



- **Aula 011 – Linguagem SQL**
 - Funções de agregação.
 - Cláusula HAVING.
 - Classificação de resultados.
 - Views.



■ Funções de agregação

- A linguagem SQL pode ser utilizada para calcular vários resumos matemáticos e estatísticos como a contagem do número de linhas que apresentem uma condição específica, obtenção dos valores mínimos e máximo de um atributo, soma e média dos valores de uma coluna, dentre outros.



■ Funções de agregação

- As **funções de agregação** executam um cálculo em um conjunto de valores e retornam um único valor.
- Com exceção de **COUNT**, as funções de agregação ignoram valores nulos. As funções de agregação normalmente são usadas com a cláusula **GROUP BY** da instrução **SELECT**.



■ Funções de agregação

Operadores	Descrição
AVG	Retorna a média dos valores em um grupo. Valores nulos são ignorados.
COUNT	Retorna o número de itens não nulos de um grupo.
COUNT (*)	Retorna o número de itens de um grupo (considera valores nulos).
COUNT (DISTINCT...)	Retorna o número de itens exclusivos de um grupo.
MAX	Retorna o maior valor de um grupo.
MIN	Retorna o menor valor de um grupo.
SUM	Retorna a soma de todos os valores ou somente os valores DISTINCT na expressão. SUM pode ser usado exclusivamente com colunas numéricas. Valores nulos são ignorados.
STDEV	Retorna o desvio padrão estatístico de todos os valores da expressão especificada.
VAR	Retorna a variância estatística de todos os valores da expressão especificada.

■ Funções de agregação

- Utiliza as funções de agregação para retornar o total de
- funcionários que existem, o total geral e a média do salário
- dentre eles, o maior e o menor salário. Demonstra também a
- utilização do desvio padrão e da variância.

```
SELECT COUNT(*)           AS 'Total de Funcionários',  
       SUM(Salario)       AS 'Gasto Salarial',  
       AVG(Salario)       AS 'Salário Médio',  
       MAX(Salario)       AS 'Maior Salário',  
       MIN(Salario)       AS 'Menor Salário',  
       STDEV(Salario)     AS 'Desvio Padrão',  
       VAR(Salario)       AS 'Variância'  
  
FROM FUNCIONARIOS  
  
GO
```



■ Funções de agregação

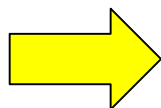
- A cláusula **GROUP BY** geralmente é utilizada quando se tem colunas de atributos combinadas com **funções agregadas**.
- Ela é utilizada para agrupar as linhas da tabela segundo um critério escolhido pelo usuário.



■ Funções de agregação

- Retorna o total de funcionários de cada sexo.
- Versão sem utilizar o GROUP BY -> Gera um erro!

```
SELECT  Sexo                AS 'Sexo',  
        COUNT(Sexo)        AS 'Total de  
Funcionários'  
FROM  FUNCIONARIOS  
GO
```



Msg 8120, Level 16, State 1, Line 546
Column 'FUNCIONARIOS.Sexo' is invalid in the
select list because it is not contained in
either an aggregate function or the GROUP BY
clause.



■ Funções de agregação

- Retorna o total de funcionários de cada sexo.
- Utiliza o GROUP BY para agrupar os dados.

```
SELECT  Sexo                                AS 'Sexo',  
        COUNT(Sexo)                        AS 'Total de  
Funcionários'
```

```
FROM FUNCIONARIOS
```

```
GROUP BY Sexo
```

```
GO
```

Agrupar pelo campo
não agregado



■ Funções de agregação

 Sexo Total de Funcionários

F 5

M 5

(2 row(s) affected)

Coluna utilizada para agrupar o resultado



■ Funções de agregação

- Exibe os dados de todos os funcionários,
- agrupando pelo ano da data de admissão.

```
SELECT  YEAR(Admissao) AS 'Ano de Admissão',  
        COUNT(*)      AS 'Total de  
Funcionários'  
FROM  FUNCIONARIOS  
GROUP BY YEAR(Admissao)  
GO
```



■ Funções de agregação

Ano de Admissão Total de Funcionários

2012	1
2014	1
2015	4
2016	1
2017	2
2018	1

Atenção: o **COUNT(*)**
leva em consideração
os valores nulos!
Utilize o **WHERE**, para
evitar a contagem de
valores nulos.

