**SCC0241 - Laboratório de Bases de Dados**

**Relatório de execução da aula prática 6**

**Aluno NUSP**

Giancarlo Malfate Caprino 12725025

Adriano da Silva de Carvalho 13692400

**Exercício 1**

**➔ Comentário**

Este espaço destina-se às questões discursivas; apenas algumas questões têm comentários.

**➔ Código**

SELECT \*

FROM DUAL;

**➔ Saída**

Resultados apresentados no terminal da interface SQL; exemplo: status de execução, mensagens de erro, dados (em forma tabular), e outras informações pertinentes. Pode-se enfatizar o texto com negrito ou itálico, mas não underline.

# **Observações:**

* O código será copiado deste documento para execução, portanto, finalize cada linha com um ponto e vírgula;
* Comentários também devem ser colocados junto com o código e iniciados pelos caracteres de comentário “--”;
* O código deve ser enfatizado com a fonte Courier New, de preferência com uma cor diferente de preto;
* Faça a cópia do exemplo acima para cada exercício, mantendo a aparência do relatório ao longo dos exercícios;
* Inicie cada exercício (mas não cada item) em uma página nova usando CTRL+ENTER;
* Itens que não valem nota, não precisam ser reportados;
* Itens curtos podem ser agrupados quando conveniente;
* Objetive a fácil leitura do documento.

**Exercício 1**

**Item a**

**➔ Código**

-- SESSÃO 1

SELECT \* FROM L05\_DISPUTA WHERE ID = 7;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

UPDATE L05\_DISPUTA SET VENCEDOR = 'Alemanha' WHERE ID = 7;

-- A alteração continua na Sessão 1 até que execute o COMMIT

COMMIT;

-- SESSÃO 2

SELECT \* FROM L05\_DISPUTA WHERE ID = 7;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

UPDATE L05\_DISPUTA SET VENCEDOR = 'Brasil' WHERE ID = 7;

-- A alteração continua na Sessão 2 até que execute o COMMIT

COMMIT;

-- iv. A Sessão 2 exibe o resultado do vencedor 'Brasil' que é o Antigo a Update realizado na Sessão 1.

-- vi. O Update da Sessão 2 fica em um estado 'infinito' de loading, aguardando que a transação da Sessão 1

-- seja commitada, assim que é realizado o commit, o Update da Sessão 2 é realizado.

-- x. A Sessão 2 exibe o resultado do vencedor 'Alemanha' conforme o Update da Sessão 1.

-- xii. A Sessão 2 concluíu o o Update, sem ficar em um estado 'infinito'.

**Item b**

**➔ Código**

-- SESSÃO 1

SELECT \* FROM L05\_DISPUTA WHERE ID = 7;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

UPDATE L05\_DISPUTA SET VENCEDOR = 'Alemanha' WHERE ID = 7;

-- A alteração continua na Sessão 1 até que execute o COMMIT

COMMIT;

-- SESSÃO 2

SELECT \* FROM L05\_DISPUTA WHERE ID = 7;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

UPDATE L05\_DISPUTA SET VENCEDOR = 'Brasil' WHERE ID = 7;

-- A alteração continua na Sessão 2 até que execute o COMMIT

COMMIT;

-- Os resultado em relação ao item a) são iguais em relação as consultas, a principal diferença é que com

-- 'SERIALIZABLE' o Update que está ocorrendo na Sessão 2 não é concluído e é exibida a mensagem de erro:

-- Erro a partir da linha : 5 no comando -

-- UPDATE L05\_DISPUTA SET VENCEDOR = 'Brasil' WHERE ID = 7

-- Relatório de erros -

-- ORA-08177: não é possível serializar o acesso para esta transação

**Exercício 2**

**Item a**

**➔ Código**

CREATE OR REPLACE VIEW V\_Ranking\_Medalhas\_Olimpiadas AS

WITH Vitorias AS (

-- Contagem de vitórias por país, olimpíada e modalidade

SELECT VENCEDOR AS Pais,

OLIMPIADA,

MODALIDADE,

COUNT(\*) AS Numero\_Vitorias,

RANK() OVER (PARTITION BY OLIMPIADA, MODALIDADE ORDER BY COUNT(\*) DESC, MAX(DATA\_HORA) DESC) AS Ranking

FROM L05\_DISPUTA

GROUP BY VENCEDOR, OLIMPIADA, MODALIDADE

),

Medalhistas AS (

-- Seleciona os países que ganharam ouro, prata e bronze

SELECT

OLIMPIADA,

MODALIDADE,

MAX(CASE WHEN Ranking = 1 THEN Pais END) AS Ouro,

MAX(CASE WHEN Ranking = 2 THEN Pais END) AS Prata,

MAX(CASE WHEN Ranking = 3 THEN Pais END) AS Bronze

FROM Vitorias

GROUP BY OLIMPIADA, MODALIDADE

),

Medalhas\_Totais AS (

-- Agrega o total de ouros, pratas e bronzes por país

SELECT

Ouro AS Pais,

COUNT(Ouro) AS Total\_Ouros,

0 AS Total\_Pratras,

0 AS Total\_Bronzes

FROM Medalhistas

WHERE Ouro IS NOT NULL

GROUP BY Ouro

UNION ALL

SELECT

Prata AS Pais,

0 AS Total\_Ouros,

COUNT(Prata) AS Total\_Pratras,

0 AS Total\_Bronzes

FROM Medalhistas

WHERE Prata IS NOT NULL

GROUP BY Prata

UNION ALL

SELECT

Bronze AS Pais,

0 AS Total\_Ouros,

0 AS Total\_Pratras,

COUNT(Bronze) AS Total\_Bronzes

FROM Medalhistas

WHERE Bronze IS NOT NULL

GROUP BY Bronze

)

SELECT

Pais,

SUM(Total\_Ouros) AS Total\_Ouros,

SUM(Total\_Pratras) AS Total\_Pratras,

SUM(Total\_Bronzes) AS Total\_Bronzes,

RANK() OVER (ORDER BY SUM(Total\_Ouros) DESC, SUM(Total\_Pratras) DESC, SUM(Total\_Bronzes) DESC) AS Ranking\_Geral

FROM Medalhas\_Totais

GROUP BY Pais;

SELECT \* FROM V\_Ranking\_Medalhas\_Olimpiadas;

**Item b**

**➔ Código**

CREATE OR REPLACE VIEW V\_Ranking\_Medalhas\_Rio2016 AS

WITH Vitorias AS (

-- Contagem de vitórias por país, para a olimpíada de 2016 e modalidade

SELECT VENCEDOR AS Pais,

OLIMPIADA,

MODALIDADE,

COUNT(\*) AS Numero\_Vitorias,

RANK() OVER (PARTITION BY MODALIDADE ORDER BY COUNT(\*) DESC, MAX(DATA\_HORA) DESC) AS Ranking

FROM L05\_DISPUTA

WHERE OLIMPIADA = 2016

GROUP BY VENCEDOR, OLIMPIADA, MODALIDADE

),

Medalhistas AS (

-- Seleciona os países que ganharam ouro, prata e bronze para a olimpíada de 2016

SELECT

MODALIDADE,

MAX(CASE WHEN Ranking = 1 THEN Pais END) AS Ouro,

MAX(CASE WHEN Ranking = 2 THEN Pais END) AS Prata,

MAX(CASE WHEN Ranking = 3 THEN Pais END) AS Bronze

FROM Vitorias

GROUP BY MODALIDADE

),

Medalhas\_Totais AS (

-- Agrega o total de ouros, pratas e bronzes por país

SELECT

Ouro AS Pais,

COUNT(Ouro) AS Total\_Ouros,

0 AS Total\_Pratras,

0 AS Total\_Bronzes

FROM Medalhistas

WHERE Ouro IS NOT NULL

GROUP BY Ouro

UNION ALL

SELECT

Prata AS Pais,

0 AS Total\_Ouros,

COUNT(Prata) AS Total\_Pratras,

0 AS Total\_Bronzes

FROM Medalhistas

WHERE Prata IS NOT NULL

GROUP BY Prata

UNION ALL

SELECT

Bronze AS Pais,

0 AS Total\_Ouros,

0 AS Total\_Pratras,

COUNT(Bronze) AS Total\_Bronzes

FROM Medalhistas

WHERE Bronze IS NOT NULL

GROUP BY Bronze

)

SELECT

Pais,

SUM(Total\_Ouros) AS Total\_Ouros,

SUM(Total\_Pratras) AS Total\_Pratras,

SUM(Total\_Bronzes) AS Total\_Bronzes,

RANK() OVER (ORDER BY SUM(Total\_Ouros) DESC, SUM(Total\_Pratras) DESC, SUM(Total\_Bronzes) DESC) AS Ranking\_Geral

FROM Medalhas\_Totais

GROUP BY Pais;

SELECT \* FROM V\_Ranking\_Medalhas\_Rio2016;

