

R

Banco de  
Dados

# Banco de Dados

## Aula 3

# Linguagem SQL

## Select – Funções de Agregação

### SUM

Soma um campo numérico.

**Sintaxe:** SELECT SUM(Nome\_Campo) FROM Nome\_Tabela;

### AVG

Calcula o valor médio de um conjunto de valores para um campo numérico.

**Sintaxe:** SELECT AVG(Nome\_Campo) FROM Nome\_Tabela;

### COUNT

Conta a quantidade de dados para um campo.

**Sintaxe:** SELECT COUNT(Nome\_Campo) FROM Nome\_Tabela;



# Linguagem SQL

## MAX

Mostra o maior valor em um conjunto de dados para um campo.

**Sintaxe:**

```
SELECT MAX(Nome_Campo) FROM Nome_Tabela;
```

## MIN

Mostra o menor valor em um conjunto de dados para um campo.

**Sintaxe:**

```
SELECT MIN(Nome_Campo) FROM Nome_Tabela;
```



# Linguagem SQL

**Curso**

CodC	NomeC	DuracaoC	MensC
C1	Análise sist.	4	400
C2	Eng. mecatrônica	5	600
C3	Ciência comp.	4	450
C4	Eng. elétrica	4	600
C5	Turismo	3	350

**Disciplina**

CodD	NomeD	CargaD	AreaD	PreReqD
D1	TLP1	2	Computação	D2
D2	Cálculo 1	4	Matemática	null
D3	Inglês	2	Humanas	null
D4	Ed. física	3	Saúde	null
D5	G. analítica	5	Matemática	D2
D6	Projeto final	6	null	D1

**Exemplo:** valor mínimo, máximo e médio das mensalidades dos cursos cuja duração é de 4 anos.

```
SELECT MIN(MensC), MAX(MensC), AVG(MensC)
FROM Curso
WHERE DuracaoC = 4
```

# Linguagem SQL

## Agrupamentos

### Cláusula GROUP BY

Permite agrupar o conteúdo por uma ou mais colunas.

Id	Nome	Fabricante	Quantidade	VUnitario	Tipo
1	Playstation 3	Sony	100.00	1999.00	Console
2	Core 2 Duo 4GB Ram 500GB HD	Dell	200.00	1899.00	Notebook
3	Xbox 360 120GB	Microsoft	350.00	1299.00	Console
4	GT-I6220 Quad Band	Samsung	300.00	499.00	Celular
5	iPhone 4 32GB	Apple	50.00	1499.00	Smartphone
6	Playstation 2	Sony	100.00	399.00	Console
7	Sofá Estofado	Coréia	200.00	499.00	Sofá
8	Armário de Serviço	Aracaju	50.00	129.00	Armário
9	Refrigerador 420L	CCE	200.00	1499.00	Refrigerador
10	Wii 120GB	Nintendo	250.00	999.00	Console

# Linguagem SQL

**Exemplo:** Obter o número de produtos em estoque, agrupados pelo tipo, para que depois seja feita a soma da quantidade existente em cada um dos grupos. Para isso usamos a função SUM() em conjunto com o GROUP BY.

```
SELECT Tipo, SUM(Quantidade) AS 'Quantidade em Estoque'  
FROM Produtos  
GROUP BY Tipo
```

Tipo	Quantidade em Estoque
Armário	50.00
Celular	300.00
Console	800.00
Notebook	200.00
Refrigerador	200.00
Smartphone	50.00
Sofá	200.00

# Linguagem SQL

**Exemplo:** somar a quantidade de produtos em estoque de acordo com os tipos e fabricantes disponíveis. Primeiro, será agrupados os produtos de acordo com os tipos e fabricantes, para que depois seja feita a soma de cada um desses grupos.

**SELECT** Tipo, Fabricante, SUM(Quantidade) **AS** 'Quantidade em Estoque'  
**FROM** Produtos  
**GROUP BY** Tipo, Fabricante

Tipo	Fabricante	Quantidade em Estoque
Smartphone	Apple	50.00
Armário	Aracaju	50.00
Refrigerador	CCE	200.00
Sofá	Coréia	200.00
Notebook	Dell	200.00
Console	Microsoft	350.00
Console	Nintendo	250.00
Celular	Samsung	300.00
Console	Sony	200.00



# Linguagem SQL

## Clausula HAVING

### Cláusula HAVING com GROUP BY

A cláusula HAVING determina uma condição de busca para um grupo ou um conjunto de registros, definindo critérios para limitar os resultados obtidos a partir do agrupamento de registros.

