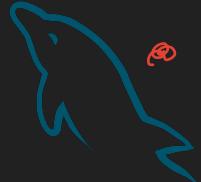


GRUPO

SQL x MySQL

oh

- SQL é a **linguagem** e MySQL é o **SGBD**;
- SGBD é um acrônimo de **Sistema Gerenciador de Bancos de Dados**;
- Ou seja, MySQL é um **software** que **gerencia os nossos bancos de dados**;
- Enquanto **SQL** é a linguagem que vamos utilizar para **manipular e criar os bancos**;



Banco →

SGBD

O que é um banco de dados?

5

- É onde **armazenamos os dados no nosso sistema**;
- Vamos criar os bancos de dados **através da SQL**;
- E o MySQL vai ajudar-nos a **gerenciar o banco e os dados**;
- O banco de dados possui algumas entidades fundamentais para o seu correto funcionamento, como **tabelas**;
- As tabelas guardam nossos **dados**:



• Entradas = Tabelas.



Database
DB

Principais elementos de um BD

b

- **Diagrama do Banco:** é o projeto do banco, parte fundamental e mais avançada, que define o sucesso do projeto;
- **Banco de dados:** A entidade que vai guardar todos os elementos do banco;
- **Tabelas:** A categoria dos dados;
- **Colunas:** Tipos de informações que precisamos salvar;
- **Dados:** O dado final entregue pelo usuário;



coluna	nome	CPF

colunas

Instalação MySQL Windows

7

- Vamos instalar o **XAMPP**;
- Este software nos dá o PHP, **MySQL** e também o Apache;
- Com ele podemos facilmente utilizar o banco de dados;
- Depois instalaremos um software para trabalhar com MySQL;

Root
Mn@2359txw
mysql --version



Arquivos de Ambiente
Arq. programs > MySQL Server >
bin > copia > criancisble
Ambiente > patch > editar
mono > colar o bin
bin

MySQL no terminal (Windows)

- Para algumas situações vamos precisar do **MySQL no terminal**;
- Para isso precisamos **adicionar o executável do MySQL as variáveis de ambiente**;
- Para conectar no MySQL vamos digitar: **mysql -u root**
- Vamos lá!



Instalação MySQL Linux

- Vamos instalar o MySQL pelo Linux com o **gerenciador de pacotes**;
- Depois estaremos aptos a **utilizar o MySQL pelo terminal**;
- Porém **vamos seguir o curso na interface gráfica**, para facilitar a visualização;
- A instalação dos softwares está na próxima aula;



Instalação Workbench

- O **Workbench** é uma ferramenta do MySQL para visualização de dados;
- Com ela podemos também criar **diagramas relacionais**;
- Vamos **adotar este software** para utilização no curso!



Instalação HeidiSQL

- O HeidiSQL é uma outra ferramenta de visualização de dados;
- Esta **funciona para a maioria dos SGBD's relacionais**;
- É uma outra alternativa de software, que tem **requisitos menores para instalar e rodar**;



Instalação VS Code

- O **VS Code** é um editor de código muito potente;
- Vamos adotá-lo como **ferramenta de edição de código do curso**;
- Além disso **possui um terminal integrado**, que vai nos ajudar com comandos de MySQL no terminal;
- Vamos lá!



Extensão MySQL no VS Code

- A extensão de MySQL no VS Code vai permitir **acessar o banco e fazer manipulações direto no editor**;
- **Basta ir no marketplace e digitar mysql**, é o primeiro resultado;
- Você pode seguir o curso com esta extensão também!



Criando um banco de dados

f d

- Nesta aula vamos colocar a mão na massa!
- Lembrando que esse conceito será explicado em detalhes futuramente :)
- Para criar um banco de dados utilizamos a instrução: **CREATE DATABASE <nome>**
- Desta forma agora temos um recipiente onde **podemos criar tabelas e inserir dados**:

CREATE DATABASE cliente;



Show DATABASES; → mostra o banco.

Sintaxe do SQL

- A sintaxe é a maneira como escrevemos instruções;
- Em SQL, por convenção, todas as instruções são em maiúsculos e os nomes são em minúsculo (banco, tabela, coluna), ex:
- CREATE DATABASE teste;
- Toda instrução deve ser finalizada com um ponto e vírgula; ↴
- Alguns SGBD's não exigem ponto e vírgula, porém inserir eles em todas as instruções é uma maneira de garantir a execução;



Exercício 1



- **Crie o seu primeiro banco de dados;**
- Define um nome para o seu banco e aplique a instrução da última aula;
- Mão à obra;



O que é importação de banco?

- Importação de banco é quando temos um **arquivo pronto de banco (com tabelas e dados) e inserimos ele no nosso SGBD;**
- Geralmente originado de uma **exportação;**
- Ação simples para obter todos os dados já cadastrados em um sistema;
- Após a importação podemos utilizar **como se o banco houvesse sido criado na nossa máquina;**
- É uma prática comum no dia a dia em uma empresa;



Importando banco de dados

- **Obs:** faça o download dos arquivos do curso!
- Vamos primeiramente acessar o **MySQL pelo terminal**;
- Agora devemos criar um banco, pode ser **empresa** o nome;
- Selecionar o banco com: **USE <nome>**
- E depois utilizar o comando: **source <arquivo>**
- O comando USE faz o banco de dados ser o principal no momento;
- Precisamos estar com o terminal na pasta do arquivo;



Selecionando dados de uma tabela

- **Obs:** vamos ver seleção mais a frente e em detalhes!
- Para selecionar todos os dados de uma tabela utilizamos o comando:
 - **SELECT * FROM <nome da tabela>**
- Desta maneira receberemos os dados inseridos na tabela alvo;
- Vamos utilizar a tabela de funcionários neste exemplo;



Exercício 2



- Agora é a sua vez, selecione os dados da tabela **servicos**;
- Esta tabela também está no nosso banco de dados!
- Mão à obra!



Como tirar o máximo proveito

- Execute todas as instruções das aulas;
- Faça os exercícios;
- Crie seus próprios exemplos bancos, tabelas e dados;
- Execute as instruções de aula em outras situações;
- **Dica bônus:** ver e depois executar;

✓ 27





V

29

Introdução

Conclusão da seção





Seção 2 : 3º

Criação de bancos

Introdução da seção



Criando um banco de dados

- Agora vamos nos aprofundar em **criação de bancos e tabelas**;
- O primeiro passo é sempre criar o banco;
- Utilizamos o comando: **CREATE DATABASE <nome>**;
- Desta forma teremos uma **entidade disponível para criação de tabelas** e posteriormente inserção de dados;

Banco > tabela > coluna > dados.



Verificando bancos de dados

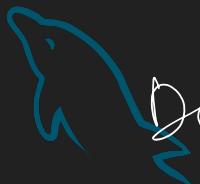
- Para garantir que um banco foi criado podemos utilizar um **comando utilitário**;
- Que é o: **SHOW DATABASES**;
- Todos os bancos que possuímos no nosso SGBD serão exibidos;



Removendo banco de dados

- Para deletar um banco de dados podemos utilizar o seguinte comando:
- **DROP DATABASE <nome>;**
- Note que esta ação frequentemente é chamada de **dropar**;
- Após esta ação **tabelas e dados são perdidos!**

Drop DATABASE <nome>;



DELETE → DELETA DADOS.

