SQL - postgreSQL

SELECT:

- usado para selecionar as colunas de uma tabela
- quando selecionar mais de uma coluna, elas devem ser separadas por virgula, sem conter vírgula antes do comando FROM
- pode-se utilizar o (*) para selecionar todas as colunas de uma tabela

```
select email
from sales.customers
```

DISTINCT:

- Comando usado para remover linhas duplicadas e mostrar apenas linhas distintas
- Muito utilizado na etapa de exploração dos dados
- Caso mais de uma coluna seja selecionada, o comando SELECT DISTINCT irá retornar todas as combinações distintas.

```
select distinct brand, model_year
from sales.products
```

WHERE:

- Comando utilizado para filtrar linhas de acordo com uma condição
- No PostgreSQL são utilizadas aspas simples para delimitar strings
- Pode-se combinar mais de uma condição utilizando os operadores lógicos
- No PostgreSQL as datas são escritas no formato 'YYYY-MM-DD' ou 'YYYYMMDD'

```


select email, state, birth_date


```

```
from sales.customers
where (state = 'SC' or state = 'MS') and birth_date < '1991-1
```

ORDER BY:

- Comando utilizado para ordenar a seleção de acordo com uma regra definida
- Por padrão o comando ordena na ordem crescente. Para mudar para a ordem decrescente usar o comando DESC
- No caso de strings a ordenação será seguirá a ordem alfabetica

```
select *
from sales.products
order by price
```

LIMIT:

- Comando utilizado para limitar o nº de linhas da consulta.
- Muito utilizado na etapa de exploração dos dados
- Muito utilizado em conjunto com o comando ORDER BY quando o que importa são
  os TOP N. Ex: "N pagamentos mais recentes", "N produtos mais caros"

```
select visit_id
from sales.funnel

limit 10
```

OPERADORES ARITMÉTICOS:

- Servem para executar operações matemáticas
  Muito utilizados para criar colunas calculadas

```
select
    email,
    birth_date,
    (current_date - birth_date) / 365 as "idade do cliente"
```

```
from sales.customers
order by "idade do cliente"
```

```
select
    first_name || ' ' || last_name as nome_completo
from sales.customers
```

OPERADORES DE COMPARAÇÃO:

Tipos -

- =
- >
- >
- ≥
- ≤
- <>

Exemplo :

```
select
    customer_id,
    first_name,
    professional_status,
    (professional_status = 'clt') as cliente_clt

from sales.customers
```

OPERADORES LÓGICOS:

AND, OR, NOT, BETWEEN, IN, LIKE, ILIKE, IS NULL

Between:

```
select *
from sales.products
where price between 100000 and 200000
```

In:

```
select *
from sales.products
where brand in ('HONDA', 'TOYOTA', 'RENAULT')
```

Like:

```
select distinct first_name
from sales.customers
where first_name like 'ANA%'
```

Ilike: igual o like, porém ignorando se as letras estão maiúsculas ou minúsculas

Is null:

```
select *
from temp_tables.regions
where population is null
```

FUNÇÕES DE AGREGAÇÃO:

- **Count:** Conta o número de linhas em um conjunto de resultados.
- **Sum:** Calcula a soma de valores em uma coluna numérica.
- **Avg:** Calcula a média de valores em uma coluna numérica.
- **Max:** Retorna o maior valor em uma coluna numérica.
- **Min:** Retorna o menor valor em uma coluna numérica.

```
select
    count(*) as total_vendas,
    sum(price) as total_receita,
    avg(price) as preco_medio,
    max(price) as preco_maximo,
    min(price) as preco_minimo
from sales.orders
```

GROUP BY:

- Serve para agrupar registros semelhantes de uma coluna, normalmente utilizado em conjunto com as Funções de agregação.
- O GROUP BY sozinho funciona como um DISTINCT, eliminando linhas duplicadas

```
select state, professional_status, count (*) as contagem
from sales.customers
group by state, professional_status
order by contagem desc
```

HAVING:

- Tem a mesma função do WHERE mas pode ser usado para filtrar os resultados das funções agregadas enquanto o WHERE possui essa limitação
- A função HAVING também pode filtrar colunas não agregadas

```
select
    state,
    count (*)
from sales.customers

group by state
having count (*) > 100
    and state <> 'MG'
```

JOINS:

- Os JOINS são utilizados para combinar registros de duas ou mais tabelas com base em uma coluna comum entre elas.
- Existem diferentes tipos de JOINS, como INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL JOIN, cada um com suas particularidades de combinação.

```
select
    c.customer_id,
    c.first_name,
    c.last_name,
    o.order_id,
    o.order_date
from sales.customers c
inner join sales.orders o on c.customer_id = o.customer_id
```