É importante lembrar que essa cláusula só pode ser usada em parceria com GROUP BY.

Obs: O HAVING é diferente do WHERE. O WHERE restringe os resultados obtidos **sempre** após o uso da cláusula FROM, ao passo que a cláusula HAVING filtra o retorno do agrupamento.



# Linguagem SQL

**Exemplo:** A cláusula GROUP BY pode ser empregada, entre outras finalidades, para agrupar os produtos de acordo com cada tipo existente. Dentro de cada um dos grupos, a cláusula HAVING pode ser usada para restringir apenas os registros que possuem uma quantidade superior a **200** unidades no estoque.

```
SELECT Tipo, SUM(Quantidade) AS 'Quantidade em Estoque'  
FROM Produtos  
GROUP BY Tipo  
HAVING SUM(Quantidade) > 200
```

Tipo	Quantidade em Estoque
Celular	300.00
Console	800.00

# Linguagem SQL

**Exemplo:** Vamos agrupar os produtos com base nos tipos e fabricantes disponíveis. Logo após, retornaremos apenas os registros cuja quantidade supera novamente as **200** unidades em estoque.

```
SELECT Tipo, Fabricante, SUM(Quantidade) AS 'Quantidade em Estoque'  
FROM Produtos  
GROUP BY Tipo, Fabricante  
HAVING SUM(Quantidade) > 200
```

Tipo	Fabricante	Quantidade em Estoque
Console	Microsoft	350.00
Console	Nintendo	250.00
Celular	Samsung	300.00

# Linguagem SQL

**Exemplo:** vamos supor que o agrupamento deverá ser feito pelo Nome. Dentro deste agrupamento, desejamos obter apenas aqueles cuja quantidade novamente supera as **200** unidades em estoque e cujo valor estocado seja igual ou superior a **100 mil**.

```
SELECT Nome, SUM(Quantidade) AS 'Quantidade em Estoque', SUM(Quantidade  
* VIUnitario) AS 'Valor em Estoque'  
FROM Produtos  
GROUP BY Nome  
HAVING SUM(Quantidade) > 200 AND SUM (Quantidade * VIUnitario) >= 10000.00
```

Nome	Quantidade em Estoque	Valor em Estoque
GT-I6220 Quad Band	300.00	149700.0000
Wii 120GB	250.00	249750.0000
Xbox 360 120GB	350.00	454650.0000

# Linguagem SQL

## Subconsulta

É uma instrução **SELECT** que está encadeada dentro de outra instrução **SELECT**. A consulta interior é designada por seleção interna e é executada em primeiro lugar, sendo o seu resultado utilizado para completar a consulta principal ou externa.

Caso a consulta interna retorne apenas uma linha, podem ser usados operadores lógicos na comparação (> , < , >= , <= , <>) Quando mais de uma linha for encontrada no resultado interno, os operadores **IN, ANY, ALL, EXISTS** que manipulam sobre conjuntos de valores devem ser utilizados.



# Linguagem SQL

**Exemplo 1:** obter os nomes dos cursos com mensalidade maior que a média das mensalidades entre os cursos.

```
SELECT NOME_C, MENSC
FROM CURSO
WHERE MENSC > (SELECT AVG(MENSC) FROM CURSO)
```

## Resultado:

1) SELECT AVG(MENSC) FROM CURSO

AVG(MENSC)
480

2) SELECT NOME\_C, MENSC FROM CURSO WHERE MENSC > 480

NomeC	MensC
Eng Mecatrônica	600
Eng Elétrica	600



# Linguagem SQL

**Exemplo 2:** nome e área das disciplinas do curso C4.

```
SELECT NomeD, AreaD
FROM Disciplina
WHERE CodD IN (SELECT CodD FROM Grade WHERE CodC = 'C4')
```

**Resultado:**

1) SELECT CodD FROM Grade WHERE CodC = 'C4'

CodD	NomeD	AreaD
------	-------	-------

D1	TLP1	Computação
----	------	------------

D3	Inglês	Humanas
----	--------	---------

2) SELECT NomeD, AreaD FROM Disciplina WHERE CodD IN (...)

