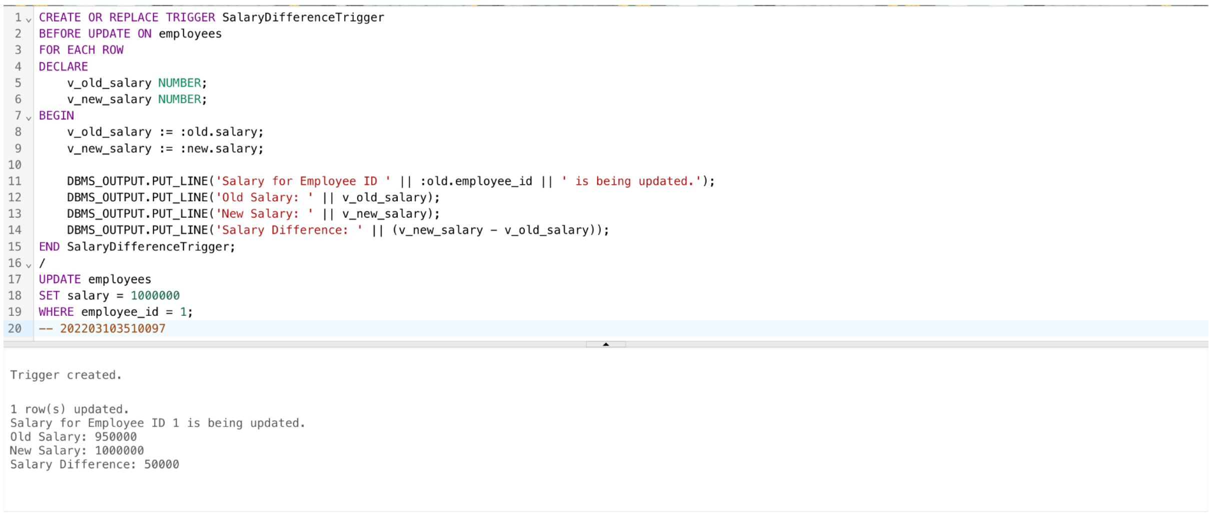
Practical No. 12

Aim: Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers

Theory:
Triggers are database objects that execute automatically in response to specific events or actions within the database. They are used to enforce business rules, audit changes, and perform other actions without requiring manual intervention. Triggers can be categorized based on when they execute (BEFORE or AFTER) and how they react to events (Row or Statement Triggers), and there are also INSTEAD OF Triggers, which allow you to customize behavior when data modification is requested.

Queries:

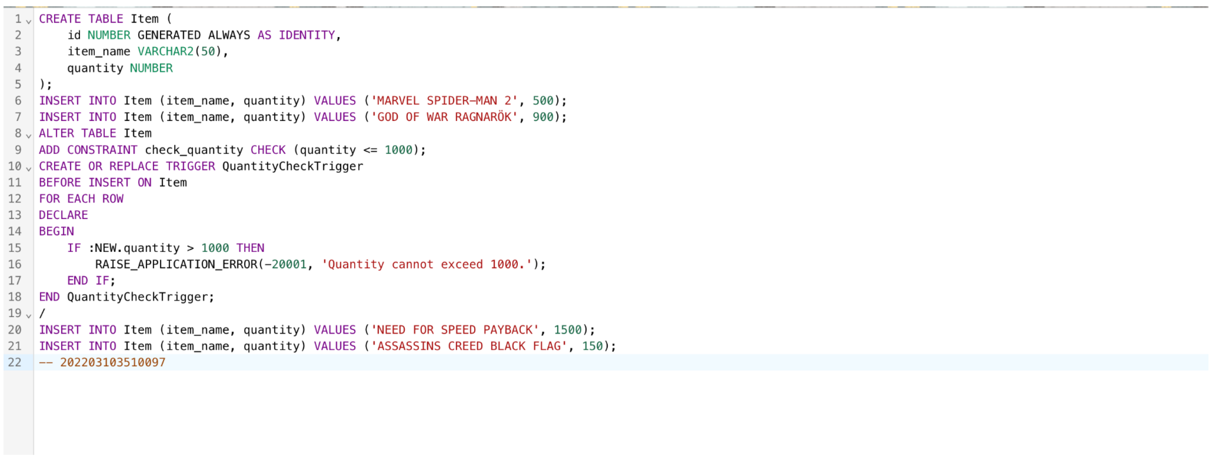
1) Trigger will display the salary difference between the old values and new values.