UNION: União simples de duas tabelas

Exemplo:

```
select * from sales.products
union all
select * from temp_tables.products_2
```

SUBQUERYS:

- Servem para consultar dados de outras consultas.
- Para que as subqueries no WHERE e no SELECT funcionem, elas devem retornar apenas um único valor.
- Não é recomendado utilizar subqueries diretamente dentro do FROM pois isso dificulta a legibilidade da query.

```
with numero_de_visitas as (

    select customer_id, count(*) as n_visitas
    from sales.funnel
    group by customer_id

)
```

```
select
    cus.*,
    n_visitas

from sales.customers as cus
left join numero_de_visitas as ndv
    on cus.customer_id = ndv.customer_id
```

CONVERSÃO DE UNIDADES:

- :: - (::numeric) para converter para números e (::text) para converter para texto
- Cast - usa-se da forma amostrada abaixo:

```
select cast('2021-10-01' as date) - cast('2021-02-01' as date
```

TRATAMENTO GERAL:

- Case when - CASE WHEN é o comando utilizado para criar respostas específicas para diferentes condições e é muito utilizado para fazer agrupamento de dados.

```
--Agrupamento de dados com CASE WHEN
--Calcule o nº de clientes que ganham abaixo de 5k, entre 5k
--15k e acima de 15k


with faixa_de_renda as (
    select
        income,
        case
            when income < 5000 then '0-5000'
            when income >= 5000 and income < 10000 then '5000
            when income >= 10000 and income < 15000 then '100
            else '15000+'
            end as faixa_renda
    from sales.customers
```

```
)

select faixa_renda, count(*)
from faixa_de_renda
group by faixa_renda
```

- COALESCE - é o comando utilizado para preencher campos nulos com o primeiro valor não nulo de uma sequência de valores.

```
-- Crie uma coluna chamada populacao_ajustada na tabela temp_
-- preencha com os dados da coluna population, mas caso esse
-- preencha com a população média (geral) das cidades do Bras

select
    *,
    coalesce(population, (select avg(population) from temp_ta

from temp_tables.regions
```

TRATAMENTO DE TEXTO:

- LOWER() é utilizado para transformar todo texto em letras minúsculas
- UPPER() é utilizado para transformar todo texto em letras maiúsculas
- TRIM() é utilizado para remover os espaços das extremidades de um texto
- REPLACE() é utilizado para substituir uma string por outra string

```
-- Corrija o primeiro elemento das queries abaixo utilizando
-- de tratamento de texto para que o resultado seja sempre TR

select 'São Paulo' = 'SÃO PAULO'
select upper('São Paulo') = 'SÃO PAULO'


select 'São Paulo' = 'são paulo'
select lower('São Paulo') = 'são paulo'
```

```
select 'SÃO PAULO     ' = 'SÃO PAULO'
select trim('SÃO PAULO     ') = 'SÃO PAULO'


select 'SAO PAULO' = 'SÃO PAULO'
select replace('SAO PAULO', 'SAO', 'SÃO') = 'SÃO PAULO'
```

TRATAMENTO DE DATAS:

- Interval - O comando INTERVAL é utilizado para somar datas na unidade desejada. Caso a unidade não seja informada, o SQL irá entender que a soma foi feita em dias

```
-- (Exemplo 1) Soma de datas utilizando INTERVAL
-- Calcule a data de hoje mais 10 unidades (dias, semanas, me
select current_date + 10
select (current_date + interval '10 weeks')::date
select (current_date + interval '10 months')::date
select current_date + interval '10 hours'
```

- Date_trunc - O comando DATE_TRUNC é utilizado para truncar uma data no início do período

```
-- (Exemplo 2) Truncagem de datas utilizando DATE_TRUNC
-- Calcule quantas visitas ocorreram por mês no site da empre

select visit_page_date, count(*)
from sales.funnel
group by visit_page_date
order by visit_page_date desc
```

- Extract - O comando EXTRACT é utilizado para extrair unidades de uma data/timestamp

```
-- (Exemplo 3) Extração de unidades de uma data utilizando EX
-- Calcule qual é o dia da semana que mais recebe visitas ao

select
    extract('dow' from visit_page_date) as dia_da_semana,
    count(*)
from sales.funnel
group by dia_da_semana
order by dia_da_semana
```