CodD	NomeD	AreaD
------	-------	-------

D1	TLP1	Computação
----	------	------------

D3	Inglês	Humanas
----	--------	---------



# Linguagem SQL

## Operador EXISTS/NOT EXISTS

Permite à consulta externa verificar se a consulta interna devolveu alguma linha.

O valor das linhas não é importante, apenas a cardinalidade do conjunto.

A correspondência é feita entre a tabela da consulta externa com a tabela da consulta interna, na cláusula **WHERE** desta última. Devolve **TRUE** se a cardinalidade for superior a 0 (zero), e **FALSE** caso seja igual a 0 (zero). Este operador pode ser negado com a cláusula **NOT**.





# Linguagem SQL

## Operador ALL

Permite a uma consulta externa fazer comparações usando < ou > com os elementos de um conjunto devolvido pela subconsulta. Este operador devolve **TRUE** se todas as linhas do conjunto satisfazem a condição, ou seja, devolve **FALSE** se alguma linha não a satisfaz. Pode ser negado com NOT.

**Exemplo:** nomes dos cursos que possuam duração maior que todos os cursos de mensalidade inferior a 500 reais.

```
SELECT NomeC  
FROM Curso  
WHERE DuracaoC ALL (SELECT DuracaoC FROM Curso WHERE MensC < 500)
```



# Linguagem SQL

## Operador ANY

Permite a uma consulta externa fazer comparações usando < ou > com os elementos de um conjunto devolvido pela subconsulta. Este operador devolve **TRUE** se uma das linhas do conjunto satisfaz a condição, ou seja, devolve **FALSE** se nenhuma satisfaz a condição. Pode ser negado com **NOT**.

**Exemplo:** nomes dos cursos que possuam duração maior que qualquer um dos cursos de mensalidade inferior a 500 reais.

```
SELECT NomeC
FROM Curso
WHERE DuracaoC > ANY (SELECT DuracaoC FROM Curso WHERE MensC <
500)
```



# Linguagem SQL

## Controle de Fluxo

Assim como muitas linguagens de programação utilizam operadores de condição, o SQL não poderia ficar de fora. Ele trabalha com esses elementos, também denominado de controle de fluxo, permitindo assim ao desenvolvedor criar lógicas para as mais variadas situações e regras de negócio de seu sistema.

Os elementos de controle de fluxo que iremos ver são, nessa ordem:

- 1 – BEGIN / END,
- 2 – IF / ELSE,
- 3 – CASE / WHEN / THEN / END,
- 4 – WHILE; e
- 5 – TRY...CATCH.



# Linguagem SQL

## BEGIN / END

Os elementos **BEGIN** e **END** tem o objetivo de iniciar e finalizar, respectivamente, um bloco de comandos, de maneira que este possa ser posteriormente executado. Podemos aninhar blocos de comando utilizando estes elementos.

Caso seja executado um bloco de comandos logo após a realização de um teste de condição, os elementos **BEGIN** e **END**, são usados logo após um comando **IF** ou **WHILE**. Veremos mais a frente exemplos com o uso destes elementos.



# Linguagem SQL

## IF / ELSE

Os elementos **IF** e **ELSE** são usados para testar condições quando um comando **Transact-SQL** é executado. O **IF** e **ELSE** funcionam similarmente aos comandos de mesmo nome usados em linguagens como **C#** por exemplo, para testar condições de execução de comandos.

### Sintaxe:

```
IF Expressao_Booleana
    { comando_sql | bloco_comando }
[ ELSE
    { comando_sql | bloco_comando } ]
```



# Linguagem SQL

**Exemplo:** IF / ELSE com o uso de BEGIN / END

IF @IdUsuario > 0

BEGIN

SELECT IdUsuario  
FROM Clientes  
WHERE IdUsuario = @IdUsuario

END

ELSE

BEGIN

SELECT TOP 1 @IdUsuario = IdUsuario  
FROM Clientes

END



# Linguagem SQL

## WHILE

Assim como o **IF/ELSE**, o comando **WHILE** funciona da mesma forma que nas linguagens de programação: ele faz com que um comando ou bloco de comandos SQL seja executado repetidamente, ou seja, é criado um loop o comando ou bloco de comandos, que será executado enquanto a condição especificada for verdadeira.

