

Laboratório de Bases de Dados

PROF. JOSÉ FERNANDO RODRIGUES JÚNIOR

AULA 3 - REVISÃO SQL/DML

MATERIAL ORIGINAL EDITADO: PROFA. ELAINE PARROS MACHADO DE SOUSA

DML - Introdução

Comandos da DML:

- INSERT
- UPDATE
- DELETE
-  ◦ SELECT

SQL importance: <https://spectrum.ieee.org/top-programming-languages-2022>

Comandos DML

SELECT – comando de consulta

- retorno \Rightarrow tabela resultado (**multiconjunto** – potencialmente um **conjunto com repetições**)

```
SELECT [DISTINCT|ALL] <lista de atributos>
FROM <lista de tabelas>
[WHERE <condições>]
[GROUP BY atributo]
[HAVING <condições>]
[ORDER BY atributo [ASC|DESC]]
```

SELECT

SELECT → **O QUE** se deseja na tabela resultado

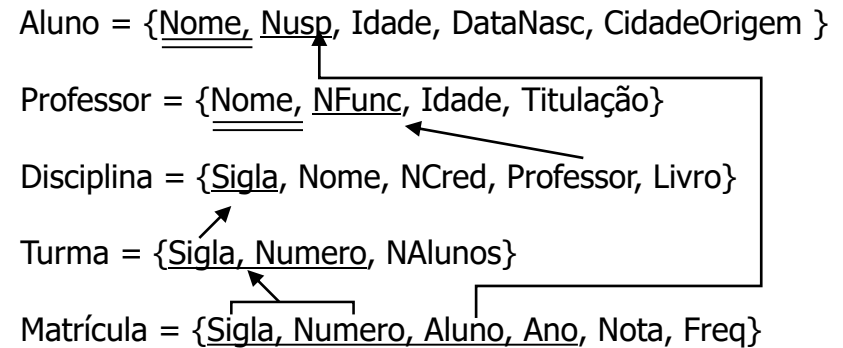
- *<lista de atributos>* ou
- * (para todos os atributos)
- **ALL** – resultado pode conter tuplas duplicadas (*default*)
- **DISTINCT** – resultado contém somente tuplas distintas

FROM → **DE ONDE** retirar os dados necessários

WHERE → **CONDIÇÕES** (predicado) da consulta

- expressão condicional booleana
- condições de seleção
- condições de junção, ...

Exemplo:



- Selecionar os alunos (NUSP) que cursam a disciplina SCC541 ou a SCC 241;

```
SELECT Aluno FROM Matricula  
WHERE Sigla IN ('SCC541', 'SCC241');
```

- Selecionar os alunos (NUSP) que cursam alguma disciplina do SCC no ano de 2010;

```
SELECT Distinct Aluno FROM Matricula  
WHERE Sigla LIKE 'SCC%' and Ano = 2010;
```

SELECT

Cláusula **FROM** com mais de uma tabela

- **Junção interna** (*Inner Join*)
- **WHERE** \Rightarrow condição de junção
 - em geral: atributos com relacionamento PK - FK


```
SELECT [DISTINCT|ALL] <atributos>  
FROM tabela1, tabela2  
WHERE tabela1.atributo1 =  
      tabela2.atributo2
```

Exemplo: Junção

Aluno = {Nome, <u>NUSP</u> }	Matricula= { <u>Sigla</u> , Numero, Aluno, Ano}
{<Zeca, 11111>, <Zico, 22222>, <Juca, 33333>, <Tuca, 44444> }	{<SCC-125, 1, 11111, 2010>, <SCC-148, 1, 11111, 2010>, <SCC-125, 2, 22222, 2010>, <SCC-148, 1, 22222, 2009>}

```
select A.nome, A.nusp, M.Sigla  
from Aluno A, Matricula M  
where A.nusp = M.aluno
```

{Nome, NUSP, Sigla}
{<Zeca, 11111, SCC-125>,
<Zeca, 11111, SCC-148>,
<Zico, 22222, SCC-125>,
<Zico, 22222, SCC-148 >}



Junção Interna – operador JOIN

```
SELECT [DISTINCT|ALL] <atributos>  
FROM tabela1 T1  
[INNER] JOIN tabela2 T2  
ON T1.atributo1 = T2.atributo2
```


Junção Interna

```
SELECT <atributos>  
  FROM tabela1 T1 , tabela2 T2  
 WHERE T1.atributo1 = T2.atributo2
```



```
SELECT <atributos>  
  FROM tabela1 T1 JOIN tabela2 T2  
 ON T1.atributo1 = T2.atributo2
```

NATURAL JOIN

- Se T1.atributo1 tem o mesmo nome que T2.atributo2:

```
SELECT <atributos>
```

```
FROM tabela1 NATURAL JOIN tabela2
```

→ Vale o mesmo para chaves compostas de mais que um atributo.

