



MBA em Inteligência Artificial e Big Data  
– Curso 3: Administração de Dados Complexos em Larga Escala –  
★ **SQL/DML *Window Functions* em SQL** ★

Caetano Traina Júnior

Grupo de Bases de Dados e Imagens – GBdI  
Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo - São Carlos

*Window Functions* para comandos de consulta analíticos em SQL.



# Roteiro



- 1 Window Functions para Análise de Dados
- 2 Sintaxe das Window Functions em SQL



# Window Functions – Motivação

## Sintaxe do Comando SQL

### Sintaxe geral de um Comando SQL

```
SELECT [ALL | DISTINCT] <lista_atributos>  
FROM <nome_tabelas> | <tabelas_joined>  
[WHERE <condicao>]  
[GROUP BY <lista_atributos>  
  [HAVING <condicao>]]  
[ORDER BY <lista_atributos> [ASC|DESC]];
```

- Comandos analíticos são geralmente executados em SQL baseados na cláusula **GROUP BY**.

★ Problema:

Quando o **GROUP BY** é executado, perde-se acesso às **tuplas individuais** que compõem cada grupo.



# Window Functions – Motivação

## Exemplo de Motivação

Suponha que se quer saber qual é a idade dos alunos mais novos e mais velhos de cada cidade:

Alunos=(Nome, NUSP, Idade, Cidade)

```
SELECT Cidade, Min(Idade), Max(Idade)
FROM Aluno A
GROUP BY Cidade;
```

Cidade	Min(Idade)	Max(Idade)
Sao Carlos	21	27
(null)	24	24
Campinas	19	19
Ibate	35	35
Araraquara	21	22
Ibitinga	21	21
Rio Claro	20	25

Alunos:

Nome	NUSP	Idade	Cidade
Carlos	1234	21	Sao Carlos
Celso	2345	22	Sao Carlos
Cicero	3456	22	Araraquara
Carlitos	4567	21	Ibitinga
Catarina	5678	23	Sao Carlos
Cibele	6789	21	Araraquara
Corina	7890	25	Rio Claro
Celina	8901	27	Sao Carlos
Celia	9012	20	Rio Claro
Cesar	123	21	Araraquara
Denise	4584	35	Ibate
Durval	1479	(null)	(null)
Daniel	1489	19	Campinas
Dora	1469	24	(null)
Dina	1459	(null)	Campinas

Mas... qual é o aluno que tem cada uma dessas idades?



# Window Functions – Motivação

Exemplo de Motivação – usando *window functions*

Para listar os alunos com as menores e maiores idades usando **GROUP BY**:

👉 é necessário ler a relação de alunos duas vezes:

```
SELECT A.Nome, A.Cidade, A.Idade, MM.IdMin, MM.IdMax
FROM Aluno A, (
    SELECT Cidade, Min(Idade) IdMin, Max(Idade) IdMax
    FROM Aluno
    GROUP BY Cidade) MM
WHERE (A.Cidade=MM.Cidade AND A.Idade=MM.IdMin) OR
      (A.Cidade=MM.Cidade AND A.Idade=MM.IdMax);
```

Nome	Cidade	Idade	IdMin	IdMax
Carlos	Sao Carlos	21	21	27
Celina	Sao Carlos	27	21	27
Cesar	Araraquara	21	21	22
Cibele	Araraquara	21	21	22
Cicero	Araraquara	22	21	22
Daniel	Campinas	19	19	19
Denise	Ibate	35	35	35
Carlitos	Ibitinga	21	21	21
Corina	Rio Claro	25	20	25
Celia	Rio Claro	20	20	25



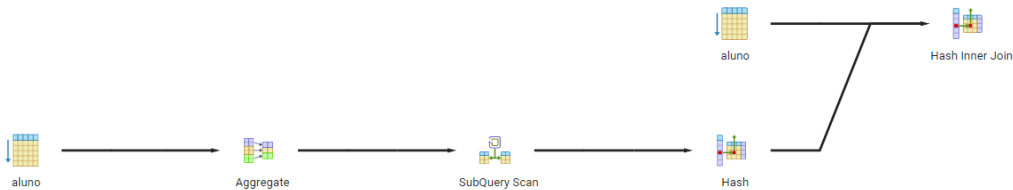
# Window Functions – Motivação

Exemplo de Motivação – usando *window functions*

Para listar os alunos com as menores e maiores idades usando **GROUP BY**:

👉 é necessário ler a relação de alunos duas vezes:

```
SELECT A.Nome, A.Cidade, A.Idade, MM.IdMin, MM.IdMax
FROM Aluno A, (
    SELECT Cidade, Min(Idade) IdMin, Max(Idade) IdMax
    FROM Aluno
    GROUP BY Cidade) MM
WHERE (A.Cidade=MM.Cidade AND A.Idade=MM.IdMin) OR
      (A.Cidade=MM.Cidade AND A.Idade=MM.IdMax);
```



# Window Functions – Conceitos


## Motivação



- O comando `GROUP BY <Atribs-Grupo>`:
  - 1 Particiona as tuplas da tabela resultante do processamento da cláusula `WHERE`, identificando todos os valores em `DISTINCT <Atribs-Grupo>`;
  - 2 Agrupa todas as tuplas de cada partição, numa única tupla, e gera uma nova tabela:
    - que tem como chave os atributos `<Atribs-Grupo>`;
    - os demais atributos são resultado de funções de agregação aplicadas sobre cada partição.