### Exemplo:

```
DECLARE @Contador AS SMALLINT
```

```
SET @Contador = 1
```

```
WHILE @Contador <= 10
```

```
    BEGIN
```

```
        SELECT @Contador
```

```
        SET @Contador = @Contador + 1
```

```
    END
```



# Linguagem SQL

## WHILE com BREAK

Podemos usar o **WHILE** com **BREAK** quando desejamos interromper o loop em um determinado ponto. O **BREAK** também pode ser usado para finalizar a execução de um loop dentro de um comando **IF/ELSE**.

### Exemplo:

```
DECLARE @Contador AS SMALLINT
```

```
SET @Contador = 1
```

```
WHILE @Contador <= 10
```

```
    BEGIN
```

```
        SELECT @Contador
```

```
        IF @Contador = 5
```

```
            BREAK
```

```
        SET @Contador = @Contador + 1
```

```
    END
```





# Linguagem SQL

## CASE

A expressão **CASE** será testada em tempo de execução do comando **SELECT** ou **UPDATE**. Como o **CASE** faz parte de outro comando, será possível colocá-lo em qualquer situação em que um valor deva ser testado.

Ao ser utilizada a cláusula **CASE** em comandos SQL é possível economizar diversas linhas de código, pois não é necessário criar blocos de programação para testar condições.

Podemos trabalhar de várias maneiras com **CASE**, é muito prático para criarmos triggers.



# Linguagem SQL

## SELECT

É possível utilizar o comando **SELECT** prevendo diversas condições para extração dos dados. A sintaxe é a seguinte:

### Sintaxe:

```
SELECT colunas,  
      CASE  
      WHEN condição THEN ação  
      ...  
      [ELSE condição padrão]  
      END  
FROM tabela;
```



# Linguagem SQL

## Exemplo:

nome_cd	preco_venda	venda
Mais do Mesmo	16,00	11,20
Bate-Boca	13,00	9,10
Elis Regina	20,00	14,00

```
select nome_cd, preco_venda,  
  case  
    when preco_venda < 10 then  
      preco_venda * 0.9  
    when preco_venda >=10 and preco_venda < 13 then  
      preco_venda * .8  
    else  
      preco_venda * .7  
  end venda  
from cd
```

# Linguagem SQL

## UPDATE

Podemos utilizar a mesma cláusula com o comando UPDATE. Assim, poderemos realizar atualizações com base em condições, simplificando a lógica de atualização dos dados.

### Sintaxe:

```
update cd
  set preco_venda =
  case
  when preco_venda < 10 then
    preco_venda * 0.9
  when preco_venda >=10 and preco_venda < 13 then
    preco_venda * .8
  else
    preco_venda * .7
  end;
```

nome_cd	preco_venda
Mais do Mesmo	3,47
Bate-Boca	3,18
Elis Regina	3,81



# Linguagem SQL

## CASE Compacto

É possível utilizar um teste **CASE** mais compacto que o demonstrado anteriormente. Para isso, basta colocarmos a coluna que queremos avaliar após a cláusula **CASE** e testarmos os valores após a cláusula **WHEN**. Observe que nesse caso é possível testar apenas igualdade, uma vez que nenhum operador poderá ser colocado ao lado do valor avaliado.

### Exemplo:

```
select nome_cd,  
       case codigo_gravadora  
         when 1 then 'EMI'  
         when 2 then 'BMG'  
         when 3 then 'Som Livre'  
       end  
from CD;
```



# Linguagem SQL

## Junção e junção interna (INNER JOIN)

### Junção

Uma junção entre duas tabelas corresponde a um produto cartesiano que gera uma relação resultante que contém todas as colunas das tabelas originais.

