UNIVERSIDADE ESTADUAL DE LONDRINA

LUCCA GIOVANE GOME

ATIVIDADE 1: VIEWS, CTE, WINDOW FUNCTIONS E DESENVOLVIMENTO EM PL/SQL

LONDRINA - PR
2024

Relatório

1. Criação do Banco de Dados

1. 1.1. Scripts de Criação das Tabelas

Abaixo estão os scripts utilizados para criar o banco de dados no **schema `luccagomes`** do Oracle.

```
-- ------------------------------------------------------------------------
-- 1) Tentar dropar todas as tabelas se existirem, na ordem inversa de criação
-- ------------------------------------------------------------------------
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.compra CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
     -- -942 = table or view does not exist
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.aluguel CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.nota_fiscal CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.generos_da_midia CASCADE
CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
```

```
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.idiomas_da_midia CASCADE
CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.midia CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.genero CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.idioma CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/
BEGIN
   EXECUTE IMMEDIATE 'DROP TABLE luccagomes.usuario CASCADE CONSTRAINTS';
EXCEPTION
   WHEN OTHERS THEN
     IF SQLCODE != -942 THEN RAISE; END IF;
END;
/

-- ------------------------------------------------------------------------
-- 2) Criação das tabelas no schema luccagomes
-- ------------------------------------------------------------------------

-----------------------------------------------------------------------------
-- TABELA USUARIO
-----------------------------------------------------------------------------
CREATE TABLE luccagomes.usuario (
    id          NUMBER GENERATED ALWAYS AS IDENTITY,
    nome        VARCHAR2(255)    NOT NULL,
    login       VARCHAR2(255)    NOT NULL,
```

```
    senha       VARCHAR2(255)    NOT NULL,
    nasc        DATE             NOT NULL,
    ativo       CHAR(1)          DEFAULT 'Y'  NOT NULL,
      -- 'Y' = ativo, 'N' = inativo

    CONSTRAINT pk_usuario PRIMARY KEY (id),
    CONSTRAINT uk_usuario_login UNIQUE (login)
);

-------------------------------------------------------------------------------
-- TABELA IDIOMA
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.idioma (
    id       NUMBER GENERATED ALWAYS AS IDENTITY,
    idioma   VARCHAR2(24) NOT NULL,

    CONSTRAINT pk_idioma PRIMARY KEY (id),
    CONSTRAINT uk_idioma UNIQUE (idioma)
);

-------------------------------------------------------------------------------
-- TABELA GENERO
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.genero (
    id       NUMBER GENERATED ALWAYS AS IDENTITY,
    genero   VARCHAR2(24) NOT NULL,

    CONSTRAINT pk_genero PRIMARY KEY (id),
    CONSTRAINT uk_genero UNIQUE (genero)
);

-------------------------------------------------------------------------------
-- TABELA MIDIA
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.midia (
    id            NUMBER GENERATED ALWAYS AS IDENTITY,
    titulo        VARCHAR2(255) NOT NULL,
    sinopse       CLOB,
    avaliacao     NUMBER(3,2),
    poster        VARCHAR2(255),
    atores        VARCHAR2(255),
    dt_lancamento DATE NOT NULL,
    valor         NUMBER(10,2) NOT NULL,
    duracao       NUMBER,        -- exclusivo de filme
    temporadas    NUMBER,        -- exclusivo de série
    ativo         CHAR(1)  DEFAULT 'Y' NOT NULL,
      -- 'Y' = ativo, 'N' = inativo
```

```
    CONSTRAINT pk_midia PRIMARY KEY (id),
    CONSTRAINT ck_filme_ou_serie CHECK (
        (
          duracao IS NULL
          AND temporadas IS NOT NULL
        )
        OR
        (
          duracao IS NOT NULL
          AND temporadas IS NULL
        )
    )
);

-------------------------------------------------------------------------------
-- TABELA IDIOMAS_DA_MIDIA
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.idiomas_da_midia (
    midia_id   NUMBER,
    idioma_id  NUMBER,

    CONSTRAINT pk_idiomas_da_midia PRIMARY KEY (midia_id, idioma_id),
    CONSTRAINT fk_midia_tem_idioma FOREIGN KEY (midia_id)
        REFERENCES luccagomes.midia(id),
    CONSTRAINT fk_idioma_da_midia FOREIGN KEY (idioma_id)
        REFERENCES luccagomes.idioma(id)
);

-------------------------------------------------------------------------------
-- TABELA GENEROS_DA_MIDIA
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.generos_da_midia (
    midia_id   NUMBER,
    genero_id  NUMBER,

    CONSTRAINT pk_generos_da_midia PRIMARY KEY (midia_id, genero_id),
    CONSTRAINT fk_midia_tem_genero FOREIGN KEY (midia_id)
        REFERENCES luccagomes.midia(id),
    CONSTRAINT fk_genero_da_midia FOREIGN KEY (genero_id)
        REFERENCES luccagomes.genero(id)
);

-------------------------------------------------------------------------------
-- TABELA NOTA_FISCAL
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.nota_fiscal (
```

```
    usuario_id   NUMBER,
    valor_total  NUMBER(10,2) NOT NULL,
    dt_pagamento TIMESTAMP DEFAULT SYSTIMESTAMP,

    CONSTRAINT pk_nota_fiscal PRIMARY KEY (usuario_id, dt_pagamento),
    CONSTRAINT fk_usuario_possui FOREIGN KEY (usuario_id)
        REFERENCES luccagomes.usuario(id)
);

-------------------------------------------------------------------------------
-- TABELA ALUGUEL
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.aluguel (
    usuario_id  NUMBER,
    midia_id    NUMBER,
    dt_inicio   TIMESTAMP,
    dt_expira   TIMESTAMP NOT NULL,
    valor       NUMBER(10,2) NOT NULL,

    CONSTRAINT pk_aluguel PRIMARY KEY (usuario_id, midia_id, dt_inicio),
    CONSTRAINT fk_usuario_alugou FOREIGN KEY (usuario_id, dt_inicio)
        REFERENCES luccagomes.nota_fiscal (usuario_id, dt_pagamento),
    CONSTRAINT fk_midia_foi_alugada FOREIGN KEY (midia_id)
        REFERENCES luccagomes.midia (id),
    CONSTRAINT ck_dt_aluguel CHECK (dt_inicio < dt_expira),
    CONSTRAINT ck_valor_aluguel_positivo CHECK (valor > 0)
);

-------------------------------------------------------------------------------
-- TABELA COMPRA
-------------------------------------------------------------------------------
CREATE TABLE luccagomes.compra (
    usuario_id  NUMBER,
    midia_id    NUMBER,
    dt_compra   TIMESTAMP,
    valor       NUMBER(10,2) NOT NULL,

    CONSTRAINT pk_comprou PRIMARY KEY (usuario_id, midia_id),
    CONSTRAINT fk_usuario_comprou FOREIGN KEY (usuario_id, dt_compra)
        REFERENCES luccagomes.nota_fiscal (usuario_id, dt_pagamento),
    CONSTRAINT fk_midia_foi_comprada FOREIGN KEY (midia_id)
        REFERENCES luccagomes.midia (id),
    CONSTRAINT ck_valor_compra_positivo CHECK (valor > 0)
);
```