O que se deseja é executar apenas parte desse processamento, de maneira que:

- 1 Particiona as tuplas da tabela resultante do processamento da cláusula `WHERE`, obtendo `DISTINCT <Atribs-Grupo>`;
- 2 Agrupa todas as tuplas de cada partição, ~~numa única tupla, e gera uma nova tabela:~~
- 3 Aplica as funções de agregação sobre as tuplas de cada partição.

 As *window functions* atendem a essa necessidade.




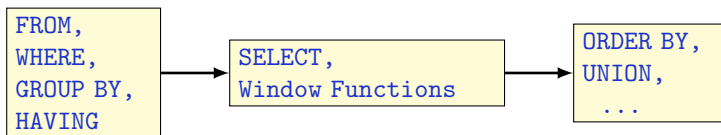
# Window Functions – Conceitos



O que são as *window functions*?

- São funções chamadas na cláusula **SELECT**

 portanto são executadas **depois** das cláusulas **FROM**, **WHERE**, **GROUP BY/HAVING**, e **antes** das cláusulas **ORDER BY** e **STOP AFTER**.



- Entre as cláusulas **GROUP BY/HAVING** e **SELECT**, é feito um outro particionamento (independente de haver ou não a cláusula **GROUP BY**), sobre o qual as *window functions* são executadas.
- Elas são uma extensão das funções de agregação, mas computam o resultado em **janelas** das tuplas de entrada em cada grupo, formadas **para cada tupla da relação original** – não para cada grupo.



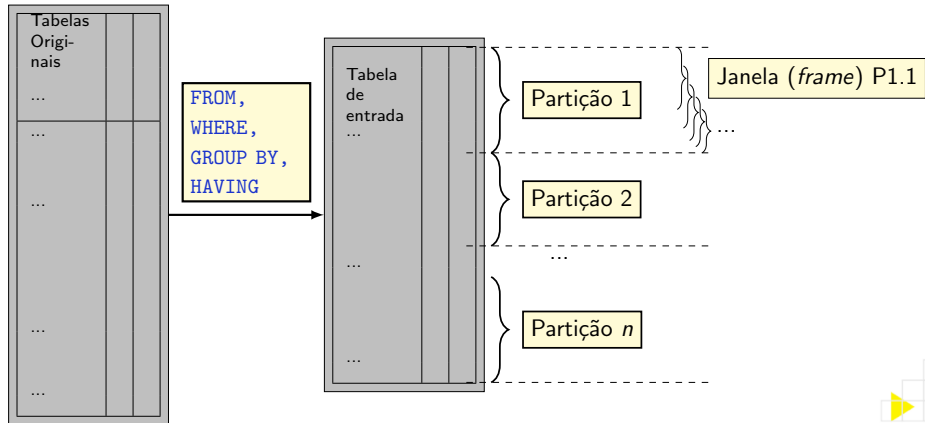


# Window Functions – Conceitos



Terminologia:

Tabela Original → Tabela de entrada → Partições → Janelas (*Windows*)



# Window Functions – Conceitos



Cada *window function* segue a sintaxe:

```
<window function> OVER ([<partição> ] [ <ordem> ] [ <frame> ])
```

A sequência de processamento é a seguinte:

- O resultado do processamento das cláusulas **FROM**, **WHERE**, **GROUP BY/HAVING** forma a **Tabela de Entrada**, sobre a qual as *Window Functions* são executadas;
- A cláusula **<partição>** separa a Tabela de Entrada em  $n$  **partições** (cada tupla está em exatamente uma partição).
- A cláusula **<ordem>** **ordena** as tuplas de cada partição.
- Cada tupla da partição tem um **frame** (chamado *sliding window*), definido pela cláusula **<frame>** como uma **sequência de tuplas da partição** numa dada ordem.
- A *window function* é executada sobre o **frame** de cada tupla da relação de entrada.

# Window Functions – Conceitos

Exemplo de Motivação – usando *window functions*

Obter a idade do(s) aluno(s) mais novo(s) e mais velho(s) em cada cidade:

Alunos=(Nome, NUSP, Idade, Cidade)

```
SELECT nome, Cidade, Idade,  
       Min(Idade) OVER(Partition by Cidade) IdMin,  
       Max(Idade) OVER(Partition by Cidade) IdMax  
FROM Aluno
```

Nome	Cidade	Idade	IdMin	IdMax
Carlos	Sao Carlos	21	21	27
Celina	Sao Carlos	27	21	27
Cesar	Araraquara	21	21	22
Cibele	Araraquara	21	21	22
Cicero	Araraquara	22	21	22
Daniel	Campinas	19	19	19
Denise	Ibate	35	35	35
Carlitos	Ibitinga	21	21	21
Corina	Rio Claro	25	20	25
Celia	Rio Claro	20	20	25
Dora	(null)	24	24	24



# Window Functions – Conceitos

Exemplo de Motivação – usando *window functions*

Obter a idade do(s) aluno(s) mais novo(s) e mais velho(s) em cada cidade:

Alunos=(Nome, NUSP, Idade, Cidade)

```
SELECT nome, Cidade, Idade,  
       Min(Idade) OVER(Partition by Cidade) IdMin,  
       Max(Idade) OVER(Partition by Cidade) IdMax  
FROM Aluno
```



aluno



Sort



Window Aggregate



# Window Functions – Conceitos

Exemplo de Motivação – usando *window functions*

Compare:

```
SELECT nome, Cidade, Idade,  
       Min(Idade) OVER(Partition by Cidade) IdMin,  
       Max(Idade) OVER(Partition by Cidade) IdMax  
FROM Aluno
```

X

```
SELECT A.Nome, A.Cidade, A.Nome, MM.IdMin, MM.IdMax  
FROM Aluno A, (  
    SELECT Cidade, Min(Idade) IdMin, Max(Idade) IdMax  
    FROM Aluno  
    GROUP BY Cidade) MM  
WHERE (A.Cidade=MM.cidade AND A.Idade=MM.IdMin) OR  
       (A.Cidade=MM.cidade AND A.Idade=MM.IdMax);
```



# Sintaxe das Window Functions em SQL

Sintaxe para chamar *window functions*

Sintaxe genérica para chamar uma *window functions*

```
SELECT ..., <window function>, ...  
FROM <tablename> ... ;
```

Onde cada *<window function>* segue a sintaxe:

```
<aggreg function call> [ FILTER (WHERE <condição>) ]  
OVER ([<partição>] [<ordem> [<frame>]])
```

```
<window function> := <agreg function> OVER (<window specification>)  
<window specification> := [<window partition>] [<window order>] [<window frame>]  
  
<aggreg function> := ROW NUMBER() | RANK() | LEAD(<atr>) | LAG(<atr>) |  
FIRST VALUE(<atr>) | LAST VALUE(<atr>) | NTH VALUE(<atr>, <n>) |  
SUM(<atr>) | MIN(<atr>) | MAX(<atr>) | AVG(<atr> | COUNT(<atr>) ...  
  
<window partition> := PARTITION BY <atrlist>  
<window order> := ORDER BY <atrlist>  
<window frame> := {ROW|RANGE|GROUP}  
{<preceding>| [BETWEEN <preceding> AND <following>]} [exclude]
```



# Window Functions em SQL

Sintaxe para chamar *window functions*



- A opção `[ FILTER (WHERE <condição>) ]` estende qualquer função de agregação (não apenas *window functions*) para apenas contabilizar as tuplas em que a condição é atendida.

Exemplo:

```
SELECT Cidade, Count(*) AS Total,  
       Count(*) FILTER (WHERE Idade<=21) AS Menor,  
       Count(*) FILTER (WHERE Idade>21) AS Adulto  
FROM Aluno  
GROUP BY Cidade  
ORDER BY Cidade NULLS FIRST;
```



# Window Functions em SQL

## Sintaxe para chamar *window functions*



Vamos analisar cada parte da sintaxe.

- Uma *window function* é qualquer função de agregação seguida de **OVER (...)**;
- As funções de agregação usuais (**MIN()**, **MAX()**, **COUNT()**, **AVG()** etc.) podem ser usadas como *window functions*.  
Diversas outras foram criadas especialmente para ser *window functions*, tais como: **RANK()** – numera as tuplas na partição;
- Todas as cláusulas entre parênteses do **OVER (...)** são opcionais;
  - 1 partição
  - 2 ordem
  - 3 *frame*
- a palavra **OVER** é necessária para indicar que a função de agregação deve ser usada como *window function*.





# Window Functions em SQL

## Sintaxe para chamar *window functions*



- Diferença de usar uma função de agregação como *window function*: uma função de agregação “**agrega**” todas as tuplas e gera apenas um resultado, numa única tupla:

```
SELECT Min(Idade)
FROM Aluno;
```

Min(Idade)
19

- uma função de agregação usada como *window function* calcula a agregação (sobre a respectiva partição, ordenação e *frame*) **para cada tupla da tabela de entrada**:

```
SELECT nome, Cidade, Idade,
       Min(Idade) OVER()
FROM Aluno;
```

Nome	Cidade	Idade	Min(Idade)
Carlos	Sao Carlos	21	19
Celina	Sao Carlos	27	19
Cesar	Araraquara	21	19
Cibele	Araraquara	21	19
Cicero	Araraquara	22	19
...			
Celia	Rio Claro	20	19
Daniel	Campinas	19	19
Durval	(null)	(null)	19
Dora	(null)	24	19
Dina	Campinas	(null)	19

# Window Functions em SQL

Sintaxe para chamar *window functions*



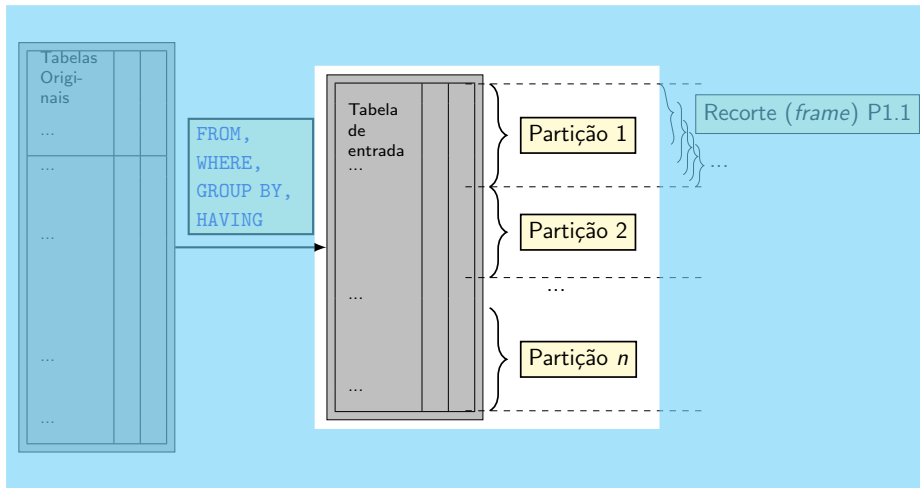
```
SELECT nome, Cidade, Idade, Min(Idade) OVER() FROM Aluno;
```

