

Ex.No:13

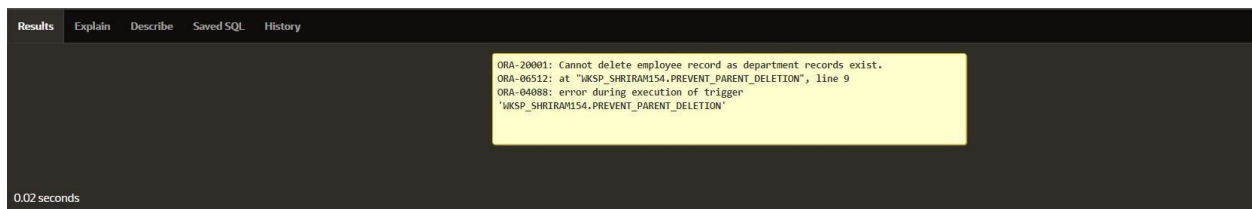
WORKING WITH TRIGGER**Program 1**

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;
```



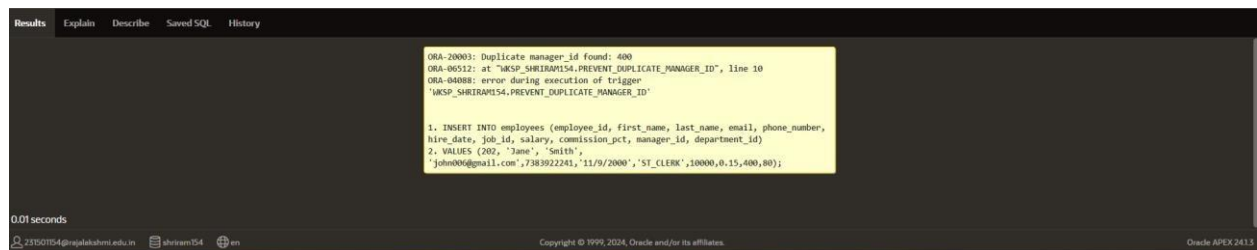
Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
CREATE OR REPLACE TRIGGER prevent_duplicate_manager_id
```

Program 2

```
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
```

Program 3

FOR EACH ROW

DECLARE

```
total_salary NUMBER; threshold
NUMBER := 100000; BEGIN
```

```
SELECT SUM(salary)
```

```
INTO total_salary
```

```
FROM employees;
```

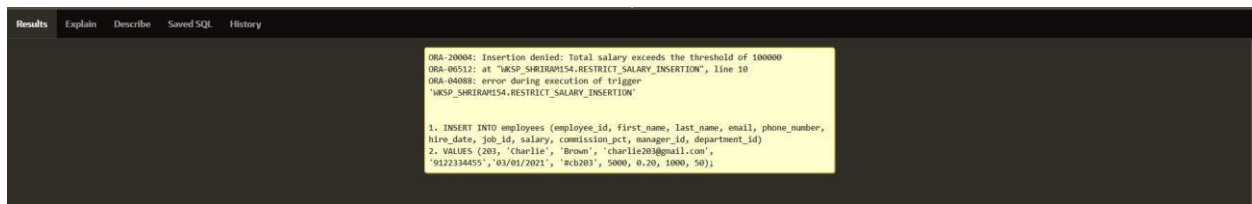
```
IF (total_salary + :NEW.salary) > threshold THEN
```

```
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
```

Program 4

```
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;
```

AUDIT_ID	EMPLOYEE_ID	OLD_SALARY	NEW_SALARY	OLD_JOB_ID	NEW_JOB_ID	CHANGE_TIMESTAMP	CHANGED_BY
1	20	50000	55000	manager	manager	15-OCT-24 10:00:00.000000 AM	admin
2	122	60000	65000	Manager	Manager	15-OCT-24 10:15:00.000000 AM	admin
5	27	45000	47000	Analyst	Senior Analyst	15-OCT-24 10:30:00.000000 AM	user1
22	176	7500	55000	#ce005	ST_CLERK	16-OCT-24 04:25:06.252580 PM	APEX_PUBLIC_USER
3	9	70000	75000	Senior Developer	Lead Developer	15-OCT-24 10:45:00.000000 AM	user2
4	4	80000	85000	Team Lead	Project Manager	15-OCT-24 11:00:00.000000 AM	admin

6 rows returned in 0.00 seconds [Download](#)

Program 5

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;
END trg_audit_employees;

```

```

INSERT INTO employees (employee_id, first_name, salary)
VALUES (3, 'Ball', 50000);

```

Results	Explain	Describe	Saved SQL	History
1 row(s) inserted.				
0.12 seconds				

```

UPDATE employees
SET salary = 55000
WHERE employee_id = 3;

```

1 row(s) updated.				
0.06 seconds				

```

DELETE FROM employees
WHERE employee_id = 3;

```

```
SELECT * FROM audit_log;
```

AUDIT_ID	ACTION	TABLE_NAME	RECORD_ID	CHANGED_BY	CHANGE_TIMESTAMP	OLD_VALUES	NEW_VALUES
1	INSERT	employees	3	APEX_PUBLIC_USER	16-OCT-24 04:39:17:957308 PM	-	employee_id: 3, first_name: Ball, salary: 50000
3	DELETE	employees	3	APEX_PUBLIC_USER	16-OCT-24 04:41:49:077471 PM	employee_id: 3, first_name: Ball, salary: 55000	-
2	UPDATE	employees	3	APEX_PUBLIC_USER	16-OCT-24 04:40:03:193035 PM	employee_id: 3, first_name: Ball, salary: 50000	employee_id: 3, first_name: Ball, salary: 55000

3 rows returned in 0.00 seconds [Download](#)

Program 6

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);

```

TRANSACTION_ID	AMOUNT	RUNNING_TOTAL
1	10000	10000
2	20000	30000

2 rows returned in 0.01 seconds [Download](#)

Program 7

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');

```

```

1 row(s) inserted.

0.03 seconds

```

```

INSERT INTO orders (order_id, item_id, quantity, order_status)
VALUES (2, 103, 20, 'Pending');

```

```

ORA-20001: Insufficient stock for item: 103
ORA-06512: at "WKSP_SHRIRAM154.VALIDATE_STOCK_BEFORE_ORDER", line 15
ORA-04088: error during execution of trigger
'WKSP_SHRIRAM154.VALIDATE_STOCK_BEFORE_ORDER'

1. INSERT INTO orders (order_id, item_id, quantity, order_status)
2. VALUES (2, 103, 20, 'Pending');

```

ITEM_ID	ITEM_NAME	STOCK_LEVEL
101	hp_laptop	10
102	keyboard	20
103	mouse	15

5 rows returned in 0.01 seconds [Download](#)

ORDER_ID	ITEM_ID	QUANTITY	ORDER STATUS
1	101	5	Pending

1 rows returned in 0.01 seconds [Download](#)