1.2. Relatório

Para essa migração para a Oracle, foi feita uma “tradução” do banco marketplace.sql (https://github.com/LuccaGiovane/UEL-BD/blob/main/marketplace.sql) feito no semestre passado para a matéria de Banco de Dados 1.

Assim, algumas mudanças foram feitas para a criação do código acima:

1. Migração de SERIAL para NUMBER GENERATED ALWAYS AS IDENTITY:

- No PostgreSQL, campos autoincrementais costumam ser declarados como SERIAL (ex.: id SERIAL). Já no Oracle, não há SERIAL, mas sim o recurso de coluna Identity:
    - id NUMBER GENERATED ALWAYS AS IDENTITY
- Isso foi aplicado em tabelas como *usuario, idioma, genero* e *midia.*

2. Tratamento de Booleanos (`ativo ou inativo`)

- No PostgreSQL, o campo `ativo` era `BOOLEAN`, geralmente `TRUE` ou `FALSE`. No Oracle, não há tipo booleano nativo armazenável em tabela, então a solução foi usar `CHAR(1)` com `'Y'` ou `'N'` como valores válidos. Em vez de `DEFAULT TRUE`, define-se `DEFAULT 'Y'`.

3. Conversão de `TEXT` para `CLOB`

- No PostgreSQL, havia o tipo `TEXT` (como em `sinopse TEXT`). No Oracle, um tipo análogo para armazenar grandes cadeias de texto é `CLOB`. Por isso, na criação da tabela `midia`, a coluna `sinopse` virou sinopse CLOB

4. Ajustes de Funções e Datas Padrão

- No PostgreSQL, utilizava o `NOW()` como valor default em colunas do tipo `TIMESTAMP`. No Oracle, utilizamos `SYSTIMESTAMP` (ou poderia usar `SYSDATE`, dependendo do contexto).

- Um exemplo é na tabela `nota_fiscal`, a coluna `dt_pagamento` passou de `TIMESTAMP DEFAULT(NOW())` (Postgres) para `TIMESTAMP DEFAULT SYSTIMESTAMP` (Oracle).

5. Diferenças de Intervalos e Checks

- No PostgreSQL, pode-se fazer algo como `dt_expira TIMESTAMP NOT NULL DEFAULT (dt_inicio + INTERVAL '30 days')`. Já no no Oracle adotei a seguinte abordagem:
    - Declarar `dt_expira` como `TIMESTAMP NOT NULL`.
    - Criar `CHECK (dt_inicio < dt_expira)` ou programar a lógica em triggers/procedures se o prazo precisar de mais dinâmica.
    - No script ficou: CONSTRAINT ck_dt_aluguel CHECK (dt_inicio < dt_expira)

De mudanças mais notáveis foram essas alem de coisas como DROP SCHEMA marketplace CASCADE por DROP TABLE ... CASCADE CONSTRAINTS o nome do schema anteriormente ser marketplace e agora luccagomes (que e meu login)

A constraint `ck_filme_ou_serie` foi mantida em ambos, pois funciona de forma similar, só mudando a sintaxe do check para o dialeto Oracle.

As chaves estrangeiras (`FOREIGN KEY`) e chaves primárias (`PRIMARY KEY`) mantiveram a mesma lógica, apenas ajustando para apontar para o schema `luccagomes` (em vez de `marketplace`).

Alguns nomes de constraints foram adaptados para o padrão Oracle (`pk_`, `fk_`, etc.), mas mantêm o mesmo significado.

