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.

```
SQL> BEGIN
  2     EXECUTE IMMEDIATE '
  3        CREATE TABLE parent_table (
  4           parent_id NUMBER PRIMARY KEY,
  5           parent_name VARCHAR2(100)
  6        )';
  7  EXCEPTION
  8     WHEN OTHERS THEN
  9        IF SQLCODE = -955 THEN
 10           NULL; -- suppresses ORA-00955 exception if table already exists
 11        ELSE
 12           RAISE;
 13        END IF;
 14  END;
 15  /

PL/SQL procedure successfully completed.

SQL> BEGIN
  2     EXECUTE IMMEDIATE '
  3        CREATE TABLE child_table (
  4           child_id NUMBER PRIMARY KEY,
  5           child_name VARCHAR2(100),
  6           parent_id NUMBER,
  7           CONSTRAINT fk_parent FOREIGN KEY (parent_id) REFERENCES parent_table(parent_id)
  8        )';
  9  EXCEPTION
 10     WHEN OTHERS THEN
 11        IF SQLCODE = -955 THEN
 12           NULL; -- suppresses ORA-00955 exception if table already exists
 13        ELSE
 14           RAISE;
 15        END IF;
 16  END;
 17  /
```

```
SQL> BEGIN
  2     INSERT INTO parent_table (parent_id, parent_name) VALUES (1, 'Parent 1');
  3     INSERT INTO parent_table (parent_id, parent_name) VALUES (2, 'Parent 2');
  4     INSERT INTO parent_table (parent_id, parent_name) VALUES (3, 'Parent 3');
  5  EXCEPTION
  6     WHEN DUP_VAL_ON_INDEX THEN
  7        NULL; -- suppresses errors if data already exists
  8     WHEN OTHERS THEN
  9        RAISE;
 10  END;
 11  /

PL/SQL procedure successfully completed.

SQL> BEGIN
  2     INSERT INTO child_table (child_id, child_name, parent_id) VALUES (1, 'Child 1', 1);
  3     INSERT INTO child_table (child_id, child_name, parent_id) VALUES (2, 'Child 2', 1);
  4     INSERT INTO child_table (child_id, child_name, parent_id) VALUES (3, 'Child 3', 2);
  5     INSERT INTO child_table (child_id, child_name, parent_id) VALUES (4, 'Child 4', 3);
  6  EXCEPTION
  7     WHEN DUP_VAL_ON_INDEX THEN
  8        NULL; -- suppresses errors if data already exists
  9     WHEN OTHERS THEN
 10        RAISE;
 11  END;
 12  /

PL/SQL procedure successfully completed.
```

```
SQL> BEGIN
  2     EXECUTE IMMEDIATE '
  3        CREATE OR REPLACE TRIGGER prevent_parent_deletion
  4        BEFORE DELETE ON parent_table
  5        FOR EACH ROW
  6        DECLARE
  7           v_child_count NUMBER;
  8        BEGIN
  9           -- Check if there are any child records
 10           SELECT COUNT(*)
 11           INTO v_child_count
 12           FROM child_table
 13           WHERE parent_id = :OLD.parent_id;
 14
 15           -- If child records exist, raise an error
 16           IF v_child_count > 0 THEN
 17              RAISE_APPLICATION_ERROR(-20001, ''Cannot delete parent record; child records exist.'');
 18           END IF;
 19        END;';
 20  END;
 21  /

PL/SQL procedure successfully completed.

SQL> DELETE FROM parent_table WHERE parent_id = 1;
DELETE FROM parent_table WHERE parent_id = 1
            *
ERROR at line 1:
ORA-20001: Cannot delete parent record; child records exist.
ORA-06512: at "SYSTEM.PREVENT_PARENT_DELETION", line 12
ORA-04088: error during execution of trigger 'SYSTEM.PREVENT_PARENT_DELETION'
```