Exercício 3

- Crie um banco de dados com o nome de computador;
- Verifique se o banco foi criado;
- Remova o banco;



Exportando bancos

- Podemos também exportar nosso banco, utilizamos o comando:
 - **mysqldump -u root <nome_banco> > <nome_arquivo>.sql**
- Desta forma todas as tabelas e dados ficarão salvas em um arquivo .sql
- Podemos testar com o nosso banco de dados empresa!

Você usa
o nome do
arquivo.



Utilizando banco de dados

- Para qualquer ação que envolva um determinado banco, precisamos utilizar ele;
- O comando é: **USE <nome>;**
- Agora os comandos serão direcionados a este banco;
- Podemos então criar tabelas nele!



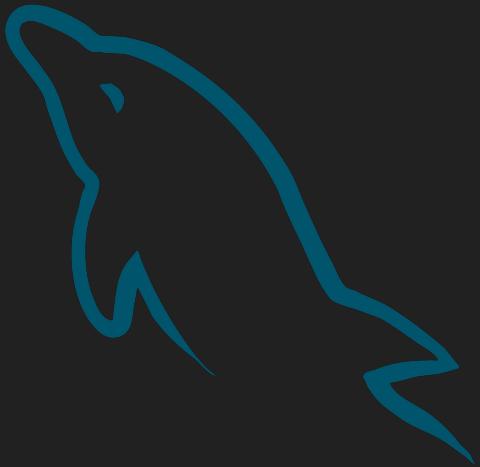


37

Criação de bancos

Conclusão da seção





Criação de tabelas

Introdução da seção



O que é uma tabela?

- É a entidade responsável por **guardar nossos dados**;
- Uma tabela possui **colunas**;
- As **colunas são como categorias** dos nossos dados: nome, profissão, idade;
- As colunas possuem **tipos de dados determinados**, como: textos, números, datas e etc;
- E **atributos**: não nulo, chave primária, auto incremento e etc;



Criando tabelas

- Agora vamos criar nossa primeira tabela, o comando é:
- **CREATE TABLE <nome> (<coluna> <tipo de dado>);**
- Podemos inserir diversas colunas e com diferentes tipos de dados;
- O mais comum e que acaba sendo mais utilizado é o **VARCHAR**, que representa os textos/strings de uma tabela;



Removendo tabelas

- Podemos também remover as tabelas que criamos;
- Lembrando que **todos os dados serão perdidos**;
- O comando é:
- **DROP TABLE <nome>;**
- Agora a tabela não existe mais no nosso banco!



Tipos de dados

- Os tipos de dados **classificam um dado** e os inserimos em uma coluna;
- Podemos ter dados como: texto, data, número e outros;
- É uma **parte extremamente importante da criação do nosso projeto** e banco de dados;
- Alguns dados também **permitem limites** serem definidos, como quantidade máxima de caracteres;
- São divididos em: **texto, numérico, data e espacial**;



Tipos de texto

- **CHAR(x)**: aceita textos com 0 a 255 caracteres;
- **VARCHAR(x)**: aceita textos com 0 a 65535 caracteres;
- Onde **x** é a quantidade máxima;
- **TINYTEXT**: apenas texto com até 255 caracteres;
- **MEDIUMTEXT**: apenas texto com até 16777215 caracteres;
- Obs: **CHAR** e **VARCHAR** aceitam números e caracteres especiais;



Exercício 4

- Agora vamos trabalhar com tipos de dados, para isso vamos precisar de um novo banco!
- Crie um banco com o nome: **teste_tipo_dados**



Vamos criar uma tabela com textos

- Para exercitar, vamos criar uma tabela com os tipos de texto;
- Utilizaremos **CHAR**, **VARCHAR** e também **MEDIUMTEXT**;
- Vamos lá!



Exercício 5

- Crie uma tabela chamada cadastro;
- Com dois campos como VARCHAR, o nome e sobrenome;
- Limite os dois para 100 caracteres;



Inserir dados

- Uma das operações mais comuns é **inserir dados**, utilizamos o comando:
- **INSERT INTO <tabela> (<colunas...>) VALUES (<valores...>)**
- Precisamos inserir o nome das colunas e também os valores para cada uma, **separados por vírgula**;
- Os **valores precisam corresponder a ordem das colunas**;
- **Obs:** valores de texto são inseridos entre aspas;



Exercício 6

- Sua vez de inserir produtos!
- Crie dois inserts para inserir produtos na nossa tabela produtos;
- Depois selecione todos os dados, dica: SELECT * FROM ...



Tipos numéricos

- Os tipos numéricos não precisam de aspas ao serem inseridos;
- **BIT(x)**: 1 a 64 caracteres;
- **TINYINT(x)**: 1 a 255 caracteres;
- **BOOL**: 0 é falso e outros valores são verdadeiros;
- **INT(x)**: valores entre -2147483648 a 2147483647;



Vamos criar uma tabela com números

- Vamos testar **BOOL** e **INT**;
- Neste exemplo vamos também inserir outro tipo de dado, para ver que isso é possível e muito utilizado;
- E por fim vamos inserir dados nesta tabela;
- Vamos lá!



Exercício 7

- Sua vez de inserir servidores;
- Insira mais dois servidores na nossa tabela;
- Depois selecione todos os dados, dica: SELECT * FROM ...



Tipos de data

- **DATE**: Aceita uma data no formato YYYY-MM-DD;
- **DATETIME**: Aceita uma data com horário no formato YYYY-MM-DD hh:mm:ss;
- **TIMESTAMP**: Aceita uma data no formato de DATETIME, porém apenas entre os anos 1970 a 2038;
- As datas também inserimos entre aspas!



Vamos criar uma tabela com datas

- Vamos utilizar para este exemplo **DATE**;
- Criaremos uma tabela com **nomes e data de nascimento**;
- E por último vamos inserir dados na tabela;
- Vamos lá!



Exercício 8

- Insira o seu nome e data de nascimento na tabela;
- Treine também a seleção de todos os dados;
- Mão à obra!



Porque escolher o tipo de dado?

- Devemos sempre **escolher o tipo de dado mais próximo da nossa necessidade**, e também limitar tamanho quando for possível;
- Isso vai **otimizar o banco**, deixando ele com respostas mais rápidas;
- Além de **economizar espaço em disco**, pois estamos salvando apenas o necessário;
- Pense bem antes de realizar a criação das colunas, para **escolher o tipo correto e limitá-lo**;



Alterando tabelas

- Há três tipos de alterações em SQL:
- **Adição de colunas;**
- **Remoção de colunas;**
- **Modificar tipo da coluna;**
- Vamos criar uma tabela para trabalhar com alterações!



Adicionando nova coluna a tabela

- Para **adicionar** uma coluna vamos utilizar o comando:
- **ALTER TABLE <tabela> ADD COLUMN <nome> <tipo>;**
- Desta maneira uma nova coluna é adicionada;



Exercício 9

- Adicione a coluna cpf com CHAR(11) na tabela funcionários;
- Adicione outra coluna da sua escolha;
- Adicione dados a tabela;
- Mãoz à obra!



