

Função CUBE

Podemos utilizar várias funções para resolver o mesmo problema com scripts SQL. Porém devemos observar a performance da consulta, principalmente se estivermos trabalhando com uma grande massa de dados. Vamos avaliar a função CUBE.

Criação da tabela vendas com carga de dados.

```
DROP TABLE IF EXISTS arq."VENDAS";
```

```
CREATE TABLE arq."VENDAS" (  
    loja VARCHAR NOT NULL,  
    categoria VARCHAR NOT NULL,  
    produto VARCHAR NOT NULL,  
    quantidade INT NOT NULL,  
    PRIMARY KEY (loja, categoria)  
);
```

```
INSERT INTO arq."VENDAS" (loja, categoria, produto, quantidade)  
VALUES  
    ('Atacadao Interlagos', 'Cerveja', 'Heineken', 1000),  
    ('Atacadao Interlagos', 'Refrigerante', 'Coca Cola 2l', 2000),  
    ('Assai Jundia', 'Cerveja', 'Heineken', 5000),  
    ('Assai Jundia', 'Refrigerante', 'Coca Cola 2L', 3000);
```

Scripts utilizando as funções UNION ALL, GROUPING SETS E CUBE.

UNION ALL - Várias instruções SQL juntas em apenas uma instrução

```
SELECT loja, categoria, SUM(quantidade)  
FROM arq."VENDAS"  
GROUP BY loja, categoria  
UNION ALL  
SELECT loja, NULL, SUM(quantidade)  
FROM arq."VENDAS"  
GROUP BY loja  
UNION ALL  
SELECT NULL, categoria, SUM(quantidade)  
FROM arq."VENDAS"  
GROUP BY categoria  
UNION ALL  
SELECT NULL, NULL, SUM(quantidade)  
FROM arq."VENDAS";
```

Retorno:

| | loja character varying 🔒 | categoria character varying 🔒 | sum bigint |
|---|-----------------------------|----------------------------------|---------------|
| 1 | Atacadao Interlagos | Cerveja | 10 |
| 2 | Assai Jundia | Cerveja | 50 |
| 3 | Assai Jundia | Refrigerante | 30 |
| 4 | Atacadao Interlagos | Refrigerante | 20 |
| 5 | Atacadao Interlagos | [null] | 30 |
| 6 | Assai Jundia | [null] | 80 |
| 7 | [null] | Cerveja | 60 |
| 8 | [null] | Refrigerante | 50 |
| 9 | [null] | [null] | 110 |

GROUPING SETS - Simplifica o UNION ALL

```
SELECT loja, categoria, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY GROUPING SETS ( (loja, categoria), (loja), (categoria), () );
```

Retorno:

| | loja [PK] character varying ✎ | categoria [PK] character varying ✎ | sum bigint 🔒 |
|---|----------------------------------|---------------------------------------|-----------------|
| 1 | [null] | [null] | 11000 |
| 2 | Atacadao Interlagos | Cerveja | 1000 |
| 3 | Assai Jundia | Cerveja | 5000 |
| 4 | Assai Jundia | Refrigerante | 3000 |
| 5 | Atacadao Interlagos | Refrigerante | 2000 |
| 6 | Atacadao Interlagos | [null] | 3000 |
| 7 | Assai Jundia | [null] | 8000 |
| 8 | [null] | Cerveja | 6000 |
| 9 | [null] | Refrigerante | 5000 |

CUBE

- Cube simplifica o Grouping Sets
- A função CUBE permite realizar vários grouping sets.

```
SELECT loja, categoria, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY CUBE (loja, categoria);
```

Retorno:

| | loja [PK] character varying | categoria [PK] character varying | sum bigint |
|---|--------------------------------|-------------------------------------|---------------|
| 1 | [null] | [null] | 11000 |
| 2 | Atacadao Interlagos | Cerveja | 1000 |
| 3 | Assai Jundia | Cerveja | 5000 |
| 4 | Assai Jundia | Refrigerante | 3000 |
| 5 | Atacadao Interlagos | Refrigerante | 2000 |
| 6 | Atacadao Interlagos | [null] | 3000 |
| 7 | Assai Jundia | [null] | 8000 |
| 8 | [null] | Cerveja | 6000 |
| 9 | [null] | Refrigerante | 5000 |

