UNIVERSIDADE FEDERAL DA BAHIA
DISCIPLINA: MATA60 – BANCO DE DADOS
DISCENTE: Miguel Santana Fialho Santos (matrícula: 222116289)

PROJETO 1 – Domínio de Aplicação

Vídeo de Dominio de Aplicação: [MATA] Domínio Aplicação
Link para o GitHub com o projeto: https://github.com/guelfialho/Mata60_ProjetoBD

1. PROTÓTIPOS DE TELAS E CÓDIGOS SQL ASSOCIADOS (CÓDIGOS SQL)

1.1. Comandos simples

Para atender aos requisitos de:

1. 10 comandos SQL para exclusão, alteração e inclusão de registros;
2. 4 buscas simples;

- **PROTÓTIPO 1**: Tela de controle dos produtos. É feita uma consulta para buscar todos os produtos ativos, para exibi-los. Em cada linha da exibição, referente a um produto, existe um botão para alteração e exclusão do mesmo. Além disso, no topo da tela tem outro botão para cadastrar outro produto no catálogo. (**RS1**)
  - DML para exibição dos produtos:

```sql
SELECT id_produto, nome, ce_categoria, marca, descricao, preco
FROM tbl_produtos
WHERE deleted = FALSE;
```

  - DML para alteração de um produto específico. Onde “Novo Nome”, “Nova Descrição” e “120.00” são valores sugeridos.

```sql
UPDATE tbl_produtos
SET nome = 'Novo Nome', descricao = 'Nova Descrição', preco = 120.00
WHERE id_produto = 1;
```

  - DML para inserção de um produto. O conteúdo de VALUES é apenas para demonstração.

```sql
INSERT INTO tbl_produtos (nome, ce_categoria, marca, descricao, preco)
VALUES ('Nome do Produto', 1, 'Marca', 'Descrição do Produto', 100.00);
```

- DML para exclusão de um produto específico. O id 1 é apenas para demonstração.

```
UPDATE tbl_produtos
SET deleted = TRUE
WHERE id_produto = 1;
```

- **PROTÓTIPO 2**: Tela de controle das categorias. É feita uma consulta para buscar todas as categorias ativas, para exibi-las. Em cada linha da exibição, referente a uma categoria, existe um botão para alteração e exclusão da mesma. Além disso, no topo da tela tem outro botão para cadastrar outra categoria no sistema. (**RS2**)
  - DML para exibição das categorias:

```
SELECT id_categoria, nome, descricao
FROM tbl_categorias
WHERE deleted = FALSE;
```

  - DML para alteração de uma categoria específica.

```
UPDATE tbl_categorias
SET nome = 'Novo Nome', descricao = 'Nova Descrição'
WHERE id_categoria = 1;
```

  - DML para inserção de uma categoria.

```
INSERT INTO tbl_categorias (nome, descricao)
VALUES ('Nome da Categoria', 'Descrição da Categoria');
```

  - DML para exclusão de uma categoria específica.

```
UPDATE tbl_categorias
SET deleted = TRUE
WHERE id_categoria = 1;
```

- **PROTÓTIPO 3**: Tela de controle dos fornecedores. É feita uma consulta para buscar todos os fornecedores ativos, para exibi-los. Em cada linha da exibição, referente a um

fornecedor , existe um botão para alteração e exclusão do mesmo. Além disso, no topo da tela tem outro botão para cadastrar outro fornecedor no sistema. (**RS3**)

- DML para exibição dos fornecedores:

```
SELECT id_fornecedor, nome, cnpj, avaliacao
FROM tbl_fornecedores
WHERE deleted = FALSE;
```

- DML para alteração de um fornecedor específico.

```
UPDATE tbl_fornecedores
SET nome = 'Novo Nome', cnpj = '00.000.000/0000-00', avaliacao = 4
WHERE id_fornecedor = 1;
```

- DML para inserção de um fornecedor.

```
INSERT INTO tbl_fornecedores (nome, cnpj, avaliacao)
VALUES ('Nome do Fornecedor', '00.000.000/0000-00', 5);
```

- DML para exclusão de um fornecedor específico.

```
UPDATE tbl_fornecedores
SET deleted = TRUE
WHERE id_fornecedor = 1;
```

- **PROTÓTIPO 4**: Tela de controle dos clientes. É feita uma consulta para buscar todos os clientes ativos, para exibi-los. Em cada linha da exibição, referente a um cliente, existe um botão para alteração e exclusão do mesmo. Além disso, no topo da tela tem outro botão para cadastrar outro cliente no sistema. (**RS4**)
  - DML para exibição dos clientes:

```
SELECT id_cliente, nome, cpf, email, realizou_compra, sexo
FROM tbl_clientes
WHERE deleted = FALSE;
```

  - DML para alteração de um cliente específico.

```
UPDATE tbl_clientes
SET nome = 'Novo Nome', email = 'novoemail@cliente.com'
WHERE id_cliente = 1;
```