(6 row(s) affected)



■ Hora do estagiário...



■ Cláusula HAVING

- A cláusula **GROUP BY** agrupa as linhas retornadas pelo comando **SELECT**, em conjunto com as funções de agregação.
- Para filtrar esses resultados, podemos utilizar a cláusula **HAVING**. Ele funciona de maneira semelhante a cláusula **WHERE**, porém, somente pode ser aplicado ao resultado de uma operação envolvendo o **GROUP BY**.



■ Cláusula HAVING

- Sintaxe do comando **WHERE**:

SELECT Colunas...

FROM Tabelas...

WHERE Condições... ← 1

GROUP BY... ← 2

ORDER BY... ← 3



■ Cláusula HAVING

- Sintaxe do comando **HAVING**:

SELECT Colunas...

FROM Tabelas...

WHERE Condições... ← 1

GROUP BY... ← 2

HAVING... ← 3

ORDER BY... ← 4



■ Cláusula HAVING

- Exibir um relatório contendo a inicial dos funcionários, o
- total de funcionários cujo nome comece com essa inicial e
- o salário total gasto com todos esses funcionários.

```
SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
        COUNT(Nome)           AS 'Quantidade de  
Funcionários',  
        SUM(Salario)          AS 'Gasto Salarial'  
FROM  FUNCIONARIOS  
GROUP BY SUBSTRING(Nome, 1, 1)  
GO
```



■ Cláusula HAVING

Inicial	Quantidade de Funcionários	Gasto Salarial

A	1	1089.00
M	6	12290.00
P	2	2244.00
T	1	1350.50

(4 row(s) affected)



■ Cláusula HAVING

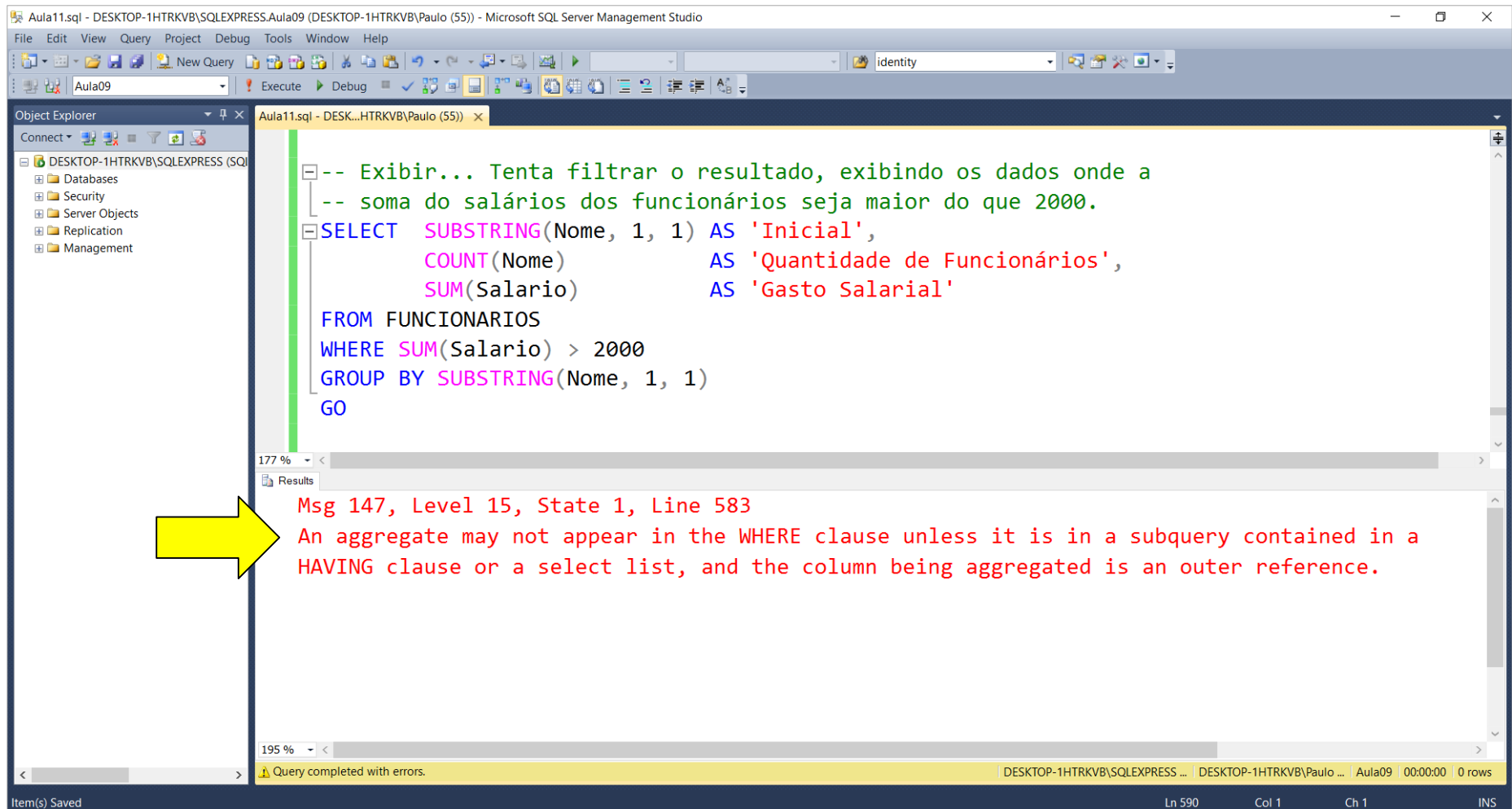
-- Exibir... Tenta filtrar o resultado, exibindo os dados
-- onde a soma do salários dos funcionários seja maior do que
-- 2000.

```
SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
        COUNT(Nome)           AS 'Quantidade de  
Funcionários',  
        SUM(Salario)          AS 'Gasto Salarial'  
FROM  FUNCIONARIOS  
WHERE  SUM(Salario) > 2000  
GROUP BY SUBSTRING(Nome, 1, 1)  
GO
```

