

# SQL Tutorial



Introdução à Linguagem de  
Consulta Estruturada

# Objetivos

---

Entender a Linguagem de Definição de Dados (DDL)

Entender a Linguagem de Manipulação de Dados (DML)

- SELECT, INSERT, UPDATE, DELETE

Utilizar a declaração **SELECT FROM WHERE** para construção de consultas

- Usar funções de agregação (SUM, COUNT, AVG, ...)

# Linguagem de Definição de Dados

---

Permite especificar as entidades e relacionamentos, bem como os atributos das entidades, incluindo:

- O banco de dados (schema) que contem as entidades e relacionamentos
- Os tipos de dados (domain) de cada atributo
- As restrições impostas às entidades (chave primaria), aos relacionamentos (chaves estrangeiras) e aos atributos (nulidade), que matêm a integridade do banco de dados

# Tipos de Dados em SQL

---

Tipo	Descrição
CHAR(n)	Cadeia de caracteres de tamanho fixo. Tamanho especificado por <b>n</b>
VARCHAR(n)	Cadeia de caracteres de tamanho variável. Tamanho especificado por <b>n</b>
INTEGER	Inteiro. Ocupa 4 Bytes (32 bits) e permite mais de 4 bilhões de números distintos
SMALLINT	Inteiro. Ocupa 2 Bytes (16 bits) e permite 65536 números distintos
FLOAT(M,D)	Número de ponto flutuante (decimal). Total de M dígitos com D dígitos decimais.
DOUBLE(M,D)	Número de ponto flutuante com precisão dupla.

- Similar aos tipos de dados utilizados nas linguagens de programação.

# CREATE DATABASE

---

Um banco de dados (schema) no MySQL é criado com o comando **CREATE DATABASE**:

- **create database** [*database name*]

Exemplo

- **create database** *banco*

# CREATE TABLE

---

Um tabela (entidade) no SQL é criada utilizando o comando **CREATE TABLE**:

- **create table** [*tablename*] ( $A_1 T_1, A_2 T_2, \dots A_n T_n,$   
    (*integrity-constraint*<sub>1</sub>),  
    ...,  
    (*integrity-constraint*<sub>k</sub>));
- $A_i$  é um campo (atributo) da tabela.
- $T_i$  é o tipo de dado (domain) do campo  $A_i$

## Exemplo

```
create table student (  
    flashlineID    char(9) not null,  
    name           varchar(30),  
    age            integer,  
    department     varchar(20),  
    primary key (flashlineID);
```

Constraint

# DROP e ALTER TABLE

---

- DROP TABLE exclui a tabela e todos os dados contidos nela.

```
drop table tabela <ação>
```

- **ALTER TABLE** é usado para adicionar, modificar ou excluir campos das tabelas:

```
alter table tabela <ação>
```

Ação pode ser:

```
add campo
```

```
drop campo
```

```
add primary key (campo,...)
```

```
drop primary key
```

# Modificando o Banco de Dados

---

## Comandos:

Incluir registro	<b>INSERT INTO</b> <i>tabela</i> <b>VALUES</b> ( $Val_1$ , $Val_2$ , ... , $Val_n$ )
Alterar registro(s)	<b>UPDATE</b> <i>tabela</i> <b>SET</b> $A_1=val_1$ , $A_2=val_2$ , ..., $A_n=val_n$ <b>WHERE</b> <i>predicado</i>
Excluir registro(s)	<b>DELETE</b> <b>FROM</b> <i>tabela</i> <b>WHERE</b> <i>predicado</i>



# Inclusão

---

Adicionar um novo registro na tabela ***student***

```
insert into student
```

```
values ('999999999', 'Mike', 18, 'Math');
```

```
insert into student (flashlineID, name, age,  
department)
```

```
values ('999999999', 'Mike', 18, 'computer science');
```

Adicionar uma novo registro na tabela ***student*** com campo ***age*** preenchido com **null**

```
insert into student
```

```
Values ('999999999', 'Mike', null, 'Math');
```

# Alteração

---

Atualizar todos os departamentos para 'computer science' da tabela **student**

```
update student  
    set department = 'computer science';
```

Na tabela **conta** (*numero\_conta*, *nome\_agencia*, *saldo*), aumente o valor do saldo de todas as contas em 6%

```
update conta  
    set saldo = saldo * 1.06;
```

