```
CREATE TABLE CLIENT_MASTER
(
CLIENT_NO VARCHAR2(6) PRIMARY KEY,
NAME VARCHAR2(20) NOT NULL,
ADDRESS1 VARCHAR2(30),
ADDRESS2 VARCHAR2(30),
CITY VARCHAR2(15),
STATE VARCHAR2(15),
PINCODE NUMBER(6),
BAL_DUE NUMBER(10,2)
);

--INSERT DATA IN CLIENT_MASTER

CREATE TABLE AUDITCLIENT
(
CLIENT_NO VARCHAR2(6),
NAME VARCHAR2(20),
BAL_DUE NUMBER(10,2),
OPERATION VARCHAR2(8),
USERID VARCHAR2(20),
ODATE DATE
);

--DON'T INSERT ANY DATA IN AUDITCLIENT

PROBLEM: Create a transparent audit systen for a table CLIENT_MASTER. The system
must keep track
-------- of the records that are being deleted or updated. The functionality being
when a record is deleted or modified the original record details and the date of
operation are stored in the audit table, then the delete or update is allowed to go
through.

SOLUTION:
---------
This trigger is fired when an update or delete is fired on the table client_master.
The trigger
first checks for the operation being performed on the table. Then depending on the
operation
being performed, a variable is assigned the value 'update' or 'delete'. Previous
values of the
modified record of the table client_master are stored into appropriate variables
declared. The
contents of these variables are then inserted into the audit table auditclient.


CREATE TRIGGER audit_trial
        AFTER UPDATE OR DELETE ON client_master
        FOR EACH ROW
DECLARE
        oper varchar2(8);
        client_no varchar2(6);
        name varchar2(20);
        bal_due number(10,2);
BEGIN
        IF updating THEN
                oper := 'update';
        END IF;

        IF deleting THEN
                oper := 'delete';
        END IF
```

```
        client_no := :old.client_no;
        name := :old.name
        bal_due := :old.bal_due;
        INSERT INTO auditclient
        VALUES(client_no, name, bal_due, oper, user, sysdate);
END;
```

```
/*
create table Employee(
ID                  VARCHAR2(4 BYTE)          NOT NULL,
First_Name          VARCHAR2(10 BYTE),
Last_Name           VARCHAR2(10 BYTE),
Start_Date          DATE,
End_Date            DATE,
Salary              Number(8,2),
City                VARCHAR2(10 BYTE),
Description         VARCHAR2(15 BYTE)
);
*/

DECLARE
      e_MissingNull EXCEPTION;
      PRAGMA EXCEPTION_INIT(e_MissingNull, -1400);
BEGIN
      INSERT INTO Employee (id) VALUES (NULL);
EXCEPTION
      WHEN e_MissingNull then
        DBMS_OUTPUT.put_line('ORA-1400: Cannot insert NULL');
END;
/
```

```
DECLARE
i number(2);
BEGIN
FOR i IN 1..10 LOOP
dbms_output.put_line(i);
END LOOP;
END;
/
```

```
DECLARE
        cursor c is select * from number_table;
BEGIN
        for num_row in c loop
        insert into doubles_table values(num_row.num*2);
        end loop;
END;
/
```

```
--Creating a function

create or replace function rating_message(rating IN NUMBER) return VARCHAR2
AS
BEGIN
IF rating > 7 THEN
return 'You are great';
ELSIF rating >= 5 THEN
return 'Not bad';
ELSE
return 'Pretty bad';
END IF;
END;
/


--Calling a function

declare
paulRate:=9;
Begin
dbms_output.put_line(ratingMessage(paulRate));
end;
/



--create or replace function squareNumber(num in number) Return number
--IS
--begin
--dbms_output.put('Square of ' || num || ' is: ');
--return num*num;
--end;
--/

--USE 1

--begin
--dbms_output.put_line('======================');
--dbms_output.put_line(squareNumber(3.5));
--dbms_output.put_line('======================');
--end;
--/

--USE 2

--declare
--n number;
--begin
--n := &enter_a_number;
--dbms_output.put_line('======================');
--dbms_output.put_line(squareNumber(n));
--dbms_output.put_line('======================');
--end;
--/
```

```
DECLARE
        a number;
        msg varchar2(50);
BEGIN
        a:=&percentage;
        if a >= 60 then
        msg := 'First class';
        elseif a >=50 then
        msg := 'Second class';
        elseif a >=40 then
        msg := 'Third class';
        else
        msg := 'Fail class';
        end if;
dbms_output.put_line(msg);
END;
/
```

```
DECLARE
        num_row number_table%ROWTYPE;
BEGIN
        select *
        into num_row
        from number_table;
        dbms_output.put_line(1/num_row.num);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                dbms_output.put_line('No data!');
        WHEN TOO_MANY_ROWS THEN
                dbms_output.put_line('Too many!');
        WHEN OTHERS THEN
                dbms_output.put_line(‘Error’);
END;
/
```

```
CREATE TABLE ITEM_MASTER
(
ITEMID NUMBER(4) PRIMARY KEY,
DESCRIPTION VARCHAR2(20),
BAL_STOCK NUMBER(3)
);

CREATE TABLE ITEM_TRANSACTION
(
ITEMID NUMBER(4),
DESCRIPTION VARCHAR2(20),
QTY NUMBER(3)
);
```

