Desenvolvimento de Banco de Dados

• Linguagem de Definição de Dados (DDL):
CREATE – cria banco de dados, tabelas, colunas.
DROP – exclui banco de dados
ALTER – altera banco de dados
TRUNCATE – esvazia toda a tabela.

• Linguagem de Manipulação dos Dados (DML):
INSERT – insere dados em uma tabela.
UPDATE – atualiza os dados existentes em uma tabela.
DELETE – exclui registros de uma tabela.

• Linguagem de Consulta a Dados (DQL):
SELECT – principal instrução de consulta do SQL.
SHOW – exibe todas as informações além dos dados (metadata).
HELP – exibe informações do manual de referência do MySQL.

• Linguagem de Controle de Dados (DCL):
GRANT – essa instrução concede privilégios às contas de usuário.
REVOKE – essa instrução permite revogar os privilégios da conta de usuário.

• Linguagem de Transação de Dados (DTL):
START TRANSACTION – inicia uma nova transação.
SAVEPOINT – identifica um determinado ponto em uma transação.
COMMIT – é uma instrução de entrega ao SGBD, todas as alterações sejam permanentes.
ROLLBACK– reverte toda a transação, cancelando todas as alterações ou até determinado ponto da transação.
RELEASE SAVEPOINT – instrução para remoção de um SAVEPOINT.

Abaixo temos os tipos de dados que o MySQL pode armazenar, classificados em 3 diferentes tipos: numéricos, data e hora e texto.

Nomenclatura

- **M:** tamanho do dado
  Seu valor máximo está relacionado ao tipo de dado
- **D:** quantidade de dígitos “depois da vírgula”.
  Valor máximo é 30 (ou M-2)
- **[ ]:** partes opcionais na definição dos tipos.
- **UNSIGNED:** não aceita números negativos
- **ZEROFILL:** preencher com zeros à esquerda. E recebe o atributo UNSIGNED.
- **fsp:** tipos de data e hora.
  Representa a quantidade de dígitos das “frações de segundo” que será armazenada.

Tipos Numéricos:

- **SMALLINT [(M)]** [UNSIGNED] [ZEROFILL]
  Inteiro no intervalo de -32768 a 32767. O intervalo sem sinal é de 0 a 65535.
- **INT [(M)]** [UNSIGNED] [ZEROFILL]
  Inteiro no intervalo de -21473648 a 21473647. O intervalo sem sinal é de 0 a 42949695.
- **BIGINT [(M)]** [UNSIGNED] [ZEROFILL]
  Intervalo de -922337203685808 a 922337205807.
- **FLOAT [(M, D)]** [UNSIGNED] [ZEROFILL]
  Ponto flutuante, de precisão simples. Os valores admissíveis são -3,402823466E+38 a -1,175494351E38, 0 e 1,175494351E-38 a 3,402823466E+38.
- **DOUBLE [(M, D)]** [UNSIGNED] [ZEROFILL]
  Ponto flutuante de precisão dupla. -1,7976931348623157E+308 a -2,2250738585072014E-308,0 Sinônimo: DOUBLE PRECISION.

Tipo Data e hora:

- **DATE**
  MySQL exibe no formato 'YYYY-MM-DD’
  Permite números ou strings
- **DATETIME** [(fsp)]
  Combinação de data e hora
  Formato 'AAAA-MM-DD HH: MM: SS [fracao]’
- **TIMESTAMP** [(fsp)], TIME [(fsp)], YEAR [4]
  Os valores de TIMESTAMP são armazenados como o número de segundos desde a época ('1970-01-01 00:00:00' UTC).
- **TIME** [(fsp)]
  O MySQL exibe valores “TIME” no formato 'HH: MM: SS [fração]';
- **YEAR [(4)]** - Um ano no formato de quatro dígitos.
  O MySQL exibe valores YEAR (ano) no formato YYYY

