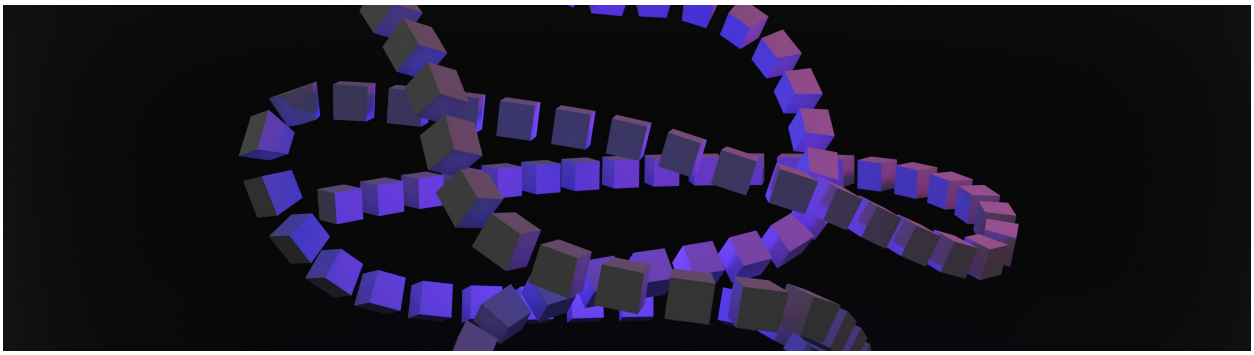


SQL Notebook

Objetivos

Com o objetivo de criar e disponibilizar um arquivo com explicações e exemplos da linguagem SQL explicando os formatos de busca, junção e mesclagem de tabelas, assim como funções introdutórias e avançadas é fornecer um recurso útil e acessível para aqueles que desejam aprender e aprimorar suas habilidades em SQL. Com um entendimento sólido da linguagem e suas funções, os usuários podem criar e gerenciar bancos de dados de maneira mais eficiente, realizar análises mais avançadas e desenvolver aplicativos mais sofisticados. Além disso, o arquivo pode ser usado como um recurso de referência para consulta rápida de sintaxe e funções específicas do SQL.

Nesse notebook utilizaremos nos baseamos nas especificidades do PostgreSQL ou Postgres, é um banco de dados relacional de código aberto que suporta muitos recursos avançados, como transações, views, gatilhos e procedimentos armazenados. Além disso, o Postgres possui recursos específicos, como herança de tabelas, tipos de dados personalizados e suporte para JSON. A sintaxe e as funções do SQL podem variar de acordo com o banco de dados utilizado, por isso é importante entender as especificidades de cada "flavor" ou variação do SQL.

Sumario

Relacionamento entre tabelas

1. Relacionamento Um-para-Um (One-to-One): uma linha de uma tabela está relacionada a apenas uma linha de outra tabela e vice-versa.
2. Relacionamento Um-para-Muitos (One-to-Many): uma linha de uma tabela pode estar relacionada a várias linhas de outra tabela, mas uma linha da segunda tabela está relacionada apenas a uma linha da primeira tabela.

3. Relacionamento Muitos-para-Muitos (Many-to-Many): várias linhas de uma tabela podem estar relacionadas a várias linhas de outra tabela, sendo necessário uma terceira tabela de associação para estabelecer esse relacionamento.

Essas classificações são importantes para definir a estrutura e as regras de integridade referencial do banco de dados, permitindo que as tabelas sejam conectadas e consultadas de maneira eficiente e precisa.