# Exclusão

---

Excluir todos os registros da tabela **students**

```
delete from student;
```

Excluir todos os registros da tabela **students** em que o departamento seja computer science

```
delete from student
```

```
where department = 'computer science';
```

# Consultas (Query)

---

Uma típica consulta em SQL tem a forma:

```
select  $A_1, A_2, \dots, A_n$   
from  $tabela_1, tabela_2, \dots, tabela_m$   
where  $R$ 
```

- $A_i$  são os atributos (campos)
- $tabela_i$  são as tabelas
- $R$  são as restrições (condições da consulta)

A consulta é equivalente a seguinte expressão em algebra relacional:

$$\prod_{A_1, A_2, \dots, A_n} (\sigma_R(tabela_1, tabela_2, \dots, tabela_m))$$

Example

```
select flashlineID, name  
from student  
where department = 'computer science'
```

# SELECT – Registros Duplicados

---

SQL não remove automaticamente os registros duplicados no resultado da consulta.

Para eliminar as duplicidades devemos utilizar a palavra reservada **distinct** após o **select**.

- Exemplo: Encontrar os nomes de todos os alunos da universidade e remover as duplicidades

```
select distinct name  
from student;
```

# SELECT – Expressão *as*

---

Um asterisco em um comando **select** retorna todos os campos da tabela.

```
select * from student
```

Um campo pode receber um apelido utilizando a expressão **as**

- Exemplo

```
select FlashlineID as ID  
from student
```

**Nota:** **as** também pode ser usado para apelidar as tabelas

```
select s.name as myname  
from student as s
```

# WHERE

---

A cláusula **WHERE** define as condições (restrições) que satisfazem a consulta

- Podem utilizar os operadores de comparação e lógicos:
  - Operador de comparação: <, <=, >, >=, =, <>
  - Operador lógico: and, or, not
- Exemplo: Encontrar os nomes dos estudantes de computer science department com idade inferior a 18 anos

```
select names
from student
where department = 'computer science'
and age < 18;
```

# ORDER BY

---

A cláusula **ORDER BY** ordena o resultado de uma consulta

- Podem utilizar os operadores ASC (ascendente) ou DESC (descendente), ou ainda, uma combinação dos dois.
- Exemplo: Encontrar o nome e a idade dos estudantes do departamento de computer science com idade inferior a 18 anos. Ordenar o resultado por nome ascendente e idade decendente.

```
select names, age
from student
where department = 'computer science'
and age < 18
order by name asc, age desc;
```



# Agregação

---

As funções de agregação operam nos campos de valor de um conjunto de registros e retornam o valor computado com os dados

**avg**(*campo*): valor médio

**min**(*campo*): campo com menos valor

**max**(*campo*): campo com o maior valor

**sum**(*campo*): soma dos valores dos campos

**count**(*campo*): quantidade de registros no conjunto de dados

Para remover as duplicidades de valor no campo do conjunto de dados, utilize a palavra **distinct** antes do campo:

**avg**(**distinct** *campo*)

# Agregação – Valor NULL

---

Encontrar o estudante mais novo da universidade

```
select  *  
  from  student  
 where  age = min(age)
```

- A consulta acima ignora os valores do campos **age** iguais a **NULL**
- O resultado será **NULL** se todas as idades estiverem com valor **NULL**

As funções de agregação, com exceção de **count**, ignoram os campos com valor igual a **NULL**.

# Agregação – GROUP BY

---

- **GROUP BY** opera da seguinte forma
  1. Agrupa os dados do conjunto em subconjuntos de campos do **select**
  2. Agrega os valores de cada subconjunto
  3. Retorna o valor agregado para os subconjuntos de campos
- Exemplo: Listar cada departamento e a quantidade de estudantes em cada um deles.

```
select department, count(name) as number  
  from student  
group by department
```

**Nota:** Se um **select** contem uma função de agregação, então todos os campos que não participam da agregação têm que estar relacionados no **group by**.  
No exemplo acima, como department não participa da função de agregação, então ele tem que estar no **group by**.

# Agregação – HAVING

---

A cláusula **HAVING** filtra (impõe uma condição) o resultado de uma agregação.