Tipo Texto

• **CHAR [(M)]-** [CHARACTER SET charset_name]
Uma cadeia de comprimento fixo que é sempre preenchida à direita com espaços para o comprimento especificado quando armazenada.
• **VARCHAR (M)-** [CHARACTER SET charset_name]
Cadeia de comprimento variável.
• **ENUM-** ('valor1', 'valor2', ...) [CHARACTER SET charset_name]
. Objeto de string que pode ter apenas um valor, escolhido na lista de valores 'valor1', 'valor2', [...], NULO, ou vazio.
Armazenado como inteiro pelo banco.
• **TEXT-** TEXTO/DESCRIÇÃO
• **BINARY [(M)]-** O tipo BINARY é semelhante ao tipo CHAR, mas armazena cadeias de bytes binários em vez de cadeias de caracteres não binários.
• **VARBINARY (M)-** O tipo VARBINARY é semelhante ao tipo VARCHAR, mas armazena cadeias de bytes binários em vez de cadeias de caracteres não binários. M representa o comprimento máximo da coluna em bytes.
• **TINYBLOB-** Uma coluna “BLOB” com um comprimento máximo de 255 (28 - 1) bytes. Cada valor TINYBLOB é armazenado usando um prefixo de 1 byte.
• **BLOB [(M)]-** Uma coluna “BLOB” com um comprimento máximo de 65.535 (216 - 1) bytes. Cada valor BLOB é armazenado usando um prefixo de 2 bytes

Ex 1: Criar um Banco de Dados

```
create database cadastro;
```

Criando uma tabela

```
create table if not exists teste(    # criar tabela se não existir
  id int not null auto_increment,    # (1, 2, 3...)
  nome varchar(10) not null,         # não pode ser nulo (obrigatório)
  nascimento date,
  sexo enum ('F', 'M'),
  altura decimal (3,2),          # (default)- se ninguém colocar nada, padrão o Brasil
  nacionalidade varchar (30) default 'Brasil'
)default charset = utf8;         #designa um conjunto de símbolos
```

Inserindo dados na tabela

```
insert into teste value    # valores
('1', 'João', '1998-20-04', 'M', '1.70', 'Japão'),       # sempre entre parentes simples
('2', 'Maria', '1999-03-12', 'F', '1.60', 'Portugal');   #decimal com ponto, nao virgula
```

Mostrar todas a tabela

```
select * from teste;
```

Ex 2: Criar um Banco de Dados

```
create table cursos(
idcurso int not null,
nome varchar (30),
descricao text,
carga int unsigned,        # (unsigned) não aceita números negativos
totaulas int unsigned
)default charset = utf8;
```

Inserir dados na tabela

```
insert into cursos values
('1', 'HTML4', 'curso HTML5', '37', '10'),
('2', 'PGP', 'curso de PHP', '40', '20'),
('3', 'Jarva', 'Linguagem Java', '10', '29');
```

Alterando campos

```
update cursos                  # update = atualiza
set nome = 'HTML5'             # set = configura
where idcurso = 1;             # where = onde
```

```
update cursos
set nome = 'PHP'               # ja adicionando os nomes corretos
where idcurso = 2;
```

```
update cursos
set nome = Java, carga = 40
where idcurso = 3
limit = 1;       # limit = segurança, alterar somente 1
```

Deletar uma linha da tabela

```
delete from cursos
where idcurso = 2;       # vai deletar o id 2
```

Apagar dados da tabela, mas não a estrutura

```
truncate table cursos;
```

Para excluir a tabela toda

```
drop table if not exists cursos;
```

Para adicionar nova coluna

```
alter table cursos
add column ano int(4);
```

Para adicionar nova coluna numa posição específica

```
alter table cursos
add column ano int(4) after nome;    #after - depois do nome
```

Para adicionar nova coluna na 1º posição

```
alter table cursos
add ano int(4) first;                    #first - primeira posição
```

Para alterar estrutura da definição

```
alter table cursos                    #modificar o tipo primitivo do campo
modify column ano int(5);             e todos os constraints (int(4) -> int(5))
```