- Não foi indicado como fazer as partições. Então, a tabela de entrada inteira é “a” (única) partição, que inclui todas as tuplas;
- não foram indicados frames. Então existe um único *frame*, que inclui toda a partição;
- e como não foi indicada uma ordenação, as tuplas ficam numa ordem arbitrária.
- Portanto, **Min(Idade) OVER()**:
  - é calculada sobre todas as tuplas da tabela de entrada
  - é equivalente a um **GROUP BY Idade** (só que não tem uma cláusula **GROUP BY** no comando!)
- MAS: todas as tuplas da tabela de entrada são preservadas!
  - retorna o mesmo valor para todas as tuplas da tabela de entrada.



# Sintaxe das Window Functions em SQL

## Definindo partições



# Sintaxe das Window Functions em SQL

## Definindo partições



### Partição

Todas as tuplas da **tabela de entrada** são separadas em uma ou mais partições. Cada tupla está em exatamente uma partição.

- Uma partição é indicada por uma lista de atributos (ao menos um):
- É equivalente ao agrupamento executado pelo comando **GROUP BY**, mas sem colapsar as tuplas em uma só.



# Sintaxe das Window Functions em SQL

## Definindo partições



Sintaxe para expressar as **partições**:

```
<window function> := <agreg function> OVER (<window specification>)  
  
<window specification> := [<window partition>] [<window order>] [<window frame>]  
  
<window partition> := PARTITION BY <atrlist>  
<window order> := ORDER BY <atrlist>  
<window frame> := {ROW|RANGE} {<preceding>|[BETWEEN <preceding> AND <following>]}
```

- Tanto a sintaxe do parâmetro **PARTITION BY** quanto sua ação é semelhante às da cláusula **GROUP BY**:
  - Particiona o conjunto de tuplas da relação de entrada em subconjuntos mutuamente exclusivos:
  - Cada tupla fica em exatamente **um** subconjunto, chamado uma **Partição**.



# Window Functions em SQL

## Definindo partições

- Definindo uma partição para a *window function*, ela calcula a agregação sobre o conjunto de tuplas da partição, ainda para **cada tupla da tabela de entrada**:

```
SELECT nome, Cidade, Idade,  
       Min(Idade)  
       OVER(PARTITION BY Cidade)  
FROM Aluno;
```

Nome	Cidade	Idade	Min(Idade)
Carlos	Sao Carlos	21	21
Celina	Sao Carlos	27	21
Cesar	Araraquara	21	21
Cibele	Araraquara	21	21
Cicero	Araraquara	22	21
...			
Celia	Rio Claro	20	20
Daniel	Campinas	19	19
Dina	Campinas	(null)	19
Dora	(null)	24	24
Durval	(null)	(null)	24

- Cada *window function* pode particionar as tuplas de maneira independente das demais,

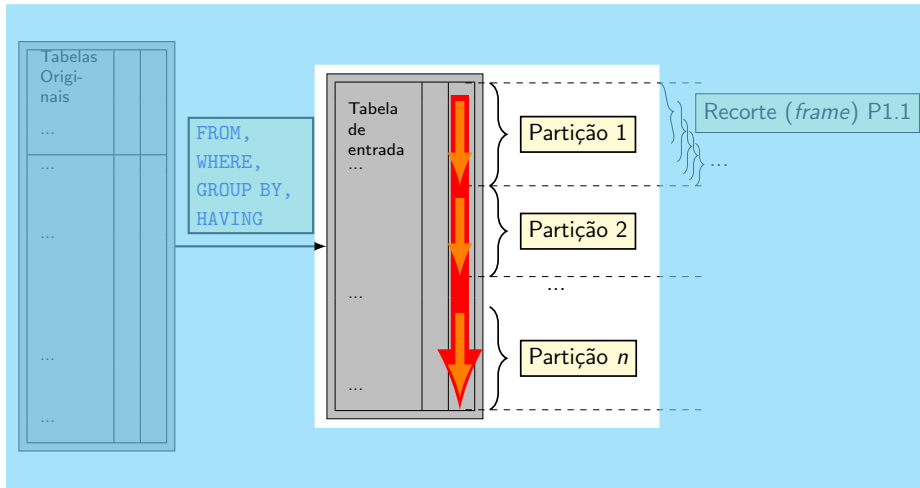
```
SELECT nome, Cidade, Idade,  
       Min(Idade) OVER(PARTITION BY Cidade)  
       Count(Idade) OVER(PARTITION BY idade)  
FROM Aluno;
```

até no mesmo comando:



# Sintaxe das Window Functions em SQL

## Definindo ordenações



# Sintaxe das Window Functions em SQL

## Definindo ordenações



- Ordenações são especificadas sobre uma lista de atributos, de maneira muito semelhante àquela da cláusula **ORDER BY**:

```
<window function> := <agreg function> OVER (<window specification>)  
<window specification> := [<window partition>] [<window order>] [<window frame>]  
  
<window partition> := PARTITION BY <atrlist>  
<window order> := ORDER BY <atrlist> [ASC|DESC] [NULLS {FIRST|LAST}]  
<window frame> := {ROW|RANGE} {<preceding>|[BETWEEN <preceding> AND <following>]}
```

- Tanto a sintaxe do parâmetro **ORDER BY** quanto sua ação são semelhantes às da cláusula **ORDER BY** do comando **SELECT**, cada partição é ordenada separadamente,
- e cada *window function* pode ordenar as tuplas de maneira independente das demais, até no mesmo comando.





