FACULDADE DE INFORMATICA E ADMINISTRAÇÃO PAULISTA ANÁLISE E DESENVOLVIMENTO DE SISTEMA

PEDRO HENRIQUE DOS SANTOS – RM:559064

THIAGO THOMAZ – RM:557992

VINÍCIUS DE OLIVEIRA COUTINHO - RM556182

MOTTUGRID

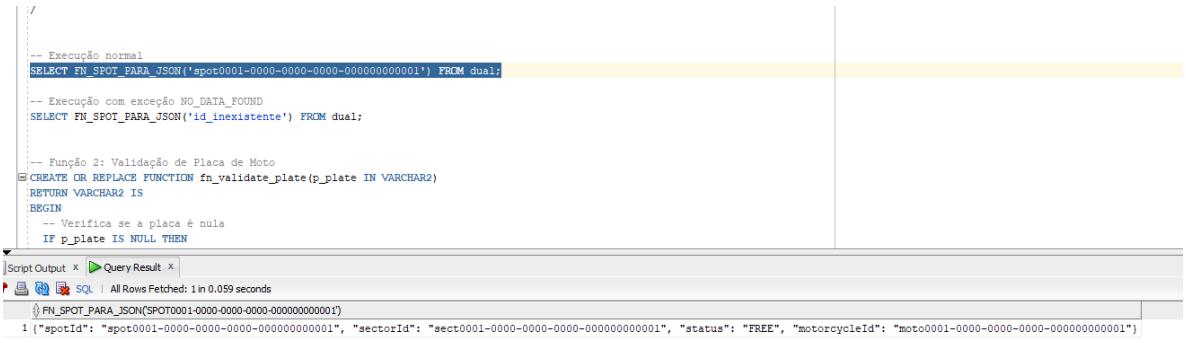
DATABASE

SÃO PAULO 2025

Function 1

```
-- Função 1: Converter Dados de uma Vaga (Spot) em JSON
CREATE OR REPLACE FUNCTION FN_SPOT_PARA_JSON (
    p_spot_id IN CHAR
) RETURN VARCHAR2 IS
    v_spot_id      SPOTS.SPOT_ID%TYPE;
    v_sector_id    SPOTS.SECTOR_ID%TYPE;
    v_status       SPOTS.STATUS%TYPE;
    v_motorcycle_id SPOTS.MOTORCYCLE_ID%TYPE;
    v_json         VARCHAR2(4000);
BEGIN
    -- Busca os dados relacionais
    SELECT SPOT_ID, SECTOR_ID, STATUS, NVL(MOTORCYCLE_ID, 'NULL')
    INTO v_spot_id, v_sector_id, v_status, v_motorcycle_id
    FROM SPOTS
    WHERE SPOT_ID = p_spot_id;

    -- Monta o JSON manualmente
    v_json := '{' ||
                '"spotId": "' || v_spot_id || '", ' ||
                '"sectorId": "' || v_sector_id || '", ' ||
                '"status": "' || v_status || '", ' ||
                '"motorcycleId": "' || v_motorcycle_id || '"' ||
              '}';

    RETURN v_json;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN '{"erro": "Vaga não encontrada."}';
    WHEN TOO_MANY_ROWS THEN
        RETURN '{"erro": "Mais de uma vaga encontrada com o mesmo ID."}';
    WHEN OTHERS THEN
        RETURN '{"erro": "Erro inesperado: ' || SQLERRM || '"}';
END;
/
```

Execução sem retornar exception



Execução com exception

```
-- Execução com exceção NO_DATA_FOUND
SELECT FN_SPOT_PARA_JSON('id_inexistente') FROM dual;


-- Função 2: Validação de Placa de Moto
CREATE OR REPLACE FUNCTION fn_validate_plate(p_plate IN VARCHAR2)
RETURN VARCHAR2 IS
BEGIN
  -- Verifica se a placa é nula
  IF p_plate IS NULL THEN
```

Query Result

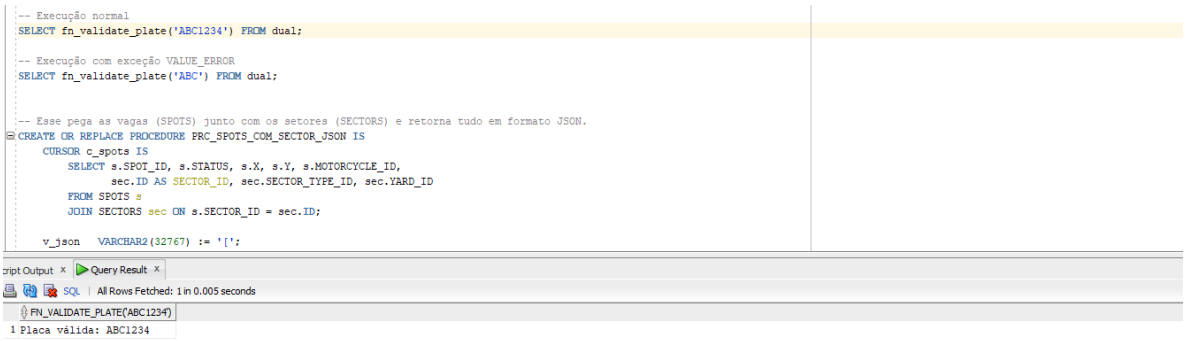
SQL | All Rows Fetched: 1 in 0.007 seconds