Para renomear o nome da coluna

```
alter table cursos
change ano anos int(5);        #mudar o nome antigo(ano), para o novo (anos)
```

Para renomear a tabela inteira

```
alter table cursos
rename to cursos_guanabara;
```

Order by (as linhas serão atualizadas na ordem específica)

```
select * from cursos
order by nome;  ou                  #os nomes vão ficar na
order by nome asc;                    ordem crescente (A - Z)
ou
order by nome desc;                 # nomes vão ficar na decrescente (Z - A)
```

Group by (agrupando os valores iguais, os resultados das seleções podem ser organizados em grupos, baseados no conteúdo existente em uma ou mais colunas)

```
select carga, count(nome) from cursos
group by carga       #agrupar os valores iguais de carga
order by carga;      #deixando a carga na ordem crescente
```

Having (tendo somente - dentro do group by)

```
select carga, count(nome) from cursos
group by carga
having count(nome) > 3;  #vai mostrar somente quem tem o contador maior que 3
ou
select ano, count(*) from cursos
where totatulas > 30          #onde totaulas é maior que 30
group by ano                  #agrupando por ano
having ano > 2013             #mostrando somente o ano acima de 2013
order by count(*) asc;        #ordenando o total na ordem crescente
```

Select (filtrar **colunas**)

```
select nome, carga, ano from cursos  # vai mostrar somente estes 3 campos
order by nome;
ou
order by ano, nome;    # ano vai ficar na ordem crescente em seguida o nome
```

Select (filtrar **linhas**)

```
select * from cursos
where ano = '2016'        # vai mostrar somente os cursos de 2016
order by nome;            # com nomes na ordem crescente
where ano <= '2015'       # onde o ano é menor ou igual a 2015
```

Between - and (entre uma coisa e outra)

```
select nome, ano from cursos
where ano between '2014' and '2016'    # vai mostrar o ano entre 2014 e 2016
order by nome,
ou
order by ano desc, nome asc;
```

In (selecionar valores específicos)

```
select idcurso, nome from cursos
where ano in ('2014', '20161', '2018'),  #só vai mostrar os cursos desses anos
ou
where carga > 35 and totaulas < 30,
                  #mostrando carga maior que 35, e totaualas menor que 30
order by nome;
```

Like (%)

```
select * from cursos
where nome like 'P%',          #nomes que começam com a letra P
where nome like 'Sor%',        #nomes que começam com a letra Sor
where nome like '%A',          #nomes que terminam com a letra A
where nome like '%O%',         #nomes que tem a letra O
where nome not like '%O%',     #nomes que não tem a letra O
where nome like 'PH%P',        #começa com PH e termina com P
where nome like 'PH%P%',       #começa com PH, e q tem mais letras depois de P
where nome like 'PH%P_';       #obrigatório ter carácter depois de P
```

Distinct (não repete os mesmos nomes, ano..)

```
select distinct carga from cursos;
select distinct nome from teste;
```

Count (mostra a quantidade total)

```
select count(*) from cursos;   # vai mostrar a quantidade de cursos
ou
select count(*) from cursos    # vai contar quantos cursos
where carga > 40;              # carga é maior que 40
```

Max (o máximo dentro da tabela)

```
select max(totaulas) from cursos;
ou
select max(totaulas) from cursos where ano = '2016';
```

Min (o mínimo dentro da tabela)

```
select min(cargas) from cursos;
ou
select min(nome) from cursos
```

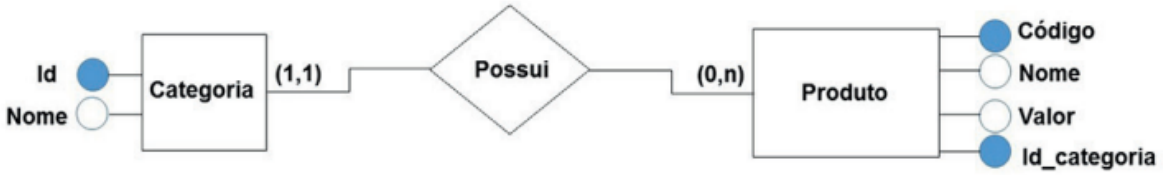
Sum (vai somar)

```
select sum(cargas) from cursos;
ou
select sum(ano) from cursos
```