1. Comandos básicos e intermediários

1. SELECT: usado para selecionar dados de uma tabela ou visualização.
2. FROM: usado para especificar a tabela ou visualização de onde os dados devem ser selecionados.
3. WHERE: usado para filtrar os dados selecionados com base em uma condição.
4. ORDER BY: usado para ordenar os dados selecionados em ordem crescente ou decrescente com base em uma ou mais colunas.
5. GROUP BY: usado para agrupar os dados selecionados com base em uma ou mais colunas.
6. JOIN: usado para combinar dados de duas ou mais tabelas com base em uma condição de junção.
7. INNER JOIN: uma forma de JOIN que retorna apenas as linhas das tabelas que correspondem aos critérios especificados.
8. LEFT JOIN: uma forma de JOIN que retorna todas as linhas da tabela à esquerda e as correspondentes da tabela à direita. Se não houver correspondência, retorna NULL.
9. RIGHT JOIN: uma forma de JOIN que retorna todas as linhas da tabela à direita e as correspondentes da tabela à esquerda. Se não houver correspondência, retorna NULL.
10. FULL OUTER JOIN: uma forma de JOIN que retorna todas as linhas das tabelas à esquerda e à direita, incluindo NULL se não houver correspondência.
11. UNION: usado para combinar os resultados de duas ou mais consultas em uma única tabela.
12. INSERT: usado para inserir novos dados em uma tabela.
13. UPDATE: usado para atualizar os dados existentes em uma tabela.
14. DELETE: usado para excluir dados existentes de uma tabela.
15. CREATE TABLE: usado para criar uma nova tabela.
16. ALTER TABLE: usado para alterar a estrutura de uma tabela existente.
17. DROP TABLE: usado para excluir uma tabela existente.
18. INDEX: usado para criar um índice em uma ou mais colunas de uma tabela para melhorar a velocidade de pesquisa.
19. VIEW: usado para criar uma visualização que é uma representação lógica dos dados de uma ou mais tabelas.
20. TRANSACTION: usado para agrupar uma ou mais operações de banco de dados em uma única transação atômica.

```
Exemplos e aplicações dos comandos básicos e intermediários

# Colta simples
SELECT COUNT (col1) AS count_col1
FROM tab

# Conta e Rename de col1 e col2
SELECT COUNT (col1) AS count_col1, COUNT(col2) AS count_col2
FROM tab

# Conta todas as colunas
SELECT COUNT(*) AS total
FROM tab

# Conta valores diferentes
SELECT COUNT(DISTINCT col1) AS count_distinct_col1
```

```
FROM people


# Critérios e de seleção
  # Critérios simples
  WHERE color = 'green'
  WHERE release_year >= 1960
  WHERE release_year <> 1960

  # Mútiplos critérios (AND / OR / IN)
  WHERE color = 'yellow' OR length = 'short';
  WHERE color = 'yellow' AND length = 'short';
  WHERE buttons BETWEEN 1 AND 5;

  (WHERE (release_year = 1994 OR release_year = 1995)
        AND (certification = 'PG' OR certification = 'R'));

  WHERE release_year IN (1920, 1930, 1940);


# Seleção de textos
  #LIKE é usado para buscar nomes com o valor indicado
  WHERE name LIKE 'Ade%';       #nomes que começam com Ade
  WHERE name LIKE 'Ev_';        #nomes no formato Ev_
  WHERE name NOT LIKE 'A.%';    #nomes sem A. na composição

  #IS NULL e IS NOT NULL
  WHERE col1 IS NULL;           #valores de col1 is null
  WHERE col1 IS NOT NULL;       #valores de col1 is not null

  # Cálculos básicos
  SELECT AVG(budget)            #calcula a média
  SELECT SUM(budget)            #calcula soma
  SELECT MIN(budget)            #calcula a min   (LOWEST)
  SELECT MAX(budget)            #calcula a máx   (HIGHEST)
  SELECT AVG(budget) Where release_year = 2010;
  SELECT ROUND(AVG(budget), 2)  #limita em duas casas decimais
  SELECT COUNT(deathdate) * 100.0 / COUNT(*) AS percentage_dead


# Organizando resultados
  #ORDER BY (ordena)
  ORDER BY budget;
  ORDER BY budget ASC;
  ORDER BY wins ASC, Imdb_score DESC;
  WHERE budget IS NOT NULL  ORDER BY budget DESC;   #retira vazios da orgem

  # GROUP BY (agrupa)
  SELECT col1, col2, COUNT(col3) as count_col3
  FROM tab
  GROUP BY col1, col2
  ORDER BY count_col3, col1;

    #agrupando com média
    SELECT region, AVG(gdp_percapita) AS avg_gdp
    FROM countries AS c
    LEFT JOIN economies AS e
    USING(code)
    WHERE year = 2010
    GROUP BY region;


    #HAVING
  SELECT certification, COUNT(title) AS title_count
  FROM films
  WHERE certification IN ('G','PG','PG-13')
  GROUP BY certification
  HAVING COUNT (tittle) > 500      # coldição de >500 na contagem
  ORDER BY title_count DESC
  LIMIT 3;
```

2. Joining Data in SQL

É o processo de combinar dados de duas ou mais tabelas em um único conjunto de resultados usando a cláusula JOIN em uma consulta SQL.

Existem vários tipos de junções em SQL, incluindo INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL OUTER JOIN, cada um com seu próprio conjunto de regras para combinar os dados.

Ao unir dados em SQL, é importante identificar as colunas relacionadas entre as tabelas e usar essas colunas como chaves de junção.

A junção de dados em SQL é amplamente utilizada na análise de dados e na criação de relatórios para combinar informações de diferentes fontes em um formato útil para tomada de decisões informadas.