2. Procedimento em PL/SQL para Carga de Dados (Fator de Escala)

Objetivo: Criar um procedimento que gere dados (semi)aleatórios nas tabelas, recebendo como parâmetro um fator de escala. O fator 1 gera, por exemplo, 5 registros em cada tabela; o fator 2 gera 10, etc.

2. 2.1. Script de Criação do Procedimento

```
-------------------------------------------------------------------------------
-
-- carrega_dados.sql
-- Script que:
--   1) Cria/recria o procedimento sp_carga_dados
--   2) Executa sp_carga_dados(p_fator)
-------------------------------------------------------------------------------
-

PROMPT ========== CRIANDO/RECRIANDO O PROCEDIMENTO sp_carga_dados ==========

CREATE OR REPLACE PROCEDURE luccagomes.sp_carga_dados(p_fator IN NUMBER) AS
BEGIN
    -- 1) Limpando as tabelas (apenas DELETE, sem DROP)
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.idiomas_da_midia';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.generos_da_midia';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.aluguel';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.compra';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.nota_fiscal';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.midia';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.genero';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.idioma';
    EXECUTE IMMEDIATE 'DELETE FROM luccagomes.usuario';

    COMMIT;

    DECLARE
        v_count      NUMBER := 1;
        v_limite     NUMBER := 5 * p_fator;  -- base de 5, ajustada pelo fator

        -- Variáveis para armazenar os IDs reais dos idiomas inseridos
        v_idioma1    NUMBER;
        v_idioma2    NUMBER;
        v_idioma3    NUMBER;

        -- Variáveis para armazenar os IDs reais dos gêneros inseridos
        v_genero1    NUMBER;
        v_genero2    NUMBER;
        v_genero3    NUMBER;
```

```
    -- Variáveis auxiliares para compra
    v_midia_compra   NUMBER;
    v_count_compra   NUMBER;
BEGIN
    ------------------------------------------------------------------------
    -- 2) Inserir usuários
    ------------------------------------------------------------------------
    WHILE v_count <= v_limite LOOP
        INSERT INTO luccagomes.usuario (nome, login, senha, nasc, ativo)
        VALUES (
            'Usuario ' || v_count,
            'login' || v_count,
            'senha' || v_count,
            TRUNC(TO_DATE('01/01/1990','DD/MM/YYYY')
                  + DBMS_RANDOM.value(1,10000)),  -- datas aleatórias
            CASE WHEN MOD(v_count, 2) = 0 THEN 'Y' ELSE 'N' END
        );
        v_count := v_count + 1;
    END LOOP;

    ------------------------------------------------------------------------
    -- 3) Inserir idiomas (capturando o ID real gerado)
    ------------------------------------------------------------------------
    INSERT INTO luccagomes.idioma (idioma)
    VALUES ('Portugues')
    RETURNING id INTO v_idioma1;

    INSERT INTO luccagomes.idioma (idioma)
    VALUES ('Vulcano')
    RETURNING id INTO v_idioma2;

    INSERT INTO luccagomes.idioma (idioma)
    VALUES ('Alemao')
    RETURNING id INTO v_idioma3;

    ------------------------------------------------------------------------
    -- 4) Inserir gêneros (capturando o ID real gerado)
    ------------------------------------------------------------------------
    INSERT INTO luccagomes.genero (genero)
    VALUES ('Genero_1')
    RETURNING id INTO v_genero1;

    INSERT INTO luccagomes.genero (genero)
    VALUES ('Genero_2')
    RETURNING id INTO v_genero2;

    INSERT INTO luccagomes.genero (genero)
```

```
        VALUES ('Genero_3')
        RETURNING id INTO v_genero3;

        ------------------------------------------------------------------------
        -- 5) Inserir mídias
        ------------------------------------------------------------------------
        v_count := 1;
        WHILE v_count <= v_limite LOOP
            INSERT INTO luccagomes.midia
                (titulo, sinopse, avaliacao, poster, atores, dt_lancamento,
                 valor, duracao, temporadas, ativo)
            VALUES (
                'Midia ' || v_count,
                'Sinopse da midia ' || v_count,
                ROUND(DBMS_RANDOM.value(0,10),2),
                'poster' || v_count || '.jpg',
                'Ator_' || v_count || ', Atriz_' || v_count,
                TRUNC(TO_DATE('01/01/2000','DD/MM/YYYY')
                      + DBMS_RANDOM.value(1,5000)),   -- datas aleatórias
                ROUND(DBMS_RANDOM.value(10,100),2),
                CASE WHEN MOD(v_count,2)=0 THEN
ROUND(DBMS_RANDOM.value(90,180))
                     ELSE NULL END,   -- Se par, é filme (duracao != NULL)
                CASE WHEN MOD(v_count,2)=1 THEN TRUNC(DBMS_RANDOM.value(1,6))
                     ELSE NULL END,   -- Se ímpar, é série (temporadas !=
NULL)