Na cláusula **FROM**, utilizamos os nomes das tabelas que possuem os campos que queremos trazer.



# Linguagem SQL

## Junção interna (INNER JOIN)

Na junção interna entre duas tabelas, somente são recuperadas as linhas da tabela resultante que satisfazem a condição de junção.

Os atributos comuns às duas tabelas quando utilizados no comando **SELECT** ou no critério de junção devem ser qualificados da seguinte forma:

**TABELA.ATRIBUTO**. Se um ALIAS for utilizado na cláusula **FROM**, este substitui o nome da tabela dentro da consulta.

Existem duas possibilidades de sintaxe para a junção interna.



# Linguagem SQL

**Exemplo 1:** Todas as informações sobre a grade e as disciplinas do curso C4.

```
SELECT *  
FROM Grade, Disciplina  
WHERE Disciplina.CodD = Grade.CodD  
AND CodC = 'C4'
```

Nessa sintaxe, as tabelas são separadas por vírgula na cláusula FROM, e a condição de junção fica na cláusula.





# Linguagem SQL

A sintaxe equivalente é:

```
SELECT *  
FROM Grade INNER JOIN Disciplina  
ON Disciplina.CodD = Grade.CodD  
WHERE CodC = 'C4'
```

Nessa sintaxe, as tabelas são separadas na cláusula **FROM** pelo comando **INNER JOIN** (ou apenas **JOIN** em alguns bancos de dados), e a condição de junção fica após a cláusula **ON**. Outros critérios para filtrar linhas ficam na cláusula **WHERE**.



# Linguagem SQL

## Usando Joing Using

Caso as colunas usadas na junção tenham o mesmo nome nas duas relações, podemos usar o comando USING.

**Exemplo:** todas as informações sobre a grade e as disciplinas do curso C4.

```
SELECT *  
FROM Grade INNER JOIN Disciplina  
USING (CodD)  
WHERE CodC = 'C4'
```



# Linguagem SQL

## Junção Natural (natural join)

Para fazer uma junção em que os atributos de junção têm o mesmo nome nas duas relações, podemos realizar uma junção natural usando o comando NATURAL JOIN. Os atributos repetidos são removidos da relação resultante.

**Exemplo:** todas as informações sobre a grade e as disciplinas do curso C4, sem repetição de colunas.

```
SELECT *  
FROM Grade NATURAL JOIN Disciplina  
ON Grade.CodD = Disciplina.CodD  
WHERE CodC = 'C4'
```



# Linguagem SQL

## Usando ALIAS

Podemos substituir o nome da relação dentro da consulta usando um ALIAS. As referências à relação podem ser substituídas em qualquer local da consulta.

**Exemplo:** código, nome, carga horária, sala e curso das disciplinas que estão na grade.

```
SELECT G.CodD, NomeD, CargaD, Sala, CodC  
FROM Disciplina D, Grade G  
WHERE D.CodD = G.CodD
```

### Ou na sintaxe

```
SELECT G.CodD, NomeD, CargaD, Sala, CodC  
FROM Disciplina D INNER JOIN Grade G  
ON D.CodD = G.CodD
```



# Linguagem SQL

## Várias Tabelas no JOIN

Podemos realizar a junção com mais de duas relações na consulta. Basta que cada par de relações tenha um critério de junção correspondente.

## Autojunção

Ocorre quando uma relação faz junção com ela mesma.

**Exemplo 1:** nomes das disciplinas cujo pre-requisito é “Cálculo1”.

```
SELECT Disc1.NomeD  
FROM Disciplina Disc1, Disciplina Disc2  
WHERE Disc2.CodD = Disc1.PreReqD AND Disc2.NomeD = “Cálculo1”
```



# Linguagem SQL

## Tipos de junção e operações

Vejam agora outros tipos junção e operações e suas características.

### Junção Externa (OUTER JOIN)

Uma junção externa é uma seleção que não requer que os registros de uma tabela possuam registros equivalentes em outra.

Esse tipo de junção se subdivide dependendo de qual tabela não terá correspondência de registros: a tabela esquerda, a direita ou ambas.

