**EXP-13 WORKING WITH TRIGGER**

**NAME:M.Praveenraj**

**REG NO:230701396**

**DATE:22.10.2024**

**Program 1**

**Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.**

**CREATE OR REPLACE TRIGGER prevent\_parent\_deletion BEFORE DELETE ON employees**

# FOR EACH ROW DECLARE

**pl\_dept\_count NUMBER; BEGIN**

# SELECT COUNT(\*)

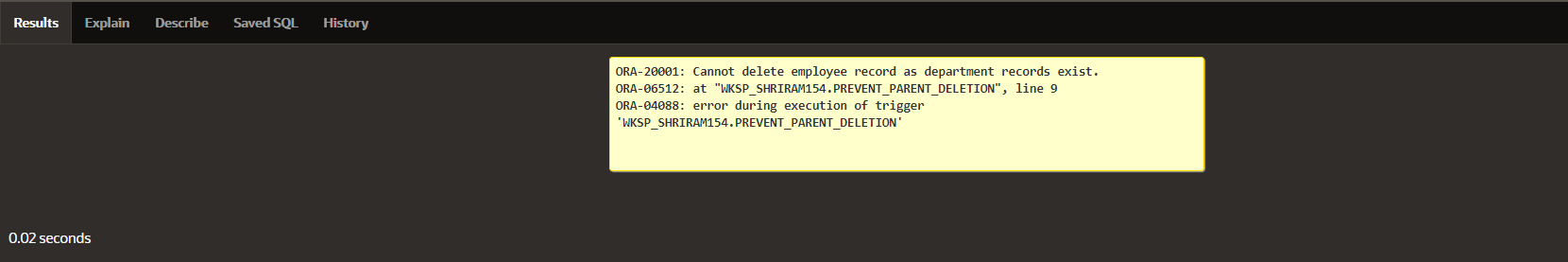
**INTO pl\_dept\_count FROM department**

**WHERE dept\_id = :OLD.employee\_id; IF pl\_dept\_count > 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete employee record as department records exist.');**

# END IF; END;

**DELETE FROM employees WHERE employee\_id = 70;**



**Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.**

**CREATE OR REPLACE TRIGGER prevent\_duplicate\_manager\_id BEFORE INSERT OR UPDATE ON employees**

# FOR EACH ROW DECLARE

**pl\_count NUMBER; BEGIN**

# SELECT COUNT(\*)

**INTO pl\_count FROM employees**

**WHERE manager\_id = :NEW.manager\_id AND employee\_id != :NEW.employee\_id; IF pl\_count > 0 THEN**

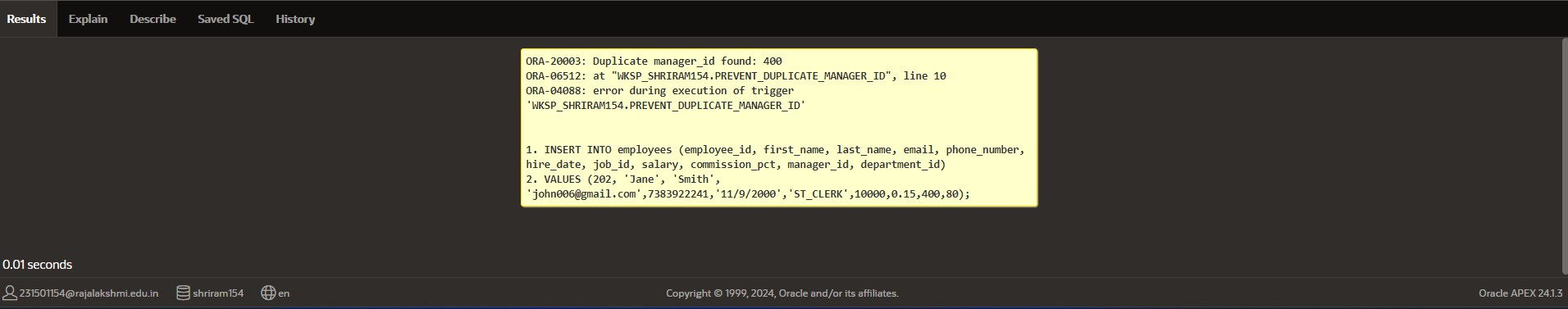
**RAISE\_APPLICATION\_ERROR(-20003, 'Duplicate manager\_id found: ' ||**

