TRIGGERS

PROGRAMAÇÃO AVANÇADA DE BANCO DE DADOS

Programação Avançada de Banco de Dados

Professor

- Me. Carlos Augusto Lombardi Garcia

Grupo VII

- Francilei Augusto dos Santos
- Jonathan Cezar de Souza Silva
- Matheus Henrique de Gonçalves
- Railson Tales de Oliveira
- Robson de Sousa

O que são Triggers (Gatilhos)

- Triggers são eventos mapeados pelo banco de dados que executam blocos PL/SQL.
- São dependentes das tabelas ou visões nas quais são declarados.
- Triggers são muito utilizados para auditoria e log.

Propriedades dos Triggers

- Tempo: Momento em que o Trigger será ativado. Pode ser ativado antes ou depois da operação
- Evento: Qual operação que ativa o Trigger. Permite também múltiplos eventos, separados por OR.
- Tipo: comando (executa apenas uma vez para o comando) ou linha (executa o trigger uma vez para cada linha atualizada/adicionada)
- Corpo: bloco PL/SQL
- Podem ter propriedades combinadas

Características do corpo do Trigger

- Permite usar :old.coluna e :new.coluna para acessar valores antes da atualização, e o valor atualizado.
- Permite fazer ações diferentes para cada operação, usando IF. Por exemplo:

```
IF DELETING THEN
-- Ação
ELSIF INSERTING THEN
 -- Ação
```

Características do corpo do Trigger

- No caso do UPDATING, pode ser especificado uma coluna específica, em que o IF/ELSIF será ativado apenas para atualização da mesma.

```
IF UPDATING(‘salary') THEN
-- Ação
ELSIF UPDATING(‘job_id’) THEN
-- Ação
```

Exercício - Objetivo

- Substituir Trigger existente que é executado sempre que há uma mudança de cargo ou departamento
- Mudança deve registrar salário recebido pelo empregado antes e depois da mudança de cargo, e exibir mensagem com o aumento salarial

Primeiro Passo – Novos campos

- Adição de campos novos para registrar dados (salário antigo, novo ID departamento, novo salário)

```
ALTER TABLE job_history ADD (
        sal_old number(8, 2), new_department_id number(4, 0),
        new_job_id varchar2(10), sal_new number(8, 2)
);
```

Segundo Passo – Procedure

- A Procedure *add_job_history* efetivamente faz a inserção do registro na tabela *job_history*
- Os campos antigos e novos são os parâmetros, e faz com que, quando ativado, tente adicionar um novo registro com os dados novos
- Como a tabela tem uma constraint em que o employee_id deve ser único, é usada a Exception DUP_VAL_ON_INDEX, e caso já tenha um registro do funcionário, ele atualiza o registro existente

```
CREATE OR REPLACE PROCEDURE add_job_history
  (  p_emp_id          job_history.employee_id%type
   , p_start_date      job_history.start_date%type
   , p_end_date        job_history.end_date%type
   , p_job_id          job_history.job_id%type
   , p_new_job_id      job_history.new_job_id%type
   , p_department_id   job_history.department_id%type
   , p_new_department_id job_history.new_department_id%type
   , p_new_salary job_history.sal_new%type
   , p_old_salary job_history.sal_old%type
   )
IS
BEGIN
  INSERT INTO job_history (employee_id, start_date, end_date,
                      job_id, department_id, new_department_id,
                      sal_new, sal_old, new_job_id)
    VALUES(p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id,
    p_new_department_id, p_new_salary, p_old_salary, p_new_job_id);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        UPDATE job_history
        SET end_date=p_end_date, department_id=p_department_id,
            new_department_id=p_new_department_id,
            job_id=p_job_id, new_job_id=p_new_job_id,
            sal_old=p_old_salary, sal_new=p_new_salary
        WHERE employee_id=p_emp_id;
END add_job_history;
```

Terceiro Passo – Atualização do Trigger

- Atualização do Trigger para que chame a procedure *add_job_history* com os campos antigos e novos, e exiba a mensagem do aumento do salário
- O Trigger foi configurado para ser ativado após o UPDATE dos campos *job_id* e/ou *department_id*, realizando a operação para cada linha atualizada

Terceiro Passo – Atualização do Trigger

```
CRATE OR REPLACE TRIGGER update_job_history
  AFTER UPDATE OF job_id, department_id ON employees
  FOR EACH ROW

BEGIN
    add_job_history(:old.employee_id, :old.hire_date, sysdate,
                    :old.job_id, :new.job_id, :old.department_id,
                    :new.department_id, :new.salary, :old.salary);

    DBMS_OUTPUT.PUT_LINE('Aumento foi de: R$'|| to_char(:new.salary -
:old.salary));
END;
```

REFERÊNCIAS

- Create Trigger – Oracle Online Documentation. Disponível em: <https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_7004.htm>. Acesso em: 24/10/2020
- Create Procedure – Oracle Online Documentation. Disponível em: <https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6009.htm>. Acesso em: 24/10/2020
- Errors and Exception Handling - Oracle Online Documentation. Disponível em: <https://docs.oracle.com/cd/E11882_01/timesten.112/e21639/exceptions.htm#TTPLS191>. Acesso em: 25/10/2020