#### Left Outer Join

O resultado dessa seleção sempre contém todos os registros da tabela esquerda (a primeira tabela mencionada na consulta), mesmo quando não existam registros correspondentes na tabela direita. Quando não há correspondência, retorna um valor **NULL**. Exemplo: nome dos professores e código da disciplina ministrada, incluindo os professores sem alocação.



# Linguagem SQL

## Exemplo:

```
SELECT NomeP, CodD  
FROM Professor P LEFT OUTER JOIN Grade G  
ON P.CodP = G.CodP
```

## Resultado:

CodD	NomeP
D6	Joaquim
D2	Paulo
D2	Paulo
D1	Gil
D3	Andre
D4	Andre
D4	Gil
Null	Juliana

# Linguagem SQL

## Right Outer Join

Inversa à anterior e retorna sempre todos os registros da tabela à direita (a segunda tabela mencionada na consulta), mesmo se não existir registro correspondente na tabela à esquerda.

O valor **NULL** é retornado quando não há correspondência.





# Linguagem SQL

**Exemplo:** código do curso e nome das disciplinas relacionadas, incluindo as disciplinas não alocadas.

```
SELECT CodC, NomeD
FROM Grade G RIGHT OUTER JOIN Disciplina D
ON G.CodD = D.CodD
```

## Resultado:

CodC	NomeD
C1	Projeto Final
C2	Cálculo1
C3	Cálculo1
C4	TLP1
C4	Inglês
C5	Ed Física
C5	Ed Física
Null	G Analítica



# Linguagem SQL

## Full Outer Join

Essa operação apresenta todos os dados das tabelas à esquerda e à direita, mesmo que não possuam correspondência em outra tabela.

Assim, a tabela combinada possuirá todos os registros de ambas as tabelas e apresentará valores nulos para os registros sem correspondência.

## Operação de Conjunto

As operações de conjunto **UNION**, **INTERSECT** e **EXCEPT** operam sobre relações e correspondem às operações da álgebra relacional  $\cup$ ,  $\cap$  e  $-$ .

Pelo menos duas consultas estarão envolvidas na operação, e as colunas relacionadas a elas devem ser do mesmo tipo.

Cada uma das operações acima elimina automaticamente os registros duplicados. Para manter todos os registros duplicados, deve-se utilizar os comandos **UNION ALL**, **INTERSECT ALL** e **EXCEPT ALL**.



# Linguagem SQL

**Vejam as tabelas de exemplo:**

empréstimo (**numero\_emprestimo**, nome\_agencia, quantia)

tomador (**nome\_cliente**, numero\_emprestimo)

conta (**numero\_conta**, nome\_agencia, saldo)

depositante (**nome\_cliente**, numero\_conta)

## UNION

**Exemplo:** encontra todos os clientes que possuem conta e/ou empréstimo.

```
(SELECT NOME_CLIENTE FROM DEPOSITANTE)
```

```
UNION
```

```
(SELECT NOME_CLIENTE FROM TOMADOR)
```



# Linguagem SQL

**Vejam as tabelas de exemplo:**

empréstimo (**numero\_emprestimo**, nome\_agencia, quantia)

tomador (**nome\_cliente**, numero\_emprestimo)

conta (**numero\_conta**, nome\_agencia, saldo)

depositante (**nome\_cliente**, numero\_conta)

## INTERSECT

**Exemplo:** encontra todos os clientes que possuem um empréstimo e uma conta.

```
(SELECT NOME_CLIENTE FROM TOMADOR)  
INTERSECT  
(SELECT NOME_CLIENTE FROM DEPOSITANTE)
```



# Linguagem SQL

**Vejam as tabelas de exemplo:**

empréstimo (**numero\_emprestimo**, nome\_agencia, quantia)

tomador (**nome\_cliente**, numero\_emprestimo)

conta (**numero\_conta**, nome\_agencia, saldo)

depositante (**nome\_cliente**, numero\_conta)

**EXCEPT**

**Exemplo:** encontra todos os clientes que possuam conta, mas não possuam empréstimo.

```
(SELECT NOME_CLIENTE FROM DEPOSITANTE)  
EXCEPT  
(SELECT NOME_CLIENTE FROM TOMADOR)
```





f i t @rederecode | y @recoderede

<https://recode.org.br>