Exemplo: Listar cada departamento e a quantidade de estudantes em cada um deles, somente para os departamentos com mais de dois estudantes.

```
select department, count(name) as number  
  from student  
 group by department  
 order by department  
having count(distinct name) > 2
```

# Junções - Join

---

As junções entre duas ou mais tabelas podem ser realizadas através de:

- Cross join
- Inner join
- Left outer join
- Right outer join
- Union (Outer Full join)




# Junções - Join

Para definir e exemplificar as junções acima citadas vamos criar as tabelas a seguir:




```
CREATE TABLE Funcionario (  
  Matricula int not null,  
  NomeFuncionario varchar(50),  
  Codcargo char(2),  
  primary key (Matricula),  
  foreign key (Codcargo)  
  references cargo (Codcargo)  
) Engine = InnoDB;
```

```
CREATE TABLE Cargo (  
  Codcargo char(2) not null,  
  Nomecargo varchar(50),  
  Valorcargo double null,  
  primary key (codcargo)  
) Engine = InnoDB;
```

**FUNCIONARIO**

 Matricula	INTEGER	NOT NULL
 NomeFuncionario	VARCHAR(80)	NOT NULL
 CodCargo (FK)	CHAR(2)	NULL

**CARGO**

 CodCargo	CHAR(2)	NOT NULL
 NomeCargo	VARCHAR(50)	NOT NULL
 ValorCargo	MONEY(10,0)	NOT NULL



# Junções - Join

Insira os seguintes registros nas tabelas:

```
SELECT * FROM CARGO AS C --> Apelidamos a tabelas Cargo de C
SELECT * FROM FUNCIONARIO AS F --> Apelidamos funcionário de F
```

Results Messages

	CodCargo	NomeCargo	ValorCargo
1	C1	CAIXA	800,00
2	C2	VENDEDOR	1200,00
3	C3	GERENTE	2400,00

**CARGO AS C**

	Matricula	NomeFuncionario	CodCargo
1	100	JOÃO	C1
2	110	MARIA	C2
3	120	CARLOS	C1
4	130	TADEU	NULL

**FUNCIONARIO AS F**

# Junções – Cross Join

---

Quando queremos juntar duas ou mais tabelas por **cruzamento** (produto cartesiano).

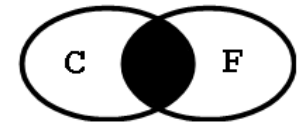
No nosso exemplo, para cada linha da tabela FUNCIONARIO queremos todos os CARGOS ou vice-versa.

```
SELECT F.NomeFuncionario, C.NomeCargo
FROM CARGO AS C
CROSS JOIN FUNCIONARIO AS F
```



# Junções – Inner Join

---



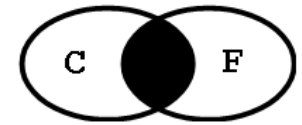
Quando queremos juntar duas ou mais tabelas por coincidência.

No nosso exemplo, para cada linha da tabela FUNCINARIO queremos o CARGO correspondente que internamente (INNER), em seus valores de atributos, coincidam.

No caso de FUNIONÁRIO e CARGO os atributos internos coincidentes são **codigoCargo** na tabela **CARGO** e **codigoCargo** na tabela **FUNCIONARIO**.

# Junções – Inner Join

---

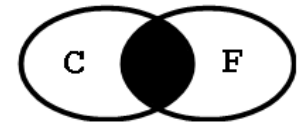


Lembrem que **codigoCargo** é chave primária (**PK**) da tabela CARGO e chave estrangeira (**FK**) na tabela FUNCIONARIO.

Para efetivarmos a junção das duas tabelas se fará necessário ligar (**ON**) as duas tabelas por seus atributos internos (INNER) coincidentes.

# Junções – Inner Join

---

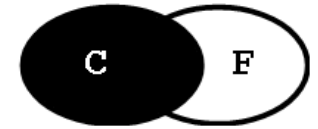


No nosso exemplo, para cada linha da tabela CARGO queremos todos os FUNCIONARIOS.

```
SELECT F.NomeFuncionario, C.NomeCargo
      FROM CARGO AS C
      INNER JOIN FUNCIONARIO AS F
            ON F.CodCargo = C.CodCargo;
```

# Junções – Left Outer Join

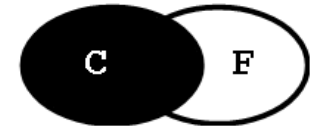
---



E se desejarmos listar todos os funcionários com seus respectivos cargos, incluindo os funcionários sem cargos?