Removendo coluna da tabela

- Para **remover** uma coluna vamos utilizar o comando:
- **ALTER TABLE <tabela> DROP COLUMN <nome>;**
- Desta maneira uma nova coluna é removida;



Exercício 10

- Remova a coluna que você adicionou anteriormente no exercício 9;
- Mãoz à obra!



Modificando coluna

- Para **alterar um tipo** de dados vamos utilizar:
- **ALTER TABLE <tabela> MODIFY COLUMN <coluna> <tipo>;**
- Agora a coluna está com o novo tipo em vigor;
- Vamos fazer um teste!

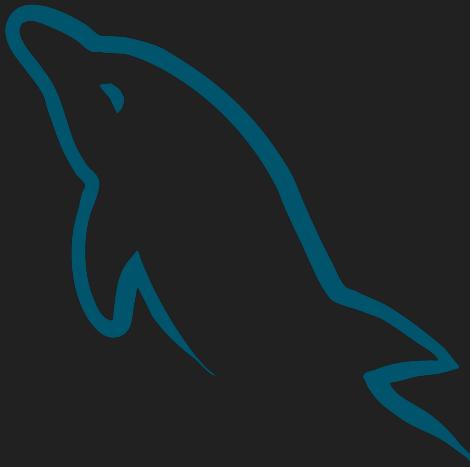




Criação de tabelas

Conclusão da seção





Comandos do CRUD

Introdução da seção



Exercício 11

- Crie um banco de dados chamado cadastro;
- Crie uma tabela chamada pessoas;
- Colunas nome, rg, cpf e limite;
- nome, rg e cpf, são dados de texto;
- limite é um valor numérico;
- Insira 5 dados;



O que é CRUD?

- **CRUD** são as ações que mais são utilizadas em todas as aplicações;
- **Create** = criar / inserir dados (insert);
- **Read** = ler dados (select);
- **Update** = atualizar dados (update);
- **Delete** = deletar / remover dados (delete);
- **Toda aplicação web com banco de dados** tem pelo menos uma destas operações e possivelmente todas;



Selecionar todos os dados

- Para selecionar todos os dados de uma tabela utilizamos:
- **SELECT * FROM <tabela>;**
- Desta maneira receberemos **todas as colunas da tabela e também todos os registros;**
- Conforme a base vai crescendo, essa consulta pode se tornar muito custosa e também **lenta;**



Selecionar colunas específicas

- Para selecionar colunas específicas trocamos o * por o nome das colunas, veja:
- **SELECT coluna1, coluna2 FROM <tabela>;**
- Desta maneira **criamos um filtro**, deixando nossa consulta um pouco mais leve;
- Ao longo do curso aprenderemos outras **técnicas de otimização de consulta**;



Inserindo dados

- Para inserir dados vamos utilizar a instrução INSERT:
- **INSERT INTO <tabela> (<colunas...>) VALUES (<valores...>)**
- Os valores e as colunas devem ser **separados por vírgula**;



A importância do WHERE

- O WHERE é uma cláusula de algumas queries, que determina **quais registros vamos atualizar**;
- Por exemplo: **WHERE id = 1**;
- Se não inserirmos esta instruções em atualizações ou remoções, vamos **aplicar a todos os dados**, o que pode ser um enorme problema;
- Então **lembre-se sempre de inserir WHERE** em UPDATE e DELETE;
- Depois veremos sua aplicação com SELECT;



Atualizando dados

- Para atualizar dados vamos utilizar a instrução **UPDATE**:
- **UPDATE tabela SET <coluna=valor> WHERE <condição>**
- Lembre-se de utilizar o **WHERE**;
- Podemos inserir mais colunas, separando elas por **vírgula**;



Exercício 12

- Atualize o limite do quarto registro da sua tabela para 1000;
- Mãoz à obra!



Deletando dados

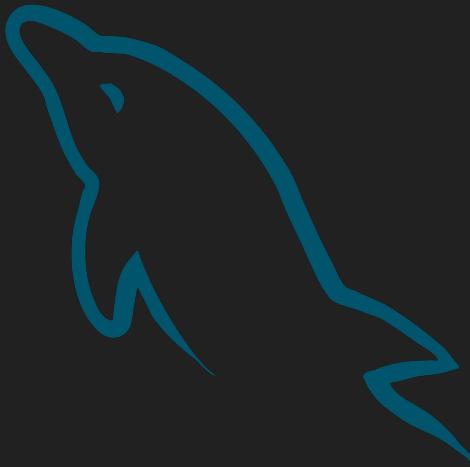
- Para deletar dados vamos utilizar a instrução **DELETE**:
- **DELETE FROM <tabela> WHERE <condição>**
- Lembre-se de utilizar o **WHERE**, se não deletaremos todos os dados;
- A quantidade de dados removidos depende do WHERE;



Exercício 13

- Delete registros que tenham limites maiores que 2000;
- Mãoz à obra!





Comandos do CRUD

Conclusão da seção





Avançando em SELECT

Introdução da seção



Instalar um banco

- Vamos instalar um banco maior para trabalhar nesta seção;
- Os dados a mais ajudarão na imersão dos comandos novos;
- Utilizaremos este banco: https://github.com/datacharmer/test_db
- Vamos instalar juntos!



A importância do SELECT

- A maioria das queries em um banco de dados **são de consulta**;
- E é também o comando com **mais variações**;
- Para receber detalhados resultados, precisamos aprender todo o poder do SELECT;
- Desta maneira criaremos **filtros avançados** e conseguiremos atingir o resultado desejado facilmente;



Operadores

- **Comparação** ($>$, $<$, \geq , \leq , $=$): estes operadores vão filtrar dados baseados nas comparações;
- **BETWEEN**: Seleção entre um intervalo;
- **LIKE**: Seleção por meio de algum padrão;
- **IN**: Seleção entre um conjunto de valores específicos;



Cláusula WHERE

- O **WHERE** será utilizado junto dos operadores vistos da última aula;
- Assim conseguimos **filtrar os dados de maneira objetiva**;
- Exemplo: WHERE id = 10;
- Receberemos apenas o resultado onde o id é 10;



Exercício 14

- Selecione na tabela titles todos as colunas, porém apenas as que emp_no for maior ou igual a 11500;
- Mãoz à obra!



Utilizando o DISTINCT

- O **DISTINCT** vai selecionar apenas as variações de valores;
- Por exemplo: temos 10 cidades diferentes de usuários no sistema, só receberemos 10 resultados;
- O comando é:
- **SELECT DISTINCT <coluna> FROM <tabela>**
- Volta apenas os valores diferentes da coluna selecionada;



Operadores lógicos

- Podemos **combinar a cláusula WHERE** com operadores lógicos como:
- **AND**: Recebe duas condições, só volta os resultados que atendem as duas;
- **OR**: Recebe duas condições, volta os resultados que atendem pelo menos uma;
- **NOT**: Invertemos uma cláusula;
- Exemplo: WHERE id > 10 OR salario > 5000;



Utilizando o AND

- Todos estes operadores lógicos **são muito utilizados na programação**;
- Com o **AND** temos um filtro duplo para resultados;
- Por exemplo: WHERE salario > 1000 AND cargo = 'programador';
- No caso acima, apenas **programadores com salário maior que 1000** serão retornados no nosso SELECT;



Utilizando o OR

- O **OR** tem a possibilidade de retornar qualquer uma das condições impostas;
- Exemplo: WHERE salario > 5000 || profissao = 'programador';
- Neste caso, teremos **programadores com salários menores que 5000 e também outros tipos de cargos com salários acima de 5000**;



Utilizando o NOT

- O **NOT** inverte uma cláusula;
- Exemplo: WHERE NOT profissao = 'programador';
- Ou seja, teremos todos os dados retornados, **menos os que são de programadores**;
- Pode ser utilizado neste sentido de exclusão de dados;



Utilizando a ORDER BY

- A **ORDER BY** é uma instrução para ordenação de resultados;
- Podemos utilizar de forma ascendente (**ASC**);
- E descendente (**DESC**);
- Utilizamos **após o WHERE**, se tiver;
- E é **baseada em alguma coluna**, exemplo:
- ORDER BY salario ASC;



Exercício 15

- Selecione na tabela `titles` todos as colunas, porém ordene os títulos por ordem descendente;
- Mãoz à obra!