2. 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.

```
SET SERVEROUTPUT ON;

BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE employees (
        employee_id NUMBER PRIMARY KEY,
        employee_name VARCHAR2(100)
    )';
    DBMS_OUTPUT.PUT_LINE('Table created.');
END;
/

BEGIN
    INSERT INTO employees (employee_id, employee_name) VALUES (1, 'John Doe');
    INSERT INTO employees (employee_id, employee_name) VALUES (2, 'Jane Smith');
    INSERT INTO employees (employee_id, employee_name) VALUES (3, 'Alice Johnson');
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Initial values inserted.');
END;
/

BEGIN
    EXECUTE IMMEDIATE '
    CREATE OR REPLACE TRIGGER check_duplicate_employee_id
    BEFORE INSERT OR UPDATE ON employees
    FOR EACH ROW
    DECLARE
        v_count NUMBER;
    BEGIN
        -- Check for duplicate employee_id
        SELECT COUNT(*)
        INTO v_count
        FROM employees
        WHERE employee_id = :NEW.employee_id
        AND ROWID != :NEW.ROWID;

        IF v_count > 0 THEN
            RAISE_APPLICATION_ERROR(-20001, ''Duplicate employee_id value found.'');
        END IF;
    END;';
    DBMS_OUTPUT.PUT_LINE('Trigger created.');
END;
/
```

```
BEGIN

    INSERT INTO employees (employee_id, employee_name) VALUES (4, 'Bob Brown');
    DBMS_OUTPUT.PUT_LINE('Inserted employee_id 4: Success.');


    BEGIN
        INSERT INTO employees (employee_id, employee_name) VALUES (1, 'Chris Green');
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;

    COMMIT;
END;
/
```

Output:

```
Table created.

PL/SQL procedure successfully completed.

Initial values inserted.

PL/SQL procedure successfully completed.

Trigger created.

PL/SQL procedure successfully completed.

Inserted employee_id 4: Success.
Error: ORA-20001: Duplicate employee_id value found.
```

3. 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 TABLE my_table (
    id NUMBER PRIMARY KEY,
    value_column NUMBER
);

INSERT INTO my_table (id, value_column) VALUES (1, 200);
INSERT INTO my_table (id, value_column) VALUES (2, 300);
INSERT INTO my_table (id, value_column) VALUES (3, 400);

CREATE OR REPLACE TRIGGER check_total_value
BEFORE INSERT ON my_table
FOR EACH ROW
DECLARE
    total_value NUMBER;
    threshold CONSTANT NUMBER := 1000; -- Set your threshold value here
BEGIN

    SELECT NVL(SUM(value_column), 0) INTO total_value FROM my_table;

    total_value := total_value + :NEW.value_column;

    IF total_value > threshold THEN
        RAISE_APPLICATION_ERROR(-20001, 'Total value exceeds the allowed threshold of ' || threshold);
    END IF;
END;
/

BEGIN

    INSERT INTO my_table (id, value_column) VALUES (4, 50);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

```
BEGIN

    INSERT INTO my_table (id, value_column) VALUES (5, 1000);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Output:

```
SQL> @E:\DBMS\tigger_test3.sql

Table created.


1 row created.


1 row created.


1 row created.


Trigger created.


PL/SQL procedure successfully completed.

Error: ORA-20001: Total value exceeds the allowed threshold of 1000
```

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

```
SQL> CREATE TABLE employees (
  2      employee_id NUMBER PRIMARY KEY,
  3      first_name VARCHAR2(50),
  4      last_name VARCHAR2(50),
  5      salary NUMBER
  6  );

Table created.

SQL>
SQL> CREATE TABLE salary_audit (
  2      audit_id NUMBER PRIMARY KEY,
  3      employee_id NUMBER,
  4      old_salary NUMBER,
  5      new_salary NUMBER,
  6      changed_at TIMESTAMP,
  7      changed_by VARCHAR2(50)
  8  );

Table created.

SQL>
SQL> CREATE SEQUENCE salary_audit_seq
  2  START WITH 1
  3  INCREMENT BY 1
  4  NOCACHE
  5  NOCYCLE;

Sequence created.
```

```
CREATE OR REPLACE TRIGGER audit_salary_changes
AFTER UPDATE OF salary ON employees
FOR EACH ROW
DECLARE
    v_user VARCHAR2(50);
BEGIN

    SELECT USER INTO v_user FROM dual;

    INSERT INTO salary_audit (
        audit_id,
        employee_id,
        old_salary,
        new_salary,
        changed_at,
        changed_by
    ) VALUES (
        salary_audit_seq.NEXTVAL,  -- Use the sequence for audit_id
        :OLD.employee_id,
        :OLD.salary,
        :NEW.salary,
        SYSTIMESTAMP,
        v_user
    );
END;
/

INSERT INTO employees (employee_id, first_name, last_name, salary) VALUES (1, 'John', 'Doe', 50000);
INSERT INTO employees (employee_id, first_name, last_name, salary) VALUES (2, 'Jane', 'Smith', 60000);
COMMIT;
/

UPDATE employees SET salary = 55000 WHERE employee_id = 1;
COMMIT;
/
```

Output:

```
SQL> @E:\DBMS\trigger_test4.sql

Trigger created.