Podemos conseguir esse feito com a junção FUNCIONARIO/CARGO através da declaração `FUNCIONARIO OUTER LEFT JOIN CARGO`, que promove a junção interna (INNER) de todos os funcionários a cargos e lista ainda outros (EXTERNOS/OUTER) não associados.

# Junções – Left Outer Join



Uma observação importante é que a ordem da ligação (ON) não faz diferença, ou seja:

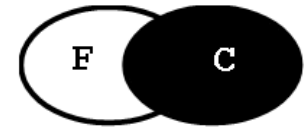
“ON F.codCargo = C.codCargo”  
é exatamente igual a

- “ON C.codCargo = F.codCargo”

```
SELECT F.NomeFuncionario, C.NomeCargo
FROM FUNCIONARIO AS F
LEFT OUTER JOIN CARGO AS C
ON C.CodCargo = F.CodCargo;
```

# Junções – Right Outer Join

---



Se desejarmos listar todos os CARGOS e seus respectivos FUNCIONARIOS, incluindo os CARGOS sem FUNCIONÁRIOS, usamos a junção RIGHT OUTER JOIN.

Novamente a ordem da ligação (ON) não faz diferença, ou seja:

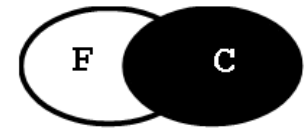
“ON F.codCargo = C.codCargo”

é exatamente igual a

“ON C.codCargo = F.codCargo”

# Junções – Right Outer Join

---



```
SELECT F.NomeFuncionario, C.NomeCargo
      FROM FUNCIONARIO AS F
      RIGHT OUTER JOIN CARGO AS C
        ON C.CodCargo = F.CodCargo;
```

# Junções – Union



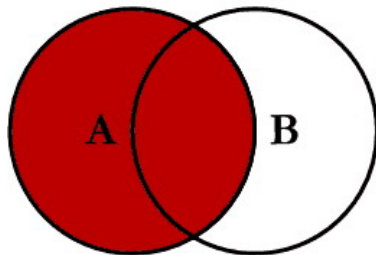
Aqui juntamos o resultado das junções (JOIN) internas (INNER) à listagem de todas as outras linhas não associadas, tanto do lado direito (RIGHT) como do lado ESQUEDO (LEFT) da junção.

```
SELECT F.NomeFuncionario, C.NomeCargo
      FROM FUNCIONARIO AS F
      LEFT OUTER JOIN CARGO AS C
            ON C.CodCargo = F.CodCargo
UNION
SELECT F.NomeFuncionario, C.NomeCargo
      FROM FUNCIONARIO AS F
      RIGHT OUTER JOIN CARGO AS C
            ON C.CodCargo = F.CodCargo
```

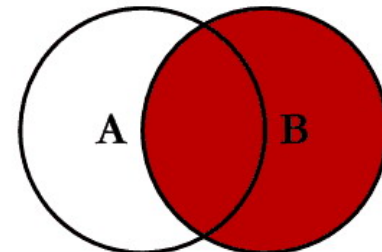


# Junções – Union

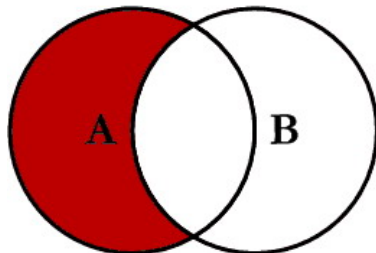
## SQL JOINS



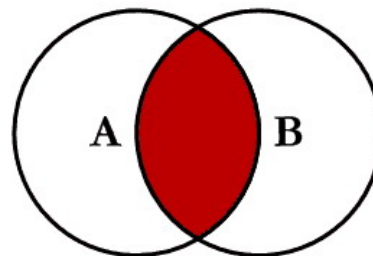
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



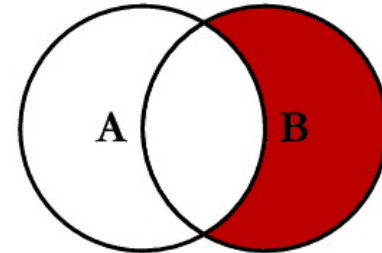
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



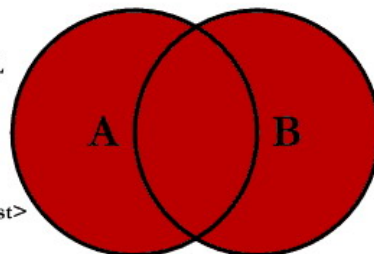
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



