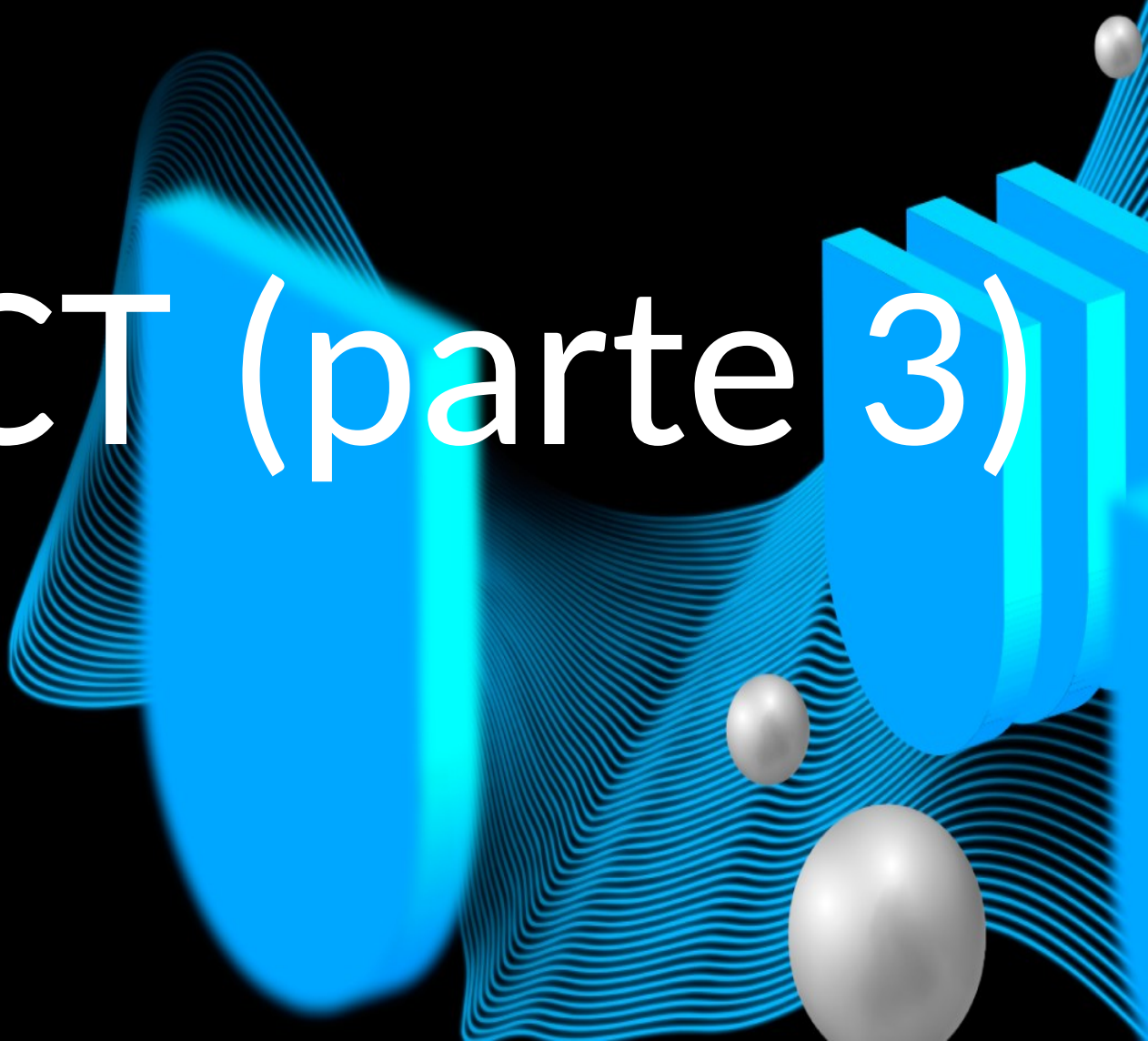

UNITAU
Universidade de Taubaté

SELECT (parte 3)



CREATE TABLE

É possível criar uma tabela com o resultado de um **SELECT**.

Exemplo:

```
CREATE TEMPORARY TABLE resumo AS ( SELECT fabricante.nome AS fabricante, revenda.estado, revenda.cidade, venda.valor FROM venda, revenda, automovel, fabricante WHERE revenda.codigo = venda.revenda AND automovel.codigo = venda.automovel AND fabricante.codigo = automovel.fabricante );
```

CREATE TABLE

```
SELECT * FROM resumo;
 fabricante | estado |   cidade    |  valor
------------+--------+-------------+----------
 Ford       | SP     | Sao Paulo   | 17500.00
 Volkswagen | SP     | Taubate     | 28000.00
 Fiat       | RJ     | Macae       | 28000.00
 Volkswagen | RJ     | Macae       | 42000.00
 Chevrolet  | MG     | Betim       | 11500.00
 Volkswagen | MG     | Contagem    | 29500.00
 Ford       | SP     | Taubate     | 22100.00
 Chevrolet  | SP     | Sao Paulo   | 15500.00
 Fiat       | MG     | Betim       | 24500.00
 Volkswagen | SP     | Cacapava    | 39500.00
 Volkswagen | SP     | Sao Paulo   | 31000.00
 Chevrolet  | SP     | Sao Paulo   | 17500.00
 Ford       | RJ     | Resende     | 21500.00
```

CREATE TABLE

É possível criar uma tabela com a mesma estrutura de uma tabela já existene.