	FN_SPOT_PARA_JSON('ID_INEXISTENTE')
1	{"erro": "Vaga não encontrada."}

Function 2

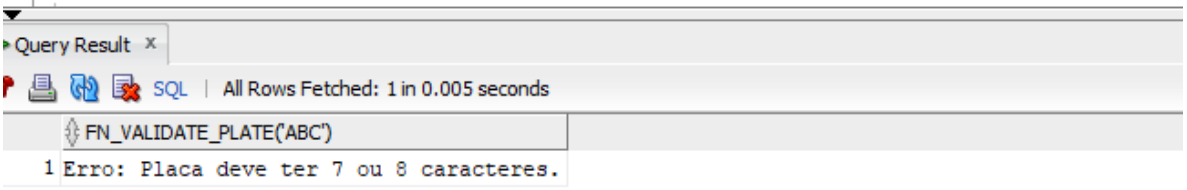
```
-- Função 2: Validação de Placa de Moto
CREATE OR REPLACE FUNCTION fn_validate_plate(p_plate IN VARCHAR2)
RETURN VARCHAR2 IS
BEGIN
  -- Verifica se a placa é nula
  IF p_plate IS NULL THEN
    RAISE NO_DATA_FOUND; -- força uma exception específica
  END IF;

  -- Verifica tamanho inválido
  IF LENGTH(p_plate) != 7 AND LENGTH(p_plate) != 8 THEN
    RAISE VALUE_ERROR; -- força a exception
  END IF;

  -- Verifica se corresponde ao padrão AAA0A00 ou AAA0000
  IF REGEXP_LIKE(p_plate, '^[A-Z]{3}[0-9][A-Z0-9][0-9]{2}$')
     OR REGEXP_LIKE(p_plate, '^[A-Z]{3}[0-9]{4}$') THEN
    RETURN 'Placa válida: ' || p_plate;
  ELSE
    RETURN 'Placa inválida: formato não corresponde.';
  END IF;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 'Erro: A placa não pode ser nula.';
  WHEN VALUE_ERROR THEN
    RETURN 'Erro: Placa deve ter 7 ou 8 caracteres.';
  WHEN OTHERS THEN
    RETURN 'Erro inesperado: ' || SQLERRM;
END;
/
```

Execução sem retornar exception



Execução com exception

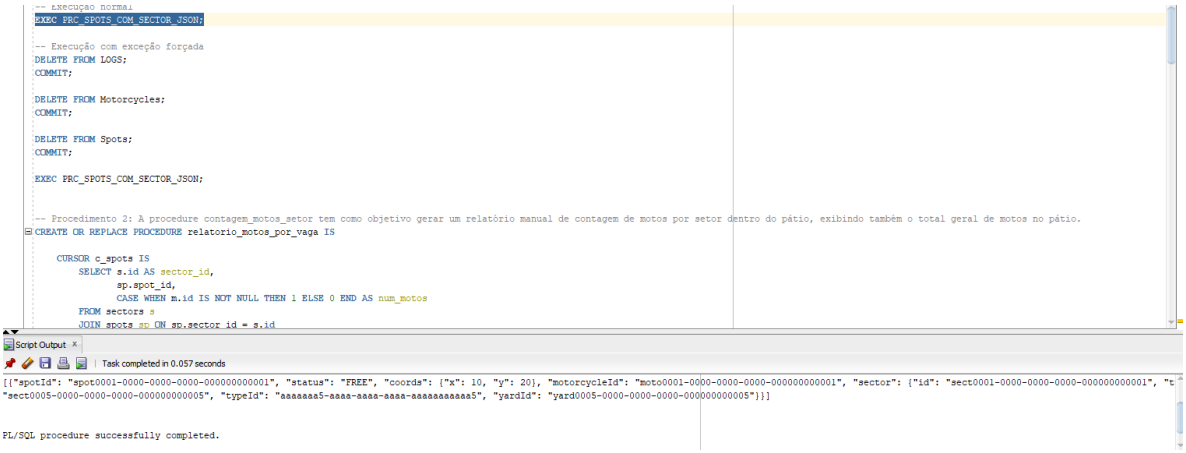
```
-- Execução com exceção VALUE_ERROR
SELECT fn_validate_plate('ABC') FROM dual;


-- Esse pega as vagas (SPOTS) junto com os setores (SECTORS) e retorna tudo em formato J
CREATE OR REPLACE PROCEDURE PRC_SPOTS_COM_SECTOR_JSON IS
    CURSOR c_spots IS
```