1 row created.


1 row created.


Commit complete.


Commit complete.


1 row updated.


Commit complete.


Commit complete.

SQL> SELECT * FROM salary_audit;

  AUDIT_ID EMPLOYEE_ID OLD_SALARY NEW_SALARY
---------- ----------- ---------- ----------
CHANGED_AT
---------------------------------------------------------------------------
CHANGED_BY
--------------------------------------------------
         1           1      50000      55000
20-MAY-24 02.40.59.351000 PM
SYSTEM
```

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 TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    department_id NUMBER
);

CREATE TABLE departments (
    department_id NUMBER PRIMARY KEY,
    department_name VARCHAR2(50)
);

CREATE TABLE audit_log (
    log_id NUMBER PRIMARY KEY,
    table_name VARCHAR2(50),
    operation VARCHAR2(10),
    user_name VARCHAR2(50),
    timestamp TIMESTAMP
);

CREATE SEQUENCE audit_log_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

CREATE OR REPLACE TRIGGER employees_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'employees', 'INSERT', USER, SYSTIMESTAMP);
    ELSIF UPDATING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'employees', 'UPDATE', USER, SYSTIMESTAMP);
    ELSIF DELETING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'employees', 'DELETE', USER, SYSTIMESTAMP);
    END IF;
END;
/
```

```
CREATE OR REPLACE TRIGGER departments_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON departments
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'departments', 'INSERT', USER, SYSTIMESTAMP);
    ELSIF UPDATING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'departments', 'UPDATE', USER, SYSTIMESTAMP);
    ELSIF DELETING THEN
        INSERT INTO audit_log (log_id, table_name, operation, user_name, timestamp)
        VALUES (audit_log_seq.NEXTVAL, 'departments', 'DELETE', USER, SYSTIMESTAMP);
    END IF;
END;
/
-- Insert values into the employees table
INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (1, 'John', 'Doe', 1);
INSERT INTO employees (employee_id, first_name, last_name, department_id) VALUES (2, 'Jane', 'Smith', 2);

-- Insert values into the departments table
INSERT INTO departments (department_id, department_name) VALUES (1, 'Engineering');
INSERT INTO departments (department_id, department_name) VALUES (2, 'Marketing');
```

Output:

```
SQL> @E:\DBMS\trigger_test5.sql

Table created.


Table created.


Table created.


Sequence created.


Trigger created.


Trigger created.


1 row created.


1 row created.


1 row created.


1 row created.
```

```
SQL> SELECT * FROM audit_log;

    LOG_ID TABLE_NAME                                          OPERATION
---------- -------------------------------------------------- ----------
USER_NAME
--------------------------------------------------
TIMESTAMP
---------------------------------------------------------------------------
         1 employees                                          INSERT
SYSTEM
20-MAY-24 02.51.53.491000 PM

         2 employees                                          INSERT
SYSTEM
20-MAY-24 02.51.53.494000 PM

    LOG_ID TABLE_NAME                                          OPERATION
---------- -------------------------------------------------- ----------
USER_NAME
--------------------------------------------------
TIMESTAMP
---------------------------------------------------------------------------

         3 departments                                        INSERT
SYSTEM
20-MAY-24 02.51.53.498000 PM

         4 departments                                        INSERT
SYSTEM

    LOG_ID TABLE_NAME                                          OPERATION
---------- -------------------------------------------------- ----------
USER_NAME
--------------------------------------------------
TIMESTAMP
---------------------------------------------------------------------------
20-MAY-24 02.51.53.502000 PM
```

6. 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 sales (
    sale_id NUMBER PRIMARY KEY,
    sale_amount NUMBER,
    running_total NUMBER
);

CREATE OR REPLACE TRIGGER update_running_total
BEFORE INSERT ON sales
FOR EACH ROW
BEGIN
    IF :new.sale_id = 1 THEN
        :new.running_total := :new.sale_amount;
    ELSE
        SELECT running_total + :new.sale_amount
        INTO :new.running_total
        FROM sales
        WHERE sale_id = (SELECT MAX(sale_id) FROM sales);
    END IF;
END;
/

INSERT INTO sales (sale_id, sale_amount) VALUES (1, 100);
INSERT INTO sales (sale_id, sale_amount) VALUES (2, 150);
INSERT INTO sales (sale_id, sale_amount) VALUES (3, 200);
```

Output:

```
SQL> @E:\DBMS\trigger_test7.sql

Table created.