                CASE WHEN MOD(v_count,3)=0 THEN 'N' ELSE 'Y' END
            );
            v_count := v_count + 1;
        END LOOP;

        ------------------------------------------------------------------------
        -- 6) Inserir relacionamentos entre mídias e idiomas
        ------------------------------------------------------------------------
        FOR mid IN (SELECT id FROM luccagomes.midia) LOOP
            -- idioma 1
            INSERT INTO luccagomes.idiomas_da_midia (midia_id, idioma_id)
            VALUES (mid.id, v_idioma1);

            -- chance aleatória de inserir o idioma 2
            IF DBMS_RANDOM.value(0,1) > 0.5 THEN
                INSERT INTO luccagomes.idiomas_da_midia (midia_id, idioma_id)
                VALUES (mid.id, v_idioma2);
            END IF;
        END LOOP;

        ------------------------------------------------------------------------
```

```
    -- 7) Inserir relacionamentos entre mídias e gêneros
    ------------------------------------------------------------------------
    FOR mid IN (SELECT id FROM luccagomes.midia) LOOP
        -- Usa o ID real do 1º gênero
        INSERT INTO luccagomes.generos_da_midia (midia_id, genero_id)
        VALUES (mid.id, v_genero1);

        -- chance aleatória de inserir também o 2º gênero
        IF DBMS_RANDOM.value(0,1) > 0.5 THEN
            INSERT INTO luccagomes.generos_da_midia (midia_id, genero_id)
            VALUES (mid.id, v_genero2);
        END IF;

        -- Exemplo: inserir o 3º se quiser
        /*
        IF DBMS_RANDOM.value(0,1) > 0.7 THEN
            INSERT INTO luccagomes.generos_da_midia (midia_id, genero_id)
            VALUES (mid.id, v_genero3);
        END IF;
        */
    END LOOP;

    ------------------------------------------------------------------------
    -- 8) Inserir notas fiscais (2 por usuário ativo)
    ------------------------------------------------------------------------
    FOR usr IN (SELECT id FROM luccagomes.usuario WHERE ativo = 'Y') LOOP
        INSERT INTO luccagomes.nota_fiscal (usuario_id, valor_total)
        VALUES (usr.id, ROUND(DBMS_RANDOM.value(50,300),2));

        INSERT INTO luccagomes.nota_fiscal (usuario_id, valor_total)
        VALUES (usr.id, ROUND(DBMS_RANDOM.value(50,300),2));
    END LOOP;

    ------------------------------------------------------------------------
    -- 9) Inserir alugueis e compras para cada nota fiscal
    ------------------------------------------------------------------------
    FOR nf IN (
        SELECT usuario_id, dt_pagamento, valor_total
        FROM   luccagomes.nota_fiscal
    ) LOOP

        -- 9.1) Aluguel (chance ~50%)
        IF DBMS_RANDOM.value(0,1) > 0.5 THEN
            INSERT INTO luccagomes.aluguel
                (usuario_id, midia_id, dt_inicio, dt_expira, valor)
            VALUES (
                nf.usuario_id,
```

```
                    (SELECT id FROM (SELECT id FROM luccagomes.midia
                                     ORDER BY DBMS_RANDOM.value)
                     WHERE ROWNUM = 1),
                    nf.dt_pagamento,
                    nf.dt_pagamento + NUMTODSINTERVAL(3,'DAY'),
                    ROUND(DBMS_RANDOM.value(10,50),2)
                );
            END IF;

            -- 9.2) Compra (chance ~70%)
            IF DBMS_RANDOM.value(0,1) > 0.3 THEN
                -- Escolhe aleatoriamente uma mídia
                SELECT id
                  INTO v_midia_compra
                  FROM (
                    SELECT id
                    FROM   luccagomes.midia
                    ORDER BY DBMS_RANDOM.value
                  )
                  WHERE ROWNUM = 1;

                -- Verifica se esse usuário já comprou essa mídia
                SELECT COUNT(*)
                  INTO v_count_compra
                  FROM luccagomes.compra
                  WHERE usuario_id = nf.usuario_id
                    AND midia_id   = v_midia_compra;

                -- Se ainda não comprou, faz a compra
                IF v_count_compra = 0 THEN
                   INSERT INTO luccagomes.compra
                       (usuario_id, midia_id, dt_compra, valor)
                   VALUES (
                       nf.usuario_id,
                       v_midia_compra,
                       nf.dt_pagamento,
                       ROUND(DBMS_RANDOM.value(15,80),2)
                   );
                END IF;
            END IF;

        END LOOP;

        COMMIT;
    END;
END;
/
```

```
SHOW ERRORS;


-------------------------------------------------------------------------------
-
-- Invoca a Procedure
-------------------------------------------------------------------------------
-
PROMPT ========== EXECUTANDO sp_carga_dados(p_fator) ==========

BEGIN
    -- Ajuste aqui o fator da escala 1, 5 etc
    luccagomes.sp_carga_dados(1);  -- p_fator = 1
END;
/
```

