INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ

IFCE *CAMPUS CRATO*

Trabalho Prático SQL - N1

Curso: Sistemas de Informação

Disciplina: Banco de Dados II

Professor: Marco André Santos Machado

Tema: Extração de informação de um BD “real”

Equipe:

- Amanda de Lira Silva
- Ana Laís Macêdo Fonte
- Rozane Raquel da Silva Gonçalves
- Matheus Soares do Nascimento

Banco escolhido: Brazilian E-Commerce Public Dataset by Olist

Para este trabalho, nossa equipe escolheu o banco de dados 'Olist E-Commerce'. O motivo principal é que ele é um dataset real, bastante complexo, e atende perfeitamente a todos os requisitos solicitados. Ele tem 9 tabelas relacionadas e muitos dados temporais, como datas de compra, envio e entrega que foram vitais para as análises de tempo e logística de pedidos.

O banco de dados reflete o modelo de negócio da Olist, que não opera como uma loja tradicional, mas sim como um marketplace, conectando vendedores (sellers) a clientes (customers). O funcionamento do banco segue a jornada de um pedido: um cliente faz um pedido (orders), que contém os itens (order_items) desses vendedores. Na sequência, o cliente realiza o pagamento (order_payments)

e, após receber o produto, deixa sua avaliação (order_reviews). Essa estrutura nos permitiu analisar todo o fluxo de ponta a ponta, da venda até a satisfação do cliente.

.

Resolução das questões:

Q1 - Panorama temporal e ranking (funções de janela)

Esse código gera um panorama mensal das vendas de cada vendedor em 2018, calculando a evolução das vendas em relação ao mês anterior (diferença absoluta e percentual) e criando um ranking mensal. Assim, é possível identificar os Top-5 vendedores de cada mês e acompanhar a tendência de desempenho ao longo do ano, destacando os destaques do período.

- Script:

```
WITH vendas_mensais AS (
  SELECT
    DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m-01') AS mes,
    i.seller_id,
    COUNT(DISTINCT o.order_id) AS total_vendas
  FROM olist_orders_dataset o
  JOIN olist_order_items_dataset i
    ON o.order_id = i.order_id
  WHERE o.order_status = 'delivered'
   AND YEAR(o.order_purchase_timestamp) = 2018
  GROUP BY mes, i.seller_id
),
tendencia AS (
  SELECT
    mes,
```

```
    seller_id,
    total_vendas,
    LAG(total_vendas) OVER (PARTITION BY seller_id ORDER BY mes) AS
vendas_mes_anterior,
    total_vendas - LAG(total_vendas) OVER (PARTITION BY seller_id ORDER BY
mes) AS variacao_absoluta,
    ROUND(
      (total_vendas - LAG(total_vendas) OVER (PARTITION BY seller_id ORDER
BY mes))
      / NULLIF(LAG(total_vendas) OVER (PARTITION BY seller_id ORDER BY
mes),0) * 100,
      2
    ) AS variacao_percentual
  FROM vendas_mensais
),
ranking AS (
  SELECT
    mes,
    seller_id,
    total_vendas,
    variacao_absoluta,
    variacao_percentual,
    RANK() OVER (PARTITION BY mes ORDER BY total_vendas DESC) AS
posicao
  FROM tendencia
)
SELECT *
```

```
FROM ranking

WHERE posicao <= 5  -- Top-5 vendedores por mês

ORDER BY mes, posicao;
```

Q2 - View Analítica (monitorar qualidade de dados e priorização)

A view *view_entregas* mostra os pedidos do e-commerce analisando os atrasos de entrega de diferentes formas. Cada pedido recebe uma pontuação de prioridade de 1 a 4, sendo 1 os casos mais graves, como atrasos longos ou entregas não realizadas. A view traz três tipos de análise: por mês, mostrando a quantidade de pedidos, atrasos críticos e a porcentagem de atrasos; por estado do cliente, mostrando quais regiões tiveram mais problemas; e por categoria de produto, mostrando quais tipos de produto tiveram mais atrasos. Dessa forma, é possível acompanhar o desempenho das entregas ao longo do tempo, identificar regiões ou produtos com problemas e ajudar na tomada de decisão para melhorar a logística.