# Sintaxe das Window Functions em SQL

## Definindo ordenações



Conceito associado:

Tuplas parceiras

*Peer rows*

São todas as tuplas que têm o mesmo valor que a tupla em questão nos atributos da ordenação.

- Tuplas parceiras são aquelas que o critério de ordenação não consegue desempatar.
- Dependendo dos parâmetros da *window function*, tais tuplas podem:
  - ser tomadas individualmente em qualquer ordem; ou
  - ser tomadas como um grupo só.



# Sintaxe das Window Functions em SQL

*Window functions* vinculadas à ordenação

Existem diversas *window functions* vinculadas à **ordem** das tuplas numa partição, entre elas:

## Row\_Number()

Número sequencial da tupla na partição (começa em 1).

## Rank()

Número sequencial da tupla na partição, repetido para tuplas parceiras.  
Tem 'pulos'.

## Dense\_Rank()

Número sequencial do grupo de tuplas parceiras.  
Sem 'pulos'.



# Sintaxe das Window Functions em SQL

Window functions vinculadas à ordenação

Exemplo:

```
SELECT Idade, Cidade, Nome,  
       Row_Number() OVER(ORDER BY Idade) "R#(Idade)",  
       Rank() OVER(ORDER BY Idade) "Rank(Idade)",  
       Dense_Rank() OVER(ORDER BY Idade) "DRank(Idade)",  
       Dense_Rank() OVER(ORDER BY Idade NULLS FIRST) "DRank(I/C)",  
       Row_Number() OVER(PARTITION BY Idade ORDER BY Cidade) "R#(I/C)"  
FROM Aluno  
ORDER BY Idade NULLS FIRST, Cidade;
```

Idade	Cidade	Nome	R#(Idade)	Rank(Idade)	DRank(Idade)	DRank(I/C)	R#(I/C)
(null)	Campinas	Dina	14	14	10	1	1
(null)	(null)	Durval	15	14	10	1	2
19	Campinas	Daniel	1	1	1	2	1
20	Rio Claro	Celia	2	2	2	3	1
21	Araraquara	Cesar	4	3	3	4	2
21	Araraquara	Cibele	3	3	3	4	1
21	Ibitinga	Carlitos	5	3	3	4	3
21	Sao Carlos	Carlos	6	3	3	4	4
22	Araraquara	Cicero	7	7	4	5	1
22	Sao Carlos	Celso	8	7	4	5	2
23	Sao Carlos	Catarina	9	9	5	6	1
24	(null)	Dora	10	10	6	7	1
25	Rio Claro	Corina	11	11	7	8	1
27	Sao Carlos	Celina	12	12	8	9	1
35	Ibate	Denise	13	13	9	10	1



# Window Functions em SQL

## Definindo ordenações – o *frame default*



- Cada tupla de uma partição está associada a um “recorte” (*frame*).
- Se o *frame* não for explicitado, usa-se um *default*, que depende da existência ou não do parâmetro **ORDER BY**:
  - ☞ Se não houver **ORDER BY**, todas as tuplas da partição têm o mesmo *frame*, que corresponde a todas as tuplas da partição;
  - ☞ Se houver **ORDER BY**, o *frame* de uma tupla corresponde a todas as tuplas desde a primeira da ordenação até aquela tupla.
- A *window function* é aplicada sobre o *frame* de cada tupla para cada tupla da partição.

### Conceito associado:

#### *Frame default*

É o *frame* assumido para cada tupla quando a ordenação não o declara explicitamente. Corresponde a **RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**.