- Calculo da diferença entre datas - O cálculo da diferença entre datas com o operador de subtração (-) retorna valores em dias. Para calcular a diferença entre datas em outra unidade é necessário fazer uma transformação de unidades (ou criar uma função para isso)

```
-- (Exemplo 4) Diferença entre datas com operador de subtraçã
-- Calcule a diferença entre hoje e '2018-06-01', em dias, se

select (current_date - '2018-06-01')
select (current_date - '2018-06-01')/7
select (current_date - '2018-06-01')/30
select (current_date - '2018-06-01')/365
```

FUNÇÕES:

- Para criar funções, utiliza-se o comando CREATE FUNCTION
- Para que o comando funcione é obrigatório informar:
  - quais as unidades dos INPUTS
  - quais as unidades dos OUTPUTS
  - em qual linguagem a função será escrita
- O script da função deve estar delimitado por $$
- Para deletar uma função utiliza-se o comando DROP FUNCTION

```
-- Crie uma função chamada DATEDIFF para calcular a diferença
-- duas datas em dias, semanas, meses, anos

$$

    select
        case
            when unidade in ('d', 'day', 'days') then (data_f
            when unidade in ('w', 'week', 'weeks') then (data
            when unidade in ('m', 'month', 'months') then (da
            when unidade in ('y', 'year', 'years') then (data
            end as diferenca

$$

-- Após a função ser criada


select datediff('years', '2021-02-04', current_date)
```

CRIAÇÃO E DELEÇÃO DE TABELAS:

- Para criar tabelas a partir de uma query basta escrever a query normalmente e colocar o comando INTO antes do FROM informando o schema e o nome da tabela a ser criada

ex:

```
-- (Exemplo 1) Criação de tabela a partir de uma query
-- Crie uma tabela chamada customers_age com o id e a idade d
-- Chame-a de temp_tables.customers_age
select
    customer_id,
    datediff ('years', birth_date, current_date) idade_client


    into temp_tables.customer_age
from sales.customers
```

```
select *
from temp_tables.customer_age
```

- Para criar uma tabela vazia do zero é necessário:
  - criar uma tabela vazia com o comando CREATE TABLE
  - Informar que valores serão inseridos com o comando INSERT INTO seguido do nome das colunas
  - Inserir os valores manualmente em forma de lista após o comando VALUES

ex:

```
--Criação de tabela a partir do zero
-- Crie uma tabela com a tradução dos status profissionais do
-- Chame-a de temp_tables.profissoes

select distinct professional_status
from sales.customers

create table temp_tables.profissoes (
    professional_status varchar,
    status_professional varchar
)

insert into temp_tables.profissoes
(professional_status, status_professional)

values
('freelancer', 'freelancer'),
('retired', 'aposentado'),
('clt', 'clt'),
('self_employed', 'autônomo(a)'),
('businessman', 'empresário(a)'),
('civil_servant', 'funcionário_público(a)'),
('student', 'estudante')
```

```
select * from temp_tables.profissoes
```

- Para deletar uma tabela utiliza-se o comando DROP TABLE

```
drop table temp_tables.profissoes
```

INSERÇÃO, ATUALIZAÇÃO E DELEÇÃO DE LINHAS NA TABELA:

- Para inserir linhas em uma tabela é necessário:
  - Informar que valores serão inseridos com o comando INSERT INTO seguido do nome da tabela e nome das colunas
  - Inserir os valores manualmente em forma de lista após o comando VALUES

ex:

```
insert into temp_tables.profissoes
(professional_status, status_professional)

values
('unemployed', 'desempregado(a)'),
('trainee', 'estagiario(a)')
```

- Para atualizar as linhas de uma tabela é necessário:
  - Informar qual tabela será atualizada com o comando UPDATE
  - Informar qual o novo valor com o comando SET
  - Delimitar qual linha será modificada utilizando o filtro WHERE

ex:

```
update temp_tables.profissoes
set professional_status = 'intern'
where status_professional = 'estagiario(a)'
```

- Para deletar linhas de uma tabela é necessário
    - Informar de qual tabela as linhas serão deletadas com o comando DELETE FROM
    - Delimitar qual linha será deletada utilizando o filtro WHERE

ex:

```
delete from temp_tables.profissoes
where status_professional = 'desempregado(a)'
or status_professional = 'estagiario(a)'
```

INSERÇÃO, ATUALIZAÇÃO E DELEÇÃO DE COLUNAS:

- Para fazer qualquer modificação nas colunas de uma tabela utiliza-se o comando ALTER TABLE seguido do nome da tabela.

```
alter table sales.customers
```

- Para adicionar colunas utiliza-se o comando ADD seguido do nome da coluna e da unidade da nova coluna.

```
-- Insira uma coluna na tabela sales.customers com a idade do
alter tables sales.customers
add customer_age int

select * from sales.cutomers limit 10

update sales.customers
set custmer_age = dat_diff('years', birth_date, current_date)
where true
```