**:NEW.manager\_id); END IF;**

# END;

**INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id)**

**VALUES (202, 'Jane', 'Smith', 'john006@gmail.com',7383922241,'11/9/2000','ST\_CLERK',10000,0.15,400,80);**



**Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a**

**column's values exceeds a certain threshold.**

**CREATE OR REPLACE TRIGGER restrict\_salary\_insertion BEFORE INSERT ON employees**

# FOR EACH ROW DECLARE

**total\_salary NUMBER; threshold NUMBER := 100000;**

# BEGIN

**SELECT SUM(salary) INTO total\_salary FROM employees;**

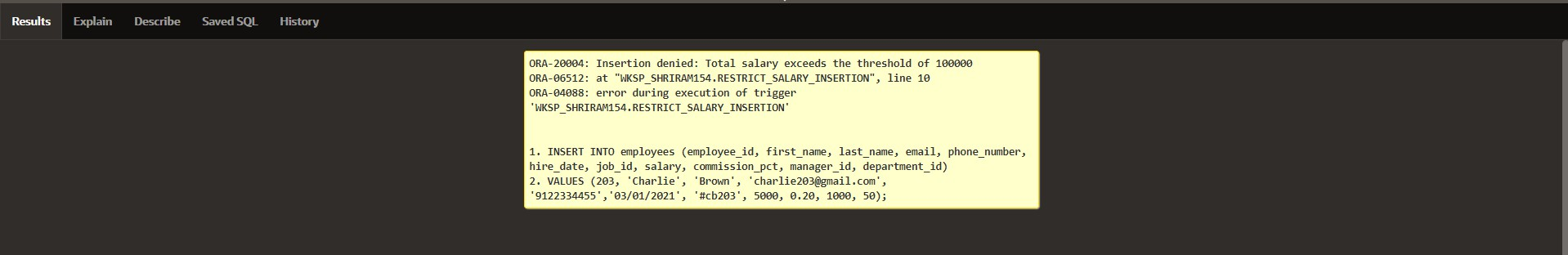
**IF (total\_salary + :NEW.salary) > threshold THEN**

**RAISE\_APPLICATION\_ERROR(-20004, 'Insertion denied: Total salary exceeds the threshold of ' || threshold);**

# END IF; END;

**INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id)**

**VALUES (203, 'Charlie', 'Brown', 'charlie203@gmail.com', '9122334455','03/01/2021', '#cb203', 5000, 0.20, 1000, 50);**



**Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.**

**CREATE OR REPLACE TRIGGER audit\_changes AFTER UPDATE OF salary, job\_id ON employees FOR EACH ROW**

# BEGIN

**IF :OLD.salary != :NEW.salary OR :OLD.job\_id != :NEW.job\_id THEN INSERT INTO employee\_audit (**

**employee\_id, old\_salary, new\_salary, old\_job\_title, new\_job\_title, change\_timestamp, changed\_by**

# ) VALUES (

**:OLD.employee\_id,**

**:OLD.salary,**

**:NEW.salary,**

**:OLD.job\_id,**

**:NEW.job\_id, SYSTIMESTAMP, USER**

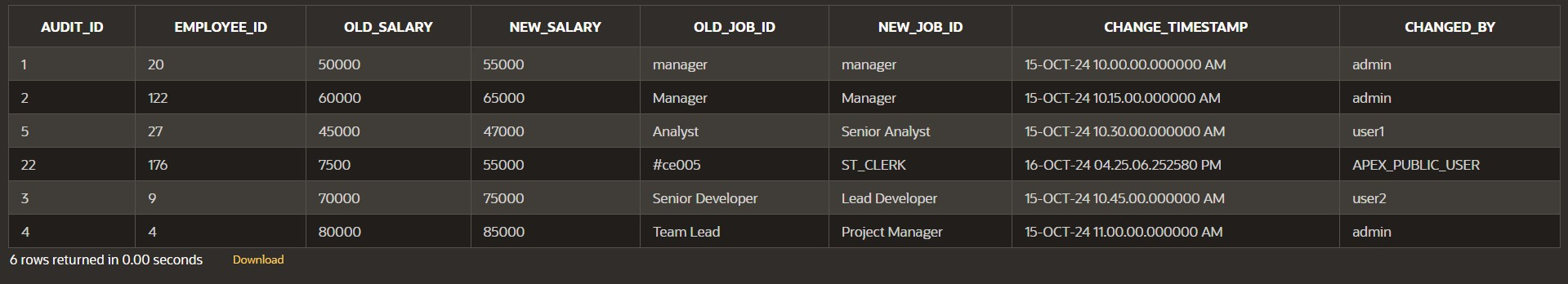
# ); END IF;

