Encontro 4 – Triggers e Tratamento de Exceções (4h)

Objetivos

- Entender e criar *triggers* (gatilhos) em PL/SQL.
- Diferenciar tipos de *triggers* (row-level, statement-level, before, after, instead of).
- Aplicar tratamento de exceções em blocos PL/SQL.
- Criar exceções personalizadas com `RAISE_APPLICATION_ERROR`.

1. Introdução às Triggers

Uma **trigger** é um bloco PL/SQL que é **automaticamente executado** em resposta a um evento em uma tabela (como `INSERT`, `UPDATE` ou `DELETE`).

Sintaxe básica

```
CREATE [OR REPLACE] TRIGGER nome_trigger
[BEFORE | AFTER | INSTEAD OF]
[INSERT | UPDATE | DELETE]
ON nome_tabela
[FOR EACH ROW]
BEGIN
  -- corpo da trigger
END;
```

2. Tipos de Triggers

Tipo	Descrição	Nível
BEFORE	Executa **antes** do evento.	Row-level ou Statement-level
AFTER	Executa **depois** do evento.	Row-level ou Statement-level
INSTEAD OF	Substitui a ação (usado em *views*).	Statement-level

FOR EACH ROW	Executa **uma vez para cada linha afetada**.	Row-level
(sem FOR EACH ROW)	Executa **uma vez por comando DML**.	Statement-level

---

3. Exemplo prático – Trigger de auditoria

Tabelas base

```
CREATE TABLE empregados (
  id NUMBER PRIMARY KEY,
  nome VARCHAR2(50),
  cargo VARCHAR2(30),
  salario NUMBER(10,2)
);

CREATE TABLE log_emp (
  id_log NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  id_emp NUMBER,
  acao VARCHAR2(20),
  data_acao DATE
);
```

Trigger de log

```
CREATE OR REPLACE TRIGGER trg_log_emp
AFTER INSERT ON empregados
FOR EACH ROW
BEGIN
  INSERT INTO log_emp (id_emp, acao, data_acao)
  VALUES (:NEW.id, 'INSERIDO', SYSDATE);
END;
/
```

Explicação:

- `:NEW` → representa o valor **novo** do registro.

- :OLD → representa o valor **antigo** (antes da alteração).

4. Trigger BEFORE UPDATE – Auditoria de alterações

```
CREATE OR REPLACE TRIGGER trg_audita_update
BEFORE UPDATE ON empregados
FOR EACH ROW
BEGIN
  INSERT INTO log_emp (id_emp, acao, data_acao)
  VALUES (:OLD.id, 'ATUALIZADO', SYSDATE);
END;
/
```

5. Trigger que impede exclusão de gerente

```
CREATE OR REPLACE TRIGGER trg_no_delete_gerente
BEFORE DELETE ON empregados
FOR EACH ROW
BEGIN
  IF :OLD.cargo = 'GERENTE' THEN
    RAISE_APPLICATION_ERROR(-20001, 'Não é permitido excluir um
gerente!');
  END IF;
END;
/
```

6. Tratamento de Exceções em PL/SQL

Estrutura

```
BEGIN
  -- código principal
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Nenhum registro encontrado.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Mais de um registro retornado.');
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Erro: divisão por zero.');
  WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Erro inesperado: ' || SQLERRM);
END;
/
```

7. Exceções Personalizadas

Você pode criar mensagens personalizadas usando:

```
RAISE_APPLICATION_ERROR(-20000, 'Mensagem personalizada');
```

Exemplo:

```
BEGIN
  IF 100 / 0 = 0 THEN
    NULL;
  END IF;
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    RAISE_APPLICATION_ERROR(-20010, 'Operação inválida: divisão por
zero!');
END;
/
```

8. Exemplo completo – Trigger com exceção personalizada

```
CREATE OR REPLACE TRIGGER trg_valida_salario
BEFORE INSERT OR UPDATE ON empregados
FOR EACH ROW
BEGIN
  IF :NEW.salario < 1500 THEN
    RAISE_APPLICATION_ERROR(-20011, 'Salário abaixo do permitido!');
  END IF;
END;
/
```

Atividades Práticas

Exercício 1 – Trigger de auditoria

Crie uma **trigger** que registre em uma tabela `log_salarios` toda vez que um salário for alterado, armazenando:

- `id_emp`
- `salario_antigo`
- `salario_novo`
- `data_alteracao`

💡 *Dica:* use `:OLD.salario` e `:NEW.salario`.

Exercício 2 – Impedir exclusão de gerentes

Crie uma trigger que **bloqueie exclusões** de empregados com cargo `'GERENTE'`.

💡 *Dica:* utilize `BEFORE DELETE` e `RAISE_APPLICATION_ERROR`.

Exercício 3 – Tratamento de exceções

Crie um bloco PL/SQL que:

1. Busque um funcionário por ID.
2. Se não encontrar, exiba uma mensagem “Empregado não encontrado”.
3. Caso retorne mais de um, exiba “Mais de um resultado encontrado”.

Exercício 4 – Exceção personalizada

Crie um bloco PL/SQL que:

- Solicite um número.

- Se for menor que zero, gere uma exceção personalizada `RAISE_APPLICATION_ERROR(-20020, 'Número negativo não permitido.')`