# Window Functions em SQL

## Definindo ordenações – Exemplo

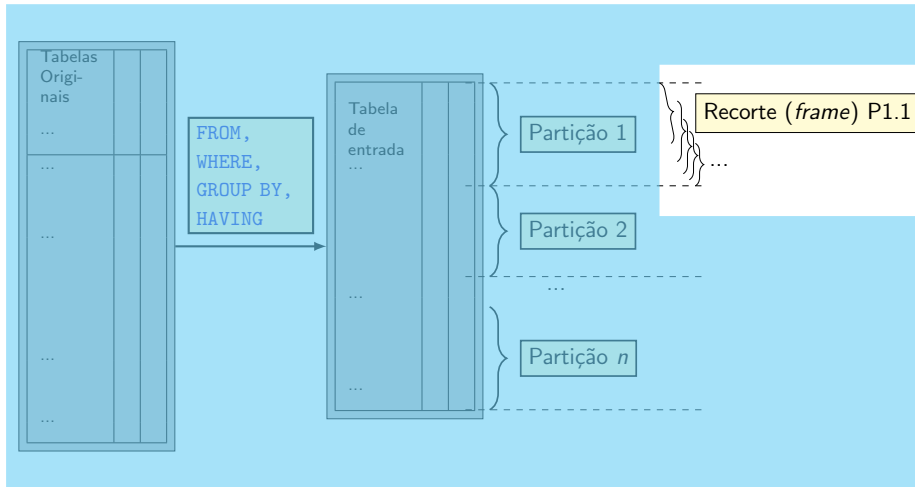
```
Select Nome, Cidade, Idade,  
       rank() OVER (PARTITION BY Cidade) AS "(1)"  
       min(idade) OVER (PARTITION BY Cidade), AS "(2)"  
       Max(idade) OVER (PARTITION BY Cidade), AS "(3)"  
       TRUNC(Avg(idade) OVER (PARTITION BY Cidade),2), AS "(4)"  
       rank() OVER (PARTITION BY Cidade ORDER BY Nome), AS "(5)"  
       min(idade) OVER (PARTITION BY Cidade ORDER BY Nome), AS "(6)"  
       Max(idade) OVER (PARTITION BY Cidade ORDER BY Nome), AS "(7)"  
       TRUNC(Avg(idade) OVER (PARTITION BY Cidade ORDER BY Nome),2) AS "(8)"  
FROM Aluno  
ORDER BY Cidade;
```

Nome	Cidade	Idade	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Cesar	Araraquara	21	1	21	22	21.33	1	21	21	21.00
Cibele	Araraquara	21	1	21	22	21.33	2	21	21	21.00
Cicero	Araraquara	22	1	21	22	21.33	3	21	22	21.33
Daniel	Campinas	19	1	19	19	19.00	1	19	19	19.00
...										
Carlos	Sao Carlos	21	1	21	27	23.25	1	21	21	21.00
Catarina	Sao Carlos	23	1	21	27	23.25	2	21	23	22.00
Celina	Sao Carlos	27	1	21	27	23.25	3	21	27	23.66
Celso	Sao Carlos	22	1	21	27	23.25	4	21	27	23.25
Dora		24	1	24	24	24.00	1	24	24	24.00
Durval			1	24	24	24.00	2	24	24	24.00



# Sintaxe das Window Functions em SQL

## Especificando *frames*



# Sintaxe das Window Functions em SQL

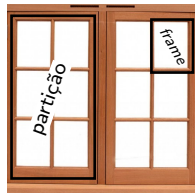
Especificando *frames*



## Recorte – ou *Frame*

É um subconjunto das tuplas de uma partição, formado por tuplas consecutivas de acordo com uma ordenação.

(Se não for indicada, uma ordenação arbitrária é usada.)



## Tupla corrente

*Current row*

Cada tupla de uma partição tem um *frame* associado. A tupla corrente é a tupla à qual o *frame* está associado.

(A tupla corrente não precisa fazer parte de seu *frame*)



# Sintaxe das Window Functions em SQL

## Especificando *frames*



- A cláusula **<window frame>** permite explicitar como o *frame* associado a cada tupla é formado:

```
<window function> := <agreg function> OVER (<window specification>)  
<window specification> := [<window partition>] [<window order>] [<window frame>]  
  
<window partition> := PARTITION BY <atrlist>  
<window order> := ORDER BY <atrlist> ...  
<window frame> := {ROW|RANGE|GROUP} {<qual>|[BETWEEN <qual> AND <qual>]} [exclude]  
  
<qual> := [UNBOUNDED PRECEDING | <quanto> PRECEDING |  
          | CURRENT ROW | <expr> FOLLOWING | [UNBOUNDED FOLLOWING]  
<exclude> := {EXCLUDE CURRENT ROW | EXCLUDE GROUP | EXCLUDE TIES |  
              EXCLUDE NO OTHERS}
```

- Somente tem sentido declarar *Frames* explicitamente se a ordem for indicada





# Window Functions em SQL

## Especificando *frames*

- *Window functions* permitem trabalhar com outras tuplas além da corrente, dentro da partição, relativas à ordem indicada.

LAG (<expressão> [,offset [, valor default]])  
ORDER BY <expressão da ordem> [ASC | DESC]  
Acessa uma tupla anterior à corrente.

LEAD (<expressão> [,offset [, valor default]])  
ORDER BY <expressão da ordem> [ASC | DESC]  
Acessa uma tupla posterior à corrente.

```
Select Nome, Cidade, Idade,  
       Idade - LAG(Idade) OVER (PARTITION BY Cidade ORDER BY Idade)  
FROM Aluno  
ORDER BY Cidade;
```



# Window Functions em SQL

## Especificando *frames*

- Cada *frame* é indicado pela **tupla inicial** e pela **tupla final**: todas as tuplas que na ordenação ficaram entre essas duas (inclusive) formam o *frame*;
  - ★ e podem ser obtidas pelas *window functions*: `First_Value`, `Last_Value` e `Nth_Value`!

`FIRST_VALUE (<expressão> [PARTITION BY <Expressão de partição>])  
ORDER BY <expressão da ordem> [ASC | DESC]`  
Acessa a primeira tupla do *frame*.

`LAST_VALUE (<expressão> [PARTITION BY <Expressão de partição>])  
ORDER BY <expressão da ordem> [ASC | DESC]`  
Acessa a última tupla do *frame*.

`Nth_VALUE (<expressão> ,offset)  
ORDER BY <expressão da ordem> [ASC | DESC]  
<especificação do frame>` Acessa a primeira tupla do *frame*.