Procedure 1

```
-- Esse pega as vagas (SPOTS) junto com os setores (SECTORS) e retorna tudo em formato JSON.
CREATE OR REPLACE PROCEDURE PRC_SPOTS_COM_SECTOR_JSON IS
    CURSOR c_spots IS
        SELECT s.SPOT_ID, s.STATUS, s.X, s.Y, s.MOTORCYCLE_ID,
               sec.ID AS SECTOR_ID, sec.SECTOR_TYPE_ID, sec.YARD_ID
        FROM SPOTS s
        JOIN SECTORS sec ON s.SECTOR_ID = sec.ID;

    v_json   VARCHAR2(32767) := '[';
    v_count  NUMBER := 0;
BEGIN
    FOR rec IN c_spots LOOP
        v_count := v_count + 1;

        v_json := v_json ||
            CASE WHEN v_count > 1 THEN ',' ELSE '' END ||
            '{' ||
              '"spotId": "' || rec.SPOT_ID || '", ' ||
              '"status": "' || rec.STATUS || '", ' ||
              '"coords": {"x": ' || rec.X || ', "y": ' || rec.Y || '}, ' ||
              '"motorcycleId": "' || NVL(TO_CHAR(rec.MOTORCYCLE_ID), 'NULL') || '", ' ||
              '"sector": {' ||
                '"id": "' || rec.SECTOR_ID || '", ' ||
                '"typeId": "' || rec.SECTOR_TYPE_ID || '", ' ||
                '"yardId": "' || rec.YARD_ID || '"' ||
              '}' ||
            '}';
    END LOOP;

    v_json := v_json || ']';

    -- Força NO_DATA_FOUND se não houver registros
    IF v_count = 0 THEN
        RAISE NO_DATA_FOUND;
    END IF;

    DBMS_OUTPUT.PUT_LINE(v_json);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('{"erro": "Nenhum dado encontrado"}');
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('{"erro": "Erro de conversão de valor"}');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('{"erro": "Erro inesperado: ' || SQLERRM || '"}');
END;
/
```

Execução sem retornar exception



Execução com exception

```
-- Execução com exceção forçada
DELETE FROM LOGS;
COMMIT;

DELETE FROM Motorcycles;
COMMIT;

DELETE FROM Spots;
COMMIT;

EXEC PRC_SPOTS_COM_SECTOR_JSON;


-- Procedimento 2: A procedure contagem_motos_setor tem como objetivo gerar um relatório manual de contagem
CREATE OR REPLACE PROCEDURE relatorio_motos_por_vaga IS

    CURSOR c_spots IS
        SELECT s.id AS sector_id,
               sp.spot id,
```

Query Result x | Script Output x

Task completed in 0.05 seconds

```
Commit complete.


7 rows deleted.


Commit complete.


6 rows deleted.


Commit complete.

{"erro": "Nenhum dado encontrado"}
```

Procedure 2

```
-- Procedimento 2: A procedure contagem_motos_setor tem como objetivo gerar um relatório manual de contagem de motos por setor dentro do pátio, exibindo também o total geral de motos no pátio.
CREATE OR REPLACE PROCEDURE relatorio_motos_por_vaga IS

    CURSOR c_spots IS
        SELECT s.id AS sector_id,
               sp.spot_id,
               CASE WHEN m.id IS NOT NULL THEN 1 ELSE 0 END AS num_motos
        FROM sectors s
        JOIN spots sp ON sp.sector_id = s.id
        LEFT JOIN Motorcycles m ON m.spotid = sp.spot_id
        ORDER BY s.id, sp.spot_id;

    v_sector         sectors.id%TYPE;
    v_spot           spots.spot_id%TYPE;
    v_num_motos      NUMBER;

    v_sector_atual   sectors.id%TYPE := NULL;
    v_subtotal       NUMBER := 0;
    v_total_patio    NUMBER := 0;
    v_count          NUMBER := 0; -- contador de registros

BEGIN
    DBMS_OUTPUT.PUT_LINE('SECTOR | SPOT | NUM_MOTOS');

    FOR rec IN c_spots LOOP
        v_count := v_count + 1;

        BEGIN
            v_sector := rec.sector_id;
            v_spot   := rec.spot_id;
            v_num_motos := rec.num_motos;

            -- Se mudou de setor, imprime subtotal do setor anterior
            IF v_sector_atual IS NOT NULL AND v_sector_atual != v_sector THEN
                DBMS_OUTPUT.PUT_LINE(v_sector_atual || ' | SUBTOTAL | ' || v_subtotal);
                v_subtotal := 0; -- reseta subtotal
            END IF;

            -- Exibe linha detalhada por vaga
            DBMS_OUTPUT.PUT_LINE(v_sector || ' | ' || v_spot || ' | ' || v_num_motos);

            -- Acumula subtotal e total geral
            v_subtotal := v_subtotal + v_num_motos;
            v_total_patio := v_total_patio + v_num_motos;

            -- Atualiza setor atual
            v_sector_atual := v_sector;
```

```
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Erro: registro de vaga ou setor não encontrado');
            WHEN VALUE_ERROR THEN
                DBMS_OUTPUT.PUT_LINE('Erro: valor inválido encontrado em num_motos');
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Erro desconhecido: ' || SQLERRM);
        END;
    END LOOP;

    -- Se não encontrou nenhum registro, lança NO_DATA_FOUND
    IF v_count = 0 THEN
        RAISE NO_DATA_FOUND;
    END IF;

    -- Imprime subtotal do último setor
    IF v_sector_atual IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE(v_sector_atual || ' | SUBTOTAL | ' || v_subtotal);
    END IF;

    -- Imprime total geral do pátio
    DBMS_OUTPUT.PUT_LINE('TOTAL PATIO ' || v_total_patio);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Erro geral: não foram encontrados registros');
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('Erro geral: valor inválido durante a execução');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erro geral desconhecido: ' || SQLERRM);

END relatorio_motos_por_vaga;
/
```