← Erro!



■ Cláusula HAVING



Aula11.sql - DESKTOP-1HTRKVB\SQLEXPRESS.Aula09 (DESKTOP-1HTRKVB\Paulo (55)) - Microsoft SQL Server Management Studio

Object Explorer

Connect

DESKTOP-1HTRKVB\SQLEXPRESS (SQL)

Databases

Security

Server Objects

Replication

Management

Aula11.sql - DESKTOP-1HTRKVB\Paulo (55)

```
-- Exibir... Tenta filtrar o resultado, exibindo os dados onde a
-- soma dos salários dos funcionários seja maior do que 2000.
SELECT SUBSTRING(Nome, 1, 1) AS 'Inicial',
       COUNT(Nome) AS 'Quantidade de Funcionários',
       SUM(Salario) AS 'Gasto Salarial'
FROM FUNCIONARIOS
WHERE SUM(Salario) > 2000
GROUP BY SUBSTRING(Nome, 1, 1)
GO
```

177 %

Results

Msg 147, Level 15, State 1, Line 583
An aggregate may not appear in the WHERE clause unless it is in a subquery contained in a
HAVING clause or a select list, and the column being aggregated is an outer reference.

195 %

Query completed with errors.

DESKTOP-1HTRKVB\SQLEXPRESS ... | DESKTOP-1HTRKVB\Paulo ... | Aula09 | 00:00:00 | 0 rows

Item(s) Saved

Ln 590 Col 1 Ch 1 INS

■ Cláusula HAVING

- Exibir... Tenta filtrar o resultado, exibindo os dados
- onde a soma do salários dos funcionários seja maior do que
- 2000.

```
SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
        COUNT(Nome)          AS 'Quantidade de  
Funcionários',  
        SUM(Salario)         AS 'Gasto Salarial'  
FROM  FUNCIONARIOS  
HAVING SUM(Salario) > 2000  
GROUP BY SUBSTRING(Nome, 1, 1)  
GO
```

← Sintaxe errada!



■ Cláusula HAVING

-- Exibir... Tenta filtrar o resultado, exibindo os dados
-- onde a soma do salários dos funcionários seja maior do que
-- 2000.

```
SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
        COUNT(Nome)           AS 'Quantidade de  
Funcionários',  
        SUM(Salario)          AS 'Gasto Salarial'  
FROM    FUNCIONARIOS  
GROUP BY SUBSTRING(Nome, 1, 1)  
HAVING  SUM(Salario) > 2000  
GO
```

← Sintaxe correta!