2.2. Relatório

O objetivo do script acima é:

- Criar um procedimento para carregar dados (semi)aleatórios nas tabelas, recebendo como parâmetro um fator de escala (p_fator).
- O fator de escala multiplica a quantidade de registros inseridos em cada tabela, de forma que p_fator=1 insere poucas tuplas, p_fator=2 insere o dobro etc.

O que o código faz:

- Primeiro limpa as tabelas como DELETE para não ter dados residuais de transações anteriores
- Depois é criado um loop que insere 5 * p_fator usuários, preenchendo campos como nome, login, senha, data de nascimento e status (ativo ou não).
- São inseridos três idiomas e três gêneros, capturando seus IDs gerados para posterior relacionamento com mídias.
- Do mesmo jeito é feito um loop que insere 5 * p_fator mídias, variando se a mídia é filme (campo duracao preenchido) ou série (campo temporadas preenchido). O script controla a flag de ativo (Y ou N) de forma pseudo-aleatória.
- Para cada usuário ativo, são gerados dois registros de nota_fiscal, cada um com valor total aleatório.
- Para o aluguel e compra de midias:
    - Para cada nota fiscal, há uma probabilidade de ~50% de inserir um aluguel (com dt_inicio e dt_expira, além do valor).
    - E ~70% de inserir uma compra, garantindo que o usuário não compre a mesma mídia duas vezes.
- O relacionamento entre as tabelas de midia e genero se dão da seguinte forma:
    - Para cada mídia, insere-se pelo menos um idioma (o “Português”) e possivelmente um segundo idioma de forma aleatória.
    - Para gêneros, toda mídia ganha pelo menos um gênero, e existe chance de receber um segundo.

3. Implementação de uma visão computada e uma visão materializada

3. 3.1. Script de Criação das Visões

```
PROMPT ========== CRIANDO/RECRIANDO VIEW (VISÃO COMPUTADA) ==========

-- 1) Criando (ou recriando) uma visão simples que retorna
--    somente as mídias ativas.
--    "Computada" significa que não armazena em disco as linhas;
--    ela reflete as alterações na tabela base em tempo real.

CREATE OR REPLACE VIEW luccagomes.vw_midia_ativa AS
SELECT
    id,
    titulo,
    dt_lancamento,
    valor,
    avaliacao,
    poster,
    atores
FROM
    luccagomes.midia
WHERE
    ativo = 'Y';


PROMPT ========== EXEMPLO DE USO DA VIEW (CONSULTA) ==========

-- Verificando quantas mídias ativas temos no momento
SELECT COUNT(*) AS total_midia_ativa
  FROM luccagomes.vw_midia_ativa;


PROMPT ========== FAZENDO ATUALIZACAO QUE AFETA A VIEW ==========

DECLARE
    v_id_midia NUMBER;
BEGIN
    -- Pegamos 1 mídia que esteja ativa
```

```
    SELECT id
      INTO v_id_midia
      FROM luccagomes.midia
     WHERE ativo = 'Y'
       AND ROWNUM = 1;  -- pega a primeira que encontrar

    -- Desativa essa mídia (muda de Y para N)
    UPDATE luccagomes.midia
       SET ativo = 'N'
     WHERE id = v_id_midia;

    COMMIT;

    DBMS_OUTPUT.put_line(
       'Midia ' || v_id_midia || ' foi desativada. Verifique a
vw_midia_ativa.'
    );
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.put_line('Nenhuma mídia ativa encontrada para
desativar.');
END;
/

-- Agora, se consultarmos novamente a view, essa mídia não aparecerá mais
(caso exista)
SELECT id, titulo
  FROM luccagomes.vw_midia_ativa
 WHERE ROWNUM <= 5;  -- Exibe até as 5 primeiras mídias ativas


PROMPT ========== CRIANDO/RECRIANDO MATERIALIZED VIEW ==========

-- 2) Criando uma visão materializada que resume as compras por usuário.
--    Ela agrupa por usuario_id e soma o valor total.
--    "REFRESH ON DEMAND" para atualizar manualmente via DBMS_MVIEW.REFRESH.
--    BUILD IMMEDIATE = cria e povoa a MV agora mesmo.

-- Antes de recriar, vamos garantir que a MV não exista:
BEGIN
    EXECUTE IMMEDIATE 'DROP MATERIALIZED VIEW luccagomes.mv_compras_resumo';
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE != -12003 THEN  -- ORA-12003: materialized view does not
exist
            RAISE;
        END IF;
```

```
END;
/

CREATE MATERIALIZED VIEW luccagomes.mv_compras_resumo
BUILD IMMEDIATE
REFRESH ON DEMAND
AS
SELECT
    c.usuario_id,
    COUNT(*)    AS total_compras,
    SUM(c.valor) AS valor_total
FROM
    luccagomes.compra c
GROUP BY
    c.usuario_id;


PROMPT ========== EXEMPLO DE USO DA MATERIALIZED VIEW ==========

-- Consulta a MV para ver quantas compras por usuário
SELECT * FROM luccagomes.mv_compras_resumo
ORDER BY usuario_id;


PROMPT ========== EXEMPLO DE ATUALIZACAO QUE AFETA O CONTEUDO DA MV ==========

DECLARE
    v_usuario   NUMBER;
    v_dt_pag    TIMESTAMP;
    v_midia     NUMBER;
BEGIN
    -- 1) Pegamos algum usuario_id e dt_pagamento que existam em nota_fiscal
    SELECT usuario_id, dt_pagamento
      INTO v_usuario, v_dt_pag
      FROM luccagomes.nota_fiscal
     WHERE ROWNUM = 1;

    -- 2) Pegamos alguma midia que ESTE USUÁRIO ainda NÃO comprou
    SELECT m.id
      INTO v_midia
      FROM luccagomes.midia m
     WHERE ROWNUM = 1
       AND NOT EXISTS (
           SELECT 1
             FROM luccagomes.compra c
            WHERE c.usuario_id = v_usuario
              AND c.midia_id   = m.id
```

```
        );

    -- 3) Insere uma nova compra na tabela base (compra),
    --    usando esse usuário, essa mídia e a data de pagamento acima
    INSERT INTO luccagomes.compra (usuario_id, midia_id, dt_compra, valor)
    VALUES (v_usuario, v_midia, v_dt_pag, 40);

    COMMIT;

    DBMS_OUTPUT.put_line('Inserida compra do usuario ' || v_usuario
        || ' na midia ' || v_midia
        || ' no dt_pagamento ' || v_dt_pag
        || ' com valor R$40');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.put_line(
            'Não foi possível inserir compra: não há nota_fiscal, midia ou '
||
            'todas as mídias já foram compradas por esse usuário.'
        );
END;
/

-- 4) Consulta novamente a MV (sem REFRESH) para ver se já atualizou
--    (não vai atualizar até fazermos o REFRESH, pois é "REFRESH ON DEMAND")
SELECT * FROM luccagomes.mv_compras_resumo
ORDER BY usuario_id;


PROMPT ========== REALIZANDO O REFRESH MANUAL DA MV ==========

-- forçar o refresh para a MV recalcular.
EXEC DBMS_MVIEW.REFRESH('luccagomes.mv_compras_resumo');

-- 5) Consulta após o REFRESH
SELECT * FROM luccagomes.mv_compras_resumo
ORDER BY usuario_id;
/
```