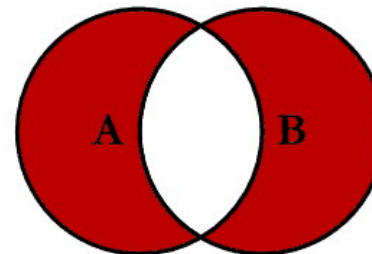
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

# Funções

---

- LIMIT
- LIKE
- AVG
- COUNT
- MAX
- MIN
- SUM
- ROUND
- NOW

# Função - LIMIT

---

LIMIT – Especifica a quantidade de registros que serão retornados.

```
SELECT F.*  
    FROM FUNCIONARIO AS F  
LIMIT 2;
```

# Função - LIKE

---

LIKE – Operador de comparação baseado em um padrão.

```
SELECT F.*  
  FROM FUNCIONARIO AS F  
 WHERE F.NomeFuncionario LIKE 'J%';
```

```
SELECT F.*  
  FROM FUNCIONARIO AS F  
 WHERE F.NomeFuncionario LIKE '%A';
```

```
SELECT F.*  
  FROM FUNCIONARIO AS F  
 WHERE F.NomeFuncionario LIKE '%R%';
```

# Função - COUNT

---

COUNT - Retorna a quantidade de registros.

```
SELECT COUNT (C.CODCARGO)  
FROM CARGO AS C;
```

# Função - AVG

---

AVG – Média dos valores de uma coluna do tipo numérica.

```
SELECT AVG (C.VALORCARGO)  
FROM CARGO AS C;
```

# Função - MAX

---

MAX - Retorna o maior valor de uma coluna.

```
SELECT MAX (C.VALORCARGO)  
FROM CARGO AS C;
```

# Função - MIN

---

MIN – Retorna o menor valor de uma coluna.

```
SELECT MIN (C.VALORCARGO)  
FROM CARGO AS C;
```



# Função - SUM

---

SUM – Retorna a soma dos valores de uma coluna.

```
SELECT SUM(C.VALORCARGO)  
FROM CARGO AS C;
```

# Função - ROUND

---

**ROUND** – Retorna o valor de uma coluna arredondado.

```
SELECT ROUND (AVG (C.VALORCARGO) )  
FROM CARGO AS C;
```

```
SELECT ROUND (AVG (C.VALORCARGO) , 2)  
FROM CARGO AS C;
```

# Função – NOW

---

NOW() – Retorna a data atual no MYSQL.

```
SELECT NOW ()  
FROM CARGO AS C;
```

# Exercícios

---

- 1) Utilizando as tabelas Cargo e Funcionario:
  - a) Crie o banco de dados EMPRESA.
  - b) Crie as tabelas CARGO e FUNCIONARIO no banco de dados EMPRESA.
  - c) Crie os registros do **slide 23** nas tabelas CARGO e FUNCIONARIO.
  - d) Faça uma seleção dos funcionários que começam com "J" e que o cargo seja "CAIXA".
  - e) Exiba os dois maiores valores de cargo em ordem decrescente.
  - f) Exiba os funcionários ordenados pelo nome.
  - g) Crie dois novos cargos "ESTOQUISTA" e "CONTADOR", com valor do cargo de 1000 e 2000, respectivamente.
  - h) Crie três novos funcionários, "PEDRO PAULO", "FLAVIO" e "MARCOS" para o cargo "ESTOQUISTA".

# Exercícios - Continuação

---

- i) Crie uma nova funcionária "CAMILA" para o cargo "CONTADOR".
- j) Aplique um aumento de 6% para o cargo "CAIXA".
- k) Exiba a quantidade de funcionários por cargo.
- l) Exiba todos os funcionários que possuam a letra "E" no nome.
- m) Exiba todos os funcionários e seus respectivos cargos que tenham o valor do cargo abaixo da média dos valores de todos os cargos.
- n) Exiba todos os funcionários que tenham o maior valor de cargo.
- o) Exiba todos os funcionários que tenham o menor valor de cargo.
- p) Apague todos os registros das tabelas CARGO e FUNCIONARIO.
- q) remova as tabelas CARGO e FUNCIONARIO.
- r) remova o banco de dados EMPRESA.