Execução sem retornar exception

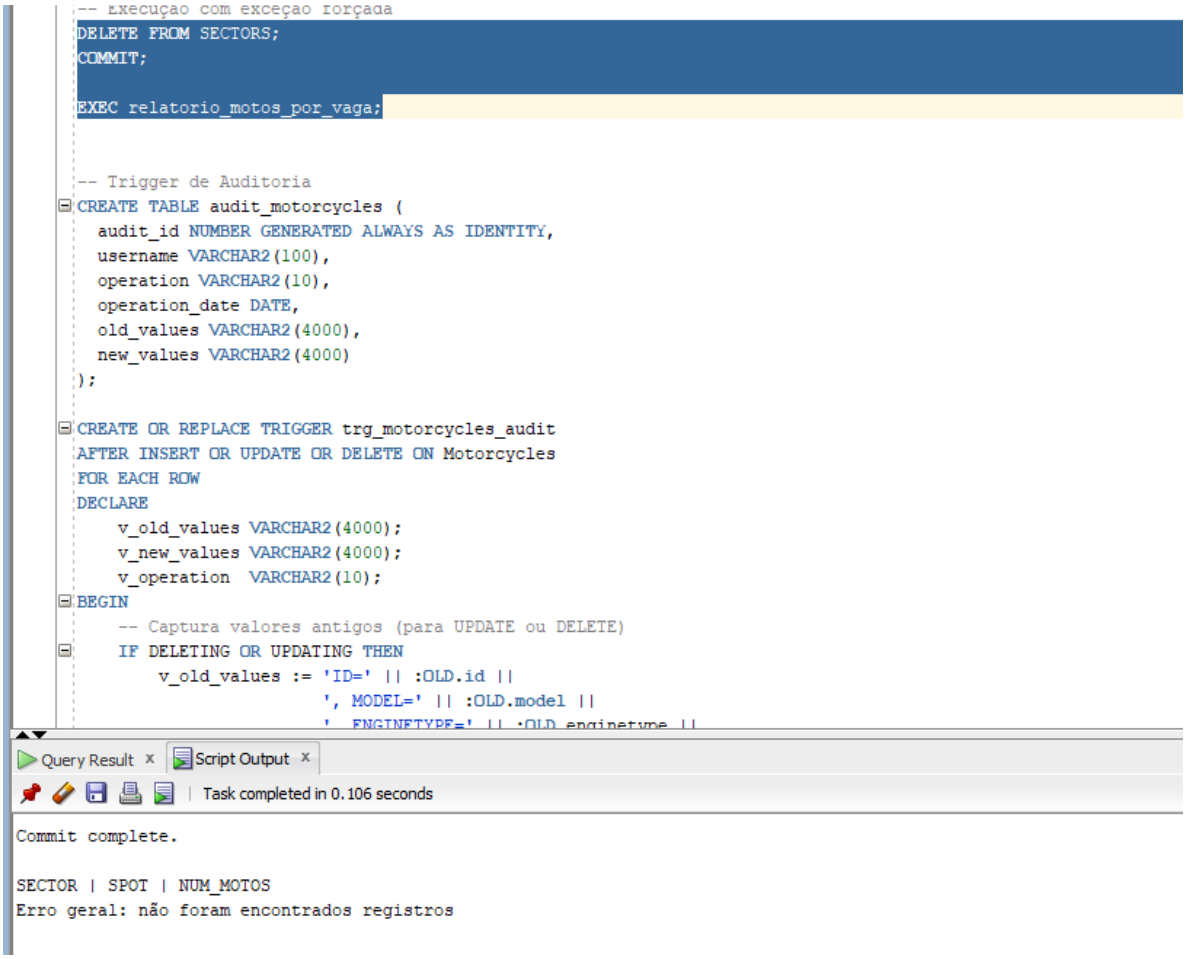
```
-- Execução normal
EXEC relatorio_motos_por_vaga;

-- Execução com exceção forçada
DELETE FROM SECTORS;
COMMIT;

EXEC relatorio_motos_por_vaga;


-- Trigger de Auditoria
CREATE TABLE audit_motorcycles (
  audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
  username VARCHAR2(100),
  operation VARCHAR2(10),
  operation_date DATE,
  old_values VARCHAR2(4000),
  new_values VARCHAR2(4000)
```

Script Output

Task completed in 0.036 seconds

```
PL/SQL procedure successfully completed.

SECTOR | SPOT | NUM_MOTOS
sect0001-0000-0000-0000-000000000001 | spot0001-0000-0000-0000-000000000001 | 1
sect0001-0000-0000-0000-000000000001 | spot0001-0000-0000-0000-000000000002 | 1
sect0001-0000-0000-0000-000000000001 | SUBTOTAL | 2
sect0002-0000-0000-0000-000000000002 | spot0002-0000-0000-0000-000000000002 | 1
sect0002-0000-0000-0000-000000000002 | SUBTOTAL | 1
sect0003-0000-0000-0000-000000000003 | spot0003-0000-0000-0000-000000000003 | 1
sect0003-0000-0000-0000-000000000003 | SUBTOTAL | 1
sect0004-0000-0000-0000-000000000004 | spot0004-0000-0000-0000-000000000004 | 1
sect0004-0000-0000-0000-000000000004 | SUBTOTAL | 1
sect0005-0000-0000-0000-000000000005 | spot0005-0000-0000-0000-000000000005 | 1
sect0005-0000-0000-0000-000000000005 | SUBTOTAL | 1
TOTAL PATIO 6
```