Junções Externas

```
SELECT [DISTINCT|ALL] <atributos>  
FROM tabela1 T1  
[LEFT | RIGHT | FULL] JOIN tabela2 T2  
ON T1.atributo1 = T2.atributo2
```

```
SELECT [DISTINCT|ALL] <atributos>  
FROM tabela1 T1, tabela2 T2  
WHERE T1.atributo1[(+)] = T2.atributo2[(+)]
```

Junções Externas

- *LEFT JOIN COM (+)*

```
SELECT [DISTINCT|ALL] <atributos>  
      FROM tabela1 T1, tabela2 T2  
      WHERE T1.atributo1 = T2.atributo2 (+)
```

- *RIGHT JOIN COM (+)*

```
SELECT [DISTINCT|ALL] <atributos>  
      FROM tabela1 T1, tabela2 T2  
      WHERE T1.atributo1 (+) = T2.atributo2
```

Junções Externas

- LEFT JOIN COM (+)

SELECT

FROM

WHERE

- RIGHT JOIN

SELECT

FROM

O (+) não é interpretado de acordo com qual lado ele está (left ou right); ele apenas indica de qual tabela serão aceitos valores null.

(+)

WHERE *T1.atributo1 (+) = T2.atributo2*

Junção

O que é mais eficiente, usar JOIN explicitamente, ou implicitamente via uma lista de nomes de tabelas separadas por vírgula (e condição de junção no WHERE)?

R.: O otimizador de consultas possui diversos mecanismos para a execução de junções eficientemente; estes recursos sempre serão usados quando uma consulta envolver junções expressas explicitamente com JOIN ou com uma lista de tabelas.

Todavia, quando usa-se a palavra JOIN explicitamente, o otimizador sempre saberá que há uma junção envolvida, pois é simples de detectar na sintaxe. Quando se usa uma lista de tabelas com condição WHERE, o otimizador precisa inferir a junção, existindo a possibilidade de não identifica-la, o que leva à execução de um caro produto cartesiano.

Isto pode acontecer quando a cláusula WHERE envolver um conjunto grande e complexo de predicados e múltiplas tabelas.

Portanto, o uso do JOIN explícito é recomendado, mas não garante melhor desempenho na maioria dos casos.

Estas são afirmações que se aplicam ao PostgreSQL. Em Oracle, não há documentação afirmando a mesma coisa. A princípio o otimizador do Oracle sempre resolve da melhor maneira.

Mais detalhes: <https://www.postgresql.org/docs/current/explicit-joins.html>

Exemplo: Junção Externa

Aluno = {Nome, NUSP}

{<Zeca, 11111>,
<Zico, 22222>,
<Juca, 33333>,
<Tuca, 44444> }

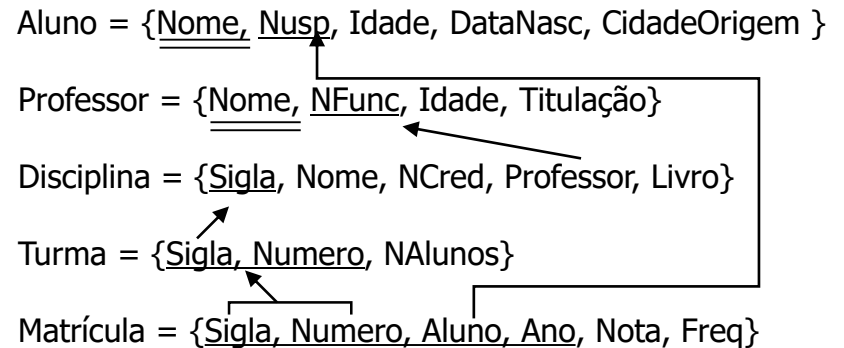
Matricula= {Sigla, Numero, Aluno, Ano}

{<SCC-125, 1, 11111, 2010>,
<SCC-148, 1, 11111, 2010>,
<SCC-125, 2, 22222, 2010>,
<SCC-148, 1, 22222, 2009>}

```
select A.nome, A.nusp, M.Sigla  
from Aluno A left join Matricula M  
ON A.nusp = M.aluno
```