# Window Functions em SQL

## Especificando *frames*



- Cada *frame* é indicado pela **tupla inicial** e pela **tupla final**: todas as tuplas que na ordenação ficaram entre essas duas (inclusive) formam o *frame*;
- ★ e podem ser obtidas pelas *window functions*: `First_Value`, `Last_Value` e `Nth_Value`!
- Quando a ordem não é indicada, as tuplas ficam em qualquer ordem;
- Quando o parâmetro **<window specification>** é omitido, o *frame* associado a cada tupla (que é chamada a **tupla corrente**), é definido por *default* da seguinte maneira:
  - a **primeira tupla** da partição é a **tupla inicial** (e portanto é a mesma para todos os *frames*),
  - a **tupla final** é a **tupla corrente** (e portanto cada *frame* vai sucessivamente acrescentando a tupla corrente ao *frame* da tupla anterior;
- Quando o parâmetro **<window specification>** indica apenas o início do *frame* então a **tupla final** é a **tupla corrente**.

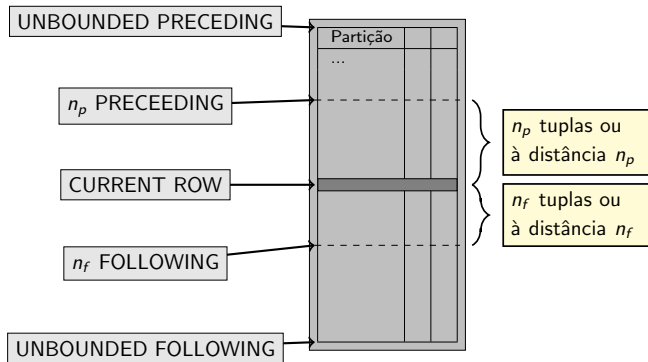


# Window Functions em SQL

## Especificando *frames*



A especificação do *frame* pode indicar tuplas ou grupos antes e/ou depois da tupla corrente:



# Window Functions em SQL

## Especificando *frames*



A indicação do *frame* pode ser feita de três modos:

**ROWS** Em modo de tuplas, o início e o fim correspondem simplesmente à tupla (primeira, última,  $n$ -ésima) indicada;

**RANGE** Em modo de faixas, o início e o fim correspondem à tupla (primeira, última, que está à distância  $n$  da tupla corrente), e:

👉 se houver empate entre tuplas à mesma distância segundo o critério de ordenação, todo o grupo é tomado junto;

👉 deve ser especificada exatamente uma coluna para ordenação.

**GROUP** Em modo de grupos, o início e o fim correspondem à tupla (primeira, última, do  $n$ -ésimo grupo a partir do grupo da tupla corrente);

👉 podem ser especificadas uma ou mais colunas para a ordenação.

👉 As indicações de início e fim do *frame* mudam de acordo com o modo.



# Window Functions em SQL

Especificando *frames* em modo de tuplas



Em modo de tuplas (**ROWS**):

**UNBOUNDED PRECEDING** - Corresponde à primeira tupla da partição

$n_p$  **PRECEDING** - Corresponde a  $n_p$  tuplas antes da tupla corrente (se existir)

**CURRENT ROW** - Corresponde à tupla corrente

$n_f$  **FOLLOWING** - Corresponde a  $n_f$  tuplas depois da tupla corrente (se existir)

**UNBOUNDED FOLLOWING** - Corresponde à última tupla da partição

- Tuplas que não existem são tratadas como nulas.



# Window Functions em SQL

Especificando *frames* em modo de faixas

Em modo de faixas (**RANGE**):

**UNBOUNDED PRECEDING** - Corresponde à primeira tupla e a todas as que empatem no critério de ordenação

$n_p$  **PRECEDING** - Corresponde a todas as tuplas que estão até à distância  $n_p$  antes da tupla corrente

**CURRENT ROW** - Corresponde à tupla corrente

$n_f$  **FOLLOWING** - Corresponde a todas as tuplas que estão até à distância  $n_p$  depois da tupla corrente

**UNBOUNDED FOLLOWING** - Corresponde à última tupla e a todas as que empatem no critério de ordenação

★ As especificações  $n_p$  **PRECEDING** e  $n_f$  **FOLLOWING** somente podem ser usadas quando o critério de ordenação é de tipo numérico ou data:

☞ SQL não tem como calcular distâncias entre valores de tipos textuais ou **BLOB**.

- Quando o *frame* está em modo faixa (**RANGE**), só pode ter um atributo no critério de ordenação.



# Window Functions em SQL

Especificando *frames*: exclusão de tuplas



A indicação de exclusão permite eliminar tuplas ao redor da **CURRENT ROW** no *frame* corrente:

**EXCLUDE CURRENT ROW** exclui a tupla corrente do *frame*

**EXCLUDE GROUP** exclui a tupla corrente e todas as que empatam no critério de ordenação com ela.

**EXCLUDE TIES** exclui as tuplas empatadas mas não a tupla corrente

**EXCLUDE NO OTHERS** é o padrão: não exclui nada.





# Window Functions em SQL

## Especificando *frames* – Exemplo 1

Para o exemplo seguinte, considere a relação *Matricula* definida como:

```
Matricula=(NUSP, Sigla, CodTurma, NotaP1, ... Registro DATE, ...)
```