**Item c**

**➔ Código**

CREATE OR REPLACE VIEW V\_Contratos\_Patrocinio AS

SELECT

p.NOME AS Nome\_Patrocinador,

a.NOME AS Nome\_Atleta,

a.PAIS AS Pais\_Atleta,

pa.INICIO AS Inicio\_Contrato,

ADD\_MONTHS(pa.INICIO, pa.VIGENCIA) AS Termino\_Contrato, -- Calcula a data final do contrato em meses

pa.VIGENCIA AS Tempo\_Vigencia

FROM

L09\_PATROCINA pa

JOIN

L08\_PATROCINADOR p ON pa.PATROCINADOR = p.ID

JOIN

L06\_ATLETA a ON pa.ATLETA = a.PASSAPORTE;

SELECT \* FROM V\_Contratos\_Patrocinio;

**Item d**

**➔ Código**

CREATE OR REPLACE VIEW V\_Disputas\_Atleta AS

SELECT

a.NOME AS Nome\_Atleta,

m.ESPORTE AS Esporte,

a.GENERO AS Genero,

COUNT(j.DISPUTA) AS Total\_Disputas

FROM

L06\_ATLETA a

JOIN

L07\_JOGA j ON a.PASSAPORTE = j.ATLETA

JOIN

L03\_MODALIDADE m ON a.MODALIDADE = m.ID

GROUP BY

a.NOME, m.ESPORTE, a.GENERO

ORDER BY Total\_Disputas DESC;

SELECT \* FROM V\_Disputas\_Atleta;

**Item e**

**➔ Código**

CREATE OR REPLACE VIEW V\_Ranking\_Atletas AS

WITH Atletas\_Disputas AS (

SELECT

a.NOME AS Nome\_Atleta,

m.ESPORTE AS Esporte,

a.GENERO AS Genero,

COUNT(j.DISPUTA) AS Total\_Disputas

FROM

L06\_ATLETA a

JOIN

L07\_JOGA j ON a.PASSAPORTE = j.ATLETA

JOIN

L03\_MODALIDADE m ON a.MODALIDADE = m.ID

GROUP BY

a.NOME, m.ESPORTE, a.GENERO

)

SELECT

Nome\_Atleta,

Esporte,

Genero,

Total\_Disputas,

RANK() OVER (ORDER BY Total\_Disputas DESC) AS Ranking

FROM

Atletas\_Disputas;

SELECT \* FROM V\_Ranking\_Atletas;

**Exercício 3**

**Item a**

**➔ Comentário**

O programa Python apresentará um erro ao tentar executar a consulta SQL, pois a coluna PASSAPORTE não existe mais.

**Item b**

**➔ Código**

-- Criar uma VIEW da tabela Atletas, para que caso alguma coluna seja alterada,

-- também deverá ser alterada a VIEW passando o nome da antiga coluna.

-- Criando a View no banco de dados caso 'PASSAPORTE' seja trocado para 'DOC\_IDENTIDADE'

CREATE OR REPLACE VIEW V\_ATLETA AS

SELECT

DOC\_IDENTIDADE AS PASSAPORTE, -- Mapeia o novo nome da coluna para o nome antigo

NOME,

PAIS

FROM

L06\_ATLETA;

-- Alteração do código Python para funcionar em caso de renomear colunas futuras:

import cx\_Oracle

con = cx\_Oracle.connect("username/password@localhost/ORCL1")

cur = con.cursor()

cur.execute("SELECT \* FROM V\_ATLETA")

for row in cur:

passaporte, nome, pais = row

print(f"Passaporte: {passaporte}, Nome: {nome}, País: {pais}")

cur.close()

con.close()

**Exercício 4**

**Item a**

**➔ Comentário**

Usuários utilizados: L13692400, Teste16 e Teste17.

**➔ Código**

-- Usuário L13692400 (Adriano Carvalho)

GRANT SELECT ON L01\_PAIS TO Teste16 WITH GRANT OPTION;

GRANT SELECT ON L02\_OLIMPIADA TO Teste16 WITH GRANT OPTION;

-- Usuário Teste16

CREATE OR REPLACE VIEW V\_USER1 AS

SELECT

p.CONTINENTE,

p.NOME,

o.ANO,

o.CIDADE\_SEDE,

o.DATA\_INICIO,

o.DATA\_ENCERRAMENTO

FROM L13692400.L01\_PAIS p

RIGHT JOIN L13692400.L02\_OLIMPIADA o

ON p.NOME = o.PAIS

ORDER BY o.ANO DESC;