Execução com exception



```
-- Execução com exceção forçada
DELETE FROM SECTORS;
COMMIT;

EXEC relatorio_motos_por_vaga;


-- Trigger de Auditoria
CREATE TABLE audit_motorcycles (
  audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
  username VARCHAR2(100),
  operation VARCHAR2(10),
  operation_date DATE,
  old_values VARCHAR2(4000),
  new_values VARCHAR2(4000)
);

CREATE OR REPLACE TRIGGER trg_motorcycles_audit
AFTER INSERT OR UPDATE OR DELETE ON Motorcycles
FOR EACH ROW
DECLARE
    v_old_values VARCHAR2(4000);
    v_new_values VARCHAR2(4000);
    v_operation  VARCHAR2(10);
BEGIN
    -- Captura valores antigos (para UPDATE ou DELETE)
    IF DELETING OR UPDATING THEN
        v_old_values := 'ID=' || :OLD.id ||
                        ', MODEL=' || :OLD.model ||
                        ', ENGINETYPE=' || :OLD.enginetype ||
```

Query Result | Script Output

Task completed in 0.106 seconds

```
Commit complete.

SECTOR | SPOT | NUM_MOTOS
Erro geral: não foram encontrados registros
```

Trigger

```
-- Trigger de Auditoria
CREATE TABLE audit_motorcycles (
  audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
  username VARCHAR2(100),
  operation VARCHAR2(10),
  operation_date DATE,
  old_values VARCHAR2(4000),
  new_values VARCHAR2(4000)
);

CREATE OR REPLACE TRIGGER trg_motorcycles_audit
AFTER INSERT OR UPDATE OR DELETE ON Motorcycles
FOR EACH ROW
DECLARE
    v_old_values VARCHAR2(4000);
    v_new_values VARCHAR2(4000);
    v_operation  VARCHAR2(10);
BEGIN
    -- Captura valores antigos (para UPDATE ou DELETE)
    IF DELETING OR UPDATING THEN
        v_old_values := 'ID=' || :OLD.id ||
                        ', MODEL=' || :OLD.model ||
                        ', ENGINETYPE=' || :OLD.enginetype ||
                        ', PLATE=' || :OLD.plate ||
                        ', LASTREVISIONDATE=' || TO_CHAR(:OLD.lastrevisiondate, 'DD/MM/YYYY') ||
                        ', SPOTID=' || :OLD.spotid;
    END IF;

    -- Captura valores novos (para INSERT ou UPDATE)
    IF INSERTING OR UPDATING THEN
        v_new_values := 'ID=' || :NEW.id ||
                        ', MODEL=' || :NEW.model ||
                        ', ENGINETYPE=' || :NEW.enginetype ||
                        ', PLATE=' || :NEW.plate ||
                        ', LASTREVISIONDATE=' || TO_CHAR(:NEW.lastrevisiondate, 'DD/MM/YYYY') ||
                        ', SPOTID=' || :NEW.spotid;
    END IF;

    -- Define o tipo de operação
    IF INSERTING THEN
        v_operation := 'INSERT';
    ELSIF UPDATING THEN
        v_operation := 'UPDATE';
    ELSIF DELETING THEN
        v_operation := 'DELETE';
    END IF;
```

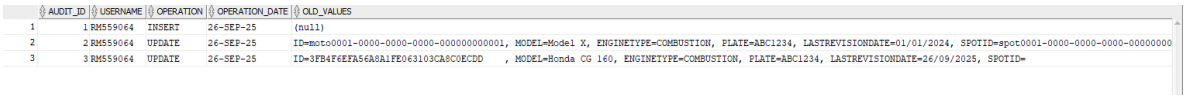
```
    -- Inserção na tabela de auditoria
    INSERT INTO audit_motorcycles (
        username,
        operation,
        operation_date,
        old_values,
        new_values
    ) VALUES (
        USER,
        v_operation,
        SYSDATE,
        v_old_values,
        v_new_values
    );

END;
/
```

```
-- Execução normal (INSERT + UPDATE)
INSERT INTO Motorcycles (id, model, enginetype, plate, lastrevisiondate)
VALUES (
    SYS_GUID(),
    'Honda CG 160',
    'COMBUSTION',
    'ABC1234',
    SYSDATE
);

UPDATE Motorcycles
SET model = 'Honda CG 160 Titan',
    plate = 'XYZ9876'
WHERE plate = 'ABC1234';

SELECT *
FROM audit_motorcycles
ORDER BY audit_id;
```

Script Output x | Query Result x

Task completed in 0.413 seconds

```
Trigger TRG_MOTORCYCLES_AUDIT compiled


1 row inserted.


2 rows updated.

>>Query Run In:Query Result
```