- Script:

```
use trabalhobd2;

create or replace view view_entregas as
with pedidos_prioridade as (
   select
      o.order_id,
      o.customer_id,
      o.order_status,
      o.order_purchase_timestamp,
      o.order_delivered_customer_date,
      o.order_estimated_delivery_date,
      datediff(o.order_delivered_customer_date, o.order_estimated_delivery_date) as
dias_atraso,
      case
         when o.order_delivered_customer_date is null then 1
```

```
            when datediff(o.order_delivered_customer_date,
o.order_estimated_delivery_date) > 10 then 1
            when datediff(o.order_delivered_customer_date,
o.order_estimated_delivery_date) between 5 and 10 then 2
            when datediff(o.order_delivered_customer_date,
o.order_estimated_delivery_date) between 1 and 4 then 3
            else 4
        end as prioridade_score
    from olist_orders_dataset o
    where o.order_status in ('delivered', 'shipped', 'invoiced', 'processing')
),
atrasos_por_mes as (
    select
        year(order_purchase_timestamp) as ano,
        month(order_purchase_timestamp) as mes,
        count(*) as total_pedidos,
        sum(case when prioridade_score = 1 then 1 else 0 end) as atrasos_criticos,
        round(sum(case when prioridade_score = 1 then 1 else 0 end)/count(*)*100,2)
as porcentagem_criticos
    from pedidos_prioridade
    group by ano, mes
),
atrasos_por_estado as (
    select
        c.customer_state as estado,
        count(*) as total_pedidos,
        sum(case when p.prioridade_score = 1 then 1 else 0 end) as atrasos_criticos,
        round(sum(case when p.prioridade_score = 1 then 1 else 0
end)/count(*)*100,2) as porcentagem_criticos
    from pedidos_prioridade p
    join olist_customers_dataset c on p.customer_id = c.customer_id
    group by c.customer_state
),
atrasos_por_categoria as (
    select
        pr.product_category_name as categoria,
        count(*) as total_pedidos,
        sum(case when p.prioridade_score = 1 then 1 else 0 end) as atrasos_criticos,
        round(sum(case when p.prioridade_score = 1 then 1 else 0
end)/count(*)*100,2) as porcentagem_criticos
    from pedidos_prioridade p
    join olist_order_items_dataset i on p.order_id = i.order_id
    join olist_products_dataset pr on i.product_id = pr.product_id
    group by pr.product_category_name
```

```
)
select
    'por_mes' as tipo_analise,
    concat(lpad(mes, 2, '0'), '/', ano) as chave,
    total_pedidos,
    atrasos_criticos,
    porcentagem_criticos
from atrasos_por_mes
union all
select
    'por_estado' as tipo_analise,
    estado as chave,
    total_pedidos,
    atrasos_criticos,
    porcentagem_criticos
from atrasos_por_estado
union all
select
    'por_categoria' as tipo_analise,
    categoria as chave,
    total_pedidos,
    atrasos_criticos,
    porcentagem_criticos
from atrasos_por_categoria;
```

— teste
Esse comando mostra em quais meses o e-commerce teve mais atrasos críticos, ajudando a identificar períodos problemáticos.

```
select * from view_entregas
where tipo_analise = 'por_mes'
order by atrasos_criticos desc;
```

Q3 - Garantia de regra de negócio (trigger com auditoria)

Para garantir a regra do negócio e garantir que as avaliações dos clientes no banco de dados sejam sempre confiáveis, foi criado um trigger que monitora automaticamente a tabela de avaliações (`olist_order_reviews_dataset`) e

aplica uma regra essencial: a nota de avaliação (`review_score`) precisa estar sempre entre 1 e 5.

Sempre que alguém tenta inserir uma nota fora desse intervalo — como “9” ou “-1” por exemplo, o sistema reage. Primeiro, ele registra a tentativa de erro em uma tabela de auditoria, criando um histórico de violações. Em seguida, bloqueia a inserção do dado incorreto antes que ele comprometa a base de dados.

Script:

```
USE atividade_bdii;

CREATE TABLE `log_violacoes_regras` (
  `id_log` INT AUTO_INCREMENT PRIMARY KEY,
  `timestamp_ocorrencia` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  `usuario_db` VARCHAR(255),
  `tabela_afetada` VARCHAR(255),
  `tipo_operacao` VARCHAR(50),
  `descricao_violacao` TEXT,
  `dados_tentativa` TEXT
)
ENGINE=MyISAM;

DELIMITER $$

CREATE TRIGGER trg_validar_e_logar_nota_review
BEFORE INSERT ON olist_order_reviews_dataset
FOR EACH ROW
BEGIN
    DECLARE v_mensagem_erro TEXT;
    DECLARE v_dados_tentativa TEXT;

    IF NEW.review_score < 1 OR NEW.review_score > 5 THEN

        SET v_mensagem_erro = 'Erro: A pontuação do review (review_score) deve
estar entre 1 e 5.';

        SET v_dados_tentativa = CONCAT(
            'order_id: ', IFNULL(NEW.order_id, 'NULL'),
            ', review_id: ', IFNULL(NEW.review_id, 'NULL'),
            ', review_score: ', IFNULL(NEW.review_score, 'NULL')
        );
```

```
        INSERT INTO log_violacoes_regras
        (
            usuario_db, tabela_afetada, tipo_operacao,
            descricao_violacao, dados_tentativa
        )
        VALUES
        (
            USER(), 'olist_order_reviews_dataset', 'INSERT',
            v_mensagem_erro, v_dados_tentativa
        );

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = v_mensagem_erro;

    END IF;
END$$

DELIMITER ;

- - Testes para mostrar o funcionamento do Trigger com auditoria - -


 - - Mostra o bloqueio quando foge da regra do negócio

INSERT INTO olist_order_reviews_dataset
(review_id, order_id, review_score)
VALUES
('review_teste_99', 'order_teste_99', 9);

 - - Mostra as tentativas de violações

SELECT * FROM log_violacoes_regras;
```