AVG (tirar a média)

```
select avg(totaulas) from cursos;
```

Modelo Relacional



Criando uma tabela

```
create table if not exists gafanhotos(
id int,
nome varchar(30),
profissao varchar(30)
)default charset = utf8;
```

Adicionar uma nova coluna

```
alter table gafanhotos
add column cod_cursospreferido int;
```

Adicionando chave primária

```
alter table gafanhotos
add constraint pk_gafanhotos
primary key (id);
```

Adicionando chave estrangeira

```
alter table gafanhotos
add constraint fk_gafanhotos
foreign key (cod_cursopreferido)
references cursos(idcurso);
```

OBS: quando se cria chave estrangeira, ela tem que ser do mesmo tipo primitivo, da chave primária pela qual vai se referir.

Adicionando um curso preferido para cada gafanhoto

```
update gafanhotos set cod_cursopreferido = '6'
where id = 1;                    #o aluno 1 vai preferir o curso 6
```

OBS: nao pode apagar(deletar) mais o curso 6, pois tem relação.

Cláusula as

Renomear um campo, sendo atributo ou tabela:

nome-antigo as nome-novo

Ou para transformar em apelido:

cursos as c, gafanhotos as g

Qualificadores

- CASCADE: qualificador que exclui ou atualiza a linha da tabela pai e exclui ou atualiza automaticamente as linhas correspondentes na tabela filha.
- SET NULL: exclui ou atualiza a linha da tabela pai e define como NULL a coluna ou colunas de chave estrangeira na tabela filha.
- SET NULL e ON UPDATE SET NULL: são suportados
- RESTRICT: rejeita a operação de exclusão ou atualização da tabela pai.
- NO ACTION: essa é uma palavra-chave do SQL padrão. No MySQL, a equivalente é a RESTRICT

Join (junção - unir duas ou mais tabelas. Uma premissa para se garantir a eficiência em sua utilização é a necessidade de que as tabelas existentes no banco de dados estejam normalizadas.)

```
select gafanhotos, nome, cod_cursopreferido, cursos.ano from gafanhotos as g
join cursos as c      #juntando o gafanhotos e cursos
on c.idcurso = g.cod_cursopreferido;
    #fazendo a relação entre a chave primária com a estrangeira
```

Criando nova tabela

```
create table gafanhotos_assiste_curso(
id int not null auto_increment,
data date,                          #colocando o idgafanhoto como chave primária da
idgafanhoto int,                     tabela gafanhotos, e o idcurso da tabela curso.
idcurso int,
primary key (idgafanhoto) references gafanhotos(id),
primary key (idcurso) references curso(idcurso)
)default charset = utf8;
```

Inserindo dados

```
insert into gafanhoto_assiste_curso values
(default, '2014-03-01', '1', '2')
(default, '2015-03-04', '2', '5')
```

Inner Join (é um tipo de junção interna)

```
select g.nome, g.id, c.nome, a.idgafanhoto, a.idcurso from gafanhotos as g
join gafanhoto_assiste_curso as a   #juntar o gafanhoto com gafanhoto_assiste_curso
on g.id = a.idgafanhoto             #juntar a chave primária com estrangeira
join cursos as c                    #juntar o gafanhoto com cursos
on c.idcurso = a.idcurso;           #juntar a chave primária com estrangeira
```

OBS: Juntando 3 entidades com o Join

Left Join (é um tipo de junção externa)

```
select g.nome, g.id, c.nome, c.idcurso from cursos as c
left join gafanhotos as c
on c.idcurso = g.id;
```

OBS: as linhas da tabela da esquerda são projetadas na seleção juntamente com as linhas não combinadas da tabela da direita. Ou seja, como resultado dessa seleção, algumas linhas em que não haja relacionamento entre as tabelas da esquerda para a direita retornarão o valor nulo (NULL).