- DML para inserção de um cliente.

```
INSERT INTO tbl_clientes (nome, cpf, email, sexo)
VALUES ('Nome do Cliente', '000.000.000-00', 'email@cliente.com', 'M');
```

- DML para exclusão de um cliente específico.

```
UPDATE tbl_fornecedores
SET deleted = TRUE
WHERE id_fornecedor = 1;
```

1.2. Comandos Intermediários

Para atender aos requisitos de:

1. Três buscas intermediárias.

- **PROTÓTIPO 5**: Tela de dashboards do sistema,contém gráficos diversos que fornecem informações relevantes para tomada de decisão e acompanhamento dos resultados da TechStore.
  - DML para obter as informações para o gráfico de barras que exibe o perfil demográfico das vendas por sexo (**RS9**)

```
SELECT
    c.sexo,
    COUNT(v.id_venda) AS quantidade_compras
FROM
    tbl_clientes c
JOIN
    tbl_vendas v ON c.id_cliente = v.ce_id_cliente
WHERE
    c.deleted = FALSE
GROUP BY
    c.sexo
ORDER BY
    quantidade_compras DESC;
```

- DML para obter a quantidade de clientes que efetuaram compras em relação a quantidade total de clientes (**RS12**)

```
SELECT
    COUNT(DISTINCT v.ce_id_cliente) AS clientes_compraram,
    COUNT(DISTINCT c.id_cliente) AS total_clientes,
    COUNT(DISTINCT v.ce_id_cliente) * 100.0 / COUNT(DISTINCT c.id_cliente)
    AS proporcao_compraram
FROM
    tbl_clientes c
LEFT JOIN
    tbl_vendas v ON c.id_cliente = v.ce_id_cliente
WHERE
    c.deleted = FALSE;
```

- DML para obter a quantidade total de um produto em estoque considerando todos os seus fornecedores (**RS5**)

```
SELECT
  p.nome AS produto,
  SUM(pf.quantidade_em_estoque) AS quantidade_total_estoque
FROM tbl_produtos p
JOIN tbl_produtos_fornecedores pf ON p.id_produto = pf.ce_id_produto
GROUP BY p.nome
ORDER BY quantidade_total_estoque DESC;
```

1.3. Comandos Avançados

Para atender aos requisitos de:

1. Três buscas avançadas.

- **PROTÓTIPO 6**: Segunda tela de dashboards do sistema, contém gráficos diversos que fornecem informações relevantes para tomada de decisão e acompanhamento dos resultados da TechStore.

- DML para obter as informações para o gráfico de ticket médio de vendas por categoria (**RS11**)

```sql
SELECT
    c.nome AS categoria,
    SUM(v.valor_total) / COUNT(v.id_venda) AS ticket_medio
FROM
    tbl_vendas v
JOIN
    tbl_produtos_fornecedores pf
      ON v.ce_id_produto_fornecedor = pf.id_produto_fornecedor
JOIN
    tbl_produtos p ON pf.ce_id_produto = p.id_produto
JOIN
    tbl_categorias c ON p.ce_categoria = c.id_categoria
WHERE
    v.criado_em IS NOT NULL
    AND c.deleted = FALSE
    AND p.deleted = FALSE
GROUP BY
    c.nome
ORDER BY
    ticket_medio DESC;
```

- DML para obter as informações para construção do gráfico de identificação dos produtos com as melhores avaliações por fornecedor.

```sql
SELECT
    f.nome AS fornecedor,
    p.nome AS produto,
    cat.nome AS categoria,
    AVG(a.nota) AS media_avaliacao
FROM
    tbl_avaliacoes a
JOIN
    tbl_produtos p ON a.ce_id_produto = p.id_produto
JOIN
    tbl_categorias cat ON p.ce_categoria = cat.id_categoria
JOIN
    tbl_produtos_fornecedores pf ON p.id_produto = pf.ce_id_produto
JOIN
    tbl_fornecedores f ON pf.ce_id_fornecedor = f.id_fornecedor
GROUP BY
    f.nome, p.nome, cat.nome
HAVING
    AVG(a.nota) >= 4
ORDER BY
    media_avaliacao DESC;
```

- DML para identificar a diversidade das categorias dos produtos adquiridos por cada um dos clientes e quais clientes possuem um perfil de adquirir produtos de

diversas categorias:

```sql
SELECT
    c.nome AS cliente,
    COUNT(DISTINCT cat.id_categoria) AS total_categorias
FROM
    tbl_vendas v
JOIN
    tbl_produtos_fornecedores pf ON v.ce_id_produto_fornecedor = pf.id_produto_fornecedor
JOIN
    tbl_produtos p ON pf.ce_id_produto = p.id_produto
JOIN
    tbl_categorias cat ON p.ce_categoria = cat.id_categoria
JOIN
    tbl_clientes c ON v.ce_id_cliente = c.id_cliente
GROUP BY
    c.nome
ORDER BY
    total_categorias DESC;
```

2. SUB-ROTINAS DE SUPORTE

Essas stored procedures em conjunto, irão atender aos requisitos **RS6** e **RS7**.