3.2. Relatório

O objetivo do script acima:

- Criar uma view que faz sentido com o schema do trabalho
- Criar uma materialized view que faz uma agregação atualizando manualmente

O que o código faz:

- View Computada:
  - Chamada ‘vw_midia_ativa’ seleciona todas as colunas relevantes da tabela `midia` onde `ativo = 'Y'`. Essa visão reflete dinamicamente as mídias que ainda estão ativas.
- View Materializada:
  - Com o nome de ‘mv_compras_resumo’ ela agrega as informações de compra por usuário, contando quantas compras foram feitas (`COUNT(*)`) e somando o valor total (`SUM(c.valor)`), agrupando por `usuario_id`.

4. Implementação de Common Table Expressions

4. 4.1. Script de Criação das CTE

```
PROMPT ========== CONSULTA NAO RECURSIVA (WITH) ==========

-- Descrição:
--   Esta consulta exemplifica o uso de CTE não recursiva para:
--     1) Calcular quantas mídias existem em cada gênero (cte_gen_count).
--     2) Calcular quantas mídias existem em cada idioma (cte_idioma_count).
--   Em seguida, fazemos um SELECT final que mostra o "top 3" gêneros e
--   o "top 3" idiomas com mais mídias cadastradas, para ilustrar
--   como podemos utilizar múltiplas CTEs na mesma consulta.

WITH
  cte_gen_count AS (
    SELECT
      g.genero,
      COUNT(gdm.midia_id) AS total_midia
    FROM
      luccagomes.genero g
      LEFT JOIN luccagomes.generos_da_midia gdm
             ON gdm.genero_id = g.id
    GROUP BY
      g.genero
  ),
  cte_idioma_count AS (
    SELECT
      i.idioma,
      COUNT(idm.midia_id) AS total_midia
    FROM
      luccagomes.idioma i
```

```
      LEFT JOIN luccagomes.idiomas_da_midia idm
             ON idm.idioma_id = i.id
    GROUP BY
      i.idioma
  )

SELECT
    'Gênero'          AS tipo,
    genero            AS nome,
    total_midia
FROM
    (
      SELECT genero, total_midia
      FROM cte_gen_count
      ORDER BY total_midia DESC
      FETCH FIRST 3 ROWS ONLY   -- top 3 gêneros
    )
UNION ALL
SELECT
    'Idioma'          AS tipo,
    idioma            AS nome,
    total_midia
FROM
    (
      SELECT idioma, total_midia
      FROM cte_idioma_count
      ORDER BY total_midia DESC
      FETCH FIRST 3 ROWS ONLY   -- top 3 idiomas
    )
ORDER BY
    tipo DESC,
    total_midia DESC;

PROMPT ========== FIM DA CONSULTA NAO RECURSIVA ==========


PROMPT
PROMPT ========== CONSULTA RECURSIVA (WITH RECURSIVE) ==========

-- Descrição:
--   Nesta consulta, é gerado uma lista de dias (datas) que compreendem
--   o intervalo mínimo e máximo de "dt_inicio" na tabela ALUGUEL.
--   Em seguida, para cada dia, mostraremos quantos aluguéis começaram
--   naquele dia.


WITH RECURSIVE cte_calendario (dia, dia_fim) AS (
```

```
    -- 1) Âncora: pega a data mínima de dt_inicio e a data máxima de dt_inicio
    SELECT
        TRUNC(MIN(a.dt_inicio)) AS dia,
        TRUNC(MAX(a.dt_inicio)) AS dia_fim
    FROM
        luccagomes.aluguel a

    UNION ALL

    -- 2) Passo recursivo: avança o dia + 1, até alcançar dia_fim
    SELECT
        dia + 1,
        dia_fim
    FROM
        cte_calendario
    WHERE
        (dia + 1) <= dia_fim
)
SELECT
    c.dia,
    COUNT(a.usuario_id) AS total_alugueis_no_dia
FROM
    cte_calendario c
    LEFT JOIN luccagomes.aluguel a
           ON TRUNC(a.dt_inicio) = c.dia
GROUP BY
    c.dia
ORDER BY
    c.dia;

PROMPT ========== FIM DA CONSULTA RECURSIVA ==========
```

4.2. Relatório

Objetivo:

- Demonstrar o uso de subconsultas nomeadas (WITH), tanto não recursivas quanto recursivas, em situações onde é importante legibilidade, reutilização de lógica ou consultas mais elaboradas.