Outros exemplos:

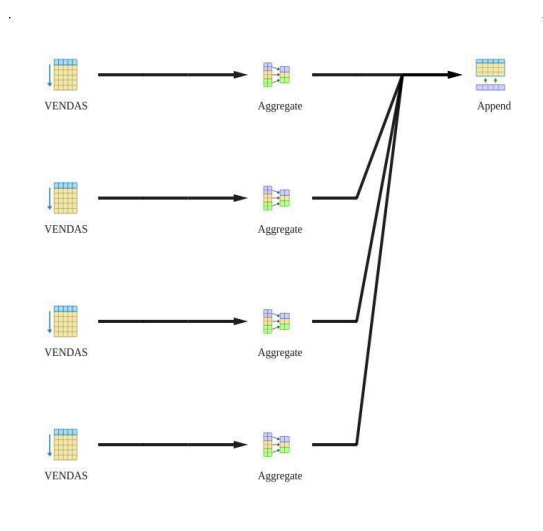
```
SELECT loja, categoria, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY CUBE (loja, categoria)
ORDER BY loja, categoria;
```

```
SELECT categoria, produto, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY CUBE (categoria, produto)
ORDER BY categoria, produto;
```

```
/*GROUP BY apenas com a coluna loja*/
SELECT loja, produto, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY loja, CUBE (produto)
ORDER BY loja, produto;
```

Uma das formas de observar a performance de cada consulta com suas respectivas funções, é através do plano de execução. No SGBD PostgreSQL existe um botão que gera automaticamente essa execução ou plano - visual.

UNION ALL



A tabela de vendas foi consultada 4 vezes, realizada a agregação e depois gerado o resultado. Imagine esse processo em uma grande massa de dados. Geralmente o UNION ALL não tem uma boa performance para geração de relatórios.

GROUPING SETS



Com a função GROUPING SETS, o plano de execução apresenta apenas 1 consulta na tabela. Geralmente é utilizada em relatórios devido alta performance, comparada com o UNION ALL.

CUBE



O plano de execução da função CUBE também apresenta uma única consulta na tabela, gerando o mesmo resultado.

Qual usar entre o GROUPING SETS E CUBE? As duas vão te dar uma performance melhor. O CUBE permite fazer vários GROUPING SETS com apenas uma instrução. Criamos uma query mais simples e lógica usando a função CUBE. As duas apresentam uma performance bem superior em comparação ao UNION ALL.

Teste as duas funções quando estiver diante de um problema onde a performance tem o maior peso. Teste e analise cada plano de execução gerado. Quantas consultas são realizadas na tabela, quantas operações são realizadas, experimente alternativas e faça as comparações.

Não existe perfeição. Busque o melhor resultado possível!

ROLLUP

Com a subcláusula CUBE é possível retornar todas as combinações possíveis de agrupamento. E se você não quiser todas as combinações possíveis de agrupamento? Se você quiser apenas uma hierarquia, totais e subtotais? Você não consegue isso com CUBE e nem com o GROUPING SETS. Mas você vai conseguir com ROLLUP.

ROLLUP é uma outra subcláusula amplamente utilizada em Data Warehouse devido às queries hierárquicas.

Na grande maioria de relatórios de DW o ROLLUP é utilizado.

Veja os scripts abaixo com CUBE e ROLLUP:

```
SELECT loja, categoria, SUM(quantidade)
FROM arq."VENDA"
GROUP BY CUBE (loja, categoria)
ORDER BY loja, categoria;
```

Resultado:

| | loja [PK] character varying | categoria [PK] character varying | sum bigint |
|---|--------------------------------|-------------------------------------|---------------|
| 1 | Assai Jundia | Cerveja | 5000 |
| 2 | Assai Jundia | Refrigerante | 3000 |
| 3 | Assai Jundia | [null] | 8000 |
| 4 | Atacadao Interlagos | Cerveja | 1000 |
| 5 | Atacadao Interlagos | Refrigerante | 2000 |
| 6 | Atacadao Interlagos | [null] | 3000 |
| 7 | [null] | Cerveja | 6000 |
| 8 | [null] | Refrigerante | 5000 |
| 9 | [null] | [null] | 11000 |

```
SELECT loja, categoria, SUM(quantidade)
FROM arq."VENDAS"
GROUP BY ROLLUP (loja, categoria)
ORDER BY loja, categoria;
```