Utilizando a LIMIT

- A instrução **LIMIT** é outra forma interessante de **limitar os resultados e tornar a consulta mais rápida**;
- Podemos especificar o número de resultados retornados, exemplo:
- **LIMIT 15;**
- Apenas os 15 primeiros resultados serão exibidos;
- **Podemos unir com WHERE** e outras instruções;



Exercício 16

- Selecione na tabela departments todos as colunas, porém limite a 5 resultados e ordene de forma ascendente;
- A ordem deve ser pela coluna de title;
- Mãoz à obra!



Funções no SQL

- **Funções** são blocos de códigos reaproveitáveis;
- Utilizadas para **extrair resultados que demandam muita programação**;
- Temos diversas no SQL, que vão nos ajudar muito nas nossas consultas;
- Exemplo: a função **MAX** retorna o maior valor, de uma determinada coluna;
- Algo que poderia ser atingido por: SELECT coluna FROM tabela ORDER BY coluna DESC LIMIT 1;



Função MIN

- A função **MIN** vai retornar o menor valor de uma coluna específica;
- Veja um exemplo:
- **SELECT MIN(<coluna>) FROM <tabela>;**
- Desta maneira vamos receber apenas um resultado, o de menor valor;



Função MAX

- A função **MAX** vai retornar o maior valor de uma coluna específica;
- Veja um exemplo:
- **SELECT MAX(<coluna>) FROM <tabela>;**
- Desta maneira vamos receber apenas um resultado, o de maior valor;



Função COUNT

- A função **COUNT** vai retornar o número de valores que combinam com algum critério;
- Podemos **utilizar com o WHERE**;
- Exemplo: **SELECT COUNT(*) FROM salaries WHERE salary > 100000;**
- Assim teremos a quantidade de salários maior que 100k;



Exercício 17

- Faça um SELECT que conte o número de registros na tabela departments;
- Mãoz à obra!



Função AVG

- A função **AVG** vai retornar a média de uma determinada coluna;
- AVG vem de average (média);
- Podemos utilizar da seguinte maneira:
- **SELECT AVG(salary) FROM salaries;**
- Temos assim a média de salários;



Função SUM

- A função **SUM** vai retornar a soma de todos os valores de uma coluna;
- Podemos utilizar da seguinte maneira:
- **SELECT SUM(salary) FROM salaries;**
- Assim teremos a soma de todos os salários;



Operador LIKE

- O **LIKE** é utilizado sempre em conjunto do WHERE;
- Ele tem a premissa de filtrar ainda mais nossos resultados;
- Utilizamos também o **coringa %**, que ajuda muito nas buscas:
- **SELECT * FROM employees WHERE first_name LIKE '%ber%';**
- Desta forma teremos a seleção de todos os nomes que contém ber, não importa se for no fim ou começo;



Exercício 18

- Faça um SELECT na tabela titles;
- Selecione todos os títulos que tenham Engineer na coluna;
- Mãoz à obra!



Operador IN

- O **IN** vai fazer uma **busca por um conjunto de valores**;
- Exemplo:
- `SELECT * FROM dept_emp WHERE dept_no IN ('d004', 'd005', 'd006');`
- Desta forma selecionamos os registros apenas dos departamentos d004, d005 e d006;



Exercício 19

- Na tabela employee faça um SELECT de todos os registros que contenham no sobrenome:
 - Facello e Peac;
 - Utilize o IN;
 - Mãoz à obra!



Operador BETWEEN

- O **BETWEEN** é parecido com o IN, mas ele vai receber uma **faixa de valores**;
- E o resultado será retornado baseado nos registros dentro da faixa, veja:
- **SELECT * FROM dept_emp WHERE dept_no BETWEEN 'd001' AND 'd008';**
- Neste exemplo retornamos os departamentos do 1 ao 8;
- O **AND** é utilizado para determinar o intervalo;



Exercício 20

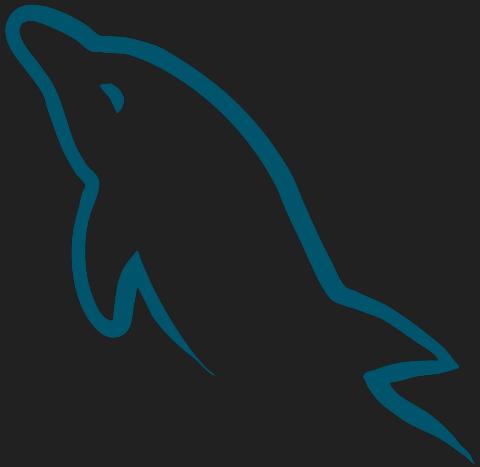
- Na tabela de salário faça um SELECT;
- Selecione os salários em uma faixa de 125000 a 175000;
- Utilize o BETWEEN;
- Mãoz à obra!



Criando um ALIAS

- O **ALIAS** pode servir para renomear uma coluna com nome não objetivo ou colunas originadas de função, por exemplo:
- **SELECT SUM(salary) AS soma_salario FROM salaries;**
- Agora a soma dos salários tem o nome de soma_salario, o que é muito mais objetivo!
- O **AS** é utilizado para determinar como a coluna deve se chamar;





Avançando em SELECT

Conclusão da seção





Constraints

Introdução da seção



O que são constraints?

- São regras que determinam como os campos serão preenchidos;
- Por exemplo: NOT NULL = o campo não pode ser nulo;
- As constraints são adicionadas na criação da tabela geralmente, porém podemos alterar tabelas para adicioná-las;
- Estas regras são de grande utilidade pois ajudam a organizar e padronizar nosso projeto;
- Vamos testá-las!