O que o código faz:

- CTE não recursiva:
    - Calcula quantas mídias existem em cada gênero (cte_gen_count) e em cada idioma (cte_idioma_count), usando JOIN e GROUP BY.

- Em seguida, exibe o “top 3” gêneros e o “top 3” idiomas com mais mídias, unindo os resultados via UNION ALL.

- CTE Recursiva:
  - Cria um “calendário” de dias entre a data mínima e a data máxima de dt_inicio em aluguel.
  - Para cada dia gerado pela recursão, mostra a quantidade de aluguéis que começaram naquele dia (juntando com a tabela aluguel).

5. Implementação de Window Functions

5. 5.1. Script de Criação das Window Functions

```
PROMPT ========== CONSULTA 1: RANKING DE MÍDIAS MAIS COMPRADAS ==========

SELECT
    midia_id,
    total_por_midia,
    RANK() OVER (ORDER BY total_por_midia DESC) AS rank_por_valor
FROM (
    SELECT
        c.midia_id,
        SUM(c.valor) OVER (PARTITION BY c.midia_id) AS total_por_midia
    FROM
        luccagomes.compra c
)
ORDER BY
    rank_por_valor;


PROMPT ========== CONSULTA 2: DIFERENÇA ENTRE COMPRAS CONSECUTIVAS ==========

SELECT
    c.usuario_id,
    c.dt_compra,
    LAG(c.dt_compra) OVER (
        PARTITION BY c.usuario_id
        ORDER BY c.dt_compra
    ) AS compra_anterior,
```

```
    (c.dt_compra - LAG(c.dt_compra) OVER (
        PARTITION BY c.usuario_id
        ORDER BY c.dt_compra
    )) AS dias_entre_compras
FROM
    luccagomes.compra c
ORDER BY
    c.usuario_id,
    c.dt_compra;

PROMPT ========== FIM DO SCRIPT DE WINDOW FUNCTIONS ==========
```

5.2 Relatório

Objetivo:

- Apresentar consultas que façam uso de funções de janela, para análises mais avançadas dentro de partições de dados.

O que o código faz:

- Primeira consulta:
  - Faz um RANK , a ideia é rankear as compras por valor.
  - Permite ver, por exemplo, quais são as maiores compras e em qual posição cada usuário se encontra.
- Segunda consulta:
  - Usa LAG() para recuperar a compra anterior de cada usuário, calculando o intervalo de dias entre uma compra e outra (dias_entre_compras).
  - Ordena por usuário e data de compra.

6. Implementação de uma Função SQL

6. 6.1 Script de Criação da Função SQL

```
PROMPT ========== CRIAÇÃO DA FUNÇÃO: FN_TOTAL_COMPRAS ==========

CREATE OR REPLACE FUNCTION luccagomes.fn_total_compras (
    p_usuario_id  IN NUMBER
)
RETURN NUMBER
IS
    v_total NUMBER(10,2);
BEGIN
    SELECT NVL(SUM(c.valor), 0)
      INTO v_total
      FROM luccagomes.compra c
     WHERE c.usuario_id = p_usuario_id;

    RETURN v_total;
END;
/

PROMPT ========== CONSULTA QUE INVOCA A FUNÇÃO FN_TOTAL_COMPRAS ==========

SELECT
    u.id AS usuario_id,
    u.nome AS nome_usuario,
    luccagomes.fn_total_compras(u.id) AS total_gasto
FROM
    luccagomes.usuario u
ORDER BY
    total_gasto DESC;
```

Relatório

Objetivo:

- Criar uma função no schema do banco que faça sentido

O que o código faz:

- A função ‘fn_total_compras(p_usuario_id)’ soma os valores de compra (valor) de um determinado usuário, retornando 0 caso não existam registros.

- Logo após eu faço um exemplo de consulta chamando fn_total_compras para cada usuário, exibindo quanto cada um já gastou.