{Nome, NUSP, Sigla}
{<Zeca, 11111, SCC-125>,
<Zeca, 11111, SCC-148>,
<Zico, 22222, SCC-125>,
<Zico, 22222, SCC-148>,
<Juca, 33333, NULL >,
<Tuca, 44444, NULL>}

Exemplo:



- Selecionar nome e nro funcional dos professores DOUTORES que ministram ou não ministram disciplinas.

```
select P.Nome, P.NFunc, D.Sigla
from Disciplina D right join Professor P
on D.Professor = P.Nfunc
where UPPER(P.Titulacao) = UPPER('doutorado')
```


GROUP BY

- Funções Agregadas
 - entrada \Rightarrow conjunto de valores
 - saída \Rightarrow um valor por grupo
 - Exemplos:
 - `AVG(nota)` \rightarrow calcula a nota média de todos os alunos
 - `AVG(nota) ... GROUP BY Disciplina` \rightarrow calcula a nota média dos alunos em cada disciplina
 - `COUNT()`
 - `count(*)` – retorna o número de tuplas de cada grupo
 - `count(atributo)` – retorna o nro de valores (com repetição) da coluna *atributo* que não tem valores null

GROUP BY

Funções Agregadas

Exemplos

- **MAX**(*atributo*) → o valor máximo da coluna *atributo*
- **MIN**(*atributo*) → o valor mínimo da coluna *atributo*
- **SUM**(*atributo*) → soma de valores da coluna *atributo*
- **AVG**(*atributo*) → média de valores da coluna *atributo*
- **MEDIAN**(*atributo*) → mediana da coluna *atributo*
- **VARIANCE**(*atributo*) → desvio padrão da coluna *atributo*
- **STDDEV**(*atributo*) → desvio padrão da coluna *atributo*
* $STDDEV = VARIANCE^{1/2}$
- **COUNT**(*) → recupera o número de tuplas

Dentre outras:

https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions001.htm#i89203

GROUP BY

GROUP BY, ou agrupamento, assume a presença de valores repetidos → portanto, apesar de aceito, não faz sentido a realização de agrupamentos sobre os atributos chave

GROUP BY

- **GROUP BY** → agrupamento de tuplas
 - para a aplicação de funções agregadas
- **HAVING** → condições aplicadas **a grupos já formados** por **GROUP BY**
- **ORDER BY** → estabelece a ordenação lógica da tabela de resultados
 - **ASC** (*default*)
 - **DESC**

Exemplo:

Aluno = {Nome, NUSP}

{<Zeca, 11111>,
<Zico, 22222>,
<Juca, 33333>,
<Tuca, 44444> }

Matricula= {Sigla, Numero, Aluno, Ano, Nota}

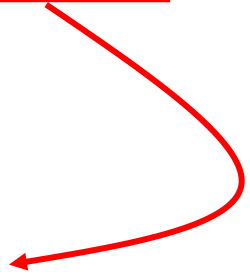
{<SCC-125, 1, 11111, 2010, 5.0>,
<SCC-148, 1, 11111, 2010, 7.0>,
<SCC-125, 2, 22222, 2010, 5.0>,
<SCC-148, 1, 22222, 2009, 4.0>}

- Selecionar, para cada aluno, seu nome e a média das notas das disciplinas em que foi aprovado (nota ≥ 5). Ordenar por nome de aluno

1º Passo: seleção e junção

```
SELECT ...  
  FROM Aluno A JOIN Matricula M  
        ON M.Aluno = A.NUSP  
 WHERE M.Nota BETWEEN 5.0 AND 10.0
```

{Nome, NUSP, Sigla, Nota}
{<Zeca, 11111, SCC-125, 5.0>,
<Zeca, 11111, SCC-148, 7.0>,
<Zico, 22222, SCC-125, 5.0>}



Exemplo: (continuação)

2º Passo: agrupamento e agregação

```
SELECT A.Nome, AVG(M.Nota) as Media  
FROM Aluno A JOIN Matricula M  
ON M.Aluno = A.NUSP  
WHERE M.Nota BETWEEN 5.0 AND 10.0  
GROUP BY A.Nome  
ORDER BY A.Nome;
```