Sobre ela é feita a contagem de quantos alunos se matriculam por dia (data do *Registro*) em cada *turma* de cada *sigla*:

```
INSERT INTO QtMatric
SELECT Registro AS Dia, Sigla, CodTurma,
       Count(*) AS Qt
FROM matricula
GROUP BY Registro, Sigla, CodTurma;
```

Dia	Sigla	CodTurma	Qt
2016-01-27	SCC0201	100	2
2016-01-27	SCC0201	200	5
2016-01-27	SCC0201	201	8
2016-01-27	SCC0202	101	3
2016-01-27	SCC0215	104	5
2016-01-27	SCC0215	204	6
2016-01-27	SCC0215	205	6
2016-01-27	SCC0215	206	10
2016-01-28	SCC0201	100	2
2016-01-28	SCC0215	102	4
2016-01-28	SCC0215	104	2
2016-01-29	SCC0201	100	3
2016-01-29	SCC0202	101	6
2016-01-29	SCC0215	102	3
2016-01-29	SCC0215	104	2
2016-01-30	SCC0215	104	1
...			



# Window Functions em SQL

## Especificando frames – Exemplo 1

```
SELECT Rank() OVER (ORDER BY Dia, Sigla) AS Seq, T.*,  
       SUM(Qt) OVER (PARTITION BY Dia) AS Tot1,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla) AS Tot2,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla, Qt) AS Tot3,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla ROWS BETWEEN  
                     UNBOUNDED PRECEDING AND CURRENT ROW) AS Tot4,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla RANGE BETWEEN  
                     UNBOUNDED PRECEDING AND CURRENT ROW) AS Tot5,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla ROWS BETWEEN  
                     2 PRECEDING AND CURRENT ROW) AS Tot6  
FROM QtMatric T  
ORDER BY Dia, Sigla;
```

Seq	Dia	Sigla	Qt	Tot1	Tot2	Tot3	Tot4	Tot5	Tot6
1	2016-03-27	SCC0201	2	45	15	2	2 =2	15	2 =2
1	2016-03-27	SCC0201	5	45	15	7	7 =2+5	15	7 =2+5
1	2016-03-27	SCC0201	8	45	15	15	15 =2+5+8	15	15 =2+5+8
4	2016-03-27	SCC0202	3	45	18	18	18 =2+5+8+3	18	16 =5+8+3
5	2016-03-27	SCC0215	5	45	45	23	33 =2+5+8+3+10+5	45	18 =3+10+5
5	2016-03-27	SCC0215	6	45	45	35	45 =2+5+8+3+10+5+6+6	45	17 =5+6+6
5	2016-03-27	SCC0215	6	45	45	35	39 =2+5+8+3+10+5+6	45	21 =10+5+6
5	2016-03-27	SCC0215	10	45	45	45	28 =2+5+8+3+10	45	21 =8+3+10
9	2016-03-28	SCC0201	4	42	12	6	12 =6+2+4	12	12 =6+2+4
9	2016-03-28	SCC0201	2	42	12	2	8 =6+2	12	8 =6+2
...									



# Window Functions em SQL

## Especificando *frames* – Exemplo 1



`Rank() OVER (ORDER BY Dia, Sigla)`

- Como não foi definida partição, todas as tuplas estão num “ranqueamento” só;
- Todas as tuplas que empatam no critério de ordenação `ORDER BY Dia, Sigla` têm o mesmo *rank*.

`PARTITION BY Dia`

- Se não existe critério de ordenação, todas as tuplas de cada partição são tomadas em conjunto e têm o mesmo resultado da *window function*;

`PARTITION BY Dia ORDER BY Sigla`

- Se é dado o critério de ordenação mas não a *window frame*, ela tem o *default*

`RANGE BETWEEN UNBOUNDED`

`PRECEDING AND CURRENT ROW`.

`ORDER BY Sigla, Qt`

- Se existem tuplas repetidas no critério de ordenação, em modo de faixa elas são tomadas como um grupo só e todas tem o mesmo valor da *window function*.

`ORDER BY Sigla`

- Se existem tuplas repetidas no critério de ordenação, em modo de tuplas elas são tomadas em qualquer ordem.



# Window Functions em SQL

## Especificando *frames* – Exemplo 2



```
SELECT Rank() OVER (ORDER BY Dia, Sigla) AS Seq, T.*,  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla ROWS BETWEEN  
                     2 PRECEDING AND CURRENT ROW) AS Tot6, -- OK!  
       SUM(Qt) OVER (PARTITION BY Dia ORDER BY Sigla RANGE BETWEEN  
                     2 PRECEDING AND CURRENT ROW) AS Tot7 -- Erro!  
FROM QtMatric T  
ORDER BY Dia, Sigla;
```

- Modo de tupla pode ser usado para qualquer critério de ordenação: Duas tuplas antes de **CURRENT ROW** está ok.
- Modo de faixa somente pode ser usado quando o critério de ordenação permite calcular distância:

```
CURRENT ROW.Sigla -2 ⇔ 'SCC0215'-2 =???
```



MBA em Inteligência Artificial e Big Data  
– Curso 3: Administração de Dados Complexos em Larga Escala –  
★ **SQL/DML *Window Functions* em SQL** ★

Caetano Traina Júnior

Grupo de Bases de Dados e Imagens – GBdI  
Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo - São Carlos



Funções de  
janelamento  
**FIM**