**END;**

**UPDATE employees**

**SET salary = 55000, job\_id = 'ST\_CLERK' WHERE employee\_id = 176;**

**SELECT \* FROM employee\_audit;**



**Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.**

**CREATE OR REPLACE TRIGGER trg\_audit\_employees AFTER INSERT OR UPDATE OR DELETE ON employees FOR EACH ROW**

# DECLARE

**v\_old\_values CLOB; v\_new\_values CLOB;**

# BEGIN

**IF INSERTING THEN**

**v\_old\_values := NULL;**

**v\_new\_values := 'employee\_id: ' || :NEW.employee\_id || ', ' || 'first\_name: ' || :NEW.first\_name || ', ' ||**

**'salary: ' || :NEW.salary;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, new\_values) VALUES ('INSERT', 'employees', :NEW.employee\_id, USER, v\_new\_values);**

# ELSIF UPDATING THEN

**v\_old\_values := 'employee\_id: ' || :OLD.employee\_id || ', ' || 'first\_name: ' || :OLD.first\_name || ', ' ||**

**'salary: ' || :OLD.salary;**

**v\_new\_values := 'employee\_id: ' || :NEW.employee\_id || ', ' || 'first\_name: ' || :NEW.first\_name || ', ' ||**

**'salary: ' || :NEW.salary;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, old\_values, new\_values)**

**VALUES ('UPDATE', 'employees', :NEW.employee\_id, USER, v\_old\_values, v\_new\_values);**

# ELSIF DELETING THEN

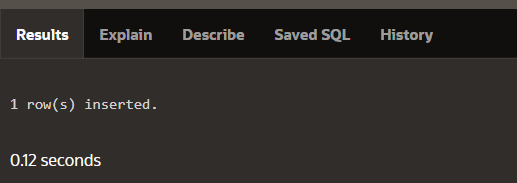
**v\_old\_values := 'employee\_id: ' || :OLD.employee\_id || ', ' || 'first\_name: ' || :OLD.first\_name || ', ' ||**

**'salary: ' || :OLD.salary; v\_new\_values := NULL;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, old\_values) VALUES ('DELETE', 'employees', :OLD.employee\_id, USER, v\_old\_values);**

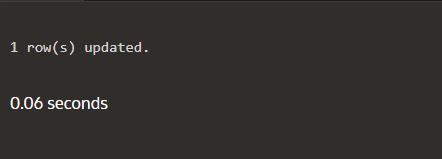
# END IF;