Representação didática de Joins

https://i.stack.imgur.com/4c5VU.png

2.1 - INNER JOIN - une tabelas com uma col em comum entre tabelas.

```
# Simple JOIN

# Juntando as colunas de 2 tabelas
SELECT *
FROM tab1
INNER JOIN tab2
USING (id)                          # Existe uma coluna KEY 'id' em cada tabela

  # Unindo colunas específicas de duas tabemas
  SELECT tab1.col1, tab2.col2         # col com mesmo nome em mais de 1 tabela
  INNER JOIN tab2
  ON id.tab1 = i_d.tab2;              # Col 'id' não possuem o mesmo nome

  # Resumindo códigos
  SELECT c.col1 AS country, l.col2 AS language, col3
  FROM table_c AS c
  INNER JOIN table_l AS l
  USING(code);

  # Juntando 2 tabelas com 2 referências
  SELECT *
  FROM left_table
  INNER JOIN right_table
  ON left_table.id = right_table.id               #colunas que se relacionam
    AND left_table.date = right_table.date;       #colunas que se relacionam


# MULTIPLE JOINS

  # Juntando 3 tabelas completas
  SELECT *
  FROM left_table                                    #tab1
    INNER JOIN right_table                           #tab2
    ON left_table.id = right_table.id;               #colunas que se relacionam
      INNER JOIN another_table                       #tab3
      ON left_table.id = another_table.id;           #colunas que se relacionam

  # Juntando 3 tabelas com 2 referencias
  SELECT col1, e.col2, col3, col4
  FROM tab_c AS c
    INNER JOIN tab_p AS p
    ON c.ID = p.country_ID                         #ID das tabelas
      INNER JOIN tab_e AS e
      USING (code)
        AND e.year = p.year;                       #join também por year
```

3.2 - LEFT JOIN, RIGHT JOIN e FULL JOIN

O LEFT JOIN retorna todas as linhas da tabela à esquerda e as correspondentes da tabela à direita. Se não houver correspondência na tabela à direita, ele retorna NULL.

O RIGHT JOIN faz o oposto, retornando todas as linhas da tabela à direita e as correspondentes da tabela à esquerda, com NULL se não houver correspondência na tabela à esquerda.

Já o FULL JOIN, também conhecido como FULL OUTER JOIN, retorna todas as linhas das duas tabelas, combinando as correspondentes, mas preenchendo com NULL onde não há correspondência.

Em resumo, o LEFT JOIN retorna todas as linhas da tabela à esquerda e as correspondentes da tabela à direita, o RIGHT JOIN faz o oposto, retornando todas as linhas da tabela à direita e as correspondentes da tabela à esquerda, e o FULL JOIN retorna todas as linhas das duas tabelas combinadas, preenchendo com NULL onde não há correspondência.

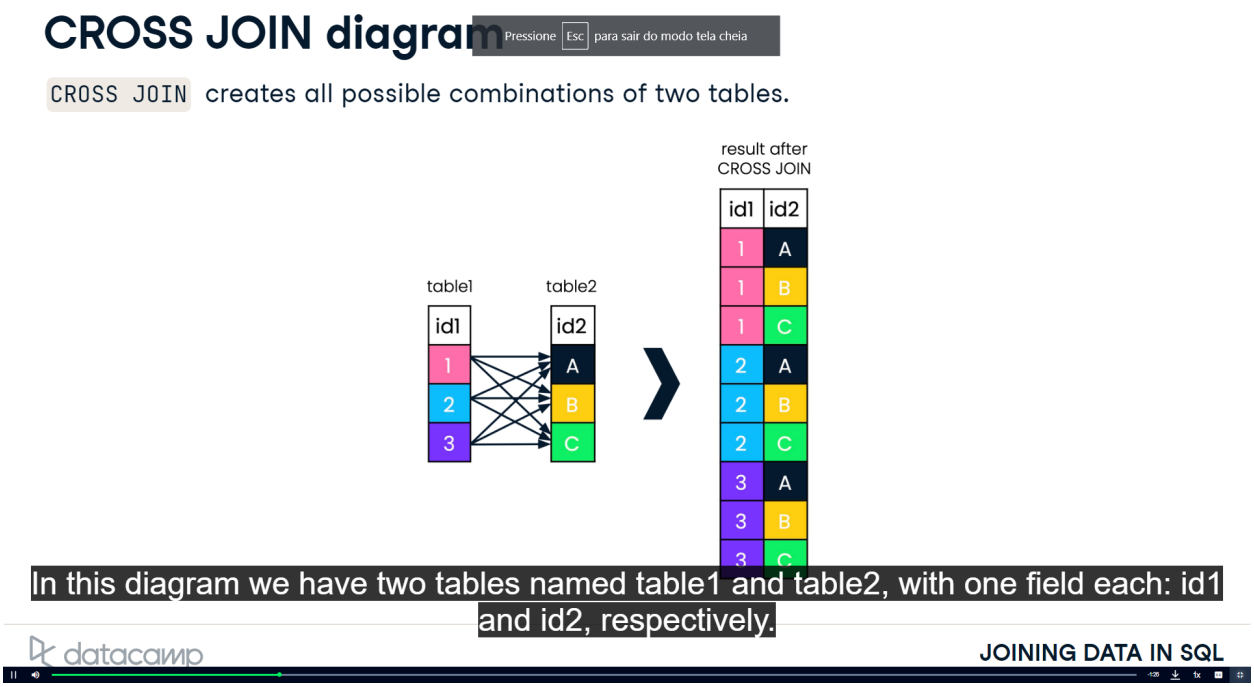
```
# LEFT JOIN - prioriza left e relaciona aos pares de right com valores null
SELECT p1.country, prime_minister, president
FROM prime_minister AS p1
LEFT JOIN presidents AS p2
USING (country)

#RIGHT JOIN - prioriza right e relaciona aos pares em left com valores null
SELECT p1.country, prime_minister, president
FROM prime_minister AS p1
RIGHT JOIN presidents AS p2
USING (country)

#FULL JOIN - junta todos os dados
SELECT left_table.id as L_id
        right_table.id as R_id
FROM left_table
FULL JOIN right_table
WHERE left_table.id = 'alguma coisa'
  OR left_table.id IS NULL
USING (id)

#FULL JOIN em 2 tabelas
SELECT
  c1.name AS country,
  region,
  l.name AS language,
  basic_unit,
  frac_unit
FROM countries as c1
FULL JOIN languages as l
USING(code)
FULL JOIN currencies AS c2
USING(code)
WHERE region LIKE 'M%esia';
```