Based on the ITEMID in the ITEM_TRANSACTION table a check is made in the ITEM_MASTER table to
see if the ITEMID exists in the ITEM_MASTER table or not.

1. If the ITEMID does not exist then an insert operation is performed and the ITEMID along with
   the DESCRIPTION and QTY is inserted into the required columns of the ITEM_MASTER table.

2. If the ITEMID exists then a modify operation is performed and the qty is updated with the
   BAL_STOCK column of the table ITEM_MASTER where ITEMID of table ITEM_MASTER is same as that
   of ITEM_TRANSACTION.

To achieve this, a package composition of:

1. A function, which will check, for the existence of ITEMID in the table ITEM_MASTER. The
   function must have one argument which receives a value for which a matching pattern of ITEMID
   in the table ITEM_MASTER. The function will return value '1' indicating that a match is found
   and a value '0' indicating that no match is found. This value returned by the function can be
   used to perform the above operation.

2. A procedure that shall insert values in the ITEM_MASTER table in case the ITEMID does not exist in the ITEM_MASTER table.

3. A procedure that shall update values in the ITEM_MASTER table in case the ITEMID already exists in the ITEM_MASTER table.

Package Specification:

```
CREATE OR REPLACE PACKAGE check_data AS
        FUNCTION f_itemidchk(vitemidno IN number) RETURN number;
        PROCEDURE proc_update(vitemidno IN number, quantity IN number);
        PROCEDURE proc_insert(vitemidno IN number, quantity IN number, descrip IN
varchar2);
END check_data; -- End of package specification
```