| | loja [PK] character varying | categoria [PK] character varying | sum bigint |
|---|--------------------------------|-------------------------------------|---------------|
| 1 | Assai Jundia | Cerveja | 5000 |
| 2 | Assai Jundia | Refrigerante | 3000 |
| 3 | Assai Jundia | [null] | 8000 |
| 4 | Atacadao Interlagos | Cerveja | 1000 |
| 5 | Atacadao Interlagos | Refrigerante | 2000 |
| 6 | Atacadao Interlagos | [null] | 3000 |
| 7 | [null] | [null] | 11000 |

Trabalhando com COALESCE

Vamos criar uma nova tabela.

```
DROP TABLE IF EXISTS arq."PRODUTOS";
```

Cria tabela

```
CREATE TABLE arq."PRODUTOS" (  
    ID serial PRIMARY KEY,  
    nome VARCHAR (100) NOT NULL,  
    preco NUMERIC NOT NULL,  
    desconto NUMERIC
```

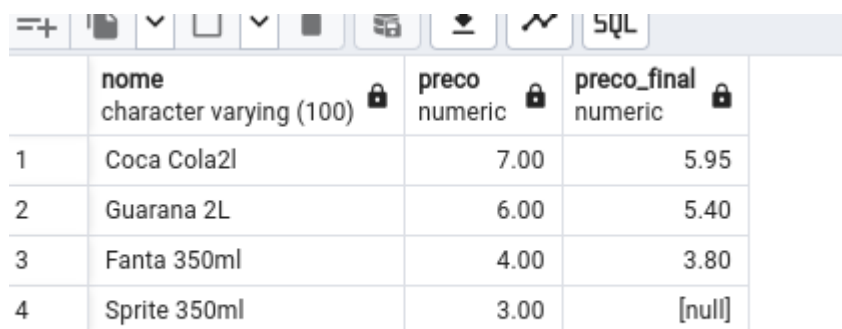
```
INSERT INTO arq."PRODUTOS" (nome, preco, desconto)  
VALUES
```

```
    ('Coca Cola' 2l',7.00 , 15),  
    ('Guarana 2L', 6.00, 10),  
    ('Fanta 350ml', 4.00, 5),  
    ('Sprite 350ml', 3.00, NULL);
```

O time de vendas solicita um relatório com o valor do preço final. No caso esse valor se dá através do preço menos a taxa de desconto.

Podemos aplicar o script abaixo:

```
SELECT nome, preco, round((preco - (preco * desconto/100)),2) AS preco_final  
FROM arq."PRODUTOS";
```



| | nome character varying (100) | preco numeric | preco_final numeric |
|---|---------------------------------|------------------|------------------------|
| 1 | Coca Cola2l | 7.00 | 5.95 |
| 2 | Guarana 2L | 6.00 | 5.40 |
| 3 | Fanta 350ml | 4.00 | 3.80 |
| 4 | Sprite 350ml | 3.00 | [null] |

Podemos ver que preco_final da linha 4 está com o valor null. Isso se dá pelo motivo da extração de dados da fonte obter valores Null.

Podemos resolver aplicando a instrução CASE:

```
SELECT nome, (preco - CASE WHEN desconto IS NULL THEN 0 ELSE round((preco *  
desconto/100),2) END)  
AS preco_final  
FROM arq."PRODUTOS";
```

| | nome character varying (100) | preco_final numeric |
|---|---------------------------------|------------------------|
| 1 | Coca Cola2l | 5.95 |
| 2 | Guarana 2L | 5.40 |
| 3 | Fanta 350ml | 3.80 |
| 4 | Sprite 350ml | 3.00 |

Também podemos utilizar a instrução COALESCE.

**SELECT nome, (preco - COALESCE(round((preco * desconto/100),2), 0)) AS
preco_final
FROM arq."PRODUTOS";**

| | nome character varying (100) | preco_final numeric |
|---|---------------------------------|------------------------|
| 1 | Coca Cola2l | 5.95 |
| 2 | Guarana 2L | 5.40 |
| 3 | Fanta 350ml | 3.80 |
| 4 | Sprite 350ml | 3.00 |

O resultado é o mesmo. Porém simplificamos com a instrução COALESCE.

Obrigado!