NOT NULL

- A constraint **NOT NULL** força um valor de uma coluna específica não ser nulo;
- Colocamos a instrução após o nome e tipo da coluna ser declarado: **nome VARCHAR(100) NOT NULL;**
- Desta maneira a coluna recebe a constraint;



UNIQUE

- A constraint **UNIQUE** garante que todos os valores em uma coluna serão diferentes;
- Um caso de uso é e-mail, não queremos e-mails duplicados na nossa base;
- Desta maneira, recebemos um erro caso o dado já tenha sido inserido;



PRIMARY KEY

- A constraint **PRIMARY KEY** só pode ser adicionada em uma única coluna da tabela, geralmente é o id;
- **O valor deve ser único** e não pode ser nulo;
- Podemos dizer que é um **identificador único de um registro na tabela**;



AUTO INCREMENT

- A constraint **AUTO INCREMENT** serve para adicionar a quantidade de um em todo registro adicionado;
- Esta constraint é **muito utilizada na coluna id**, já que ela é única e também chave primária;
- Então **não precisamos nos preocupar com este valor no INSERT**;



FOREIGN KEY

- A FOREIGN KEY é uma ligação de uma tabela a outra;
- Por exemplo: uma tabela cadastramos o usuário e em outra o endereço dele;
- Na tabela do endereço temos uma FOREIGN KEY que se refere a o id do usuário;
- Desta maneira conseguimos **impedir remoções de dados que tem ligação entre tabelas**;



INDEX

- Adicionar um índice a uma coluna **faz a consulta que envolva a mesma se tornar mais rápida**;
- Sem o índice a consulta começa da primeira a última coluna até encontrar que você precisa, com o índice as demais serão ignoradas;
- As consultas que são melhoradas pelo **INDEX** são as com **WHERE**;



Removendo INDEX

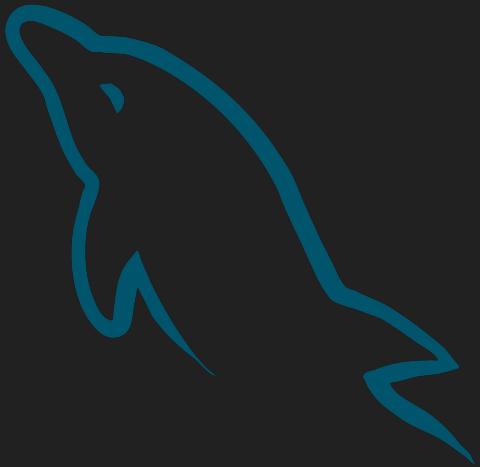
- Pode chegar um momento que um índice não é mais necessário, então precisamos remover ele;
- Lembrando que o excesso de índice pode ser prejudicial ou fazer com que os índices necessários não funcionem corretamente;
- Exemplo de remoção:
- **DROP INDEX <nome> ON <tabela>;**



Exercício 21

- Crie um banco de dados **banco** e uma tabela chamada **contas**;
- Insira as colunas **id**, **nome**, **sobrenome**, **saldo**, **data_nascimento**;
- Encontre os melhores tipos de dados para as mesmas;
- A coluna de **id** deve ser **PRIMARY KEY**, **AUTO_INCREMENT** e **NOT NULL**;
- Crie um índice em **saldo**;
- Adicione 3 registros na tabela;

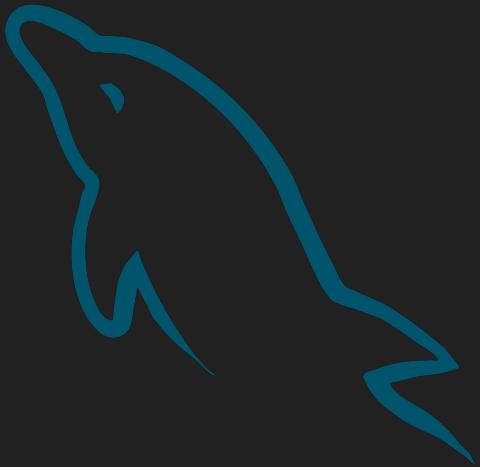




Constraints

Conclusão da seção





Joins

Introdução da seção



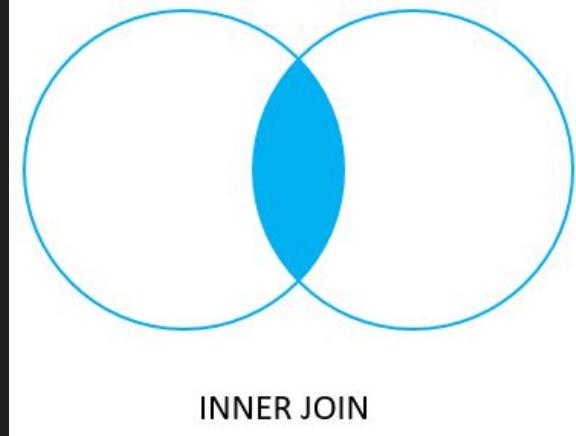
O que é um JOIN?

- São **consultas que envolvem duas ou mais tabelas**;
- As **tabelas geralmente possuem relação entre si**;
- Temos então uma consulta mais complexa e com mais dados;
- Há **três tipos mais utilizados de JOIN**:
- LEFT e RIGHT JOIN;
- INNER JOIN;



INNER JOIN

- O **INNER JOIN** vai resultar nas colunas que fazem relação entre as tabelas;
- Podemos determinar qual coluna resgatar após a instrução SELECT;
- Utilizamos a instrução **ON** para determinar as colunas que precisam ser iguais;



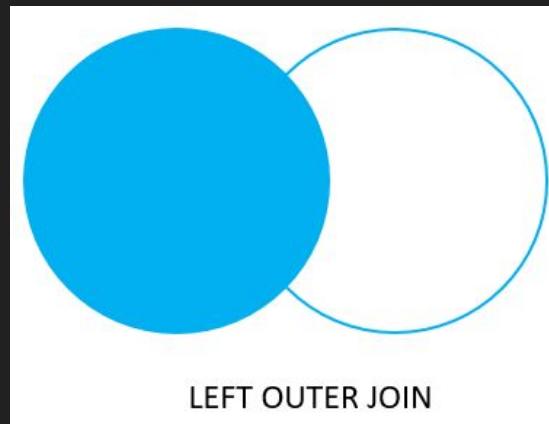
Exercício 22

- Utilize o INNER JOIN:
- Para selecionar o primeiro nome, genero e cargo;
- A relação entre as tabelas salaries e titles;



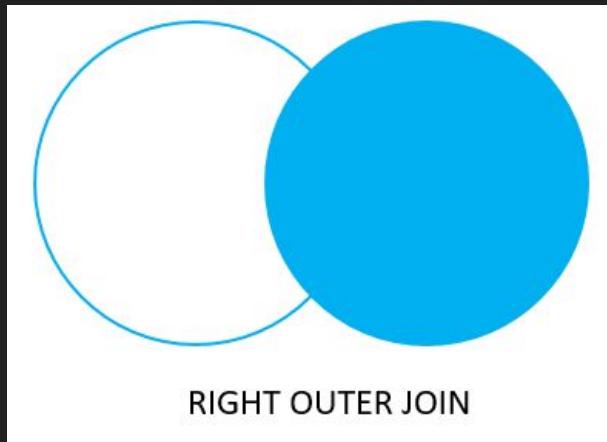
LEFT JOIN

- O **LEFT JOIN** vai retornar todos os dados da tabela da esquerda e os necessários da direita;
- Também é chamado de **LEFT OUTER JOIN**;
- Haverá dados retornados da tabela da esquerda mesmo se não corresponder com a outra;



RIGHT JOIN

- O **RIGHT JOIN** é semelhante ao LEFT, porém ele trás as colunas a mais da direita;
- Chamado também de **RIGHT OUTER JOIN**;



Existem outros JOIN's?