2.1. Stored Procedure 1

```
CREATE OR REPLACE PROCEDURE sp_atualizar_materialized_views()
    LANGUAGE plpgsql
    AS $$

    BEGIN

    REFRESH MATERIALIZED VIEW mv_quantidade_estoque_por_categoria;
    RAISE NOTICE
    'Materialized View mv_quantidade_estoque_por_categoria atualizada com sucesso!';

    REFRESH MATERIALIZED VIEW mv_perfil_demografico_clientes_compras;
    RAISE NOTICE
    'Materialized View mv_perfil_demografico_clientes_compras atualizada com sucesso!';
END;
$$;
```

Essa procedure facilita a atualização das materialized views criadas, uma vez que a diferença dela para as views é que a primeira realmente armazena esses dados enquanto que a segunda faz uma consulta dinamicamente no banco. Essa diferença gera a necessidade de dar um “refresh” na materialized view para refletir o estado atual do banco.

2.2. Stored Procedure 2 (RS6 E RS7)

```
CREATE OR REPLACE PROCEDURE sp_processar_venda(
        IN p_id_produto INT,
        IN p_id_cliente INT,
        IN p_quantidade INT
    )
  LANGUAGE plpgsql AS $$
    DECLARE
        v_id_produto_fornecedor INT;
        v_quantidade_disponivel INT;
        v_valor_total FLOAT;
        v_preco_produto FLOAT;
    BEGIN
        SELECT
            id_produto_fornecedor,
            quantidade_em_estoque
        INTO
            v_id_produto_fornecedor,
            v_quantidade_disponivel
        FROM
            tbl_produtos_fornecedores
        WHERE
            ce_id_produto = p_id_produto
            AND quantidade_em_estoque >= p_quantidade
        ORDER BY
            quantidade_em_estoque DESC
        LIMIT 1;

        IF v_id_produto_fornecedor IS NULL THEN
            RAISE EXCEPTION 'Estoque insuficiente para o produto % com a quantidade %',
            p_id_produto,
            p_quantidade;
        END IF;

        SELECT preco INTO v_preco_produto
        FROM tbl_produtos
        WHERE id_produto = p_id_produto;

        v_valor_total := v_preco_produto * p_quantidade;

        INSERT INTO tbl_vendas (
            ce_id_produto_fornecedor,
            ce_id_cliente,
            quantidade,
            valor_total
        )
        VALUES (v_id_produto_fornecedor, p_id_cliente, p_quantidade, v_valor_total);

        UPDATE tbl_clientes
        SET realizou_compra = TRUE
        WHERE id_cliente = p_id_cliente;

        UPDATE tbl_produtos_fornecedores
        SET quantidade_em_estoque = quantidade_em_estoque - p_quantidade
        WHERE id_produto_fornecedor = v_id_produto_fornecedor;

        COMMIT;
    END;
    $$;
```

Essa procedure garante que a venda ocorra de forma a respeitar os requisitos 6 e 7 do sistema. A compra só ocorre se tiver um fornecedor com a quantidade do produto

em estoque suficiente para a compra. Além disso, atualiza o valor do estoque após inserir a venda, bem como muda o atributo na tabela “tbl_clientes” que sinaliza se o mesmo realizou uma compra ou não para true.

2.3. Materialize view 1

```sql
CREATE MATERIALIZED VIEW mv_quantidade_estoque_por_categoria AS
SELECT
    c.nome AS categoria,
    SUM(pf.quantidade_em_estoque) AS quantidade_total_estoque
FROM
    tbl_categorias c
JOIN
    tbl_produtos p ON c.id_categoria = p.ce_categoria
JOIN
    tbl_produtos_fornecedores pf ON p.id_produto = pf.ce_id_produto
WHERE
    c.deleted = FALSE
    AND p.deleted = FALSE
    AND pf.deleted = FALSE
GROUP BY
    c.nome
ORDER BY
    quantidade_total_estoque DESC;
```

A materialized view em questão atende ao **RS14** de modo a identificar a quantidade de estoque dos produtos por categoria.

2.4. Materialize view 2

```sql
CREATE MATERIALIZED VIEW mv_perfil_demografico_clientes_compras AS
SELECT
    c.sexo,
    COUNT(v.id_venda) AS quantidade_compras
FROM
    tbl_clientes c
JOIN
    tbl_vendas v ON c.id_cliente = v.ce_id_cliente
WHERE
    c.deleted = FALSE
GROUP BY
    c.sexo
ORDER BY
    quantidade_compras DESC;
```

A materialized view em questão atende ao **RS9** de modo a identificar o perfil demográfico dos clientes que realizaram compras.