- Para mudar o tipo de unidade de uma coluna utiliza-se o comando ALTER COLUMN.

```
-- Alteração do tipo da coluna
-- Altere o tipo da coluna customer_age de inteiro para varch
```

```
alter table sales.customers
alter column customer_age type varchar
```

- Para renomear uma coluna utiliza-se o comando RENAME COLUMN.

```
alter table sales.customers
rename column customer_age to age
```

- Para deletar uma coluna utiliza-se o comando DROP COLUMN.

```
--Deleção de coluna
-- Delete a coluna "age"

alter table sales.customers
drop column age
```

PROJETO - DASHBOARD DE VENDAS

Este projeto tem como objetivo analisar o desempenho de vendas e o comportamento dos visitantes do site de uma empresa durante o mês de agosto de 2021. As consultas realizadas trazem insights valiosos sobre vários aspectos do negócio, organizados da seguinte forma:

1. **Resumo de leads, vendas, receita, conversão e ticket médio**: Apresenta uma visão mensal do funil de vendas, destacando o número de leads captados, vendas realizadas, receita gerada, a taxa de conversão e o ticket médio por venda, proporcionando uma visão clara da eficiência das ações comerciais.
2. **Estados com maior volume de vendas**: Identifica os 5 estados brasileiros que mais se destacaram em número de vendas, oferecendo uma visão geográfica do desempenho de mercado.
3. **Marcas mais vendidas**: Destaca as 5 marcas de produtos que obtiveram o maior número de vendas no mês, permitindo identificar quais marcas

tiveram maior sucesso.

4. **Lojas com melhor desempenho**: Apresenta as 5 lojas que registraram o maior número de vendas, proporcionando uma visão de quais pontos de venda performaram melhor.
5. **Dias da semana com maior tráfego no site**: Revela os dias da semana que concentraram o maior número de visitas ao site, ajudando a identificar padrões de comportamento dos visitantes ao longo da semana.

Com esses dados, o projeto oferece uma análise detalhada do desempenho comercial e do comportamento dos clientes, possibilitando a identificação de tendências e oportunidades estratégicas para melhorar as vendas e otimizar o atendimento ao cliente.

QUERY 1:

```
-- (Query 1) Receita, leads, conversão e ticket médio mês a m
-- Colunas: mês, leads (#), vendas (#), receita (k, R$), conv
with
    leads as (
        select
            date_trunc('month', visit_page_date)::date as vis
            count(*) as visit_page_count
        from sales.funnel
        group by visit_page_month
        order by visit_page_month
    ),

    payments as (
        select
            date_trunc('month', fun.paid_date)::date as paid_
            count(fun.paid_date) as paid_count,
            sum(pro.price * (1+fun.discount)) as receita
        from sales.funnel as fun
        left join sales.products as pro
            on fun.product_id = pro.product_id
        where fun.paid_date is not null
        group by paid_month
```

```
        order by paid_month
    )

select
    leads.visit_page_month as "mês",
    leads.visit_page_count as "leads (#)",
    payments.paid_count as "vendas (#)",
    (payments.receita/1000) as "receita (k, R$)",
    (payments.paid_count::float/leads.visit_page_count::float
    (payments.receita/payments.paid_count/1000) as "ticket mé
from leads
left join payments
    on leads.visit_page_month = paid_month
```

Essa query faz um resumo mês a mês sobre leads, vendas, receita, taxa de conversão e ticket médio. Ela está dividida em duas partes principais:

1. **Subconsulta "leads"**: Agrupa e conta quantos leads (pessoas que visitaram a página) ocorreram em cada mês, truncando as datas para o início do mês (`date_trunc('month')`).
2. **Subconsulta "payments"**: Agrupa e conta quantas vendas foram concluídas em cada mês (com data de pagamento), calculando a receita total para o mês (multiplicando o preço do produto pelos descontos aplicados).

No resultado final, a query faz o seguinte:

- Exibe o mês (`mês`), o número de leads (`leads (#)`), o número de vendas concluídas (`vendas (#)`), a receita em milhares de reais (`receita (k, R$)`), a taxa de conversão (% de vendas em relação aos leads) e o ticket médio (receita dividida pelo número de vendas, também em milhares de reais).
- Ela usa um `LEFT JOIN` para unir os dados de leads e vendas com base no mês.

Assim, a query oferece um panorama de desempenho mensal, incluindo a eficiência da conversão e o valor médio de cada venda.