■ Cláusula HAVING

Inicial	Quantidade de Funcionários	Gasto Salarial

M	6	12290.00
P	2	2244.00

(2 row(s) affected)



■ Consultas avançadas

- O **Microsoft SQL Server** conta com mais de 100 funções para agregar, analisar e manipular dados.
- Em muitos casos, deve-se utilizar as funções **CAST** e **CONVERT** para alterar o tipo de dados. Essas funções só devem ser utilizadas quando o SGBD não pode converte implicitamente os tipos de dados.
- Os valores do tipo **NULL** devem ser tratados com as funções **ISNULL** e **COALESCE**. Para retornos condicionais, pode-se utilizar a função **CASE**.



■ Consultas avançadas

- Os dados podem ser classificados com **ORDER BY**. As colunas na cláusula **ORDER BY** devem estar incluídas na lista do comando **SELECT**.
- Para pesquisas em tipos de dados alfanuméricos, pode-se utilizar o operador **LIKE**, junto com seus caracteres curingas.
- Para se recuperar dados de mais de uma tabela utilizam-se os **JOINS**.
- Para o retorno de linhas exclusivas utiliza-se a cláusula **DISTINCT**.



■ Consultas avançadas

- As funções de agregação retornam apenas um valor único para a tabela inteira. (**COUNT**, **MIN**, **MAX**, **SUM**, **AVG**, etc.).
- Após uma agregação ter sido concluída, pode-se filtrar os resultados utilizando-se a cláusula **HAVING**.
- Os resultados de uma consulta podem ser classificados utilizando-se as funções **ROW_NUMBER**, **RANK**, **DENSE_RANK** e **NTILE**.



■ Classificando resultados

- **ROW_NUMBER**: numera um conjunto de resultados sequencialmente de 1 a N, com base em uma ordem especificada pelo usuário.
- **RANK**: mesma utilidade da função ROW_NUMBER, porém exibe o mesmo valor para registros duplicados.
- **DENSE_RANK**: numera um conjunto de resultados, dando aos valores duplicados a mesma classificação, eliminando os gaps.
- **NTILE**: divide um conjunto de resultados em N blocos de tamanhos aproximadamente iguais.



■ Classificando resultados

- **ROW_NUMBER**: numera um conjunto de resultados sequencialmente de 1 a N, com base em uma ordem especificada pelo usuário.

ROW_NUMBER()

OVER ([<PARTITION BY...>]

<**ORDER BY** ... >)



■ Classificando resultados

- **RANK**: mesma utilidade da função ROW_NUMBER, porém exibe o mesmo valor para registros duplicados.

RANK ()

OVER ([<PARTITION BY...>]
 <**ORDER BY** ... >)



■ Classificando resultados

- **DENSE_RANK**: numera um conjunto de resultados, dando aos valores duplicados a mesma classificação, eliminando os gaps.

DENSE_RANK()

OVER ([<PARTITION BY...>]

<**ORDER BY** ... >)



■ Classificando resultados

- **NTILE**: divide um conjunto de resultados em blocos de tamanhos aproximadamente iguais (N blocos).

NTILE (INTEIRO)

OVER ([<PARTITION BY...>]
 <**ORDER BY**...>)



■ Classificando resultados

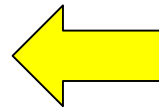
- Selecciona todos os funcionários,
- ordenando de acordo com o maior salário.

```
SELECT ID,  
        Nome,  
        Salario AS 'Salário'  
FROM FUNCIONARIOS  
ORDER BY Salario DESC  
GO
```



■ Classificando resultados

ID	Nome	Salário
---	-----	-----
7	Mônica da Silva	3300.00
1	Maria da Silva	2500.00
5	Marcelo Augusto	1900.00
9	Maria Cristina	1870.00
10	Maria Cristina	1400.00
8	Tiago Lima	1350.50
3	Maria Cristina	1320.00
6	Pedro Silva	1155.00
4	Antônio Carlos	1089.00
2	Pedro Pereira	1089.00



Exibe informações dos
funcionários que possuem
os maiores salários

(10 row(s) affected)