GRANT SELECT ON V\_USER1 TO Teste17;

-- Usuário Teste17

SELECT \* FROM Teste16.V\_USER1;

**Item b**

**➔ Comentário**

Usuários utilizados: L13692400 e Teste16

**➔ Código**

-- Usuário L13692400 (Adriano Carvalho)

CREATE TABLE L12\_MAT\_ESPORTIVO (

ID INTEGER PRIMARY KEY,

DESCRICAO VARCHAR2(100),

PRECO NUMBER(10, 2),

ATLETA VARCHAR2(10) REFERENCES L06\_ATLETA(PASSAPORTE)

);

GRANT REFERENCES ON L13692400.L12\_MAT\_ESPORTIVO TO Teste16;

INSERT INTO L12\_MAT\_ESPORTIVO (ID, DESCRICAO, PRECO, ATLETA)

VALUES (1, 'Tênis Nike', 299.00, 'BR12345');

INSERT INTO L12\_MAT\_ESPORTIVO (ID, DESCRICAO, PRECO, ATLETA)

VALUES (2, 'Canhoteira Nike', 59.00, 'BR98765');

INSERT INTO Teste16.L13\_ESTOQUE (ID, QTD\_ESTOQUE)

VALUES (1, 15);

INSERT INTO Teste16.L13\_ESTOQUE (ID, QTD\_ESTOQUE)

VALUES (2, 30);

-- Usuário Teste16

CREATE TABLE L13\_ESTOQUE (

ID INTEGER PRIMARY KEY,

QTD\_ESTOQUE INTEGER NOT NULL,

CONSTRAINT FK\_ESTOQUE\_MAT FOREIGN KEY (ID) REFERENCES L13692400.L12\_MAT\_ESPORTIVO(ID)

);

GRANT INSERT ON L13\_ESTOQUE TO L13692400;

**Item c**

**➔ Comentário**

Usuários utilizados: L13692400 e Teste16

**➔ Código**

-- Usuário L13692400 (Adriano Carvalho)

CREATE TABLE L14\_VENDAS (

ID INTEGER PRIMARY KEY,

ITEM INTEGER,

QTD INTEGER NOT NULL,

VALOR\_TOTAL NUMBER(10, 2),

CONSTRAINT FK\_ITEM\_VENDA FOREIGN KEY (ITEM) REFERENCES L12\_MAT\_ESPORTIVO(ID)

);

CREATE OR REPLACE TRIGGER trg\_atualiza\_estoque\_calcula\_valor\_total

BEFORE INSERT ON L14\_VENDAS

FOR EACH ROW

DECLARE

v\_preco\_unitario NUMBER(10, 2);

v\_estoque\_atual INTEGER;

BEGIN

-- Buscar o estoque atual do item na tabela L13\_ESTOQUE

SELECT QTD\_ESTOQUE INTO v\_estoque\_atual

FROM Teste16.L13\_ESTOQUE

WHERE ID = :NEW.ITEM;

-- Verificar se a quantidade a ser vendida é maior que a quantidade em estoque

IF :NEW.QTD > v\_estoque\_atual THEN

-- Lança um erro caso a quantidade vendida seja maior que o estoque disponível

RAISE\_APPLICATION\_ERROR(-20001, 'Estoque insuficiente para este item.');

ELSE

-- Atualiza o estoque do item na tabela L13\_ESTOQUE

UPDATE Teste16.L13\_ESTOQUE

SET QTD\_ESTOQUE = QTD\_ESTOQUE - :NEW.QTD

WHERE ID = :NEW.ITEM;

END IF;

-- Buscar o preço do item na tabela L12\_MAT\_ESPORTIVO

SELECT PRECO INTO v\_preco\_unitario

FROM L12\_MAT\_ESPORTIVO

WHERE ID = :NEW.ITEM;

-- Calcular o valor total da venda

:NEW.VALOR\_TOTAL := v\_preco\_unitario \* :NEW.QTD;

END;

-- Verificar estoque ANTES de inserir

SELECT \* FROM Teste16.L13\_ESTOQUE;

INSERT INTO L14\_VENDAS (ID, ITEM, QTD)

VALUES (5, 1, 491);

INSERT INTO L14\_VENDAS (ID, ITEM, QTD)

VALUES (4, 2, 10);

-- Verificar estoque APÓS de inserir

SELECT \* FROM Teste16.L13\_ESTOQUE;

SELECT \* FROM L14\_VENDAS;