Trigger created.


1 row created.


1 row created.


1 row created.

SQL> SELECT * FROM sales;

   SALE_ID SALE_AMOUNT RUNNING_TOTAL
---------- ----------- -------------
         1         100           100
         2         150           250
         3         200           450
```

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 items (
    item_id NUMBER PRIMARY KEY,
    item_name VARCHAR2(100),
    quantity NUMBER
);


INSERT INTO items (item_id, item_name, quantity) VALUES (1, 'Item A', 100);
INSERT INTO items (item_id, item_name, quantity) VALUES (2, 'Item B', 50);

CREATE TABLE orders (
    order_id NUMBER PRIMARY KEY,
    item_id NUMBER,
    quantity NUMBER,
    status VARCHAR2(20),
    CONSTRAINT fk_item FOREIGN KEY (item_id) REFERENCES items(item_id)
);

INSERT INTO orders (order_id, item_id, quantity, status) VALUES (1, 1, 30, 'Pending');
INSERT INTO orders (order_id, item_id, quantity, status) VALUES (2, 2, 20, 'Pending');

CREATE OR REPLACE TRIGGER validate_order_availability
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
    v_available_stock NUMBER;
    v_pending_orders NUMBER;
```

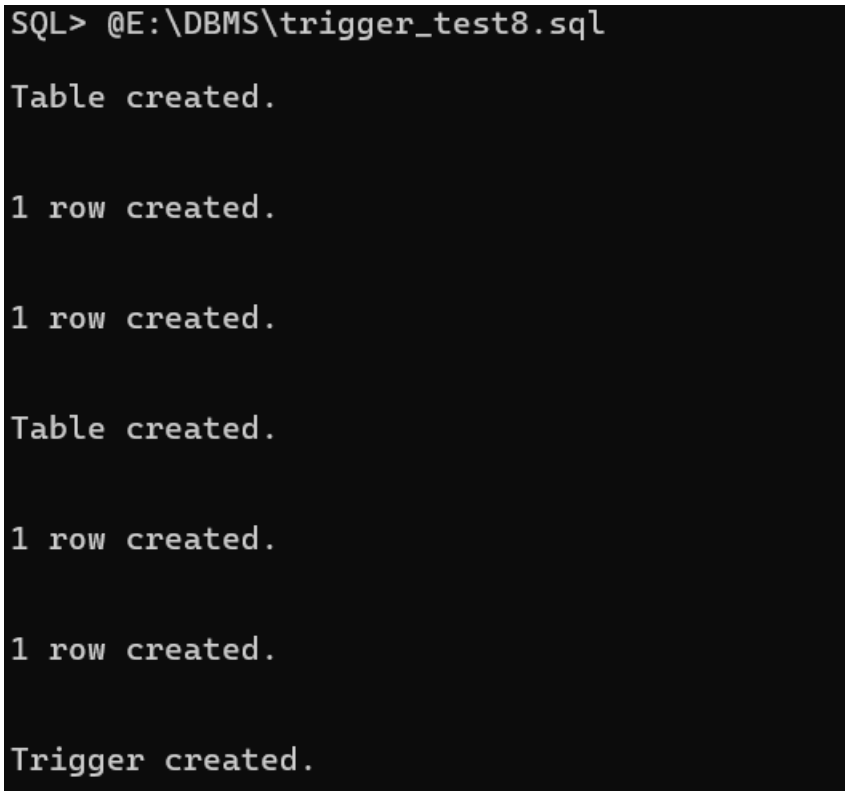
```
BEGIN

    SELECT (quantity - NVL(SUM(quantity), 0))
    INTO v_available_stock
    FROM items
    WHERE item_id = :new.item_id;

    SELECT NVL(SUM(quantity), 0)
    INTO v_pending_orders
    FROM orders
    WHERE item_id = :new.item_id
    AND status = 'Pending';

    IF (v_available_stock - v_pending_orders) < :new.quantity THEN
        RAISE_APPLICATION_ERROR(-20001, 'Not enough available stock to fulfill this order.');
    END IF;
END;
/
```

Output:


SQL> @E:\DBMS\trigger_test8.sql
Table created.
1 row created.
1 row created.
Table created.
1 row created.
1 row created.
Trigger created.