■ Classificando resultados

- Seleciona todos os funcionários, ordenando de acordo com o maior salário.
- Utiliza ROW_NUMBER, RANK, DENSE_RANK e NTILE para classificar o resultado.

```
SELECT  ID,  
        Nome,  
        Salario  AS 'Salário',  
        ROW_NUMBER() OVER (ORDER BY Salario DESC) AS 'ROW  
NUMBER',  
        RANK() OVER (ORDER BY Salario DESC)          AS 'RANK',  
        DENSE_RANK() OVER (ORDER BY Salario DESC) AS 'DENSE  
RANK',  
        NTILE(3) OVER (ORDER BY Salario DESC)          AS 'NTILE'  
FROM FUNCIONARIOS  
ORDER BY Salario
```

GO



■ Classificando resultados

ID	Nome	Salário	ROW NUMBER	RANK	DENSE RANK	NTILE
4	Antônio Carlos	1089.00	9	9	9	1
2	Pedro Pereira	1089.00	10	9	9	1
6	Pedro Silva	1155.00	8	8	8	1
3	Maria Cristina	1320.00	7	7	7	1
8	Tiago Lima	1350.50	6	6	6	2
10	Maria Cristina	1400.00	5	5	5	2
9	Maria Cristina	1870.00	4	4	4	2
5	Marcelo Augusto	1900.00	3	3	3	3
1	Maria da Silva	2500.00	2	2	2	3
7	Mônica da Silva	3300.00	1	1	1	3

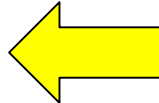
(10 row(s) affected)



■ Classificando resultados

- Seleciona todos os funcionários, ordenando de acordo com o maior salário.
- Utiliza ROW_NUMBER, RANK, DENSE_RANK e NTILE para classificar o resultado.

```
SELECT  ID,  
        Nome,  
        Salario  AS 'Salário',  
        ROW_NUMBER() OVER (ORDER BY Salario DESC) AS 'ROW  
NUMBER',  
        RANK() OVER (ORDER BY Salario DESC)          AS 'RANK',  
        DENSE_RANK() OVER (ORDER BY Salario DESC) AS 'DENSE  
RANK',  
        NTILE(3) OVER (ORDER BY Salario DESC)          AS 'NTILE'  
FROM FUNCIONARIOS  
ORDER BY Salario DESC
```



GO



■ Classificando resultados

ID	Nome	Salário	ROW NUMBER	RANK	DENSE RANK	NTILE
7	Mônica da Silva	3300.00	1	1	1	1
1	Maria da Silva	2500.00	2	2	2	1
5	Marcelo Augusto	1900.00	3	3	3	1
9	Maria Cristina	1870.00	4	4	4	1
10	Maria Cristina	1400.00	5	5	5	2
8	Tiago Lima	1350.50	6	6	6	2
3	Maria Cristina	1320.00	7	7	7	2
6	Pedro Silva	1155.00	8	8	8	3
4	Antônio Carlos	1089.00	9	9	9	3
2	Pedro Pereira	1089.00	10	9	9	3

(10 row(s) affected)



■ Classificando resultados

- Atualiza o salário dos funcionários
- de ID 5 e 8

UPDATE FUNCIONARIOS

SET Salario = 2500.00

WHERE ID IN (5, 8)

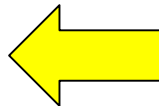
GO



■ Classificando resultados

- Seleciona todos os funcionários, ordenando de acordo com o maior salário.
- Utiliza ROW_NUMBER, RANK, DENSE_RANK e NTILE para classificar o resultado.

```
SELECT  ID,  
        Nome,  
        Salario  AS 'Salário',  
        ROW_NUMBER() OVER (ORDER BY Salario DESC) AS 'ROW  
NUMBER',  
        RANK() OVER (ORDER BY Salario DESC)          AS 'RANK',  
        DENSE_RANK() OVER (ORDER BY Salario DESC) AS 'DENSE  
RANK',  
        NTILE(3) OVER (ORDER BY Salario DESC)        AS 'NTILE'  
FROM FUNCIONARIOS  
ORDER BY Salario DESC
```