2.3 Cross Join

Combina cada linha de uma tabela com cada linha de outra tabela. Ele retorna um conjunto de resultados que é o produto cartesiano das duas tabelas envolvidas.

É útil para criar todas as possíveis combinações entre as linhas de duas tabelas.

```
# Cria todas as possibildiades com os identificadores informados
# 3 linhas em 2 identificadores geram 9 possibilidades
SELECT id1, id2
FROM table1
CROSS JOIN table2;
WHERE table1.id1 IN ('asia')               #filtros para reduzid volume
  AND table2.id2 IN ('south america');

# cria a relação de datas 2010 e 2015 da tabela populations com ela mesma
SELECT
  p1.country_code,
    p1.size AS size2010,
    p2.size AS size2015
FROM populations AS p1
INNER JOIN populations AS p2
USING(country_code)
WHERE p1.year = 2010
    AND p1.year = p2.year - 5;
```



2.4 SELF JOINS

É útil para comparar dados dentro da mesma tabela, como em casos de hierarquia ou relacionamentos entre linhas. Ele retorna um conjunto de resultados que combina as linhas da tabela com elas mesmas, com base em uma condição especificada.

```
# cria relações internas
SELECT
  p1.country
  p2.country
  p1.continent
FROM prime_ministers as p1
INNER JOIN prime_ministers as p2       # a tab se junta com ela mesma
ON p1.continent = p2.continent         # critério de junção
  AND p1.country <> p2.country;        # critério de junção
```

2.5 Set theory for SQL joins

UNION combina os resultados de duas ou mais consultas em uma única tabela, removendo duplicatas.

UNION ALL combina os resultados de duas ou mais consultas em uma única tabela, incluindo duplicatas.

INTERSECT retorna os registros que aparecem em ambas as consultas envolvidas.

EXCEPT retorna os registros da primeira consulta que não aparecem na segunda consulta.

```
# UNION - UNE TABELAS EXCLUINDO DUPLICADOS
# número e númer de colunas deve ser identico
SELECT monarch AS leader, country
FROM monarchs
  UNION
  SELECT prime_minister, country
  FROM prime_ministers
  ORDER BY country, leader             #primer_minister entra em leader
  LIMIT 10
```

```
# UNION ALL - UNE TABELAS COM DUPLICADAS
SELECT *
FROM left_table
UNION ALL
SELECT *
FROM right_table

# INTERSECT - interseção entre tabelas
# inner join na mesma situação traria linhas duplicadas
SELECT col1, col2
FROM left_table
INTERSECT
SELECT col1, col2
FROM right_table

#EXCEPT - só valores da tab1 não presentes em tab2
SELECT col1, col2
FROM left_table
EXCEPT
SELECT col1, col2
FROM right_table
```

2.6 Subquerying with semi joins and anti joins

Subqueries, ou subconsultas, são consultas aninhadas dentro de outra consulta SQL. São usadas para obter resultados que serão usados em uma condição de filtro ou comparação na consulta principal.

Uma subquery pode ser executada independentemente ou em conjunto com a consulta principal. Elas podem ser usadas em várias partes de uma consulta, como na cláusula WHERE, FROM, ou na definição de uma tabela temporária.