QUERY 2:

```
-- (Query 2) Estados que mais venderam
-- Colunas: país, estado, vendas (#)

select
    'Brazil' as país,
    cus.state as estado,
    count(fun.paid_date) as "vendas (#)"

from sales.funnel as fun
left join sales.customers as cus
    on fun.customer_id = cus.customer_id
where paid_date between '2021-08-01' and '2021-08-31'
group by país, estado
order by "vendas (#)" desc
limit 5
```

Essa query tem o objetivo de identificar os 5 estados do Brasil que mais realizaram vendas durante o mês de agosto de 2021.

Ela faz o seguinte:

1. **Seleciona o país ("Brazil")**, o estado do cliente e o número de vendas concluídas (contando as datas de pagamento não nulas).
2. Utiliza a tabela `sales.funnel` (onde estão registradas as vendas) e faz um `LEFT JOIN` com a tabela `sales.customers` para associar cada venda com o estado do cliente.
3. Aplica um filtro para considerar apenas vendas feitas entre 1º e 31 de agosto de 2021.
4. Agrupa os resultados por país e estado para contar o número de vendas por estado.
5. Ordena os estados pelo número de vendas em ordem decrescente.
6. Exibe apenas os 5 estados com o maior número de vendas.

O resultado é uma lista dos 5 estados brasileiros que mais venderam no período especificado.

QUERY 3:

```
-- (Query 3) Marcas que mais venderam no mês
-- Colunas: marca, vendas (#)

select
    pro.brand as marca,
    count(fun.paid_date) as "vendas (#)"

from sales.funnel as fun
left join sales.products as pro
    on fun.product_id = pro.product_id
where paid_date between '2021-08-01' and '2021-08-31'
group by marca
order by "vendas (#)" desc
limit 5
```

Essa query tem o objetivo de identificar as 5 marcas que mais venderam durante o mês de agosto de 2021.

Ela faz o seguinte:

1. **Seleciona a marca do produto** e o número de vendas concluídas (contando as datas de pagamento não nulas).
2. Utiliza a tabela `sales.funnel` (onde estão registradas as vendas) e faz um `LEFT JOIN` com a tabela `sales.products` para associar cada venda à marca do produto.
3. Filtra as vendas que ocorreram entre 1º e 31 de agosto de 2021.
4. Agrupa os resultados pela marca, contando o número total de vendas por marca.
5. Ordena os resultados em ordem decrescente de vendas.
6. Exibe as 5 marcas com o maior número de vendas.

O resultado é uma lista das 5 marcas que mais venderam no período especificado.

QUERY 4:

```
-- (Query 4) Lojas que mais venderam
-- Colunas: loja, vendas (#)

select
    sto.store_name as loja,
    count(fun.paid_date) as "vendas (#)"

from sales.funnel as fun
left join sales.stores as sto
    on fun.store_id = sto.store_id
where paid_date between '2021-08-01' and '2021-08-31'
group by loja
order by "vendas (#)" desc
limit 5
```

QUERY 5:

```
-- (Query 5) Dias da semana com maior número de visitas ao si
-- Colunas: dia_semana, dia da semana, visitas (#)

select
    extract('dow' from visit_page_date) as dia_semana,
    case
        when extract('dow' from visit_page_date)=0 then 'domi
        when extract('dow' from visit_page_date)=1 then 'segu
        when extract('dow' from visit_page_date)=2 then 'terç
        when extract('dow' from visit_page_date)=3 then 'quar
        when extract('dow' from visit_page_date)=4 then 'quin
        when extract('dow' from visit_page_date)=5 then 'sext
        when extract('dow' from visit_page_date)=6 then 'sába
        else null end as "dia da semana",
    count(*) as "visitas (#)"
```

```
from sales.funnel
where visit_page_date between '2021-08-01' and '2021-08-31'
group by dia_semana
order by dia_semana
```

Essa query identifica os dias da semana que tiveram o maior número de visitas ao site durante o mês de agosto de 2021.

Ela faz o seguinte:

1. **Extrai o dia da semana** (`dow` - day of the week) da data de visita ao site (`visit_page_date`). O valor vai de 0 (domingo) a 6 (sábado).
2. **Usa uma cláusula** `CASE` para converter os valores numéricos do `dow` em nomes dos dias da semana ("domingo", "segunda", etc.).
3. **Conta o número total de visitas** para cada dia da semana, agrupando os resultados pelo valor extraído de `dow`.
4. Aplica um filtro para considerar apenas as visitas feitas entre 1º e 31 de agosto de 2021.
5. Ordena os resultados pelos dias da semana, de domingo (0) a sábado (6).

O resultado é uma tabela que mostra o número de visitas por dia da semana, com os dias apresentados em ordem cronológica.