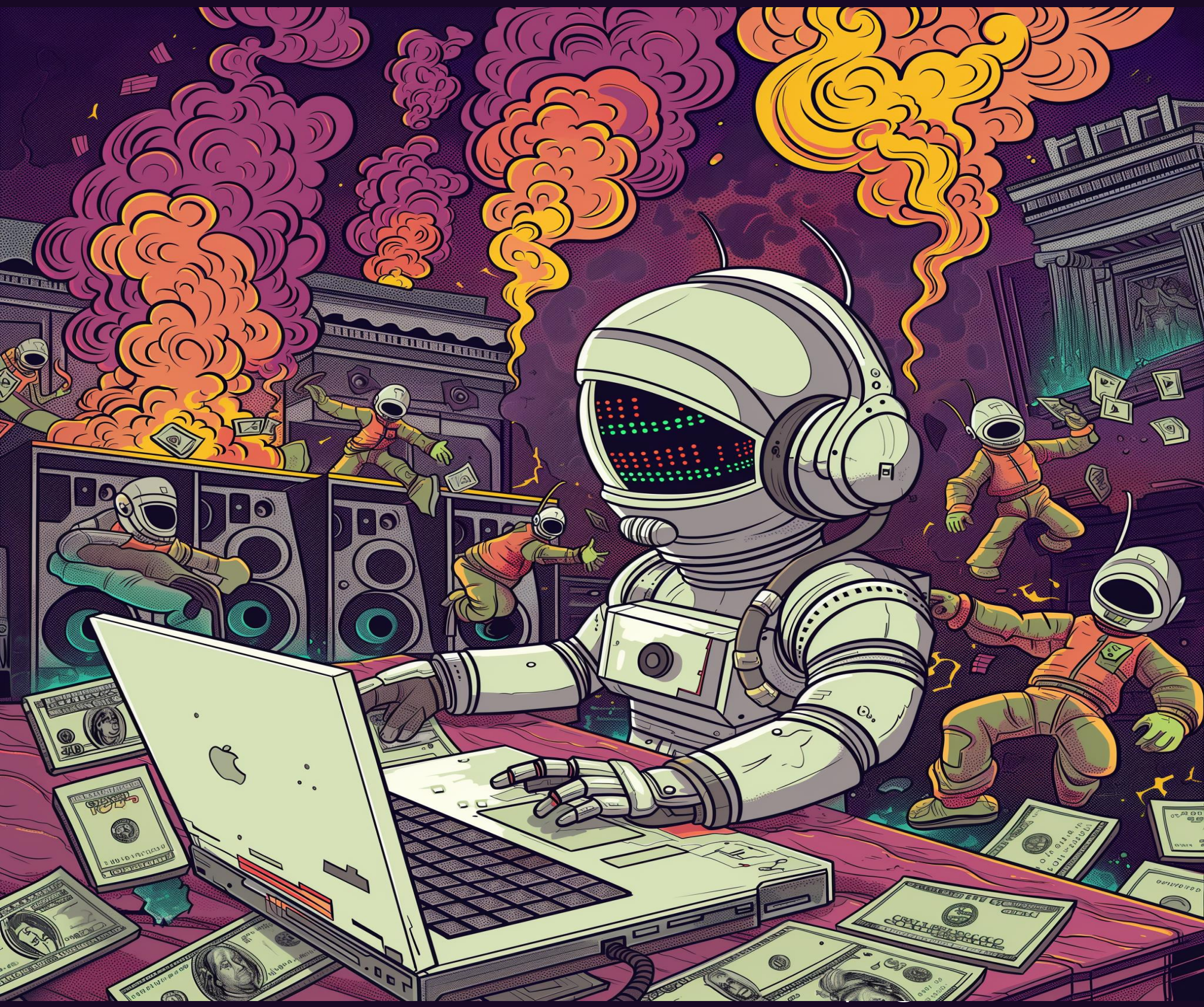
PL/SQL AVANÇADO

SALVANDO OS SEUS DADOS



LAÉRCIO CRUZ

PL/SQL AVANÇADO

Salvando os eus Dados

Bem-vindo ao eBook "PL/SQL Avançado: Salvando os Seus Dados".

Vamos explorar técnicas avançadas de PL/SQL que permitirão a você escrever o seu código com eficiência, abordando desde o uso de cursores e triggers até procedures e packages, sempre com exemplos práticos e contextos reais para facilitar o seu aprendizado.

Prepare-se para mergulhar no universo do PL/SQL avançado e transformar-se no herói que seu banco de dados precisa. Vamos lá, hora de apagar alguns incêndios!

1

CURSORS

Manipulação de Dados Linha a Linha

Cursors permitem tratar conjuntos de dados linha a linha, sendo úteis para processar resultados de uma consulta de maneira controlada.

CURSORS

Cursors são utilizados para percorrer linhas retornadas por uma consulta SQL, permitindo processar cada linha individualmente. Eles são úteis quando você precisa executar operações linha a linha, como calcular somas, atualizar registros baseados em condições específicas, ou simplesmente exibir dados de forma controlada. Podem ser **explícitos** ou **implícitos**. O cursor explícito oferece maior controle, pois você pode abrir, buscar e fechar o cursor manualmente.