■ Classificando resultados

ID	Nome	Salário	ROW NUMBER	RANK	DENSE RANK	NTILE
7	Mônica da Silva	3300.00	1	1	1	1
8	Tiago Lima	2500.00	2	2	2	1
1	Maria da Silva	2500.00	3	2	2	1
5	Marcelo Augusto	2500.00	4	2	2	1
9	Maria Cristina	1870.00	5	5	3	2
10	Maria Cristina	1400.00	6	6	4	2
3	Maria Cristina	1320.00	7	7	5	2
6	Pedro Silva	1155.00	8	8	6	3
4	Antônio Carlos	1089.00	9	9	7	3
2	Pedro Pereira	1089.00	10	9	7	3

(10 row(s) affected)



■ Classificando resultados

- Seleciona todos os funcionários, ordenando de acordo com o maior salário.
- Utiliza ROW_NUMBER, RANK, DENSE_RANK e NTILE para classificar o resultado.

```
SELECT  ID,
        Nome,
        Salario  AS 'Salário',
        ROW_NUMBER() OVER (ORDER BY Salario DESC) AS 'ROW
NUMBER',
        RANK() OVER (ORDER BY Salario DESC)          AS 'RANK',
        DENSE_RANK() OVER (ORDER BY Salario DESC) AS 'DENSE
RANK',
        NTILE(3) OVER (ORDER BY Salario DESC)          AS 'NTILE'
FROM  FUNCIONARIOS
ORDER BY Nome
```

← Ordena todas as linhas e não a classificação

GO



■ Classificando resultados

ID	Nome	Salário	ROW NUMBER	RANK	DENSE RANK	NTILE
---	-----	-----	-----	-----	-----	-----
4	Antônio Carlos	1089.00	9	9	7	3
5	Marcelo Augusto	2500.00	4	2	2	1
9	Maria Cristina	1870.00	5	5	3	2
10	Maria Cristina	1400.00	6	6	4	2
3	Maria Cristina	1320.00	7	7	5	2
1	Maria da Silva	2500.00	3	2	2	1
7	Mônica da Silva	3300.00	1	1	1	1
2	Pedro Pereira	1089.00	10	9	7	3
6	Pedro Silva	1155.00	8	8	6	3
8	Tiago Lima	2500.00	2	2	2	1

(10 row(s) affected)



■ Classificando resultados

- Selecciona todos os funcionários, ordenando de acordo com o maior salário.
- Particiona o resultado de acordo com o sexo de cada funcionário.

```
SELECT  ID,  
        Nome,  
        Sexo,  
        Salario AS 'Salário',  
        ROW_NUMBER() OVER (PARTITION BY Sexo ORDER BY Salario DESC) AS  
'ROW NUMBER',  
        RANK() OVER (PARTITION BY Sexo ORDER BY Salario DESC) AS  
'RANK',  
        DENSE_RANK() OVER (PARTITION BY Sexo ORDER BY Salario DESC) AS  
'DENSE RANK',  
        NTILE(3) OVER (PARTITION BY Sexo ORDER BY Salario DESC) AS  
'NTILE'  
FROM  FUNCIONARIOS  
GO
```



■ Classificando resultados

ID	Nome	Sexo	Salário	ROW NUMBER	RANK	DENSE RANK	NTILE
7	Mônica da Silva	F	3300.00	1	1	1	1
1	Maria da Silva	F	2500.00	2	2	2	1
9	Maria Cristina	F	1870.00	3	3	3	2
10	Maria Cristina	F	1400.00	4	4	4	2
3	Maria Cristina	F	1320.00	5	5	5	3
8	Tiago Lima	M	2500.00	1	1	1	1
5	Marcelo Augusto	M	2500.00	2	1	1	1
6	Pedro Silva	M	1155.00	3	3	2	2
4	Antônio Carlos	M	1089.00	4	4	3	2
2	Pedro Pereira	M	1089.00	5	4	3	3

(10 row(s) affected)



■ Criando VIEWS

- Uma visualização, ou **VIEW**, é uma tabela virtual baseada em uma consulta **SELECT**. Essa consulta pode conter colunas, colunas computadas, alias, funções agregadas, filtros. Ela pode envolver uma ou mais tabelas, através de uma operação de junção, ou **JOIN**. As tabelas em que a visualização se baseia são chamadas de tabelas de base.

CREATE VIEW Nome_da_View **AS SELECT** Consulta...