	AUDIT_ID	USERNAME	OPERATION	OPERATION_DATE	OLD_VALUES
1	1	RM559064	INSERT	26-SEP-25	(null)
2	2	RM559064	UPDATE	26-SEP-25	ID=moto0001-0000-0000-0000-000000000001, MODEL=Model X, ENGINETYPE=COMBUSTION, PLATE=ABC1234, LASTREVISIONDATE=01/01/2024, SPOTID=spot0001-0000-0000-0000-00000000
3	3	RM559064	UPDATE	26-SEP-25	ID=3FB4F6EFA56A8A1FE063103CA8C0ECDD , MODEL=Honda CG 160, ENGINETYPE=COMBUSTION, PLATE=ABC1234, LASTREVISIONDATE=26/09/2025, SPOTID=

CODIGOS SQL DESSA SPRIN

```
SET SERVEROUTPUT ON;

-- Função 1: Converter Dados de uma Vaga (Spot) em JSON
CREATE OR REPLACE FUNCTION FN_SPOT_PARA_JSON (
  p_spot_id  IN CHAR
) RETURN VARCHAR2 IS
  v_spot_id     SPOTS.SPOT_ID%TYPE;
  v_sector_id   SPOTS.SECTOR_ID%TYPE;
  v_status      SPOTS.STATUS%TYPE;
  v_motorcycle_id SPOTS.MOTORCYCLE_ID%TYPE;
  v_json        VARCHAR2(4000);
BEGIN
  -- Busca os dados relacionais
  SELECT SPOT_ID, SECTOR_ID, STATUS, NVL(MOTORCYCLE_ID, 'NULL')
  INTO v_spot_id, v_sector_id, v_status, v_motorcycle_id
  FROM SPOTS
  WHERE SPOT_ID = p_spot_id;
```

```
-- Monta o JSON manualmente
  v_json := '{' ||
        '"spotId": "' || v_spot_id || '", ' ||
        '"sectorId": "' || v_sector_id || '", ' ||
        '"status": "' || v_status || '", ' ||
        '"motorcycleId": "' || v_motorcycle_id || '"' ||
       '}';

  RETURN v_json;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN '{"erro": "Vaga n□o encontrada."}';
  WHEN TOO_MANY_ROWS THEN
    RETURN '{"erro": "Mais de uma vaga encontrada com o mesmo ID."}';
  WHEN OTHERS THEN
    RETURN '{"erro": "Erro inesperado: ' || SQLERRM || '"}';
END;
/
```

```
-- Função 2: Validação de Placa de Moto
CREATE OR REPLACE FUNCTION fn_validate_plate(p_plate IN VARCHAR2)
RETURN VARCHAR2 IS

BEGIN
 -- Verifica se a placa é nula
 IF p_plate IS NULL THEN
  RAISE NO_DATA_FOUND; -- for□a uma exception espec□fica
 END IF;

 -- Verifica tamanho invalido
 IF LENGTH(p_plate) != 7 AND LENGTH(p_plate) != 8 THEN
  RAISE VALUE_ERROR; -- for□a a exception
 END IF;

 -- Verifica se corresponde ao padr□o AAA0A00 ou AAA0000
 IF REGEXP_LIKE(p_plate, '^[A-Z]{3}[0-9][A-Z0-9][0-9]{2}$')
   OR REGEXP_LIKE(p_plate, '^[A-Z]{3}[0-9]{4}$') THEN
  RETURN 'Placa v□lida: ' || p_plate;
 ELSE
  RETURN 'Placa inv□lida: formato n□o corresponde.';
 END IF;
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 'Erro: A placa n□o pode ser nula.';
  WHEN VALUE_ERROR THEN
    RETURN 'Erro: Placa deve ter 7 ou 8 caracteres.';
  WHEN OTHERS THEN
    RETURN 'Erro inesperado: ' || SQLERRM;
END;
/
```

```
-- Esse pega as vagas (SPOTS) junto com os setores (SECTORS) e retorna tudo em formato JSON.
CREATE OR REPLACE PROCEDURE PRC_SPOTS_COM_SECTOR_JSON IS
  CURSOR c_spots IS
    SELECT s.SPOT_ID, s.STATUS, s.X, s.Y, s.MOTORCYCLE_ID,
        sec.ID AS SECTOR_ID, sec.SECTOR_TYPE_ID, sec.YARD_ID
    FROM SPOTS s
    JOIN SECTORS sec ON s.SECTOR_ID = sec.ID;

  v_json   VARCHAR2(32767) := '[';
  v_count  NUMBER := 0;
```

```
BEGIN
  FOR rec IN c_spots LOOP
    v_count := v_count + 1;

    v_json := v_json ||
      CASE WHEN v_count > 1 THEN ',' ELSE '' END ||
      '{' ||
       '"spotId": "' || rec.SPOT_ID || '", ' ||
       '"status": "' || rec.STATUS || '", ' ||
       '"coords": {"x": ' || rec.X || ', "y": ' || rec.Y || '}, ' ||
       '"motorcycleId": "' || NVL(TO_CHAR(rec.MOTORCYCLE_ID), 'NULL') || '", ' ||
       '"sector": {' ||
        '"id": "' || rec.SECTOR_ID || '", ' ||
        '"typeId": "' || rec.SECTOR_TYPE_ID || '", ' ||
        '"yardId": "' || rec.YARD_ID || '"' ||
       '}' ||
      '}';
  END LOOP;

  v_json := v_json || ']';
```

```
 -- Força NO_DATA_FOUND se n□o houver registros
   IF v_count = 0 THEN
     RAISE NO_DATA_FOUND;
   END IF;

   DBMS_OUTPUT.PUT_LINE(v_json);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('{"erro": "Nenhum dado encontrado"}');
   WHEN VALUE_ERROR THEN
     DBMS_OUTPUT.PUT_LINE('{"erro": "Erro de convers□o de valor"}');
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('{"erro": "Erro inesperado: ' || SQLERRM || '"}');
END;
/
```

```
-- Procedimento 2: A procedure contagem_motos_setor tem como objetivo gerar um relat□rio manual de contagem de motos por setor dentro do p□tio, exibindo tamb□m o total geral de motos no p□tio.

CREATE OR REPLACE PROCEDURE relatorio_motos_por_vaga IS