- Sim, **existem!**
- Porém 99% das vezes os três vistos aqui serão o suficiente;
- Geralmente queremos a **relação entre duas ou mais tabelas**, para pegar dados referentes a um registro;
- Então o **INNER JOIN** cai como uma luva, removendo dados não necessários;





Joins

Conclusão da seção





Agrupamentos e Subquery

Introdução da seção



UNION

- O **UNION** é utilizado para combinar o resultado de dois ou mais SELECTS;
- **As colunas precisam ter o mesmo nome;**
- Os resultados serão agregados em uma coluna só, porém **podemos selecionar mais de uma coluna** por vez;
- **Não** pode trazer resultados duplicados;



UNION ALL

- O **UNION ALL** é utilizado para combinar o resultado de dois ou mais SELECTS;
- **As colunas precisam ter o mesmo nome;**
- Os resultados serão agregados em uma coluna só, porém **podemos selecionar mais de uma coluna** por vez;
- Pode trazer resultados duplicados!



GROUP BY

- O **GROUP BY** serve para agruparmos colunas e checarmos quantidades de determinados elementos;
- Por ex: quantos programadores ou designers existem no nosso banco;
- Então agrupamos as colunas somando elas e temos o resultado de grupos;



Exercício 23

- Agrupe os trabalhadores por data de contratação, para ver se muitas pessoas foram contratadas no mesmo dia;
- A coluna é `hire_date`;



HAVING

- O **HAVING** é semelhante ao WHERE;
- Porém vamos utilizar eles com **aggregate functions (SUM, AVG, GROUP BY)**, pois o WHERE não pode ser utilizado nestes casos;
- Então sempre que estamos utilizando uma destas funções de agregação de dados, precisamos optar pelo HAVING;



Exercício 24

- Agrupe novamente os trabalhadores por data de contratação;
- Porém exiba as datas que tem menos ou 50 colaboradores contratados;



Subquery

- **Subquery** é uma query dentro de outra query;
- Teremos mais comumente dois SELECT's;
- A Subquery em alguns casos **se parece muito com um JOIN**;
- Porém as vezes **precisamos de algo muito específico**, então fazer uma subconsulta pode se tornar mais prático do que um JOIN;



EXISTS

- O **EXISTS** serve para checar se existe algum registro em alguma subquery;
- Desta maneira podemos retornar resultados, apenas se existir algum de fato;



ANY

- O **ANY** vai retornar os resultados que recebem TRUE da Subquery;
- Por meio de uma subquery receberemos apenas os resultados que condizem a uma condição;
- Ou seja: se queremos os nomes de quem ganha mais de 150000, só receberemos nomes se alguém ganhar mais que 150000;



Comentários

- Os **comentários** são utilizados por duas razões:
- Escrever instruções do que o código executa;
- Ou impedir a execução de um código, pois **o interpretador ignora código comentado**;
- Para comentários em uma linha utilizamos o símbolo: **--**



Comentários múltiplas linhas

- Os comentários de SQL também podem ser **escritos em várias linhas**;
- A sintaxe é: **`/* algum texto */`**
- Onde esta maneira também funciona para uma linha somente;



Exercício 25

- Crie uma tabela posts;
- Comente cada uma das ações que você vai fazer na tabela, ex: cada coluna;
- Você precisa inserir colunas como: id, titulo, corpo_do_post e tags;





Agrupamentos e Subquery

Conclusão da seção





Funções de String

Introdução da seção



Instalação do Sakila

- **Obs:** faça o download dos arquivos do curso!
- Para esta seção vamos importar um outro banco, o **Sakila**;
- Vamos precisar conectar ao mysql com: **mysql -u root**
- Depois **criar o banco de dados sakila e aplicar o USE**;
- E por fim utilizar o source em dois arquivos, nesta ordem:
sakila-schema.sql e depois sakila-data.sql



O que são funções?

- As **funções** são blocos de códigos já definidos que podem ser reutilizados;
- Assim como a maioria das linguagens de programação já possuem funções prontas, o SQL também;
- Elas nos ajudam a **atingir resultados de forma simples**, que com apenas queries seriam atingidos de forma complicada;
- E também permitem a **reutilização** das mesmas;



CHAR_LENGTH

- A função **CHAR_LENGTH** nos retorna o número de caracteres de uma string de uma determinada coluna;
- Ela leve um argumento que é a coluna que vamos avaliar;
- Exemplo:

CHAR_LENGTH(<nome_da_coluna>)



Exercício 26

- Selecione da tabela address:
- A coluna id, address, e aplique a função de CHAR_LENGTH em postalcode;
- Ordene os dados de forma descendente pelo id;



CONCAT

- A função **CONCAT** concatena duas ou mais strings;
- Concatenar significa **unir strings**;
- Veja um exemplo:

CONCAT(“MySQL”, “ é “, “ bom”)

- Aqui concebemos uma frase, mas podemos utilizar com colunas, o que deixa as coisas mais interessantes!



Exercício 27

- Selecione na tabela actor o id;
- E concatene nome e sobrenome dos atores;
- Coloque um espaço entre o nome e sobrenome;



CONCAT_WS

- A função **CONCAT_WS** concatena duas ou mais strings, porém com um separador comum;
- O primeiro argumento é um separador;
- Veja um exemplo:

CONCAT_WS("- ", “MySQL”, “ é ”, “ bom”)

- Os resultados seção espaçados pelo separador definido;



FORMAT

- A função **FORMAT** vai formatar um número com um número de casas determinado por argumentos;
- Além disso, em alguns casos **pode até arredondar o número**;
- Exemplo:

FORMAT(numero, 2)

- Formata o número com 2 casas decimais;



INSTR

- A função **INSTR** retorna a posição do caractere ou string que estamos buscando em uma outra string;
- Exemplo:

INSTR(<string>, ‘a');

- Retorna a posição da letra a na string alvo;



LCASE

- A função **LCASE** transforma todo o texto retornado para **lower case**, ou seja, letras minúsculas;
- Exemplo:

```
LCASE('ALGUM TEXTO')
```



LEFT

- A função **LEFT** extrai uma quantidade de caracteres de uma string;
- A extração acontece da **esquerda para direita**;
- E a quantidade é passada por parâmetro
- Exemplo:

LEFT(<string>, 5)



Exercício 28

- Selecione na tabela address as colunas:
- id e last_update;
- Em last_update aplique a função LEFT e mostre apenas a data;
- Aplique um alias em last_update para data;
- Traga apenas ids maiores que 150;



REPLACE

- A função **REPLACE** troca alguma parte de uma string por outra enviada por parâmetro;
- Exemplo:

REPLACE(<texto>, <antigo>, <novo>)

- Desta forma podemos fazer manipulações nos valores retornados;



Exercício 29

- Selecione o usuário que tem o sobrenome SMITH da tabela customer;
- Mude o nome para Mary e o sobrenome para Smith, vamos remover a caixa alta de todas as letras, deixar só nas iniciais;
- Crie alias para as duas colunas, que serão: nome e sobrenome;



RIGHT

- A função **RIGHT** extrai caracteres da direita para a esquerda;
- O número de caracteres é determinado pelo argumento da função;

RIGHT(<texto>, 10)