Em uma subquery com semi join, a consulta retorna somente as linhas da tabela principal que possuem um correspondente na subquery. Por outro lado, uma subquery com anti join retorna somente as linhas da tabela principal que não possuem correspondentes na subquery.

```
# semi-join
SELECT country
FROM states
WHERE indep_year < 1800;

# Subqueries
# seleciona presidentes
SELECT president, country, continent
FROM presidents
WHERE country IN                      #tem q ter o IN para começar a subq
    (SELECT country
      FROM states
      WHERE indep_year <1800);
      AND code NOT IN
          (SELECT code
          FROM currencies);

        # subqueries inside select, cria uma contagem de monarcas por continentes
      SELECT DISTINCT continent,
        (SELECT COUNT (*)
        FROM monarchs
        WHERE states.continent = monarch.continent) AS monarch_count
      FROM states

        # subquerie filtra life_expectancy > 1.15 em 2015
      SELECT *
      FROM populations
      WHERE life_expectancy > 1.15 *
        (SELECT AVG(life_expectancy)
         FROM populations
         WHERE year = 2015)
          AND year = 2015;

        # subquerie filtra a presenã do nome nas duas tabelas
      SELECT name, country_code, urbanarea_pop
      FROM cities
      WHERE name IN
        (SELECT capital
         FROM countries)
      ORDER BY urbanarea_pop DESC;
```

```
# Subqueries indide FROM
SELECT continent, MAX(indep_year) AS most_recent
FROM states
GROUP BY continent;


# Anti Join
# seleciona presidentes da america com independencia < 1800
SELECT president, country
FROM presidents LIKE '%America"
WHERE country NOT IN
    (SELECT country
      FROM states
      WHERE indep_year <1800);
```

3. Data Manipulation

3.1 Case statements (WHEN, THEN, ELSE, END)

Case statements são usados em SQL para avaliar uma condição e retornar um valor específico se a condição for verdadeira.

A sintaxe básica inclui as palavras-chave WHEN, THEN, ELSE e END. O CASE pode ser usado em várias partes de uma consulta, como em colunas, cláusula WHERE ou GROUP BY.

É uma maneira flexível de realizar operações condicionais em uma consulta SQL, permitindo que o usuário especifique uma ou mais condições e o valor correspondente para cada uma delas.

Um teste lógico

```
#Basic case
CASE  WHEN x = 1 THEN 'a'
      WHEN x = 2 THEN 'b'
      ELSE 'c' END AS new_column

#Criando uma nova variável
SELECT col1, col2, col3
CASE  WHEN col2 > col3 THEN '2'
      WHEN col3 > col2 THEN '3'
      ELSE 'tie' END AS outcome
FROM tab1;

#Nomeando colunas por id e contanto total_matches
SELECT
    CASE WHEN hometeam_id = 10189 THEN 'FC Schalke 04'
         WHEN hometeam_id = 9823 THEN 'FC Bayern Munich'
         ELSE 'Other' END AS home_team,
  COUNT(id) AS total_matches
FROM matches_germany
GROUP BY home_team;

# Critérios de busca com criação de coluna
SELECT
  m.date,
  t.team_long_name AS opponent,
    -- Complete the CASE statement with an alias
  CASE WHEN m.home_goal > away_goal THEN 'Barcelona win!'
         WHEN m.home_goal < away_goal THEN 'Barcelona loss :('
         ELSE 'Tie' END AS outcome
FROM matches_spain AS m
LEFT JOIN teams_spain AS t
ON m.awayteam_id = t.team_api_id
-- Filter for Barcelona as the home team
WHERE m.hometeam_id = 8634;
```

Múltiplos testes lógicos

```
# 2 testes lógicos
SELECT col1, col2, col3, col4
```

```
    CASE  WHEN col2 = 123 AND col3 > col4 THEN 'ok'
          WHEN col2 = 321 AND col3 < col3 THEN 'Ñ'
          ELSE 'Loss' END AS outcome
WHERE col1 = 1 OR col1 = 2;

# quando Null for necessário, finaliza-se somente com END AS outcome
# finalizando com valores não nulos "END IS NOT NULL"

# cria 2 colunas e uma etapa de identificação
SELECT
  date,
  CASE WHEN hometeam_id = 8634 THEN 'FC Barcelona'
       ELSE 'Real Madrid CF' END AS home,
  CASE WHEN awayteam_id = 8634 THEN 'FC Barcelona'
       ELSE 'Real Madrid CF' END AS away
FROM matches_spain
#só possibilita jogos entre um e outro
WHERE (awayteam_id = 8634 OR hometeam_id = 8634)
      AND (awayteam_id = 8633 OR hometeam_id = 8633);
```

CASE WHEN com funções

```
# Somando colunas específicas
SELECT date
  ROUND(SUM(CASE WHEN name='gabriel' THEN vitorias END),2) AS 'vitorias de gabriel' #round limita casas decimais
  ROUND(SUM(CASE WHEN name='luan' THEN vitorias END),2) AS 'vitorias de luan'
FROM tabela
GROUP BY date

# Percentages with CASE and AVG
SELECT season
  AVG(CASE  WHEN col1 = 1 AND col2>col3 THEN 1
            WHEN col1 = 1 AND col2<col3 THEN 0
            END) AS percentage
FROM table1
GROUP BY season
```

3.2 Subqueries in WHERE

É útil quando as informações necessárias para filtrar as linhas não estão diretamente disponíveis na tabela principal, mas podem ser encontradas em outra tabela.

