

```
-----  
-- PROGRAM 1 : Trigger to Enforce Referential Integrity  
-- Prevent deletion of a parent record if child records exist  
-----
```

```
-- Parent and child tables
```

```
CREATE TABLE department (  
    dept_id NUMBER PRIMARY KEY,  
    dept_name VARCHAR2(50)  
);
```

```
CREATE TABLE employee (  
    emp_id NUMBER PRIMARY KEY,  
    emp_name VARCHAR2(50),  
    dept_id NUMBER REFERENCES department(dept_id)  
);
```

```
-- Trigger
```

```
CREATE OR REPLACE TRIGGER trg_prevent_parent_delete  
BEFORE DELETE ON department  
FOR EACH ROW  
DECLARE
```

```
    v_count NUMBER;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO v_count
```

```
    FROM employee
```

```
    WHERE dept_id = :OLD.dept_id;
```

```
    IF v_count > 0 THEN
```

```
        RAISE_APPLICATION_ERROR(-20001, 'Cannot delete department with existing  
employees.');
```

```
    END IF;
```


END;

/

-- PROGRAM 2 : Trigger to Prevent Duplicate Values in a Column

-- Table for duplicate check

```
CREATE TABLE books (  
    book_id NUMBER PRIMARY KEY,  
    title VARCHAR2(100),  
    isbn VARCHAR2(20)  
);
```

-- Trigger to check duplicate ISBN

```
CREATE OR REPLACE TRIGGER trg_check_duplicate_isbn  
BEFORE INSERT OR UPDATE ON books  
FOR EACH ROW  
DECLARE  
    v_count NUMBER;  
BEGIN  
    SELECT COUNT(*) INTO v_count  
    FROM books  
    WHERE isbn = :NEW.isbn  
    AND (book_id != :NEW.book_id OR :NEW.book_id IS NULL);  
  
    IF v_count > 0 THEN  
        RAISE_APPLICATION_ERROR(-20002, 'Duplicate ISBN not allowed!');  
    END IF;  
END;
```

/

-- PROGRAM 3 : Trigger to Restrict Insertion if Total Value Exceeds Threshold

-- Table for sales

```
CREATE TABLE sales (  
    sale_id NUMBER PRIMARY KEY,  
    amount NUMBER  
);
```

-- Trigger to restrict total amount > 10000

```
CREATE OR REPLACE TRIGGER trg_restrict_total_sales  
BEFORE INSERT ON sales  
DECLARE  
    v_total NUMBER;  
    v_threshold CONSTANT NUMBER := 10000;  
BEGIN  
    SELECT NVL(SUM(amount),0) INTO v_total FROM sales;  
  
    IF (v_total + :NEW.amount) > v_threshold THEN  
        RAISE_APPLICATION_ERROR(-20003, 'Cannot insert: Total sales exceed 10000.');    END IF;  
END;  
/
```

-- PROGRAM 4 : Trigger to Capture Column Changes into an Audit Table

-- Main table

```
CREATE TABLE employees_audit_test (  
    emp_id NUMBER PRIMARY KEY,  
    emp_name VARCHAR2(50),  
    salary NUMBER  
);
```

-- Audit table

```
CREATE TABLE employees_audit_log (
```



```

    audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
    emp_id NUMBER,
    old_salary NUMBER,
    new_salary NUMBER,
    changed_on DATE,
    changed_by VARCHAR2(50)
);

-- Trigger to capture changes
CREATE OR REPLACE TRIGGER trg_audit_salary_change
AFTER UPDATE OF salary ON employees_audit_test
FOR EACH ROW
BEGIN
    INSERT INTO employees_audit_log(emp_id, old_salary, new_salary, changed_on,
    changed_by)
        VALUES(:OLD.emp_id, :OLD.salary, :NEW.salary, SYSDATE, USER);
END;
/

-----

-- PROGRAM 5 : Trigger to Record User Activity (INSERT, UPDATE, DELETE)

-----

-- Table to monitor
CREATE TABLE product (
    product_id NUMBER PRIMARY KEY,
    product_name VARCHAR2(50),
    price NUMBER
);

-- Audit log table
CREATE TABLE product_audit (
    log_id NUMBER GENERATED ALWAYS AS IDENTITY,
    operation_type VARCHAR2(10),
    product_id NUMBER,

```



```

product_name VARCHAR2(50),
price NUMBER,
action_date DATE,
performed_by VARCHAR2(50)
);

-- Trigger to record user activity
CREATE OR REPLACE TRIGGER trg_product_activity
AFTER INSERT OR UPDATE OR DELETE ON product
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO product_audit(operation_type, product_id, product_name, price,
action_date, performed_by)
        VALUES('INSERT', :NEW.product_id, :NEW.product_name, :NEW.price, SYSDATE,
USER);

    ELSIF UPDATING THEN
        INSERT INTO product_audit(operation_type, product_id, product_name, price,
action_date, performed_by)
        VALUES('UPDATE', :NEW.product_id, :NEW.product_name, :NEW.price, SYSDATE,
USER);

    ELSIF DELETING THEN
        INSERT INTO product_audit(operation_type, product_id, product_name, price,
action_date, performed_by)
        VALUES('DELETE', :OLD.product_id, :OLD.product_name, :OLD.price, SYSDATE,
USER);
    END IF;
END;
/

-----

-- END OF ALL PROGRAMS
-----

```