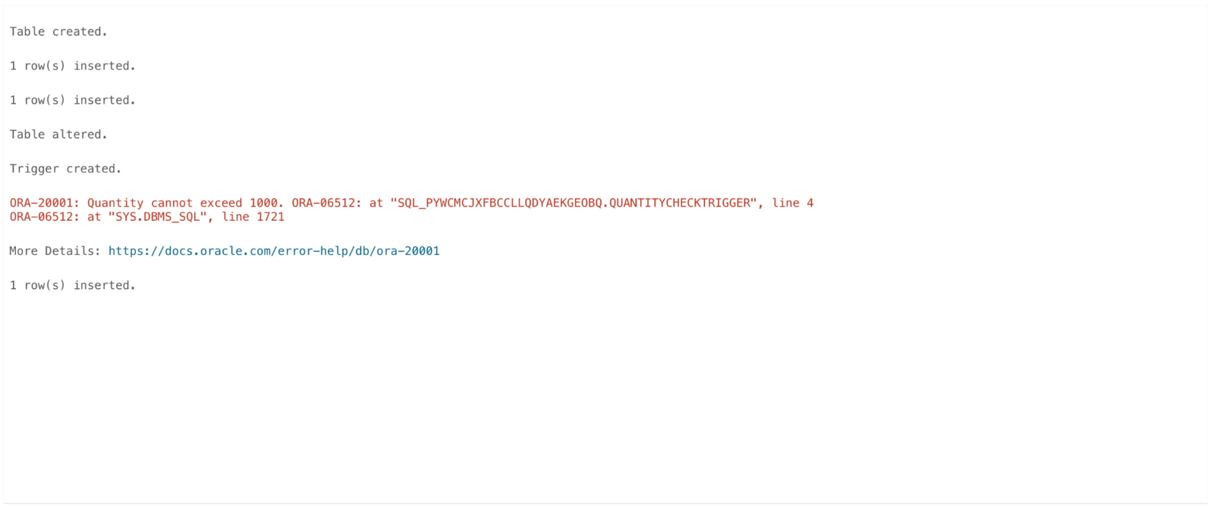
```
CREATE OR REPLACE TRIGGER SalaryDifferenceTrigger
BEFORE UPDATE ON employees
FOR EACH ROW
DECLARE
    v_old_salary NUMBER;
    v_new_salary NUMBER;
BEGIN
    v_old_salary := :old.salary;
    v_new_salary := :new.salary;

    DBMS_OUTPUT.PUT_LINE('Salary for Employee ID ' || :old.employee_id || ' is being updated.');
    DBMS_OUTPUT.PUT_LINE('Old Salary: ' || v_old_salary);
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || v_new_salary);
    DBMS_OUTPUT.PUT_LINE('Salary Difference: ' || (v_new_salary - v_old_salary));
END SalaryDifferenceTrigger;
/
UPDATE employees
SET salary = 1000000
WHERE employee_id = 1;
-- 202203103510097
```

```
Trigger created.

1 row(s) updated.
Salary for Employee ID 1 is being updated.
Old Salary: 950000
New Salary: 1000000
Salary Difference: 50000
```

2) Stop the transaction if the quantity entered (insert) by the user exceeds 1000.

