

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. 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.

### **2.1. 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**

### **3.1. 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 (SEM FILTRO) ==========

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**

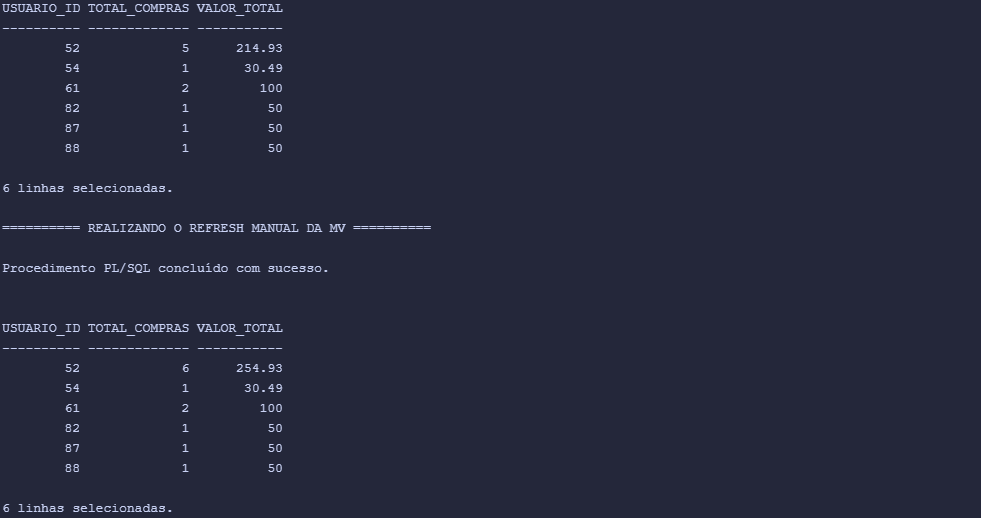
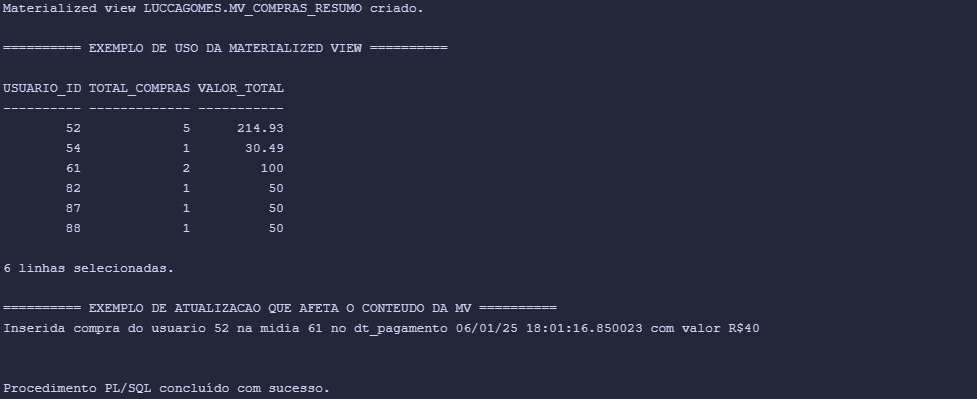
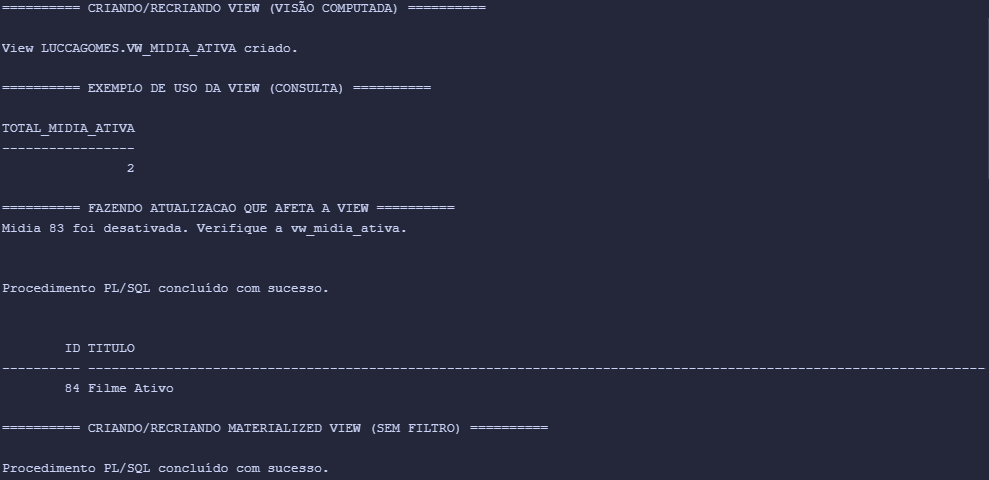
O objetivo do script acima:

* 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.

**3.3 Output.**



## **4. Implementação de Common Table Expressions**

### **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**

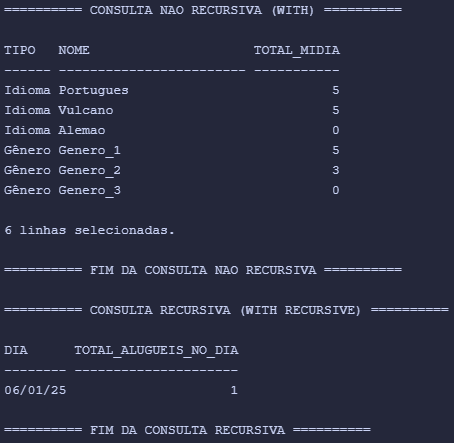
Objetivo:

* Demonstrar o uso de subconsultas nomeadas (WITH), tanto não recursivasquanto 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).