  CURSOR c_spots IS
    SELECT s.id AS sector_id,
        sp.spot_id,
        CASE WHEN m.id IS NOT NULL THEN 1 ELSE 0 END AS num_motos
    FROM sectors s
    JOIN spots sp ON sp.sector_id = s.id
    LEFT JOIN Motorcycles m ON m.spotid = sp.spot_id
    ORDER BY s.id, sp.spot_id;


  v_sector       sectors.id%TYPE;
  v_spot         spots.spot_id%TYPE;
  v_num_motos    NUMBER;


  v_sector_atual  sectors.id%TYPE := NULL;
  v_subtotal     NUMBER := 0;
  v_total_patio   NUMBER := 0;
  v_count        NUMBER := 0; -- contador de registros
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('SECTOR | SPOT | NUM_MOTOS');

  FOR rec IN c_spots LOOP
    v_count := v_count + 1;

    BEGIN
      v_sector := rec.sector_id;
      v_spot   := rec.spot_id;
      v_num_motos := rec.num_motos;

      -- Se mudou de setor, imprime subtotal do setor anterior
      IF v_sector_atual IS NOT NULL AND v_sector_atual != v_sector THEN
        DBMS_OUTPUT.PUT_LINE(v_sector_atual || ' | SUBTOTAL | ' || v_subtotal);
        v_subtotal := 0; -- reseta subtotal
      END IF;

      -- Exibe linha detalhada por vaga
      DBMS_OUTPUT.PUT_LINE(v_sector || ' | ' || v_spot || ' | ' || v_num_motos);

      -- Acumula subtotal e total geral
      v_subtotal := v_subtotal + v_num_motos;
      v_total_patio := v_total_patio + v_num_motos;

      -- Atualiza setor atual
      v_sector_atual := v_sector;
```

```
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Erro: registro de vaga ou setor n□o encontrado');
      WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('Erro: valor inv□lido encontrado em
num_motos');
      WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Erro desconhecido: ' || SQLERRM);
    END;
  END LOOP;

  -- Se n□o encontrou nenhum registro, lan□a NO_DATA_FOUND
  IF v_count = 0 THEN
    RAISE NO_DATA_FOUND;
  END IF;

  -- Imprime subtotal do □ltimo setor
  IF v_sector_atual IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE(v_sector_atual || ' | SUBTOTAL | ' || v_subtotal);
  END IF;

  -- Imprime total geral do p□tio
  DBMS_OUTPUT.PUT_LINE('TOTAL PATIO ' || v_total_patio);
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Erro geral: n□o foram encontrados registros');
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Erro geral: valor inv□lido durante a execu□□o');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Erro geral desconhecido: ' || SQLERRM);

END relatorio_motos_por_vaga;
/
```

```
-- Trigger de Auditoria
CREATE TABLE audit_motorcycles (
  audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
  username VARCHAR2(100),
  operation VARCHAR2(10),
  operation_date DATE,
  old_values VARCHAR2(4000),
  new_values VARCHAR2(4000)
);

CREATE OR REPLACE TRIGGER trg_motorcycles_audit
AFTER INSERT OR UPDATE OR DELETE ON Motorcycles
FOR EACH ROW
DECLARE
   v_old_values VARCHAR2(4000);
   v_new_values VARCHAR2(4000);
   v_operation  VARCHAR2(10);
```

```
BEGIN
  -- Captura valores antigos (para UPDATE ou DELETE)
  IF DELETING OR UPDATING THEN
    v_old_values := 'ID=' || :OLD.id ||
                    ', MODEL=' || :OLD.model ||
                    ', ENGINETYPE=' || :OLD.enginetype ||
                    ', PLATE=' || :OLD.plate ||
                    ', LASTREVISIONDATE=' || TO_CHAR(:OLD.lastrevisiondate,
'DD/MM/YYYY') ||
                    ', SPOTID=' || :OLD.spotid;
  END IF;

  -- Captura valores novos (para INSERT ou UPDATE)
  IF INSERTING OR UPDATING THEN
    v_new_values := 'ID=' || :NEW.id ||
                    ', MODEL=' || :NEW.model ||
                    ', ENGINETYPE=' || :NEW.enginetype ||
                    ', PLATE=' || :NEW.plate ||
                    ', LASTREVISIONDATE=' || TO_CHAR(:NEW.lastrevisiondate,
'DD/MM/YYYY') ||
                    ', SPOTID=' || :NEW.spotid;
  END IF;

  -- Define o tipo de opera??o
  IF INSERTING THEN
    v_operation := 'INSERT';
  ELSIF UPDATING THEN
```

```
    v_operation := 'UPDATE';
  ELSIF DELETING THEN
    v_operation := 'DELETE';
  END IF;

  -- Inser□□o na tabela de auditoria
  INSERT INTO audit_motorcycles (
    username,
    operation,
    operation_date,
    old_values,
    new_values
  ) VALUES (
    USER,
    v_operation,
    SYSDATE,
    v_old_values,
    v_new_values
  );

END;
/
```