- Neste caso teremos uma extração de 10 caracteres da direita para esquerda;



SUBSTR

- A função **SUBSTR** extrai uma string a partir de dois índices, início e fim;
- Os índices são passados via argumento;
- Exemplo:

SUBSTR(<texto>, <inicio>, <fim>)



UCASE

- A função **UCASE** vai transformar todas as letras em maiúsculas;
- É o contrário de LCASE;
- Exemplo:

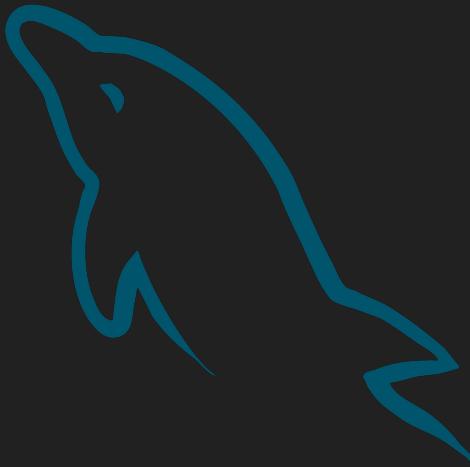
UCASE(<string>)



Exercício 30

- Selecione id e país da tabela country;
- Mude o nome dos países para letras maiúsculas;

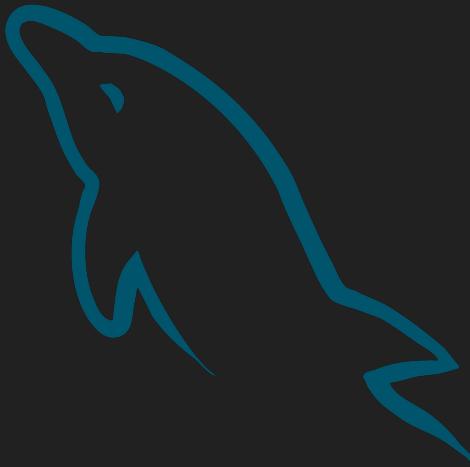




Funções de String

Conclusão da seção





Funções de Numbers

Introdução da seção



CEIL

- A função **CEIL** arredonda números com casas decimais para cima;
- Ou seja: $25.8 > 26$;
- Vamos utilizar a apenas o número/coluna como argumento:

CEIL(<numero>)



COUNT

- **Obs:** já utilizamos COUNT antes!
- A função **COUNT** conta o número de ocorrências de uma determinada coluna;
- Exemplo:

COUNT(<coluna>)

- Ideal para saber o número de registros em uma tabela;



Exercício 31

- Obtenha a quantidade de clientes cadastrados;
- Estão na tabela customer;



FLOOR

- A função **FLOOR** arredonda números com casas decimais para baixo;
- Ou seja: $25.8 > 25$;
- Vamos utilizar a apenas o número/coluna como argumento:

FLOOR(<numero>)



MAX

- A função **MAX** retorna o maior valor de uma coluna;
- Apenas passamos a coluna como argumento;
- Exemplo:

MAX(<coluna>)



MIN

- A função **MIN** retorna o menor valor de uma coluna;
- Apenas passamos a coluna como argumento;
- Exemplo:

MIN(<coluna>)



Exercício 32

- Encontre o maior salário e o menor salário da tabela salaries;
- Esta tabela está no banco employees;

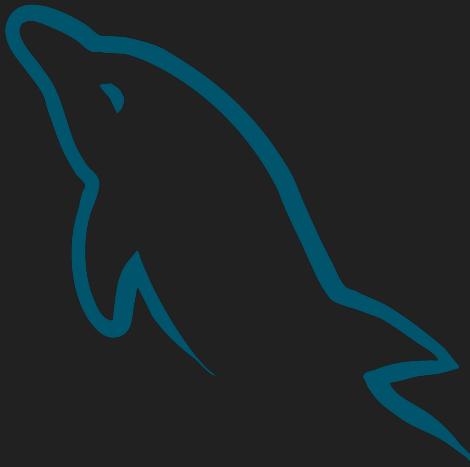


SUM

- A função **SUM** retorna a soma dos valores de uma coluna;
- Apenas passamos a coluna como argumento;
- Exemplo:

SUM (<coluna>)





Funções de Numbers

Conclusão da seção





Funções de Datas

Introdução da seção



ADDDATE

- A função **ADDDATE** adiciona ou remove uma quantidade horas, dias, meses ou anos a uma data;
- Utilizamos da seguinte forma:

ADDDATE(<coluna>, <data para adicionar>)

- Desta forma teremos uma nova data na seleção dos dados;



ADDTIME

- A função **ADDTIME** adiciona ou remove um tempo a uma data;
- Podemos utilizar o formato: yyyy-mm-dd hh:mm:ss;
- Exemplo:

ADDTIME(<coluna>, <tempo>);

- Desta maneira a data final será modificada;



DATEDIFF

- A função **DATEDIFF** calcula a diferença de duas datas;
- O valor é informado em dias;
- Vamos utilizar desta maneira:

DATEDIFF(<data1>, <data2>);

- Desta maneira será exibido o valor da diferença;



DATE_FORMAT

- A função **DATEFORMAT** formata uma data com um padrão indicado;
- Exemplo de utilização:

DATE_FORMAT(<data>, <formato>)

- Se utilizamos %Y, recebemos o ano completo, por exemplo;



DAY

- A função **DAY** retorna o dia da data utilizada na função;
- Exemplo:

DAY(<data>)

- Desta maneira teremos o número do dia, como: 15;



DAYOFWEEK

- A função **DAYOFWEEK** retorna o dia da semana de uma determinada data;
- Exemplo:

DAYOFWEEK(<data>)

- Iniciando de domingo como 1;



DAYOFYEAR

- A função **DAYOFYEAR** retorna o dia do ano de uma determinada data;
- Exemplo:

DAYOFYEAR (<data>)

- O retorno é de 1 a 365;



WEEKOFYEAR

- A função **WEEKOFYEAR** retorna a semana do ano de uma determinada data;
- Exemplo:

WEEKOFYEAR (<data>)

- O retorno é de 1 a 42;



MONTH

- A função **MONTH** extrai o mês de uma data;
- Exemplo:

MONTH(<data>)

- O retorno é de 1 a 12;



YEAR

- A função **YEAR** extrai o ano de uma determinada data;
- Exemplo:

YEAR (<data>)

- Assim obteremos de maneira fácil o ano;





Funções de Datas

Conclusão da seção





Relacionamento de tabelas

Introdução da seção



O que são relacionamentos?