Exemplo:

```
CREATE TEMPORARY TABLE resumo_sp ( LIKE resumo );
```

INSERT

É possível inserir o resultado de um SELECT diretamente em uma tabela.

Exemplo:

```
INSERT INTO resumo_sp ( SELECT fabricante.nome,
revenda.estado, revenda.cidade, venda.valor FROM venda,
revenda, automovel, fabricante WHERE revenda.codigo =
venda.revenda AND automovel.codigo = venda.automovel AND
fabricante.codigo = automovel.fabricante AND
revenda.estado='SP');

SELECT * FROM resumo_sp;
 fabricante | estado |   cidade    |  valor
------------+--------+-------------+----------
 Ford       | SP     | Sao Paulo   | 17500.00
 Volkswagen | SP     | Taubate     | 28000.00
 Ford       | SP     | Taubate     | 22100.00
 Chevrolet  | SP     | Sao Paulo   | 15500.00
 Volkswagen | SP     | Cacapava    | 39500.00
 Volkswagen | SP     | Sao Paulo   | 31000.00
 Chevrolet  | SP     | Sao Paulo   | 17500.00
```

UNION

UNION une o resultado de duas consultas.

consulta1 UNION [ALL] consulta2

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis.

Exemplo:

```
SELECT produto FROM entrada WHERE data >='2010/03/05' AND
data <= '2010/03/07' UNION SELECT produto FROM saida WHERE
data >='2010/03/05' AND data <= '2010/03/07';
```

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <= '2010/03/07'
UNION SELECT saida.produto,produto.descricao FROM
saida,produto WHERE produto.codigo=saida.produto AND
saida.data >='2010/03/05' AND saida.data <= '2010/03/07';
```

UNION

```
SELECT produto FROM entrada WHERE data >='2010/03/05' AND
data <= '2010/03/07' UNION SELECT produto FROM saida WHERE
data >='2010/03/05' AND data <= '2010/03/07';
```

```
 produto
---------
 04
 05
 01
 06
```

UNION

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <= '2010/03/07'
UNION SELECT saida.produto,produto.descricao FROM
saida,produto WHERE produto.codigo=saida.produto AND
saida.data >='2010/03/05' AND saida.data <= '2010/03/07';
```

```
 produto | descricao
---------+-----------
 06      | Caderno
 01      | Caneta
 04      | Borracha
 05      | Regua
```

UNION

As linhas duplicadas serão removidas do resultado, a menos que se utilize UNION ALL.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM entrada, produto
WHERE quantidade=8 AND produto.codigo = entrada.produto UNION ALL
SELECT saida.produto, produto.descricao FROM saida,produto WHERE
quantidade=8 AND produto.codigo=saida.produto;
```

```
 produto | descricao
---------+------------
 02      | Lapis
 02      | Lapis
 04      | Borracha
 04      | Borracha
```

```
SELECT entrada.produto,produto.descricao FROM entrada, produto
WHERE quantidade=8 AND produto.codigo = entrada.produto UNION
SELECT saida.produto, produto.descricao FROM saida,produto WHERE
quantidade=8 AND produto.codigo=saida.produto;
```

```
 produto | descricao
---------+------------
 04      | Borracha
 02      | Lapis
```

UNION

As combinações de consultas podem ser encadeadas.

Exemplo:

```
SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' UNION
SELECT produto FROM saida WHERE data='2010/03/12';
```

Essa consulta é equivalente a:

```
( SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' ) UNION
SELECT produto FROM saida WHERE data='2010/03/12';
```

UNION

```
SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' UNION
SELECT produto FROM saida WHERE data='2010/03/12';
 produto
---------
 02
 01
 06
 04
```

UNION

```
( SELECT produto FROM saida WHERE data='2010/03/03' UNION
SELECT produto FROM saida WHERE data='2010/03/05' ) UNION
SELECT produto FROM saida WHERE data='2010/03/12';
 produto
---------
 06
 01
 04
 02
```

INTERSECT

INTERSECT retorna a interseção entre os resultado de duas consultas (linhas presentes nas duas consultas).

```
consulta1 INTERSECT [ ALL ] consulta2
```

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis. As linhas duplicadas serão removidas do resultado, a menos que se utilize **INTERSECT ALL**.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM
entrada,produto WHERE produto.codigo=entrada.produto AND
entrada.data >='2010/03/05' AND entrada.data <= '2010/03/07'
INTERSECT SELECT saida.produto,produto.descricao FROM
saida,produto WHERE produto.codigo=saida.produto AND
saida.data >='2010/03/05' AND saida.data <= '2010/03/07';
 produto | descricao
---------+-----------
 04      | Borracha
```

EXCEPT

EXCEPT retorna a diferença entre os resultado de duas consultas (linhas presentes na primeira consulta mas que não estão presentes na segunda consulta).

consulta1 EXCEPT [ALL] consulta2

As duas consultas devem retornar o mesmo número de colunas, e as colunas correspondentes devem ser de tipos compatíveis. As linhas duplicadas serão removidas do resultado, a menos que se utilize **EXCEPT ALL**.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM entrada,produto WHERE
produto.codigo=entrada.produto AND entrada.data >='2010/03/05' AND
entrada.data <= '2010/03/07' EXCEPT SELECT
saida.produto,produto.descricao FROM saida,produto WHERE
produto.codigo=saida.produto AND saida.data >='2010/03/05' AND
saida.data <= '2010/03/07';
```