Por exemplo, ao selecionar todas as ordens de compra feitas por um cliente específico, pode-se usar uma subquery para selecionar o ID do cliente na tabela de clientes e usá-lo como filtro na consulta principal da tabela de pedidos.

Isso permite que sejam obtidos resultados precisos e filtrados de maneira eficiente.

```
# Simple subquerie - pode ser desenvovida independente de outra query
SELECT col1
FROM table1
WHERE col1 > (SELECT AVG(col1) FROM table1);
SELECT AVG(col1) FROM match

# Subqueries in the where
SELECT col1, col2, col3
FROM tab1
WHERE col4 = '1'
  AND col5 > (SELECT AVG(col2) FROM tab1);

# A subquery selecionará os IDs de clientes da tabela de clientes que têm país = "Brasil", e a consulta principal da tabela de pedidos usar
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT customer_id
    FROM customers
    WHERE country = 'Brazil');
```

3.3 Subqueries in FROM

são usadas para gerar tabelas temporárias que são usadas na consulta principal. Essas subconsultas são colocadas na cláusula FROM da consulta principal e são tratadas como tabelas virtuais temporárias.

Elas são úteis quando se precisa de uma tabela temporária para manipular e extrair informações de várias tabelas.

```
# A subquery in FROM está sendo usada para criar uma tabela temporária com o número total de funcionários em cada departamento. Essa tabela

SELECT d.nome, e.nome AS gerente, emp_count.total_funcionarios
FROM departamentos d
JOIN funcionarios e ON d.gerente_id = e.id
JOIN (
    SELECT dept_id, COUNT(*) AS total_funcionarios
    FROM funcionarios
    GROUP BY dept_id
) emp_count
ON d.id = emp_count.dept_id;
```

3.4 Subqueries in SELECT

É útil quando você precisa usar o resultado de uma consulta interna como uma expressão dentro de outra consulta. Ela permite que você crie consultas mais complexas e sofisticadas que não seriam possíveis de outra forma.

Um exemplo comum é quando você precisa calcular uma métrica com base em um conjunto de dados específico e, em seguida, usá-la em uma consulta mais ampla.

```
# A subquery in SELECT é usada para calcular a soma total dos valores de vendas para cada produto. Essa subquery é executada para cada linh

SELECT produto,
       (SELECT SUM(valor)
        FROM vendas
        WHERE vendas.produto = produtos.produto) AS receita_total
FROM produtos;
```

3.5 Correlated Subquaries

A melhor situação para usar uma subconsulta correlacionada é quando você precisa fazer referência a uma tabela externa na subconsulta que não está presente na subconsulta interna. Em outras palavras, você precisa correlacionar a subconsulta interna com a tabela externa para obter os resultados corretos.

```
# A subconsulta correlacionada é (SELECT COUNT(*) FROM pedidos WHERE id_cliente = c.id) que faz referência à tabela de clientes na consulta

SELECT c.nome, COUNT(*) AS num_pedidos
FROM clientes c
JOIN pedidos p ON c.id = p.id_cliente
WHERE (SELECT COUNT(*) FROM pedidos WHERE id_cliente = c.id) > 1
GROUP BY c.nome;
```

3.6 Nested Subqueries

A melhor situação para usar nested subqueries é quando precisamos filtrar dados em várias etapas, ou seja, a saída de uma subquery é usada como entrada de outra subquery.

```
# A subquery é usada para encontrar o preço máximo de um produto na categoria atual e para obter o nome da categoria desse produto.

SELECT
  nome,
  (SELECT MAX(preco) FROM produtos WHERE id_categoria = categorias.id) AS preco_maximo,
  (SELECT nome FROM categorias WHERE id = produtos.id_categoria) AS nome_categoria
FROM
  produtos
JOIN
  categorias ON produtos.id_categoria = categorias.id;
```

4. PostgreSQL Summary Stats and Window Functions

As funções de janela são usadas para obter estatísticas e resumos para um grupo de linhas (janela) em uma tabela de banco de dados, com base em uma ou mais colunas específicas. Alguns exemplos de funções de janela são ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE(), LAG(), LEAD(), SUM() OVER(), AVG() OVER(), COUNT() OVER(), entre outras.

4.1 **Fetching, ranking, and paging**

Fetching é usado para obter um determinado número de linhas de uma tabela, de acordo com os critérios definidos pelo usuário.

Ranking é usado para classificar as linhas de uma tabela em uma determinada ordem, como por exemplo, em ordem crescente ou decrescente.