7. Implementação de um Trigger DML

7. 7.1 Script de Criação do Trigger DML

```
PROMPT ========== INÍCIO DOS TRIGGERS E TESTES (DML.sql) ==========

-------------------------------------------------------------------------------
-
PROMPT 1) CRIAÇÃO DO TRIGGER PARA AJUSTE AUTOMÁTICO DO VALOR DE COMPRA
-------------------------------------------------------------------------------
-
CREATE OR REPLACE TRIGGER luccagomes.trg_auto_valor_compra
BEFORE INSERT ON luccagomes.compra
FOR EACH ROW
DECLARE
    v_valor NUMBER(10,2);
BEGIN
    -- Se o valor da compra não for informado (ou for nulo), usar o valor da
mídia
    IF :NEW.valor IS NULL THEN
        SELECT m.valor
          INTO v_valor
          FROM luccagomes.midia m
         WHERE m.id = :NEW.midia_id;

        :NEW.valor := v_valor;
    END IF;
END;
/
PROMPT ... Trigger luccagomes.trg_auto_valor_compra criado com sucesso.

-------------------------------------------------------------------------------
-
PROMPT 2) CRIAÇÃO DO TRIGGER QUE IMPEDE COMPRA DE MÍDIA INATIVA
```

```
--------------------------------------------------------------------------------
-
CREATE OR REPLACE TRIGGER luccagomes.trg_check_midia_ativa_compra
BEFORE INSERT OR UPDATE ON luccagomes.compra
FOR EACH ROW
DECLARE
    v_ativo CHAR(1);
BEGIN
    SELECT m.ativo
      INTO v_ativo
      FROM luccagomes.midia m
     WHERE m.id = :NEW.midia_id;

    IF v_ativo = 'N' THEN
        RAISE_APPLICATION_ERROR(
            -20001,
            'ERRO: Não é possível comprar mídia inativa (ID=' || :NEW.midia_id
|| ').'
        );
    END IF;
END;
/
PROMPT ... Trigger luccagomes.trg_check_midia_ativa_compra criado com sucesso.

--------------------------------------------------------------------------------
-
PROMPT 3) INSERINDO DADOS BÁSICOS PARA TESTE
--------------------------------------------------------------------------------
-

PROMPT 3.1) Inserindo um usuário (ID gerado pelo IDENTITY):
-- Se 'fulano_login' ou 'fulano_login2' já existirem, mude o valor do login
novamente.
INSERT INTO luccagomes.usuario (nome, login, senha, nasc)
VALUES ('fulano37', 'fulano_login37', '1234', DATE '1990-01-01');
/

PROMPT 3.2) Inserindo uma mídia (ID gerado pelo IDENTITY):
INSERT INTO luccagomes.midia (titulo, dt_lancamento, valor, duracao)
VALUES ('Filme Ativo', DATE '2020-01-01', 50.00, 120);
/

PROMPT 3.3) Criando uma nota_fiscal para esse usuário (valor_total=0):
-- dt_pagamento = SYSTIMESTAMP por default (PRIMARY KEY = (usuario_id,
dt_pagamento))
INSERT INTO luccagomes.nota_fiscal (usuario_id, valor_total)
SELECT u.id, 0
```

```
  FROM (
       SELECT id
         FROM luccagomes.usuario
        ORDER BY id DESC
       ) u
 WHERE ROWNUM = 1;
/

PROMPT 3.4) Consultar os dados recém inseridos:
SELECT * FROM luccagomes.usuario;
SELECT * FROM luccagomes.midia;
SELECT * FROM luccagomes.nota_fiscal;

--------------------------------------------------------------------------------
-
PROMPT 4) TESTES DOS TRIGGERS
--------------------------------------------------------------------------------
-

PROMPT 4.1) Tentando comprar com mídia ativa (deve funcionar):
INSERT INTO luccagomes.compra (usuario_id, midia_id, dt_compra, valor)
SELECT
    (SELECT id
       FROM (SELECT id FROM luccagomes.usuario ORDER BY id DESC)
      WHERE ROWNUM = 1
    ) AS usuario_id,
    (SELECT id
       FROM (SELECT id FROM luccagomes.midia ORDER BY id DESC)
      WHERE ROWNUM = 1
    ) AS midia_id,
    (SELECT dt_pagamento
       FROM (SELECT dt_pagamento FROM luccagomes.nota_fiscal ORDER BY
dt_pagamento DESC)
      WHERE ROWNUM = 1
    ) AS dt_compra,
    NULL AS valor
FROM DUAL;
/

PROMPT Consulta para ver se o valor foi preenchido automaticamente:
SELECT *
  FROM luccagomes.compra
 ORDER BY dt_compra DESC;

PROMPT 4.2) Tornar a mídia inativa e tentar comprar de novo (deve falhar):
UPDATE luccagomes.midia
   SET ativo = 'N'
```

```
 WHERE id = (
   SELECT id
     FROM (SELECT id FROM luccagomes.midia ORDER BY id DESC)
    WHERE ROWNUM = 1
 );
/

PROMPT Ao tentar comprar agora, deve disparar o erro -20001:
INSERT INTO luccagomes.compra (usuario_id, midia_id, dt_compra, valor)
SELECT
    (SELECT id
       FROM (SELECT id FROM luccagomes.usuario ORDER BY id DESC)
      WHERE ROWNUM = 1
    ),
    (SELECT id
       FROM (SELECT id FROM luccagomes.midia ORDER BY id DESC)
      WHERE ROWNUM = 1
    ),
    (SELECT dt_pagamento
       FROM (SELECT dt_pagamento FROM luccagomes.nota_fiscal ORDER BY
dt_pagamento DESC)
      WHERE ROWNUM = 1
    ),
    NULL
FROM DUAL;
/

PROMPT ... Espera-se erro ORA-20001: ERRO: Não é possível comprar mídia
inativa.

PROMPT ========== FIM DOS TRIGGERS E TESTES ==========
```

7.2 Relatório

Objetivo:

- Ter ao menos um trigger que faça uma atualização automática ou manutenção de atributo derivado.
- Ter ao menos um trigger que implemente uma restrição de integridade mais complexa, lançando exceção em caso de violação.

O que o código faz:

- **trg_auto_valor_compra (BEFORE INSERT em compra)**
  - Se o valor não for informado (:NEW.valor IS NULL), o trigger busca o valor da mídia associada e preenche automaticamente.
  - É um exemplo claro de “manutenção de atributo derivado”, pois o valor de compra padrão é definido com base no valor cadastrado em midia.

- **trg_check_midia_ativa_compra (BEFORE INSERT OR UPDATE em compra)**
    - Verifica se a mídia que se está tentando comprar está marcada como ativo = 'Y'. Caso não esteja, lança uma exceção via RAISE_APPLICATION_ERROR.
    - Esta é a restrição de integridade complexa: impede a compra de mídias inativas.