■ Características das VIEWS

- Podem ser utilizadas em qualquer posição de comandos de SQL, substituindo o nome de uma tabela.
- São atualizadas dinamicamente.
- Fornecem um **nível de segurança** no banco de dados, pois podem restringir o acesso dos usuários apenas às colunas e linhas especificadas.



■ Criando VIEWS

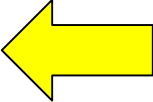
-- Seleciona os dados dos funcionários.

```
SELECT ID,  
        Nome,  
        Salario AS 'Salário'  
FROM FUNCIONARIOS  
GO
```



■ Criando VIEWS

- Cria uma VIEW simples, utilizando os
- dados dos funcionários.

```
CREATE VIEW Maiores_Salarios AS   
    SELECT ID,  
           Nome,  
           Salario AS 'Salário'  
FROM FUNCIONARIOS
```

GO



■ Criando VIEWS

- Exemplos de utilização da VIEW
- Maiores_Salarios

```
SELECT * FROM Maiores_Salarios  
GO
```

```
SELECT Nome,  
        Salário  
FROM Maiores_Salarios  
GO
```

Atenção: observe o
nome da coluna!



■ Criando VIEWS

- Cria uma VIEW baseada na consulta para exibir um relatório
- contendo a inicial dos funcionários, o total de funcionários cujo
- nome comece com essa inicial e o salário gasto com esses funcionários.

```
CREATE VIEW INICIAIS AS
```

```
    SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
            COUNT(Nome)           AS 'Quantidade de  
Funcionários',  
            SUM(Salario)          AS 'Gasto  
Salarial'
```

```
FROM FUNCIONARIOS
```

```
GROUP BY SUBSTRING(Nome, 1, 1)
```

```
GO
```



■ Criando VIEWS

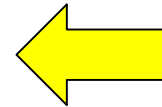
-- Seleciona todos os dados da VIEW INICIAIS

SELECT * FROM INICIAIS

GO

Inicial	Quantidade de Funcionários	Gasto Salarial
---------	----------------------------	----------------

A	1	1089.00
M	6	12890.00
P	2	2244.00
T	1	2500.00



Observe o nome
das colunas

(4 row(s) affected)

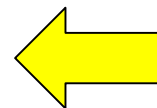


■ Criando VIEWS

-- Seleciona alguns campos da VIEW INICIAIS

SELECT Inicial,

[Quantidade de Funcionários],
"Gasto Salarial"