```
 produto | descricao
---------+-----------
 05      | Regua
```

Funções de Agregação

As funções de agregação retornam um único valor como resultado de um conjunto de valores.

COUNT(expressão) – número de valores de entrada, para os quais a expressão não é nula, para se contar todos os elementos da entrada, é usado * como expressão.

Exemplo:

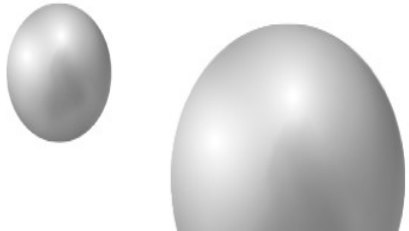
```
SELECT COUNT(*) FROM aluno;
 count
-------
    10
SELECT COUNT(curso) FROM aluno;
 count
-------
     9
```

Funções de Agregação

Pode ser usado DISTINCT para contar quantos valores distintos não nulos são retornados pela expressão.

Exemplo:

```
SELECT COUNT(DISTINCT curso) FROM aluno;
 count
-------
     2
```



Funções de Agregação

Pode ser usado CASE para agrupar o resultado.

Exemplo:

```
SELECT COUNT(CASE WHEN serie='1' THEN 1 ELSE NULL END) AS
primeiro, COUNT(CASE WHEN serie='2' THEN 1 ELSE NULL END) AS
segundo FROM aluno;

 primeiro | segundo
----------+---------
        8 |       1
```

Funções de Agregação

A cláusula **FILTER (WHERE condicao)** também pode ser usada para agrupar o resultado. Apesar de ser parte do padrão da linguagem SQL, essa cláusula não é amplamente suportada, sendo mais comum o uso de **CASE** para agrupar o resultado.

Exemplo:

```
SELECT COUNT(*) FILTER (WHERE serie='1') AS primeiro,
COUNT(*) FILTER (WHERE serie='2') AS segundo FROM aluno;
 primeiro | segundo
----------+---------
        8 |       1
```

Funções de Agregação

SUM(expressão) – soma o valor da expressão para os valores de entrada.

Exemplo:

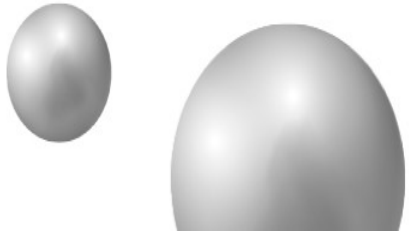
```
SELECT SUM(quantidade) FROM entrada WHERE produto='01';
 sum
-----
  35
```

Funções de Agregação

MAX(expressão) – retorna o valor máximo da expressão para os valores de entrada.

Exemplo:

```
SELECT MAX(quantidade) FROM entrada WHERE produto='01';
 max
-----
  15
```



Funções de Agregação

MIN(expressão) – retorna o valor mínimo da expressão para os valores de entrada.

Exemplo:

```
SELECT MIN(quantidade) FROM entrada WHERE produto='01';
 min
-----
   4
```

Funções de Agregação

AVG(expressão) - retorna a média aritmética da expressão para os valores de entrada.

Exemplo:

```
SELECT AVG(quantidade) FROM entrada WHERE produto='01';
        avg
--------------------
 7.000000000000000
```

Funções de Agregação

STDDEV(expressão)/STDDEV_SAMP(expressão) - retorna o desvio padrão da amostra da expressão para os valores de entrada.

Exemplo:

```
SELECT STDDEV(quantidade) FROM entrada WHERE produto='01';
       stddev
--------------------
 4.527692569068708
SELECT STDDEV_SAMP(quantidade) FROM entrada WHERE
produto='01';
    stddev_samp
--------------------
 4.527692569068708
```

Funções de Agregação

STDDEV_POP(expressão) - retorna o desvio padrão da população da expressão para os valores de entrada.

Exemplo:

```
SELECT STDDEV_POP(quantidade) FROM entrada WHERE
produto='01';
      stddev_pop
--------------------
 4.0496913462633174
```

Funções de Agregação

VARIANCE(expressão)/VAR_SAMP(expressão) - retorna a variância da amostra da expressão para os valores de entrada.

Exemplo:

```
SELECT VARIANCE(quantidade) FROM entrada WHERE produto='01';
      variance
----------------------
 20.500000000000000
SELECT VAR_SAMP(quantidade) FROM entrada WHERE produto='01';
      var_samp
----------------------
 20.500000000000000
```

Funções de Agregação

VAR_POP(expressão) - retorna a variância da população da expressão para os valores de entrada.

Exemplo:

```
SELECT VAR_POP(quantidade) FROM entrada WHERE produto='01';
        var_pop
----------------------
 16.4000000000000000
```

GROUP BY

A cláusula **GROUP BY** agrupa em uma única linha, todas as linhas com o mesmo valor para uma expressão.

Exemplo:

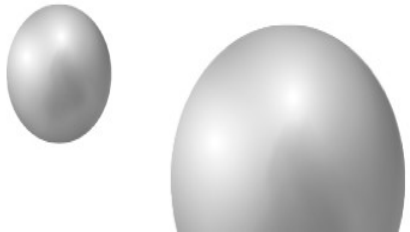
```
SELECT curso FROM aluno GROUP BY curso;
 curso
-------