Q4 - Automação de relatório (stored procedure)

A procedure SP_VENDAS consegue gerar um relatório do total vendido dos produtos de um vendedor e sua quantidade baseado em um período de tempo.

Ela recebe 3 parâmetros, os 2 primeiros sendo o intervalo de tempo o qual o usuário quer analisar e o terceiro sendo o id do vendedor o qual será analisado, no caso de nenhum id ser fornecido, será gerado um relatório de todos os vendedores.

Script:

```
DROP PROCEDURE IF EXISTS SP_VENDAS;
DELIMITER $$

CREATE PROCEDURE SP_VENDAS (
    IN data_inicio DATE,
    IN data_fim DATE,
    IN seller_id VARCHAR(50)
)
BEGIN
    IF data_inicio IS NULL OR data_fim IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'As datas de início e fim são obrigatórias.';
    END IF;

    SELECT
        items.product_id AS id_produto,
        SUM(items.price) AS total_vendido,
        COUNT(items.product_id) AS qtd_vendas
    FROM olist_orders_dataset AS orders
    JOIN olist_order_items_dataset AS items
        ON orders.order_id = items.order_id
    WHERE orders.order_purchase_timestamp BETWEEN data_inicio AND data_fim
      AND (seller_id IS NULL OR items.seller_id = seller_id)
    GROUP BY items.product_id
    ORDER BY total_vendido DESC;
END$$
```

DELIMITER ;

Exemplo de chamada:

CALL SP_VENDAS('2025-01-01', '2025-11-04',
'd1b65fc7debc3361ea86b5f14c68d2e2');

A chamada acima informa o período de 01/01/2025 a 04/11/2025 com o id de um vendedor, essa chamada irá retornar uma tabela com 3 colunas: produto, total_vendido e qntd_vendas

total_vendido será o valor total que foi vendido referente aquele produto naquele periodo, e qntd_vendas a quantidade de vezes que aquele produto foi vendido.

Q5 - Views para diferentes perfis

A view *vw_painel_vendas_gerencial* resume o desempenho mensal de vendas, facilitando a análise estratégica do negócio (volumes, receita e ticket médio).
Já a *vw_painel_vendas_detalhadas* permite que analistas explorem os dados completos de cada pedido, relacionando produtos, clientes, vendedores e pagamentos.
As duas visões atendem públicos distintos: gestores e analistas.

- Script:

```
drop view if exists vw_painel_vendas_gerencial;

create view vw_painel_vendas_gerencial as
select
   year(o.order_purchase_timestamp) as ano,
   month(o.order_purchase_timestamp) as mes,
   count(distinct o.order_id) as total_pedidos,
   sum(oi.price) as receita_total,
   sum(oi.price + oi.freight_value) as receita_bruta,
   round(sum(oi.price) / count(distinct o.order_id), 2) as ticket_medio,
   rank() over (order by sum(oi.price) desc) as ranking_receita
from olist_orders_dataset o
join olist_order_items_dataset oi on o.order_id = oi.order_id
group by ano, mes;
```

```
drop view if exists vw_painel_vendas_detalhadas;

create view vw_painel_vendas_detalhadas as
select
    o.order_id,
    o.order_purchase_timestamp,
    pr.product_id,
    pr.product_category_name,
    oi.price,
    oi.freight_value,
    pa.payment_type,
    pa.payment_value,
    c.customer_id,
    c.customer_state,
    s.seller_id,
    s.seller_state
from olist_orders_dataset o
join olist_order_items_dataset oi on o.order_id = oi.order_id
join olist_products_dataset pr on oi.product_id = pr.product_id
join olist_customers_dataset c on o.customer_id = c.customer_id
join olist_sellers_dataset s on oi.seller_id = s.seller_id
join olist_order_payments_dataset pa on o.order_id = pa.order_id;

—testes

select * from vw_painel_vendas_gerencial;

select * from vw_painel_vendas_detalhadas;
```