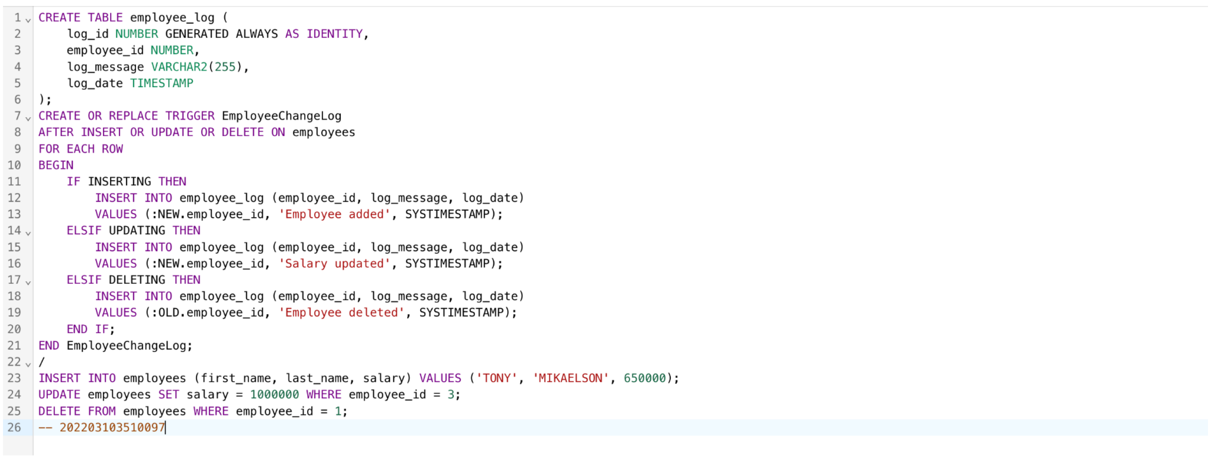


```sql
CREATE TABLE Item (
    id NUMBER GENERATED ALWAYS AS IDENTITY,
    item_name VARCHAR2(50),
    quantity NUMBER
);
INSERT INTO Item (item_name, quantity) VALUES ('MARVEL SPIDER-MAN 2', 500);
INSERT INTO Item (item_name, quantity) VALUES ('GOD OF WAR RAGNARÖK', 900);
ALTER TABLE Item
ADD CONSTRAINT check_quantity CHECK (quantity <= 1000);
CREATE OR REPLACE TRIGGER QuantityCheckTrigger
BEFORE INSERT ON Item
FOR EACH ROW
DECLARE
BEGIN
    IF :NEW.quantity > 1000 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Quantity cannot exceed 1000.');
    END IF;
END QuantityCheckTrigger;
/
INSERT INTO Item (item_name, quantity) VALUES ('NEED FOR SPEED PAYBACK', 1500);
INSERT INTO Item (item_name, quantity) VALUES ('ASSASSINS CREED BLACK FLAG', 150);
-- 202203103510097
```



```
Table created.

1 row(s) inserted.

1 row(s) inserted.

Table altered.

Trigger created.

ORA-20001: Quantity cannot exceed 1000. ORA-06512: at "SQL_PYWCMCJXFBCCLLQDYAEKGEOBQ.QUANTITYCHECKTRIGGER", line 4
ORA-06512: at "SYS.DBMS_SQL", line 1721

More Details: https://docs.oracle.com/error-help/db/ora-20001

1 row(s) inserted.
```

3) Maintain Log Table with use of trigger.



```sql
CREATE TABLE employee_log (
    log_id NUMBER GENERATED ALWAYS AS IDENTITY,
    employee_id NUMBER,
    log_message VARCHAR2(255),
    log_date TIMESTAMP
);
CREATE OR REPLACE TRIGGER EmployeeChangeLog
AFTER INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO employee_log (employee_id, log_message, log_date)
        VALUES (:NEW.employee_id, 'Employee added', SYSTIMESTAMP);
    ELSIF UPDATING THEN
        INSERT INTO employee_log (employee_id, log_message, log_date)
        VALUES (:NEW.employee_id, 'Salary updated', SYSTIMESTAMP);
    ELSIF DELETING THEN
        INSERT INTO employee_log (employee_id, log_message, log_date)
        VALUES (:OLD.employee_id, 'Employee deleted', SYSTIMESTAMP);
    END IF;
END EmployeeChangeLog;
/
INSERT INTO employees (first_name, last_name, salary) VALUES ('TONY', 'MIKAELSON', 650000);
UPDATE employees SET salary = 1000000 WHERE employee_id = 3;
DELETE FROM employees WHERE employee_id = 1;
-- 202203103510097
```

Conclusion: Triggers are essential tools in database management, enabling developers to automate actions, enforce data integrity, and customize behavior in response to database events. Understanding the different types of triggers (BEFORE, AFTER, Row, Statement, and INSTEAD OF) and their appropriate use cases is vital for effective database programming and management.