Grupo Zeca

<SCC125, 5.0>
<SCC148, 7.0>

Grupo Zico

<SCC125, 5.0>

Função **AVG** aplicada sobre cada grupo



{Nome, Media}
{<Zeca, 6.0>,
<Zico, 5.0>}

Exemplo:

Aluno = {Nome, NUSP}

{<Zeca, 11111>,
<Zico, 22222>,
<Juca, 33333>,
<Tuca, 44444> }

Matricula= {Sigla, Numero, Aluno, Ano, Nota}

{<SCC-541, 1, 11111, 2009, 3.0>,
<SCC-541, 1, 11111, 2010, 7.0>,
<SCC-240, 1, 11111, 2010, 5.0>,
<SCC-240, 1, 22222, 2009, 4.0>}

Disciplina = {Sigla, Nome}

{<SCC-541, LabBD>,
<SCC-240, BD>}

- Selecionar os nomes dos alunos que fizeram uma mesma disciplina mais de uma vez. Listar também o nome da disciplina, o nro de vezes que cursou e a nota máxima que o aluno obteve (considerando todas as vezes que cursou).

1º Passo: junção

```
select ....  
    from Aluno A join Matricula M  
        on A.NUSP = M.Aluno  
    join Disciplina D  
        on D.Sigla = M.Sigla
```

Exemplo: (continuação)

2º Passo: agrupamento e agregação

```
select A.Nome, D.Nome, count(*), max(M.Nota)
  from Aluno A join Matricula M
            on A.NUSP = M.Aluno
    join Disciplina D
    on D.Sigla = M.Sigla
 group by A.Nome, D.Nome
```

Grupo Zeca

sub-grupo LabBD

<LabBD, 3.0>

<LabBD, 7.0>

sub-grupo BD

<BD, 5.0>

Grupo Zico

sub-grupo BD

<BD, 5.0>

Funções **COUNT** e **MAX** aplicadas sobre cada sub-grupo

Exemplo: (continuação)

3º Passo: condição having

```
select A.Nome, D.Nome, count(*), max(M.Nota)
  from Aluno A join Matricula M
           on A.NUSP = M.Aluno
  join Disciplina D
    on D.Sigla = M.Sigla
  group by A.Nome, D.Nome
  having count(*) > 1
```

Grupo Zeca

sub-grupo LabBD

<LabBD, 3.0>

<LabBD, 7.0>

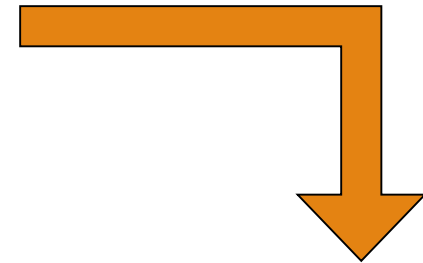
~~sub-grupo BD~~

~~<BD, 5.0>~~

Grupo Zico

~~sub-grupo BD~~

~~<BD, 5.0>~~



{A.Nome, D.Nome, count, max}
{< Zeca, LabBD, 2, 7.0>}

GROUP BY

- Problema do GROUP BY

Exemplo:

```
SELECT sigla, max(nota), max(frequenciaporc)
FROM matricula
GROUP BY sigla
```

➔ Quem são os alunos com maior nota e maior frequência?

GROUP BY

```
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

➔ Com GROUP BY, perde-se a informação de quem são os alunos com as idades mínima e máxima!

Analytic Function

- **OVER ()** → função de agregação computada sobre partições (grupos) de tuplas indicadas por um dado atributo
- Recurso chamando de Window Function ou Analytic Function

Analytic Function

- Exemplo:

SELECT Nome, Idade, Cidade

FROM Aluno

Quais alunos têm
idade mínima e máxima
em cada cidade?

	⚡ NOME	⚡ IDADE	⚡ CIDADE
1	Albertina	18	Descalvado
2	Ana	20	Jau
3	Adriana	18	Limeira
4	Adolfo	21	Limeira
5	Aline	19	Limeira
6	Andre	19	Lins
7	Petrus	30	Marilia
8	Betina	18	Sao Carlos
9	Adilson	20	Sao Carlos
10	Marcio	19	Sao Carlos
11	Bruno	21	Sao Carlos
12	Marcos	35	Vitoria

Analytic Function

- Exemplo:

```
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
1	Albertina	Descalvado	18	18	18
2	Ana	Jau	20	20	20
3	Adriana	Limeira	18	18	21
4	Adolfo	Limeira	21	18	21
5	Aline	Limeira	19	18	21
6	Andre	Lins	19	19	19
7	Petrus	Marilia	30	30	30
8	Betina	Sao Carlos	18	18	21
9	Adilson	Sao Carlos	20	18	21
10	Marcio	Sao Carlos	19	18	21
11	Bruno	Sao Carlos	21	18	21
12	Marcos	Vitoria	35	35	35

Analytic Function

```
SELECT * FROM(  
  
SELECT nome, Cidade, Idade,  
  
Min(Idade) OVER(Partition by Cidade) as IdMin,  
  
Max(Idade) OVER(Partition by Cidade) as IdMax  
  
FROM Aluno)  
  
WHERE idade = IdMin or idade = IdMax
```

	NOME	CIDADE	IDADE	IDMIN	IDMAX
1	Albertina	Descalvado	18	18	18
2	Ana	Jau	20	20	20
3	Adriana	Limeira	18	18	21
4	Adolfo	Limeira	21	18	21
5	Andre	Lins	19	19	19
6	Petrus	Marilia	30	30	30
7	Betina	Sao Carlos	18	18	21
8	Bruno	Sao Carlos	21	18	21
9	Marcos	Vitoria	35	35	35

Analytic Function x GROUP BY

```
SELECT * FROM(  
  
SELECT nome, Cidade, Idade,  
  
Min(Idade) OVER(Partition by Cidade) as IdMin,  
  
Max(Idade) OVER(Partition by Cidade) as IdMax  
  
FROM Aluno)  
  
WHERE idade = IdMin or idade = IdMax
```

	NOME	CIDADE	IDADE	IDMIN	IDMAX
1	Albertina	Descalvado	18	18	18
2	Ana	Jau	20	20	20
3	Adriana	Limeira	18	18	21
4	Adolfo	Limeira	21	18	21
5	Andre	Lins	19	19	19
6	Petrus	Marilia	30	30	30
7	Betina	Sao Carlos	18	18	21
8	Bruno	Sao Carlos	21	18	21
9	Marcos	Vitoria	35	35	35

➔ Com a cláusula OVER(), as tuplas em cada partição (grupo) são mantidas, e faz-se uma **concatenação** com o resultado da agregação

Analytic Function x GROUP BY

A operação de GROUP BY ocorre do mesmo modo, a diferença é que as tuplas envolvidas são concatenadas ao resultado das funções de agregação.

Analytic Function x GROUP BY

- Para gerar o mesmo resultado usando GROUP BY:

```
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)
ORDER BY CIDADE
```

	NOME	CIDADE	IDADE	IDMIN	IDMAX
1	Albertina	Descalvado	18	18	18
2	Ana	Jau	20	20	20
3	Adriana	Limeira	18	18	21
4	Adolfo	Limeira	21	18	21
5	Andre	Lins	19	19	19
6	Petrus	Marilia	30	30	30
7	Betina	Sao Carlos	18	18	21
8	Bruno	Sao Carlos	21	18	21
9	Marcos	Vitoria	35	35	35

Analytic Function

- Funções relacionadas à ordem das tuplas nas partições:

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'.

Analytic Function

- Funções relacionadas à ordem das tuplas nas partições:

```
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

Analytic Function

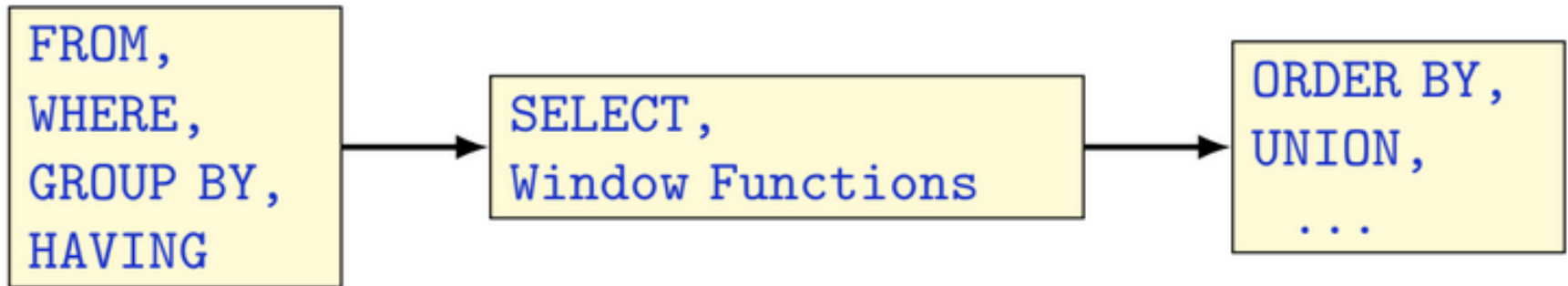
- Funções relacionadas à ordem das tuplas nas partições:

```
SELECT Idade, Cidade, Nome,
       Row_Number() OVER(ORDER BY Idade) "R#(Idade)" ordem da tupla – não há empates
       Rank() OVER(ORDER BY Idade) "Rank(Idade)" ordem da tupla - empates contam
       Dense_Rank() OVER(ORDER BY Idade) "DR" ordem da tupla – empates não contam
       Dense_Rank() OVER(ORDER BY Idade) "DR" ordem da tupla – empates não contam, nulls primeiro
       Row_Number() OVER(ORDER BY Idade, Cidade) "R#(I/C)" das pessoas com de mesma idade, qual a ordem dada pela cidade de origem
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

Analytic Function

- Ordem de execução:



Isto é, a cláusula OVER é executada depois de FROM, WHERE, GROUP BY/HAVING e antes de ORDER BY e STOP AFTER

➔ Mais:

https://docs.oracle.com/cd/E11882_01/server.112/e41084/functions004.htm#SQLRF06174

Consultas Aninhadas (Nested Queries)

Não correlacionadas – independentes

- ex: selecionar nome e nusp dos alunos com a idade mais alta

```
select nome, nusp from aluno
where idade IN
      (select max(idade)
       from aluno)
```

Consultas Aninhadas (Nested Queries)

Não correlacionadas – independentes

- ex: selecionar nome e nusp dos alunos com a idade mais alta

`select nome, nusp`
`where idade >=`

Consultas IN funcionam trazendo dados de “fora” para “dentro” da consulta principal.

`(select max(idade)`
`from aluno)`

Consultas Aninhadas

Correlacionadas – condição na cláusula WHERE da consulta interna referencia algum atributo de tabela da consulta externa

EXEMPLO:

Aluno = {Nome, Nusp, Idade, DataNasc}

Disciplina = {Sigla, Nome, NCred, Professor, Livro, Monitor}

Matrícula = {Sigla, Numero, Aluno, Ano, Nota}

- Selecionar nome e nusp dos alunos que estão matriculados em alguma disciplina e que são monitores de alguma disciplina

```
select nome, nusp from aluno A where
```

```
    EXISTS (select NULL from matricula M
            where M.aluno = A.nusp)
```

and

```
    EXISTS (select NULL from disciplina D
            where D.monitor = A.nusp )
```

EXEMPLO:

Aluno = {Nome, Nusp, Idade, DataNasc}

Disciplina = {Sigla, Nome, NCred, Professor, Livro, Monitor}

Matrícula = {Sigla, Numero, Aluno, Ano, Nota}

- Selecionar n
disciplina e q

select n

EXISTS

and

EXISTS (**select** NULL from disciplina D
where D.monitor = **A.nusp**)

Consultas EXISTS funcionam levando dados de “dentro” para “fora” da consulta principal.

A cláusula EXISTS não retorna dados, mas sim um status booleano.

n alguma

M

```
select nome, nusp from aluno A where  
exists (select NULL from matricula M  
where M.aluno = A.nusp)
```