Paging é usado para limitar o número de linhas retornadas em uma consulta, de modo a exibir apenas um número específico de linhas em uma determinada página. Essas técnicas são úteis para aumentar a eficiência e a velocidade das consultas SQL em grandes conjuntos de dados.

```
# OVER - indica que é uma windows function

# ROW_NUMBER - cria uma coluna index e pode ser um ranking
  #index
  SELECT
    ROW_NUMBER() OVER() AS Row_N
  FROM Summer_Medals
  ORDER BY Row_N ASC;

  #ranking
  ROW_NUMBER() OVER (ORDER BY Medals DESC) AS Row_N #cria

# LAG - retorna o valor anterior
# LEAD - retorna o valor posterior
SELECT
  Year, Champion,
  LAG(Champion) OVER                    # busca o campeão do ano anterior
    (ORDER BY Year ASC) AS Last_Champion
FROM Weightlifting_Gold
ORDER BY Year ASC;

#PARTITION BY - Agrupa resultados de maneira granular e é usada em conjunto com funções de agregação para calcular métricas como média, som

#FIRST_VALUE - retorna o primeiro valor em uma tabela ou pertição
#LAST_VALUE - retorna o ultimo valor em uma tabela ou partição

#RANKING
ROW_NUMBER()   #organiza sistematicamente os dados
RANK()         #pode repetir dados no formato: 1,2,2,2,5
DENSE_RANK()   #pode repetir dados no formato: 1,2,2,2,3

#PAGING - divide dados em blocos quase iguais
NTILE(n) #recebe dados e divide em páginas, n=nº de divisões
```

4.2 **Aggregate window functions and frames**

```
# FRAME - determina o tamanho dos quadros
n PRECEDING  # n = linhas antes da atual
CURRENT ROW  # a linha atual
n FOLLOWING  # n = linhas depois da atual
```

4.3 Média móvel e total

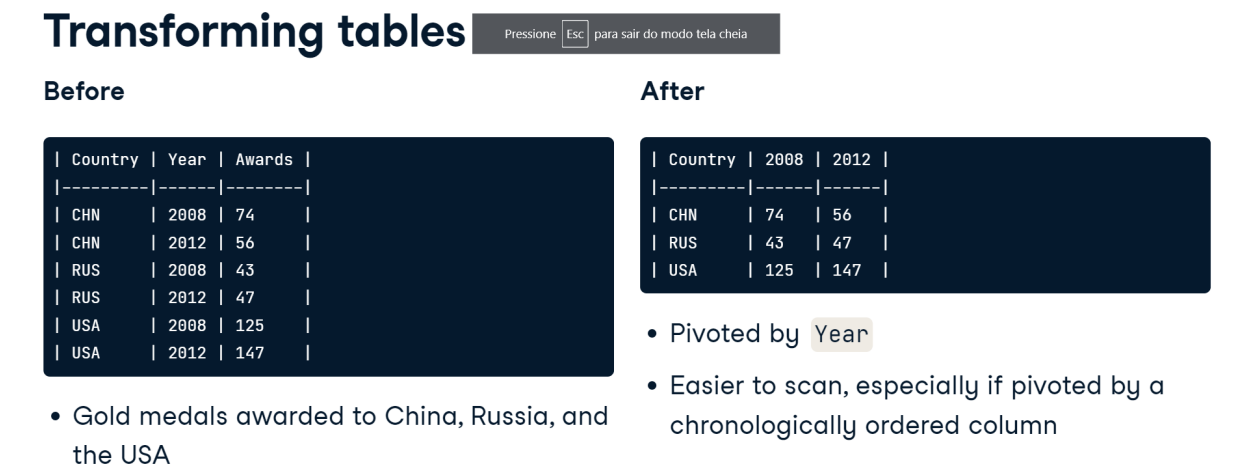
```
# média móvel de 3 linhas (atual e 2 anteriores)

WITH US_Medals AS (...)
SELECT
  Year,
  Medals,
```

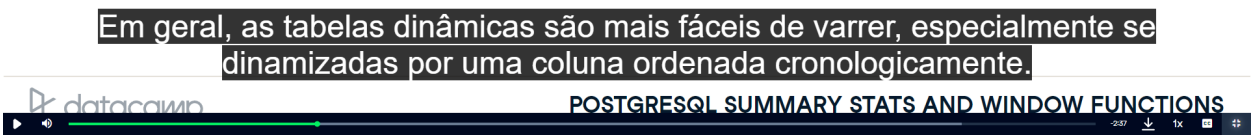
```
  AVG(Medals) OVER (ORDER BY Year ASC          #coluna de média ordenada
    ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)  #informa o intervalo da média (2 anteriores e atual)
    AS Medals_MA
FROM US_Medlas
ORDER BY Year ASC;

# RANGE BETWEEN - trata valores por intervalos, havendo variação por duplicadas
# ROWS BETWEEN - trata valores de forma continua, por linhas individuais
```