Exemplo de **Cursor Explícito**:

```
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, first_name, last_name FROM employees;
    emp_record emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('ID: ' || emp_record.employee_id || ', Nome: ' ||
                             emp_record.first_name || ' ' || emp_record.last_name);
    END LOOP;
    CLOSE emp_cursor;
END;
```

CURSORS

Neste exemplo, o **cursor implícito** é criado automaticamente pela instrução FOR no início do bloco. Ele percorre os resultados da consulta SELECT e executa as instruções dentro do loop para cada linha retornada. Dessa forma, não é necessário declarar explicitamente o cursor.

```
BEGIN
    FOR emp_record IN (SELECT employee_id, first_name, last_name FROM employees) LOOP
        DBMS_OUTPUT.PUT_LINE('ID: ' || emp_record.employee_id || ', Nome: ' ||
                             emp_record.first_name || ' ' || emp_record.last_name);
    END LOOP;
END;
```

2

TRIGGERS

Automatizando Ações no Banco de Dados

Triggers são blocos de código que são executados automaticamente em resposta a eventos específicos em uma tabela.

TRIGGERS

Triggers permitem executar código automaticamente antes ou depois de operações como INSERT, UPDATE ou DELETE em uma tabela. Eles são úteis para manter a integridade dos dados, auditar mudanças ou aplicar regras de negócio automaticamente. Por exemplo, um trigger pode ser usado para registrar a hora e o usuário que fez uma modificação em uma tabela.

Exemplo de Trigger **After Insert**:

```
CREATE OR REPLACE TRIGGER trg_after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO employee_audit (employee_id, action)
    VALUES (:NEW.employee_id, 'INSERT');
END;
```

Aqui, o trigger trg_after_insert_employee é acionado após a inserção de um novo registro na tabela employees, adicionando um log na tabela employee_audit.

TRIGGERS

O exemplo a seguir mostra um trigger **Before Insert** que calcula a comissão de cada novo funcionário pertencente ao departamento 10, antes de um registro para aquele funcionário ser inserido na tabela SALARIO:

```
CREATE OR REPLACE TRIGGER emp_comm_trig
    BEFORE INSERT ON salario
    FOR EACH ROW
BEGIN
    IF :NEW.deptID = 10 THEN
        :NEW.comm := :NEW.sal * .4;
    END IF;
END;
```

3

PROCEDURES

Reutilização e Organização de Código

Procedures são subprogramas que podem ser chamados para executar uma série de instruções PL/SQL.

PROCEDURES

Procedures permitem agrupar várias instruções PL/SQL em um único bloco de código, facilitando a reutilização e manutenção. Eles são úteis quando você precisa executar a mesma série de instruções em diferentes partes do seu código. Procedures podem aceitar parâmetros de entrada e saída, permitindo que você passe dados e receba resultados.

Exemplo de **Procedure**:

```
CREATE OR REPLACE PROCEDURE update_salary (
            p_emp_id IN employees.employee_id%TYPE,
            p_new_salary IN employees.salary%TYPE) IS
BEGIN
    UPDATE employees
    SET salary = p_new_salary
    WHERE employee_id = p_emp_id;
    COMMIT;
END;
```

Esta procedure update_salary atualiza o salário de um funcionário específico, facilitando a reutilização dessa lógica em diferentes partes do seu sistema.

4

FUNCTIONS

Retornando Valores Específicos

Functions são subprogramas que retornam um valor específico e podem ser usadas em expressões SQL.

FUNCTIONS

Functions são semelhantes às procedures, mas são usadas principalmente para calcular e retornar um valor específico. Elas podem ser chamadas a partir de instruções SQL, permitindo que você incorpore lógica PL/SQL diretamente em consultas SQL. Functions são ideais para cálculos ou operações que precisam ser reutilizadas em várias partes do sistema.

Exemplo de **Function**:

```
CREATE OR REPLACE FUNCTION get_employee_full_name (
                p_emp_id IN employees.employee_id%TYPE)
                RETURN VARCHAR2 IS
    v_full_name employees.first_name%TYPE || ' ' ||
                employees.last_name%TYPE;
BEGIN
    SELECT first_name || ' ' || last_name
    INTO v_full_name
    FROM employees
    WHERE employee_id = p_emp_id;

    RETURN v_full_name;
END;
```

5

PACKAGES

Organizando Melhor seu Código

Packages agrupam tipos, itens e subprogramas PL/SQL logicamente relacionados, melhorando a organização e encapsulamento do código.

PACKAGES

Packages permitem agrupar procedures, functions, variáveis e outros elementos PL/SQL em uma única unidade, facilitando a manutenção e reutilização de código. Eles oferecem um encapsulamento melhor e ajudam a organizar logicamente as funcionalidades relacionadas. Packages podem conter um corpo e uma especificação, onde a especificação define a interface pública e o corpo implementa a lógica.

Exemplo de **Package**:

```
CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee (p_first_name IN VARCHAR2, p_last_name IN VARCHAR2);
    FUNCTION get_employee_count RETURN NUMBER;
END emp_pkg;
/
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    PROCEDURE add_employee (p_first_name IN VARCHAR2, p_last_name IN VARCHAR2) IS
    BEGIN
        INSERT INTO employees (first_name, last_name) VALUES (p_first_name, p_last_name);
        COMMIT;
    END add_employee;

    FUNCTION get_employee_count RETURN NUMBER IS
        v_count NUMBER;
    BEGIN
        SELECT COUNT(*)
        INTO v_count
        FROM employees;

        RETURN v_count;
    END get_employee_count;
END emp_pkg;
```

CONSIDERAÇÕES FINAIS

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por Inteligência Artificial e diagramado por humano. Todo o processo de criação está detalhado no meu Github.

.

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizado uma validação minuciosa humana no conteúdo e pode conter erros gerados por uma IA.

https://github.com/laerciocruz/prompts-recipe-to-create-a-ebook