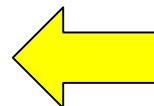


FROM INICIAIS

WHERE [Quantidade de Funcionários] > 1

GO

Inicial Quantidade de Funcionários Gasto Salarial



Inicial	Quantidade de Funcionários	Gasto Salarial
M	6	12890.00
P	2	2244.00



■ Criando VIEWS

-- Exibindo informações sobre as VIEWS

```
EXEC sp_helptext INICIAIS
```



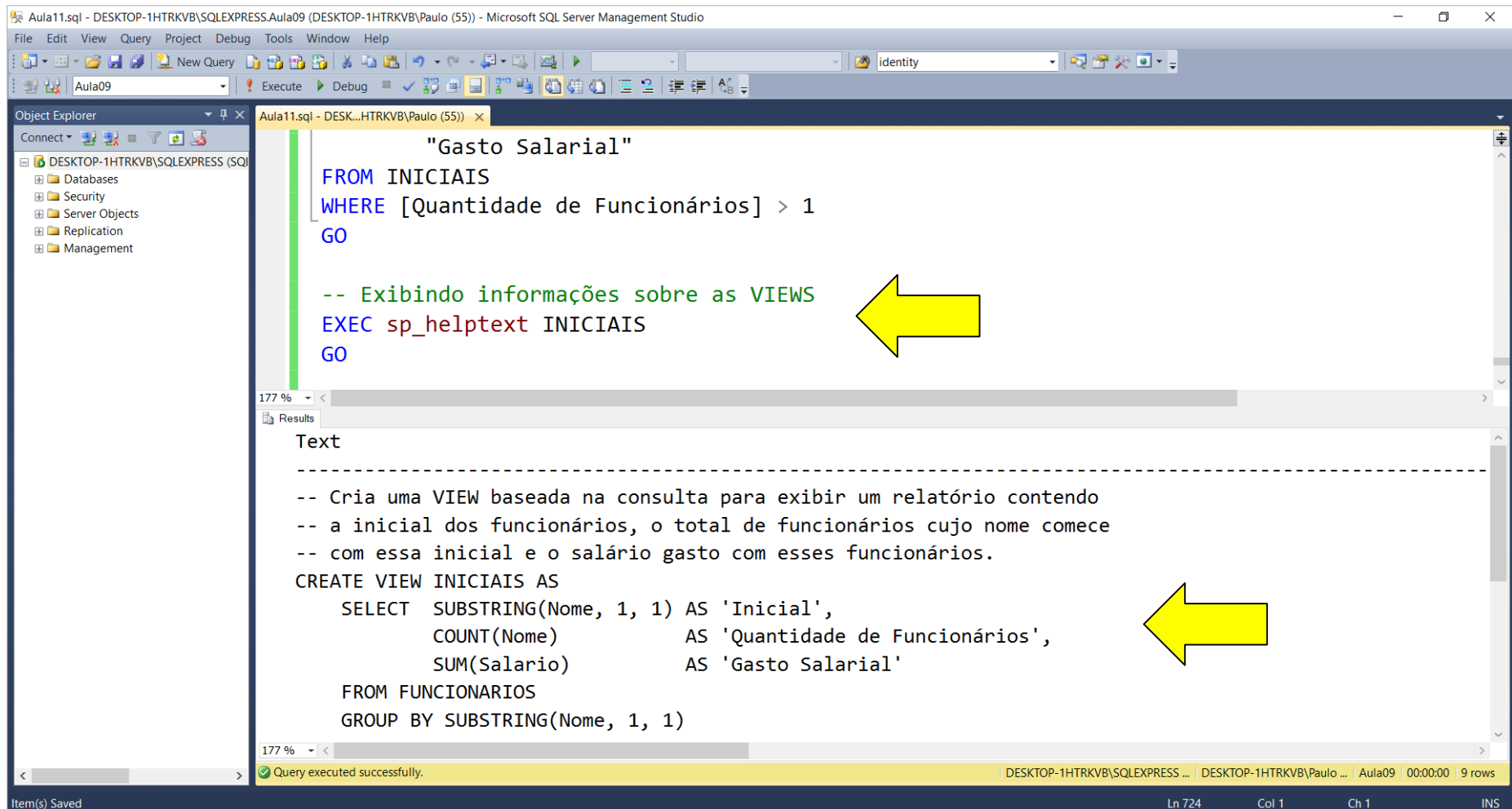
Nome da VIEW

```
GO
```

```
SELECT  TABLE_NAME          AS 'Nome da View',  
        VIEW_DEFINITION AS 'Definição da  
View'  
FROM  INFORMATION_SCHEMA.Views  
GO
```



■ Criando VIEWS



The screenshot shows the Microsoft SQL Server Management Studio interface. The main window displays a SQL script with the following content:

```
"Gasto Salarial"  
FROM INICIAIS  
WHERE [Quantidade de Funcionários] > 1  
GO  
  
-- Exibindo informações sobre as VIEWS  
EXEC sp_helptext INICIAIS  
GO
```

A yellow arrow points to the comment line: `-- Exibindo informações sobre as VIEWS`.

Below the script, the "Results" pane shows the output of the `EXEC sp_helptext INICIAIS` command:

```
Text  
-----  
-- Cria uma VIEW baseada na consulta para exibir um relatório contendo  
-- a inicial dos funcionários, o total de funcionários cujo nome comece  
-- com essa inicial e o salário gasto com esses funcionários.  
CREATE VIEW INICIAIS AS  
    SELECT  SUBSTRING(Nome, 1, 1) AS 'Inicial',  
            COUNT(Nome)          AS 'Quantidade de Funcionários',  
            SUM(Salario)          AS 'Gasto Salarial'  
FROM FUNCIONARIOS  
GROUP BY SUBSTRING(Nome, 1, 1)
```

A yellow arrow points to the `CREATE VIEW INICIAIS AS` line in the results pane.

The status bar at the bottom indicates: "Query executed successfully." and "DESKTOP-1HTRKVB\SQLEXPRESS ... | DESKTOP-1HTRKVB\Paulo ... | Aula09 | 00:00:00 | 9 rows".

- **Na próxima aula veremos**
 - Arquivos CSV, importe em massa e joins.