- SQL é uma linguagem de bancos de dados relacionais, ou seja, que **possuem relações**;
- Estas relações servem para a **separação de responsabilidades de tabelas**, ex: cadastro, endereço, pedidos;
- Há um link entre as tabelas, que são as **Foreign Keys (FKs)**;
- E há também vários **tipos de relacionamentos**: Um para um, um para muitos e muitos para muitos;



Tipos de relacionamentos

- **One to One (um para um):** Quando uma tabela possui uma conexão com outra e vice-versa;
- **One to Many (um para muitos):** Quando uma tabela possui diversos registros em outra, porém a segunda só pode possuir uma conexão;
- **Many to Many (muitos para muitos):** Quando duas tabelas podem ter conexões com diversos registros entre elas;



One to One

- Com este relacionamento teremos **no máximo um registro ligado a outro**;
- **Exemplo:** Estudante x Informações de Contato;
- Cada estudante pode ter apenas uma informação de contato e a informação de contato é apenas daquele estudante;
- A estrutura é feita por duas tabelas ligadas por uma **FOREIGN KEY**;



One to Many

- Com este relacionamento teremos **uma tabela que possui vários relacionamentos com outra**, mas o inverso não ocorre;
- **Exemplo:** Cliente x Pedido;
- Um cliente pode ter diversos pedidos na loja, porém um pedido é de um único cliente;
- Semelhante ao one to one, porém **a segunda tabela tem diversos registros ligados a um na outra**;



Many to Many

- No Many to Many as duas tabelas tem **múltiplas relações entre si**;
- **Exemplo:** Alunos x Matérias;
- Um aluno pode estar fazendo diversas matérias diferentes e uma matéria pode ter diversos alunos matriculados;
- Normalmente este recurso usa uma **pivot table**;
- Onde esta serve apenas para conter as relações entre tabelas;

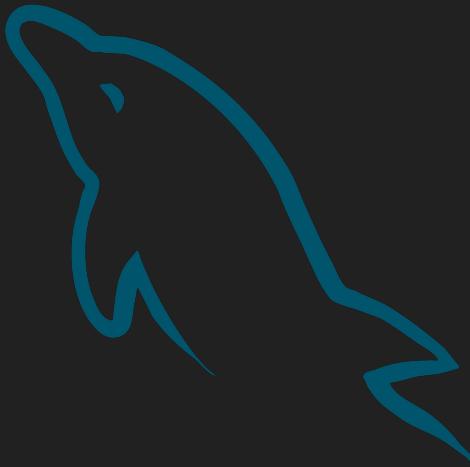




Relacionamento de tabelas

Conclusão da seção





Database Design

Introdução da seção



A importância do DB design

- Aplicações que tem o desenho do banco já iniciam com uma **documentação**;
- E ainda uma **organização prévia**, que dita as regras do sistema e como ele funciona;
- O **relacionamento entre entidades** também é previsto;
- **Erros podem ser resolvidos antes de acontecerem no código** ou ainda antecipação de problemas da própria regra de negócios;



Análise de requisitos

- É o **planejamento** e a **definição do sistema**;
- Onde **como o sistema deve funcionar** é apresentado ou uma conversa com quem precisa do sistema é feita;
- Com base nestes dados e descrições vamos **planejar o banco de dados**;
- Temos um ponto de contato com a parte não técnica e técnica;
- Aqui também podem ser **relatadas dificuldades técnicas** e criam-se alternativas para possíveis problemas;



Normalização

- É dividida em **diversos níveis**, algumas das premissas são:
- Colocar **chave primária** na tabela;
- **1FN** - Colunas guardam um único valor (**atomicidade**);
- **2FN** - Colunas que não pertencem ao tópico central da tabela, **devem virar uma outra tabela**;
- **3FN** - Deixar no banco de dados **apenas valores que não são dependentes de outros**, devem ir p/ outras camadas a responsabilidade;



Diagrama de Entidades Relacionais

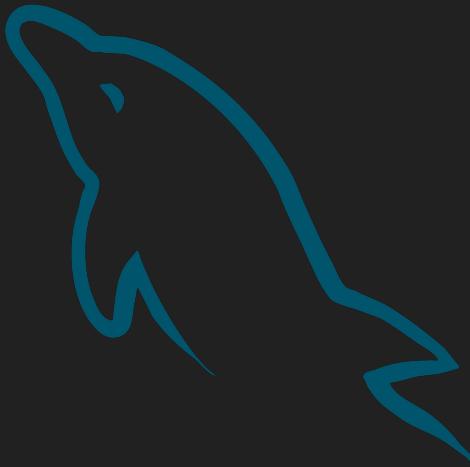
- É um quadro onde se **definem as tabelas e as relações entre si**;
- Deixando o banco de dados **visual** a todos da equipe;
- Utilizado também para **dar nome as tabelas e colunas**;
- Além de também os **tipos de dados** da mesma;
- Com este diagrama pronto o desenvolvimento da aplicação e o entendimento do projeto tornam-se mais rápidos;



Implementação

- Seguindo todas estas instruções **vamos planejar do zero um banco de dados**;
- A partir da análise de requisitos e entendimento de uma solicitação de desenvolvimento de sistema;
- Mão à obra!





Database Design

Conclusão da seção





Stored Procedures

Introdução da seção



O que são as Stored Procedures?

- **Uma query** que pode ser reutilizada;
- O recurso se assemelha a funções de linguagens de programação;
- Pode receber **parâmetros**;
- Há a necessidade do uso de um recurso chamado **DELIMITERS**;
- Para delimitar quando inicia e finaliza a procedure;



Alterando o Delimiter

- Normalmente utilizamos o delimitador o “;”
- **Cada final de query adicionamos**, para que o MySQL entenda o fim da query;
- Para utilizar procedures precisamos **modificar o delimiter padrão**;
- Vamos ver na prática!



Criando uma Procedure

- Além de alterar o Delimiter temos que utilizar outros recursos;
- **BEGIN**: inicia a procedure;
- **END**: finaliza a procedure;
- E claro: **criar a query** que será repetida;
- Podemos chamar a procedure com **CALL**
- Vamos ver na prática!



Listando todas as procedures

- Podemos **checar as procedures** que estão criadas no nosso sistema;
- O comando é **SHOW PROCEDURE STATUS**
- Ele exibe alguns **detalhes importantes** sobre nossas procedures;
- Vamos ver na prática!



Excluindo Procedures

- Podemos também **excluir procedures** do sistema;
- O comando é: **DROP PROCEDURE <nome>**
- Depois da execução a mesma não estará mais disponível;
- Vamos ver na prática!



Verificar dados de uma Procedure

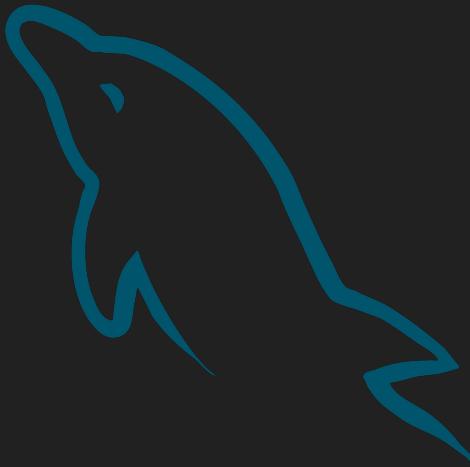
- Podemos analisar como uma procedure foi criada, ou seja, **a query base** dela;
- O comando é: **SHOW CREATE PROCEDURE <nome>**
- Vamos ver na prática!



Procedure com Parâmetros

- Precisamos **colocar os parâmetros após o nome da Procedure**, como se fossem parâmetros de função;
- **Definimos o seu tipo** também (VARCHAR, INT e etc);
- Na hora de chamar a procedure, precisamos **passar o valor do parâmetro**;
- Isso deixa as **procedures dinâmicas**;
- Vamos ver na prática!





Stored Procedures

Conclusão