**INSERT INTO employees (employee\_id, first\_name, salary) VALUES (3, 'Ball', 50000);**



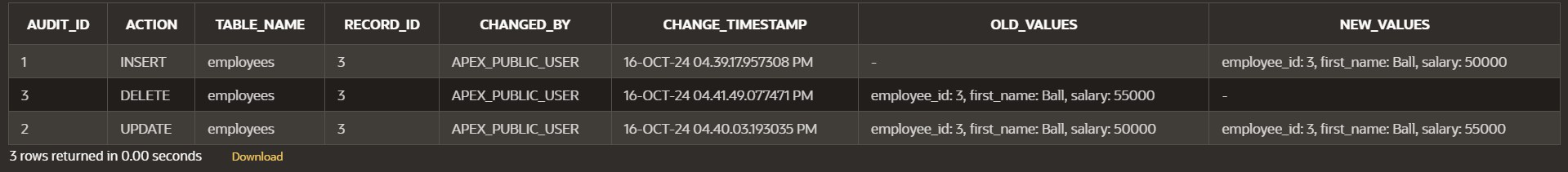
**UPDATE employees SET salary = 55000**

**WHERE employee\_id = 3;**



**DELETE FROM employees WHERE employee\_id = 3;**

**SELECT \* FROM audit\_log;**



**Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.**

**CREATE TABLE transactions ( transaction\_id NUMBER PRIMARY KEY, amount NUMBER,**

**running\_total NUMBER**

**);**

**CREATE OR REPLACE TRIGGER update\_running\_total FOR INSERT ON transactions**

# COMPOUND TRIGGER

**TYPE amount\_array IS TABLE OF NUMBER INDEX BY PLS\_INTEGER;**

**new\_amounts amount\_array;**

# BEFORE EACH ROW IS BEGIN

**new\_amounts(:NEW.transaction\_id) := :NEW.amount; END BEFORE EACH ROW;**

# AFTER STATEMENT IS BEGIN

**DECLARE**

**v\_total NUMBER; BEGIN**

**SELECT NVL(MAX(running\_total), 0) INTO v\_total**

**FROM transactions;**

**FOR i IN new\_amounts.FIRST .. new\_amounts.LAST LOOP v\_total := v\_total + new\_amounts(i);**

**UPDATE transactions**

**SET running\_total = v\_total WHERE transaction\_id = i;**

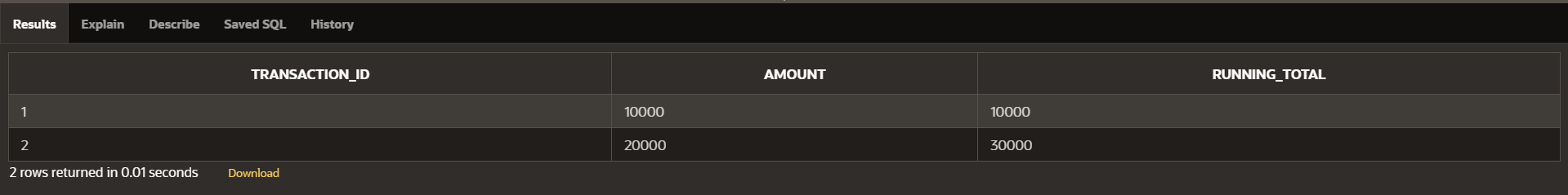
# END LOOP; END;

**END AFTER STATEMENT;**

**END update\_running\_total;**

**INSERT INTO transactions (transaction\_id, amount) VALUES (1, 10000);**

**INSERT INTO transactions (transaction\_id, amount) VALUES (2, 20000);**



**Program 7**

**Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.**

**CREATE TABLE inventory ( item\_id NUMBER PRIMARY KEY,**

**item\_name VARCHAR2(100), stock\_level NUMBER**

**);**

**CREATE TABLE orders (**

**order\_id NUMBER PRIMARY KEY, item\_id NUMBER,**

**quantity NUMBER, order\_status VARCHAR2(20),**

**CONSTRAINT fk\_item FOREIGN KEY (item\_id) REFERENCES inventory(item\_id)**

**);**

**CREATE OR REPLACE TRIGGER validate\_stock\_before\_order BEFORE INSERT ON orders**

# FOR EACH ROW DECLARE

**v\_stock\_level NUMBER; v\_pending\_orders NUMBER;**

# BEGIN

**SELECT stock\_level INTO v\_stock\_level FROM inventory**

**WHERE item\_id = :NEW.item\_id; SELECT NVL(SUM(quantity), 0) INTO v\_pending\_orders**

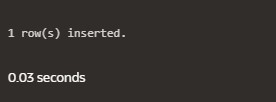
**FROM orders**

**WHERE item\_id = :NEW.item\_id AND order\_status = 'Pending';**

**IF (:NEW.quantity + v\_pending\_orders) > v\_stock\_level THEN RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient stock for item: ' || :NEW.item\_id);**

# END IF; END;

**INSERT INTO orders (order\_id, item\_id, quantity, order\_status) VALUES (1, 101, 5, 'Pending');**



**INSERT INTO orders (order\_id, item\_id, quantity, order\_status) VALUES (2, 103, 20, 'Pending');**