4.4 **Beyond window functions**

5. PIVOTING

Muda a estrutura dos dados, groupby or (transposição)





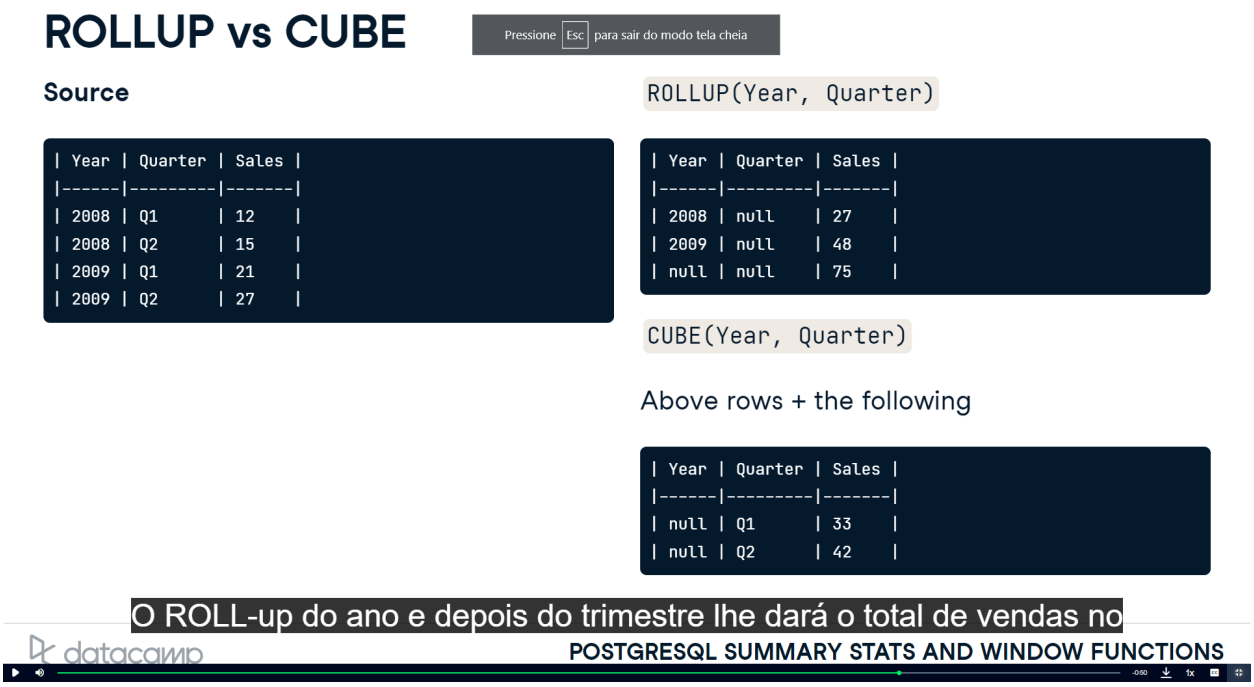

```
CREATE EXTENSION IF NOT EXISTS tablefunc;
#disponibiliza funções extras em uma extensão (biblioteca interna)
```

ROLLUP é uma subclausula de groupby que inclui linhas extras para agregação

```
SELECT country, medal, count(*) AS awards
FROM summer_medals
WHERE
  year = 2008 AND country IN ('chin','rus')
GROUP BY ROLLUP (country, medal)            #cria 2 grupos de groupby!!!!
ORDER BY country ASC, medal ASC
```

CUBE - gera agregações de níveis sem a hierarquia do rollup

```
SELECT country, medal, count(*) AS awards
FROM summer_medals
WHERE
  year = 2008 AND country IN ('chin','rus')
GROUP BY CUBE (country, medal)            #cria 2 grupos de groupby!!!!
ORDER BY country ASC, medal ASC
```



6. Functions for Manipulating Data in PostgreSQL

6.1 **Overview of Common Data Types**

```
Text data types
- CHAR    =
- VARCHAR =

numeric data types
- NUMERIC
- INT

data types
- DATE / TIME / DATE_TIME / INTERVAL

ARRAY

# Visualizando a estrutura de dados da tabela
SELECT *
 FROM INFORMATION_SCHEMA.COLUMNS
 -- Limit to the customer table
 WHERE table_name = 'actor';

# Visualizando o tipo de dado de cada coluna
 SELECT
    column_name,
    data_type
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name ='customer';

# create table e insert data
CREATE TABLE my_table(col1, col2 integer);
INSERT INTO my_table(col1,col2) VALUES('text', 12);
```

Atirhmetic operations (AGE, EXTRACT, DATE_PART, DATE_TRUNC)

```
#AGE
AGE(r.return_date, r.rental_date) AS days_rented

# INTERVL+AL
INTERVAL '1' day * f.rental_duration,
```