**4.3. Output.**



## **5. Implementação de Window Functions**

### **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**

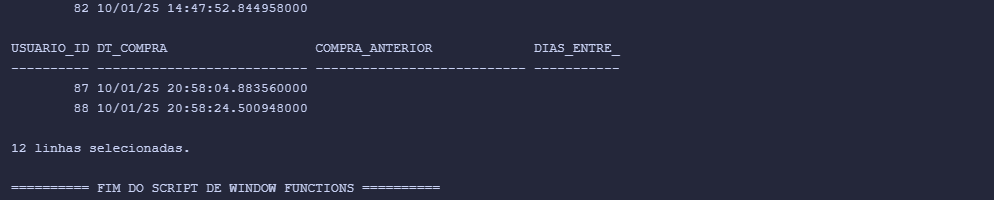
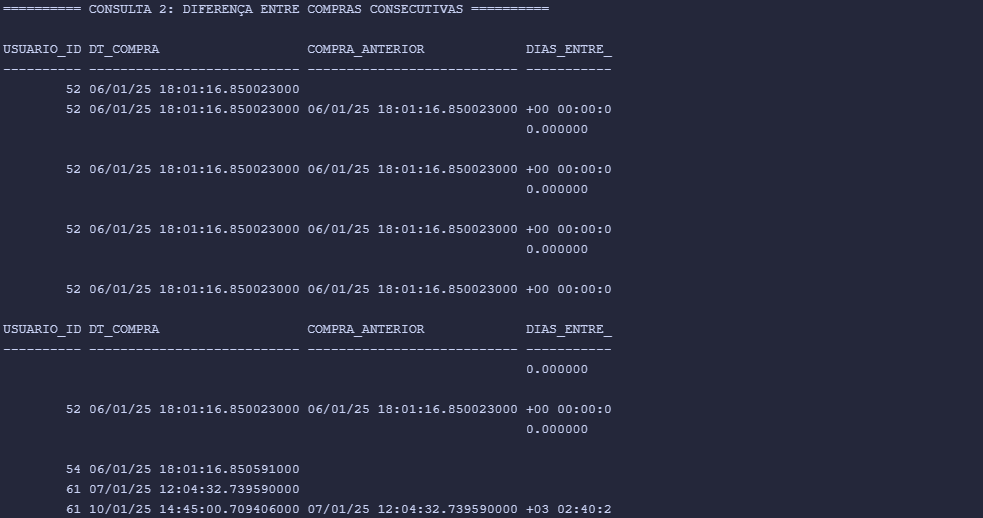
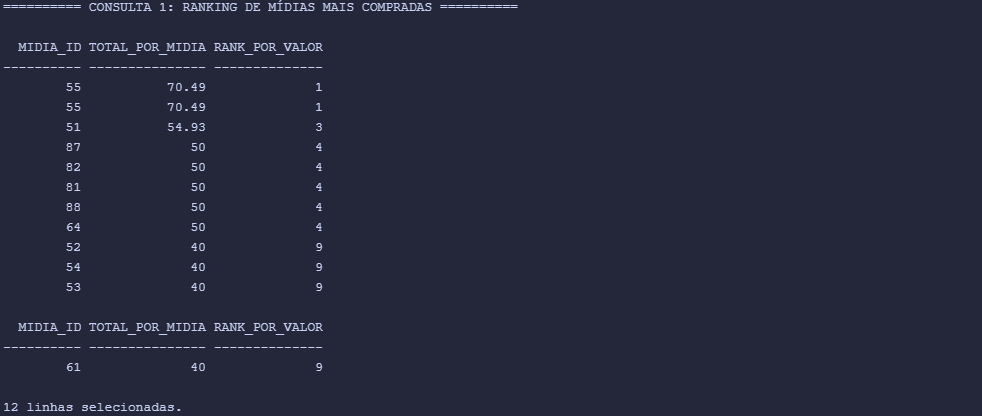
Objetivo:

* 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.

**5.3 Output.**



**6. Implementação de uma Função SQL**

### **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*;

**6.2 Relatório**

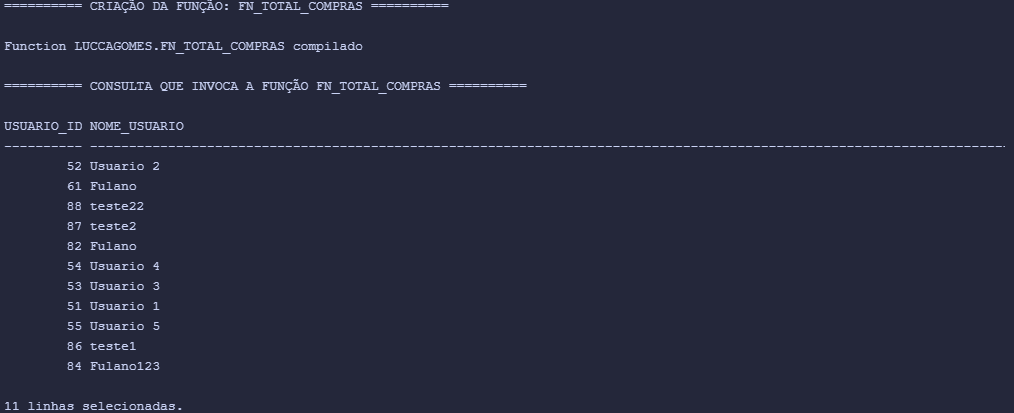
Objetivo:

* 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.

**6.3 Output**



## **7. Implementação de um Trigger DML**

### **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.

**7.3 Output.**