 0001
 0002
```

GROUP BY

Usualmente, a cláusula **GROUP BY** é utilizada com as funções de agregação. Quando se utiliza a cláusula **GROUP BY**, não é valido a lista de expressões da saída fazer referências a colunas que não são usadas na expressão de agrupamento, exceto em funções de agregação ou se for um campo com dependência funcional dos campos da expressão de agrupamento.

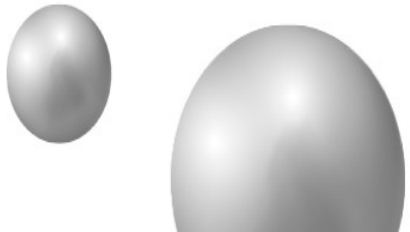
É importante saber que a cláusula **GROUP BY** é aplicada após a seleção das linhas pela cláusula **WHERE**, portanto a expressão de seleção da cláusula **WHERE** não pode fazer referências as expressões de agregação.



GROUP BY

Exemplo:

```
SELECT curso, COUNT(*) FROM aluno GROUP BY curso;
 curso | count
-------+-------
       |     1
 0001  |     6
 0002  |     3
```



GROUP BY

Campos com dependência funcional dos campos da expressão de agrupamento são campos que pertencem a tabela onde os campos da expressão de agrupamento, ou parte deles, sejam a chave primária da tabela.

Exemplo:

```
SELECT curso.nome, COUNT(*) FROM curso,aluno WHERE
aluno.curso=curso.codigo GROUP BY curso.codigo;
            nome              | count
------------------------------+-------
 Engenharia da Computacao     |     3
 Tecnologia da Informacao     |     6
```

GROUP BY

Pode ser usado CASE como expressão de agrupamento.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente' WHEN
funcao='3' THEN 'intermediario' WHEN funcao=2 OR funcao=5
THEN 'iniciante' END AS experiencia, count(*) FROM
funcionario GROUP BY CASE WHEN funcao=1 OR funcao=4 THEN
'experiente' WHEN funcao='3' THEN 'intermediario' WHEN
funcao=2 OR funcao=5 THEN 'iniciante' END;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

GROUP BY

A expressão de agrupamento pode fazer referências a colunas da lista de saída pela posição da coluna.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente' WHEN
funcao='3' THEN 'intermediario' WHEN funcao=2 OR funcao=5
THEN 'iniciante' END AS experiencia, count(*) FROM
funcionario GROUP BY 1;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

GROUP BY

A expressão de agrupamento pode fazer referências a colunas da lista de saída pelo alias da coluna.

Exemplo:

```
SELECT CASE WHEN funcao=1 OR funcao=4 THEN 'experiente' WHEN
funcao='3' THEN 'intermediario' WHEN funcao=2 OR funcao=5
THEN 'iniciante' END AS experiencia, count(*) FROM
funcionario GROUP BY experiencia;
```

```
  experiencia  | count
---------------+-------
 intermediario |     3
 experiente    |     8
 iniciante     |     5
```

GROUPING SETS

O conceito de **GROUPING SETS** permite agrupar o resultado da consulta por várias expressões de agrupamento e retornar os resultados dos agrupamentos por cada expressão em uma única consulta como em um **UNION**.

Retornar o resultado de diversas agregações em uma única consulta facilita a utilização de ferramentas de visualização para análise de dados.

GROUPING SETS

Exemplo:

```
SELECT fabricante,estado,cidade,SUM(valor) FROM resumo GROUP
BY GROUPING SETS ((fabricante),(estado,cidade),()) ORDER BY
fabricante,estado,cidade;
```

fabricante	estado	cidade	sum
Chevrolet			44500.00
Fiat			52500.00
Ford			61100.00
Volkswagen			170000.00
	MG	Betim	36000.00
	MG	Contagem	29500.00
	RJ	Macae	70000.00
	RJ	Resende	21500.00
	SP	Cacapava	39500.00
	SP	Sao Paulo	81500.00
	SP	Taubate	50100.00
			328100.00

GROUPING SETS

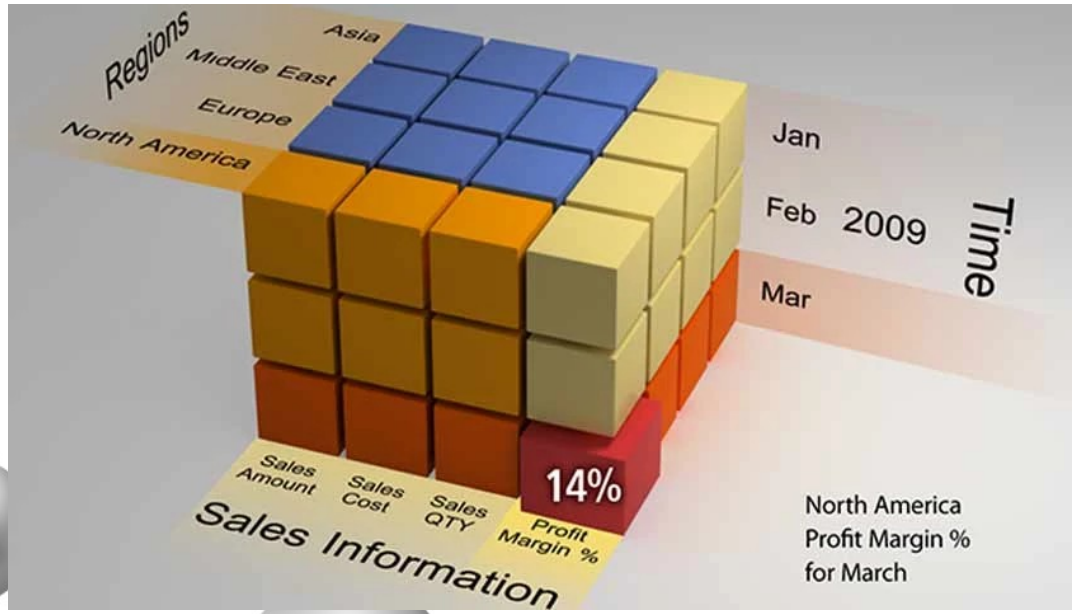
Exemplo:

```
SELECT null AS fabricante, null AS estado, null AS
cidade,SUM(valor) FROM resumo UNION SELECT fabricante, null,
null,SUM(valor) FROM resumo GROUP BY fabricante UNION SELECT
null,estado,cidade,SUM(valor) FROM resumo GROUP BY
estado,cidade ORDER BY fabricante,estado,cidade;
```

fabricante	estado	cidade	sum
Chevrolet			44500.00
Fiat			52500.00
Ford			61100.00
Volkswagen			170000.00
	MG	Betim	36000.00
	MG	Contagem	29500.00
	RJ	Macae	70000.00
	RJ	Resende	21500.00
	SP	Cacapava	39500.00
	SP	Sao Paulo	81500.00
	SP	Taubate	50100.00
			328100.00

Cubos OLAP

As aplicações de OLAP se caracterizam por apresentar visões multidimensionais das informações. O foco e a apresentação das informações em formatos de cubos, onde as dimensões dos cubos representam os componentes do negócio da organização como Tempo, Produtos, Locais, etc.



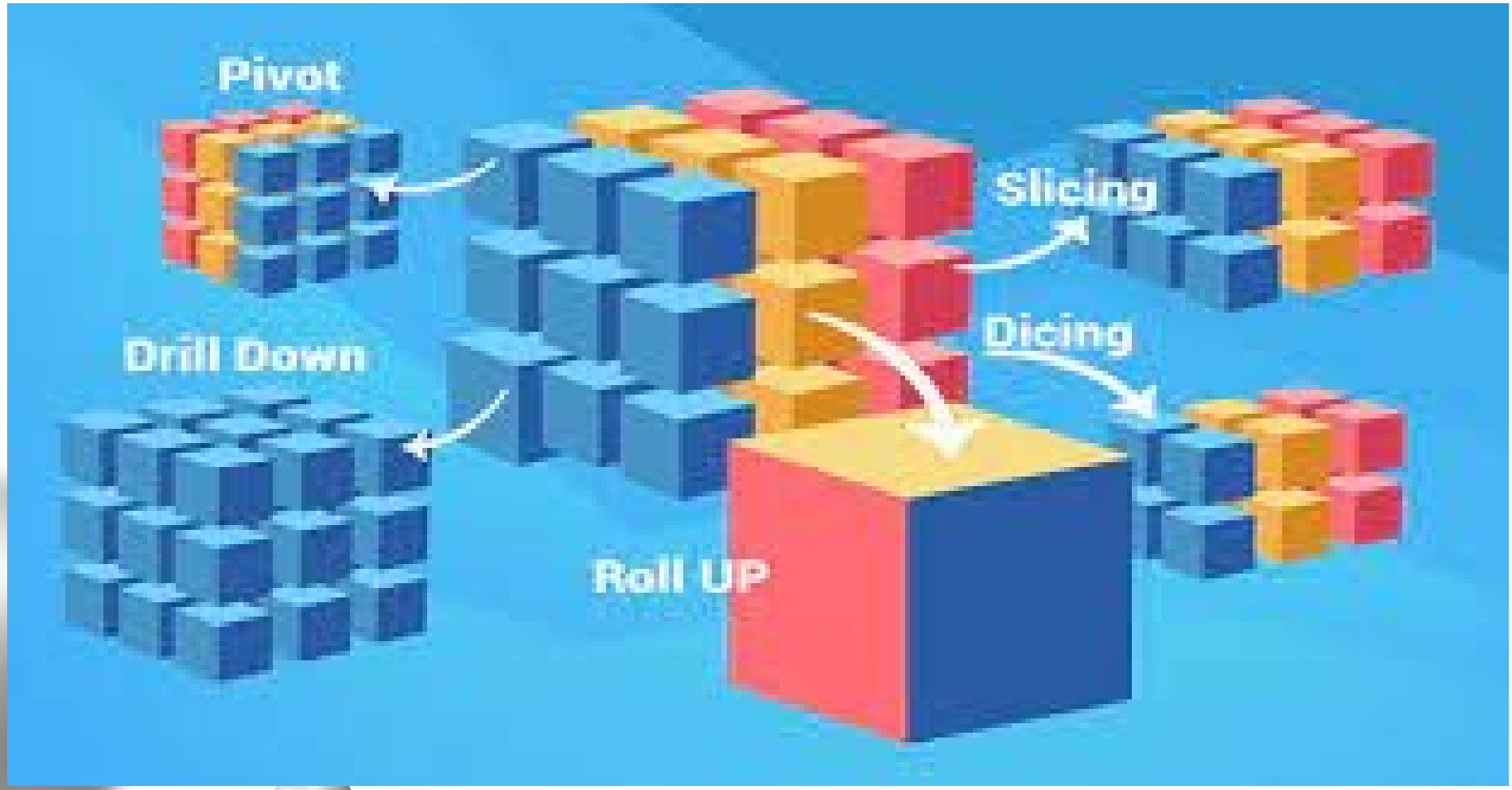
https://ayltoninacio.com.br/blog/criando-e-alimentando-um-cubo-olap-fisico-no-mysql

Cubos OLAP

As ferramentas OLAP apresentam as informações no formato de cubos com agregações e sumarização das informações e permitem algumas operações sobre as análises apresentadas:

- **Roll-Up** – apresenta os dados agrupando os valores de uma dimensão.
- **Drill-Down** – apresenta os dados com maior detalhando de uma dimensão.
- **Slice** (fatiar) - apresenta os dados separados por grupos em uma dimensão
- **Dice** (cortar) - apresentar os valores selecionando algum intervalo de valores
- **Pivoting** (rotação) - permite inverter as dimensões da apresentação

Cubos OLAP



https://www.devmedia.com.br/um-estudo-sobre-as-ferramentas-olap/6691

GROUPING SETS

As cláusulas ROLLUP e CUBE geram os agrupamentos necessários para as operações de cubos OLAP.

ROLLUP

ROLLUP (e1, e2, e3, ...)

É utilizado para gerar os agrupamentos para operações de Roll-Up e Drill-Down de cubos OLAP.

É equivalente a:

```
GROUPING SETS (
    ( e1, e2, e3, ... ),
    ...
    ( e1, e2 ),
    ( e1 ),
    ( )
)
```

ROLLUP

Exemplo:

```
SELECT fabricante,estado,cidade,SUM(valor) FROM resumo GROUP
BY ROLLUP (fabricante,estado,cidade);
```

fabricante	estado	cidade	sum
			328100.00
Ford	RJ	Resende	21500.00
Volkswagen	RJ	Macae	42000.00
Ford	SP	Sao Paulo	17500.00
Fiat	MG	Betim	24500.00
Ford	SP	Taubate	22100.00
Volkswagen	SP	Cacapava	39500.00
Chevrolet	MG	Betim	11500.00
Volkswagen	SP	Sao Paulo	31000.00
Fiat	RJ	Macae	28000.00
Volkswagen	SP	Taubate	28000.00
Chevrolet	SP	Sao Paulo	33000.00
Volkswagen	MG	Contagem	29500.00
Fiat	MG		24500.00
Chevrolet	MG		11500.00

ROLLUP

Volkswagen	SP		98500.00
Volkswagen	MG		29500.00
Ford	RJ		21500.00
Chevrolet	SP		33000.00
Fiat	RJ		28000.00
Volkswagen	RJ		42000.00
Ford	SP		39600.00
Ford			61100.00
Chevrolet			44500.00
Fiat			52500.00
Volkswagen			170000.00

CUBE

CUBE (e1, e2, ...)

É utilizado para gerar os agrupamentos necessários para as operações de cubos OLAP.

CUBE (a, b, c)

É equivalente a:

```
GROUPING SETS (
      ( a, b, c ),
      ( a, b    ),
      ( a,    c ),
      ( a       ),
      (    b, c ),
      (    b    ),
      (       c ),
      (         )
)
```

CUBE

Exemplo:

```
SELECT fabricante,estado,cidade,SUM(valor) FROM resumo GROUP
BY CUBE (fabricante,estado,cidade);
```

fabricante	estado	cidade	sum
			328100.00
Ford	RJ	Resende	21500.00
Volkswagen	RJ	Macae	42000.00
Ford	SP	Sao Paulo	17500.00
Fiat	MG	Betim	24500.00
Ford	SP	Taubate	22100.00
Volkswagen	SP	Cacapava	39500.00
Chevrolet	MG	Betim	11500.00
Volkswagen	SP	Sao Paulo	31000.00
Fiat	RJ	Macae	28000.00
Volkswagen	SP	Taubate	28000.00
Chevrolet	SP	Sao Paulo	33000.00
Volkswagen	MG	Contagem	29500.00
Fiat	MG		24500.00
Chevrolet	MG		11500.00
Volkswagen	SP		98500.00

CUBE

Volkswagen	MG		29500.00
Ford	RJ		21500.00
Chevrolet	SP		33000.00
Fiat	RJ		28000.00
Volkswagen	RJ		42000.00
Ford	SP		39600.00
Ford			61100.00
Chevrolet			44500.00
Fiat			52500.00
Volkswagen			170000.00
	MG	Betim	36000.00
	SP	Sao Paulo	81500.00
	SP	Cacapava	39500.00
	SP	Taubate	50100.00
	RJ	Macae	70000.00
	MG	Contagem	29500.00
	RJ	Resende	21500.00
	MG		65500.00
	SP		171100.00
	RJ		91500.00
Chevrolet		Sao Paulo	33000.00
Ford		Taubate	22100.00

CUBE

Volkswagen		Sao Paulo	31000.00
Fiat		Betim	24500.00
Volkswagen		Contagem	29500.00
Volkswagen		Macae	42000.00
Fiat		Macae	28000.00
Volkswagen		Taubate	28000.00
Volkswagen		Cacapava	39500.00
Ford		Resende	21500.00
Chevrolet		Betim	11500.00
Ford		Sao Paulo	17500.00
		Sao Paulo	81500.00
		Resende	21500.00
		Contagem	29500.00
		Taubate	50100.00
		Cacapava	39500.00
		Betim	36000.00
		Macae	70000.00

GROUPING SETS

A função **GROUPING(expressoes_de_agrupamento)** retorna uma máscara de bits indicando quais expressões GROUP BY não estão incluídas no conjunto de agrupamento atual. Os bits são atribuídos com o argumento mais à direita correspondente ao bit menos significativo. Cada bit é 0 se a expressão correspondente estiver incluída nos critérios de agrupamento do conjunto de agrupamento que gera a linha de resultado atual e 1 se não estiver incluída.

GROUPING SETS

Exemplo:

```
SELECT fabricante, estado, cidade, SUM(valor),
GROUPING(fabricante, estado, cidade) FROM resumo GROUP BY
CUBE (fabricante, estado, cidade);
```

fabricante	estado	cidade	sum	grouping
			328100.00	7
Ford	RJ	Resende	21500.00	0
Volkswagen	RJ	Macae	42000.00	0
Ford	SP	Sao Paulo	17500.00	0
Fiat	MG	Betim	24500.00	0
Ford	SP	Taubate	22100.00	0
Volkswagen	SP	Cacapava	39500.00	0
Chevrolet	MG	Betim	11500.00	0
Volkswagen	SP	Sao Paulo	31000.00	0
Fiat	RJ	Macae	28000.00	0
Volkswagen	SP	Taubate	28000.00	0
Chevrolet	SP	Sao Paulo	33000.00	0
Volkswagen	MG	Contagem	29500.00	0
Fiat	MG		24500.00	1
Chevrolet	MG		11500.00	1

GROUPING SETS

Volkswagen	SP		98500.00	1
Volkswagen	MG		29500.00	1
Ford	RJ		21500.00	1
Chevrolet	SP		33000.00	1
Fiat	RJ		28000.00	1
Volkswagen	RJ		42000.00	1
Ford	SP		39600.00	1
Ford			61100.00	3
Chevrolet			44500.00	3
Fiat			52500.00	3
Volkswagen			170000.00	3
	MG	Betim	36000.00	4
	SP	Sao Paulo	81500.00	4
	SP	Cacapava	39500.00	4
	SP	Taubate	50100.00	4
	RJ	Macae	70000.00	4
	MG	Contagem	29500.00	4
	RJ	Resende	21500.00	4
	MG		65500.00	5
	SP		171100.00	5
	RJ		91500.00	5
Chevrolet		Sao Paulo	33000.00	2

GROUPING SETS

Ford		Taubate	22100.00	2
Volkswagen		Sao Paulo	31000.00	2
Fiat		Betim	24500.00	2
Volkswagen		Contagem	29500.00	2
Volkswagen		Macae	42000.00	2
Fiat		Macae	28000.00	2
Volkswagen		Taubate	28000.00	2
Volkswagen		Cacapava	39500.00	2
Ford		Resende	21500.00	2
Chevrolet		Betim	11500.00	2
Ford		Sao Paulo	17500.00	2
		Sao Paulo	81500.00	6
		Resende	21500.00	6
		Contagem	29500.00	6
		Taubate	50100.00	6
		Cacapava	39500.00	6
		Betim	36000.00	6
		Macae	70000.00	6

GROUPING SETS

Vários **GROUPING SETS** podem ser utilizados na mesma consulta para gerar os agrupamentos correspondentes ao produto cruzado dos **GROUPING SETS**.

GROUP BY a, CUBE (b, c), GROUPING SETS ((d), (e))

É equivalente a:

```
GROUP BY GROUPING SETS (
        (a, b, c, d), (a, b, c, e),
        (a, b, d),    (a, b, e),
        (a, c, d),    (a, c, e),
        (a, d),       (a, e)
)
```

HAVING

A cláusula **HAVING** é aplicada após a cláusula **GROUP BY** para filtrar as linhas do resultado utilizando as expressões de agregação e as expressões de agrupamento.

Exemplo:

```
SELECT curso, COUNT(*) FROM aluno GROUP BY curso HAVING
count(*)<4;
 curso | count
-------+-------
       |     1
 0002  |     3
SELECT curso, COUNT(*) FROM aluno GROUP BY curso HAVING
count(*)<4 AND curso IS NOT NULL;
 curso | count
-------+-------
 0002  |     3
```

ORDER BY

A cláusula ORDER BY permite determinar a ordem das linhas do resultado.

Sem a determinação de uma ordem específica, não é possível esperar que as linhas venham em alguma ordem, pois não é possível determinar a ordem que as linhas serão selecionadas pelo algoritmo do SGBD.

ORDER BY

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY nome;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
         4 | Debora           | 0001
        10 | Fernanda         |
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
```

ORDER BY

Quando utilizada mais de uma expressão para a ordenação, a primeira expressão avaliada será a expressão mais a esquerda e assim por diante.

Por default, o valor nulo é classificado em uma posição mais alta do que qualquer outro valor.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso, nome;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         4 | Debora           | 0001
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
        10 | Fernanda         |
```

ORDER BY

Para alterar a posição de classificação do valor nulo pode ser utilizado **NULLS FIRST**. A opção **NULLS LAST** é equivalente ao comportamento default.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
NULLS FIRST, nome;
```

matricula	nome	curso
10	Fernanda	
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001
8	Andrea	0002
9	Carla	0002
7	Claudio	0002

ORDER BY

Para ordenação decrescente, utiliza-se **DESC** em frente a expressão que deve ser avaliada em ordem decrescente. Pode se utilizar **ASC** para especificar a ordem ascendente, porém não é necessário por ser este o padrão.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome;

SELECT matricula, nome, curso FROM aluno ORDER BY curso,
nome DESC;

SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome DESC;
```

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome;
```

matricula	nome	curso
10	Fernanda	
8	Andrea	0002
9	Carla	0002
7	Claudio	0002
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso,
nome DESC;
```

matricula	nome	curso
3	Marcelo	0001
2	Luis Claudio	0001
5	Fernanda	0001
4	Debora	0001
1	Ana Lucia	0001
6	Alvaro	0001
7	Claudio	0002
9	Carla	0002
8	Andrea	0002
10	Fernanda	

ORDER BY

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso
DESC, nome DESC;
```

matricula	nome	curso
10	Fernanda	
7	Claudio	0002
9	Carla	0002
8	Andrea	0002
3	Marcelo	0001
2	Luis Claudio	0001
5	Fernanda	0001
4	Debora	0001
1	Ana Lucia	0001
6	Alvaro	0001

ORDER BY

Para as colunas com a expressão **DESC**, o padrão passa a ser **NULLS FIRST**. Se for necessário que os valores nulos passem para o final da ordenação pode ser utilizado **NULLS LAST**.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY curso DESC
NULLS LAST, nome;
```

matricula	nome	curso
8	Andrea	0002
9	Carla	0002
7	Claudio	0002
6	Alvaro	0001
1	Ana Lucia	0001
4	Debora	0001
5	Fernanda	0001
2	Luis Claudio	0001
3	Marcelo	0001
10	Fernanda	

ORDER BY

A expressão de ordenação pode fazer referências as colunas da saída pela posição.

Exemplo:

```
SELECT matricula, nome, curso FROM aluno ORDER BY 3, 2;
 matricula |      nome        | curso
-----------+------------------+-------
         6 | Alvaro           | 0001
         1 | Ana Lucia        | 0001
         4 | Debora           | 0001
         5 | Fernanda         | 0001
         2 | Luis Claudio     | 0001
         3 | Marcelo          | 0001
         8 | Andrea           | 0002
         9 | Carla            | 0002
         7 | Claudio          | 0002
        10 | Fernanda         |
```

ORDER BY

Quando utilizado com combinação de consultas, a cláusula **ORDER BY** deve ser colocada após a última consulta e será aplicada a todo o resultado após a combinação do resultado de cada consulta.

Exemplo:

```
SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto UNION SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto ORDER BY 2;
 produto | descricao
---------+-----------
 04      | Borracha
 06      | Caderno
 01      | Caneta
 02      | Lapis
```

ORDER BY

Para aplicar a cláusula **ORDER BY** apenas a uma das consultas da combinação, essa consulta deve ser colocada entre parênteses.

Exemplo:

```
( SELECT entrada.produto,produto.descricao FROM entrada,
produto WHERE data>='2010/03/15' AND produto.codigo =
entrada.produto ORDER BY 1 ) UNION SELECT saida.produto,
produto.descricao FROM saida,produto WHERE data >=
'2010/03/15' AND produto.codigo=saida.produto;
 produto | descricao
---------+-----------
 02      | Lapis
 01      | Caneta
 04      | Borracha
 06      | Caderno
```

Exercícios

01) Obter uma lista da movimentação dos produtos com a data, código do produto, descrição do produto, tipo do movimento ('E' para entrada e 'S' para saída) e quantidade, ordenado por data e código do produto

02) Listar o total de créditos e total de débitos da conta 01, cada total em uma linha

03) Listar a descrição da fase e nome dos participantes do projeto 01 ordenados por fase, incluir a supervisão

04) Listar o total de horas de supervisão dos projetos

05) Listar a quantidade de automóveis

06) Obter o número de entradas do produto '02'

07) Listar o número de programadores participantes do projeto 02

08) Listar a quantidade de automóveis produzidos no Brasil

Exercícios

09) Obter o número de entradas para cada produto

10) Listar o país e o total de veículos produzidos no pais

11) Obter a soma das entradas para cada produto

12) Listar o país e total de valor pago nos automóveis fabricados no país

13) Listar o nome do fabricante e o total de automóveis de cada fabricante

14) Listar a descrição da plataforma e o total de horas gasto com fases dessa plataforma

15) Listar a descrição do projeto, descrição da plataforma e o total de horas gasto com fases dessa plataforma para cada projeto

16) Listar o nome do fabricante e o total do valor de venda dos automóveis do fabricante

Exercícios

17) Obter a média das quantidades das saídas de cada produto, ordenado pelo código do produto

18) Obter a maior quantidade saída de cada produto, ordenada por produto

19) Obter os produtos e total de saídas para os produtos que tiveram total de saídas maior que 20, ordenado por código do produto

20) Listar o nome das revendas e total do valor de venda para as revendas com total de vendas acima de R$ 50.000,00

21) Listar a descrição dos produtos que tiveram mais de 3 entradas

22) Listar o total de horas de cada fase dos projetos (incluindo supervisão) ordenados pelo total de horas em ordem decrescente

Exercícios

23) Listar a descrição do projeto e o custo total do projeto

24) Obter o código do produto, data e quantidade(s) da(s) ultima(s) saída(s) de cada produto, considerar apenas a data da saida

25) Obter o código, descrição do produto, total de entradas e total de saídas do produtos, ordenado pela descrição dos produtos

26) Obter o código, descrição e o saldo da movimentação dos produtos, ordenados pelo código dos produtos

27) Obter o código, descrição e o saldo da movimentação dos produtos que tem saldo menor ou igual a 15, ordenados pela descrição dos produtos

28) Obter o código, descrição, data e saldo da movimentação do produto na data ordenado por produto e data

Exercícios

29) Obter o código, descrição e total dos salários das diretorias com total de salários maior que $12000,00

30) Obter o total dos maiores salários de cada diretoria

31) Obter para cada produto o código e saldo da movimentação na primeira quinzena e na segunda quinzena do mês 03/2010