and

```
exists (select NULL from disciplina D  
where D.Monitor = A.nusp)
```



Também pode ser resolvida com junção

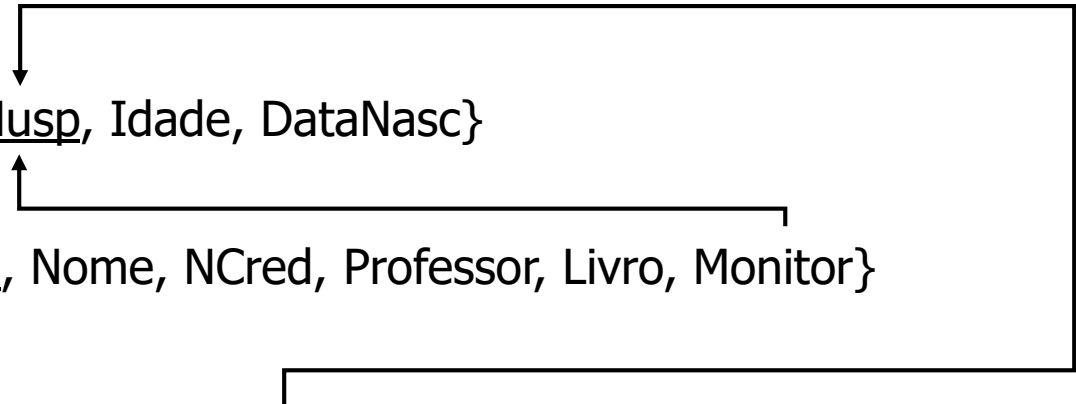
```
select distinct A.nome, A.nusp  
from aluno A, matricula M, disciplina D  
where M.aluno = A.nusp and D.monitor = A.nusp
```

EXEMPLO:

Aluno = {Nome, Nusp, Idade, DataNasc}

Disciplina = {Sigla, Nome, NCred, Professor, Livro, Monitor}

Matrícula = {Sigla, Numero, Aluno, Ano, Nota}



- Selecionar nome e nusp dos alunos que não estão matriculados em nenhuma disciplina

```
select nome, nusp from aluno A where  
NOT EXISTS
```

```
(select NULL from matricula M  
where M.aluno = A.nusp)
```

EXEMPLO:

```
select nome, nusp
from aluno A where
    NOT EXISTS
        (select NULL from matricula M
         where M.aluno = A.nusp)
```

```
select nome, nusp
from aluno A LEFT JOIN matricula M
    on M.aluno = A.nusp
where M.disciplina IS NULL
```

Outros recursos SQL do Oracle



SELECT com CASE

Usado para gerar novos dados categóricos

```
SELECT TIPO, CATEGORIA,  
       CASE TIPO  
         WHEN 'G' THEN CATEGORIA || ' Graduando '  
         WHEN 'P' THEN CATEGORIA || ' PosGraduando '  
         ELSE 'Indefinido '  
       END  
FROM Tabela
```


RANK

Gera informação de ranking de acordo com a ordem de qualquer atributo

```
SELECT Nome, NUsp, AnoIngresso,  
        RANK() OVER (ORDER BY AnoIngresso) MAIS_VELHO  
FROM Tabela
```

LISTAGEM ALEATÓRIA

```
SELECT *  
FROM (  
    SELECT *  
    FROM table  
    ORDER BY DBMS_RANDOM.RANDOM)  
WHERE rownum < 21;
```

Lógica de três valores

NULL – Lógica de três valores

- Clausulas WHERE verificam o status de cada tupla com relacao a um predicado – os possiveis status são:
 - true
 - false
 - null: informação indisponível, não se pode afirmar nada
- Cláusulas WHERE retornam apenas tuplas cujo status definido pelo predicado seja true – recusam-se tuplas com status false e NULL (unknown).

NULL – Lógica de três valores

- **IS [NOT] NULL:** retorna true se um dado valor tiver estado NULL;

Na tupla\No predicado	= NULL	= value	IS NULL
valor	NULL ("false")	false/true	false
NULL	NULL ("false")	NULL ("false")	true

Exemplo:

```
SELECT IDADE
FROM ALUNO
WHERE IDADE = NULL
```

NULL ("false") para todas
as tuplas

```
SELECT IDADE
FROM ALUNO
WHERE IDADE = value
```

Retorna NULL ("false"),
para as tuplas com valor
NULL, true ou false para
as demais

```
SELECT IDADE
FROM ALUNO
WHERE IDADE IS NULL
```

Se na tupla houver NULL
retorna TRUE, para as demais
retorna FALSE

* "false": valor NULL que não é retornado pelo SELECT, não se trata de um false de fato

Onde consultar ...

R. Elmasri, S. Navathe: *Fundamentals of Database Systems* – 4th Edition

- Capítulo 8

A. Silberschatz, H. F. Korth, s. Sudarshan: *Sistema de Banco de Dados*

- Capítulo 4

Manuais em *list of books* no site da Oracle

- *SQL Reference*

PRÁTICA 3