Right Join (é um tipo de junção externa)

```
select g.nome, g.id, c.nome, c.idcurso from gafanhotos as g
right join gafanhotos as c
on c.idcurso = g.id;
```

OBS: similar ao comando LEFT JOIN, com o comando RIGHT JOIN as linhas da tabela da direita são projetadas na seleção juntamente com as linhas não combinadas da tabela da esquerda.

Full Join (é um tipo de junção externa)

View (conceitos de visões)

O recurso SQL para gerar visões é uma alternativa para visualizar os dados de uma ou mais tabelas de um BD. Ao utilizar uma VIEW para efetuar seleção de dados, torna as consultas mais rápidas, e exige uma carga de processamento menor.

```
CREATE VIEW [nome_da_VIEW] AS
SELECT [coluna] FROM [tabela 1]
INNER JOIN [tabela 2]
WHERE [condições];
```

Ex1:

```
create view cadastro as
select nome, profissao, sexo, nacionalidade from gafanhotos
inner join cursos
where cursos.idcurso = gafanhotos.cod_cursopreferido;
```

Para exibir uma consulta

```
select * from cadastro;
```

Para excluir uma VIEW

```
drop view cadastro;
```

Ex2: criei um BD livros, 2 tabelas, e fiz um VIEW entre eles

```
create database livros;

create table livros(
id_livro int,
nome_Livro varchar(30),
quantas_pags int,
primary key (id_livro)
)default charset = utf8;

create table autores(
id_autor int,
nome_autor varchar(30)
)default charset = utf8;

alter table autores
add constraint fk_autores
foreign key (id_autor)
references livros(id_livro);

create view livroos as
select id_livro, nome_Livro as livro, id_autor, nome_autor as autor from livros
inner join autores
on id_livro = id_autor;
```

Index (índice)

O recurso SQL para aumentar a velocidade das consultas nos bancos de dados é a utilização de índices. A utilização dos índices é opcional para a seleção de dados, pois os índices são considerados estruturas redundantes. Nas buscas nas tabelas e para imposição das restrições de integridade.

Declarar um índice no desenvolvimento da tabela

```
CREATE TABLE [nomeDaTabela] (
Campo1 tipo(tamanho),
Campo2 tipo(tamanho),
INDEX(Campo1) );
```

Declarar um índice, em uma tabela existente no BD

```
CREATE TABLE [nomeDoIndice] ON
[nomeDaTabela](Campo);
```

Para nos certificarmos de que os índices foram criados

```
SHOW INDEX FROM [nomeDaTabela];
```

Para excluir um índice, a sintaxe utilizada pode ser observada a seguir

```
DROP INDEX (nomeDoIndice);
```

FullText

Esse recurso tem a capacidade de buscar um trecho dentro de várias strings, assim como a função “localizar” existente nos navegadores de internet, editores de texto, etc.

Sintaxe

```
ALTER TABLE [nome_tabela] ADD FULLTEXT
(nome_da_coluna);
```

Nesse comando, ao especificar uma determinada coluna como FULLTEXT, a mesma passa a ter as strings no interior de um texto, monitoradas.

```
SELECT [coluna] FROM nome_da_tabela
WHERE MATCH(coluna) AGAINST(‘palavra_desejada');
```

Isso permite buscar palavras dentro de longos textos, em que:

• **MATCH**(coluna): tem a função de informar ao sistema de gerenciamento de banco de dados a coluna na qual deve ser usada na consulta do FULLTEXT;

• **AGAINST**(‘palavra_desejada’): tem como objetivo informar ao sistema de gerenciamento de banco de dados a palavra chave que deve ser buscada no FULLTEXT.