CODIGO CORRIGIDO SPRINT PASSADA

```
SET SERVEROUTPUT ON;
-- 1. Motores por tipo, ano de revisÃ£o, setor e pÃ¡tio (3 JOINs)
BEGIN
  FOR rec IN (
    SELECT
      m.enginetype,
      EXTRACT(YEAR FROM m.lastrevisiondate) AS revision_year,
      st.name AS sector_type,
      y.name AS yard_name,
      COUNT(*) AS total_motorcycles
    FROM Motorcycles m
    JOIN spots s ON m.spotid = s.spot_id
    JOIN sectors sec ON s.sector_id = sec.id
    JOIN sector_types st ON sec.sector_type_id = st.id
    JOIN yards y ON sec.yard_id = y.id
    GROUP BY m.enginetype, EXTRACT(YEAR FROM m.lastrevisiondate), st.name,
y.name
    ORDER BY revision_year DESC, total_motorcycles DESC
  ) LOOP
```

```
DBMS_OUTPUT.PUT_LINE('Engine Type: ' || rec.enginetype ||
                     ', Revision Year: ' || rec.revision_year ||
                     ', Sector Type: ' || rec.sector_type ||
                     ', Yard: ' || rec.yard_name ||
                     ', Total: ' || rec.total_motorcycles);
  END LOOP;
END;
/
```

```
-- 2. PÃ¡tios, total de setores, Ã¡rea total dos setores, tipo de setor e cidade (3 JOINs)
BEGIN
  FOR rec IN (
    SELECT
      y.name AS yard_name,
      COUNT(sec.id) AS total_sectors,
      st.name AS sector_type,
      a.city AS city,
      COUNT(s.spot_id) AS total_spots
    FROM yards y
    JOIN addresses a ON y.address_id = a.id
    JOIN sectors sec ON sec.yard_id = y.id
    JOIN sector_types st ON sec.sector_type_id = st.id
    JOIN spots s ON s.sector_id = sec.id
    GROUP BY y.name, st.name, a.city
    ORDER BY total_spots DESC
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Yard: ' || rec.yard_name ||
                         ', Sectors: ' || rec.total_sectors ||
                         ', Sector Type: ' || rec.sector_type ||
                         ', City: ' || rec.city ||
                         ', Total Spots: ' || rec.total_spots);
  END LOOP;
END;
/
```

```
-- 3. Cidades, total de pÃ¡tios, total de setores, total de motos (3 JOINs)
BEGIN
  FOR rec IN (
    SELECT
      a.city,
      COUNT(DISTINCT y.id) AS total_yards,
      COUNT(DISTINCT sec.id) AS total_sectors,
      COUNT(DISTINCT m.id) AS total_motorcycles
    FROM addresses a
    JOIN yards y ON y.address_id = a.id
    JOIN sectors sec ON sec.yard_id = y.id
    JOIN spots s ON s.sector_id = sec.id
    LEFT JOIN Motorcycles m ON m.spotid = s.spot_id
    GROUP BY a.city
    ORDER BY total_motorcycles DESC
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('City: ' || rec.city ||
                         ', Yards: ' || rec.total_yards ||
                         ', Sectors: ' || rec.total_sectors ||
                         ', Motorcycles: ' || rec.total_motorcycles);
  END LOOP;
END;
/
```

```
DECLARE
 CURSOR c_sectors IS
  SELECT
   sec.id AS sector_id,
   sec.id AS sector_id_repeat  -- Alias para a segunda coluna sec.id
  FROM sectors sec
  ORDER BY sec.id;

 TYPE id_table IS TABLE OF sectors.id%TYPE INDEX BY PLS_INTEGER;

 ids      id_table;
 total    INTEGER := 0;
 v_prev_id  sectors.id%TYPE;
 v_next_id  sectors.id%TYPE;
BEGIN
 -- Carrega os dados ordenados no array
 FOR rec IN c_sectors LOOP
  total := total + 1;
  ids(total) := rec.sector_id;
 END LOOP;
```

```
-- Cabeçalho
  DBMS_OUTPUT.PUT_LINE(RPAD('Setor', 38) || RPAD('Anterior', 38) || RPAD('Atual', 38) || RPAD('Próximo', 38));

  FOR i IN 1..total LOOP
   -- Anterior
   IF i = 1 THEN
    v_prev_id := NULL;
   ELSE
    v_prev_id := ids(i - 1);
   END IF;

   -- Próximo
   IF i = total THEN
    v_next_id := NULL;
   ELSE
    v_next_id := ids(i + 1);
   END IF;
```

```
DBMS_OUTPUT.PUT_LINE(
    RPAD(ids(i), 38) ||
    RPAD(NVL(v_prev_id, 'Vazio'), 38) ||
    RPAD(ids(i), 38) ||
    RPAD(NVL(v_next_id, 'Vazio'), 38)
    );
  END LOOP;
END;
/
```