Package Body:

```
CREATE OR REPLACE PACKAGE check_data IS
        FUNCTION f_itemidchk(vitemidno IN number) RETURN number IS
                dummyitem number(4);
        BEGIN
                        SELECT itemid INTO dummyitem
                        FROM item_master WHERE itemid = vitemid;
                        RETURN 1;
        EXCEPTION
                        WHEN no_data_found THEN
                        RETURN 0;
        END;                    -- End of function

        PROCEDURE proc_insert(vitemidno IN number, quantity IN number, descrip IN
varchar2) IS
        BEGIN
                        INSERT INTO item_master(itemid, bal_stock, description)
                        VALUES(vitemidno, quantity, descrip);
        END;                    -- End of procedure

        PROCEDURE proc_update(vitemidno IN number, quantity IN number) IS
        BEGIN
                        UPDATE item_master SET item_master.bal_stock = quantity
                        WHERE itemid = vitemidno;
        END;                    -- End of procedure
END check_data;         -- End of package body

-------------------------------------------
Calling the package in the PL/SQL code block:
-------------------------------------------

DECLARE
        CURSOR scannable IS
                SELECT itemid, qty, description
                FROM item_transaction;
        vitemidno number(4);
        descrip varchar2(30);
        quantity number(3);
        valexists number(1);
BEGIN
        OPEN scannable;
        LOOP
                FETCH scannable INTO vitemidno,quantity,descrip;

                EXIT WHEN scannable%NOTFOUND;

                valexists := check_data.f_itemidchk(vitemidno);
                DBMS_OUTPUT.PUT_LINE(TO_CHAR(valexists));

                IF valexists = 0 THEN
                        check_data.proc_insert(vitemidno,quantity,descrip);
                ELSE
                        check_data.proc_update(vitemidno,quantity);
                END IF;
        END LOOP;

        CLOSE scannable;

        COMMIT;
END;
```

```
/*
• Modes:
– IN: procedure must be called with a value for the parameter.
Value cannot be changed
– OUT: procedure must be called with a variable for the
parameter. Changes to the parameter are seen by the user
(i.e., call by reference)
– IN OUT: value can be sent, and changes to the parameter
are seen by the user
• Default Mode is: IN
*/

--Creating the procedure
/*
create or replace procedure num_logged
(person IN mylog.who%TYPE, num OUT mylog.logon_num%TYPE) IS
BEGIN
select logon_num
into num
from mylog
where who = person;
END;
/
*/
--Calling the procedure

declare
howmany mylog.logon_num%TYPE;
begin
num_logged('pete',howmany);
dbms_output.put_line(howmany);
end;
/
```

```
DECLARE
        less_than_target EXCEPTION;
        sman_no salesman_master.salesman_no%TYPE;
        tgt_sales salesman_master.tgt_to_get%TYPE;
        act_sales salesman_master.ytd_sales%TYPE;
        comm_rate salesman_master.rate_of_commission%TYPE;
BEGIN
        SELECT salesman_no, rate_of_commision, tgt_to_get, ytd_sales
                INTO sman_no, comm_rate, tgt_sales, act_sales
                FROM salesman_master
                WHERE salesman_no = &sman_no;

        IF act_sales < tgt_sales THEN
                RAISE less_than_target;
        ELSE
                INSERT INTO commission_payable
                VALUES(sman_no, sysdate, act_sales * comm_rate/100);
        END IF;
EXCEPTION
        WHEN less_than_target THEN
                DBMS_OUTPUT.PUT_LINE('Salesman No ' || sman_no || ' is not entitled
to get commission');
```

```
DECLARE
        e_number1 EXCEPTION;
        cnt NUMBER;
BEGIN
        select count(*)
        into cnt
        from number_table;

        IF cnt = 1 THEN RAISE e_number1;
        ELSE dbms_output.put_line(cnt);
        END IF;
EXCEPTION
        WHEN e_number1 THEN
                dbms_output.put_line('Count = 1');
END;
/
```

```
--Creating Sequence

Create Sequence mySeq
Start with 10
Increment By 5
Maxvalue 1000
Cycle;

--Alering a sequence

Alter Sequence mySeq
Increment By 3;

--Using Sequence

Insert Into myTable values(mySeq.Nextval,'col2');

--Dropping Sequence

Drop Sequence mySeq;
```

```
DECLARE
        cursor c is select * from number_table;
        cVal c%ROWTYPE;
BEGIN
        open c;
        LOOP
                fetch c into cVal;
                EXIT WHEN c%NOTFOUND;
                insert into doubles values(cVal.num*2);
        END LOOP;
END;
/
```

SQL TO SEE DIFFERENT TYPE OF OBJECTS

```
select object_name, object_type
from user_objects
where object_type in ( 'TABLE', 'PROCEDURE', 'FUNCTION','PACKAGE', 'PACKAGE BODY' )
and rownum < 50;
```

SQL TO SEE SOURCE OF AN OBJECT

```
SELECT TEXT
  FROM ALL_SOURCE
 WHERE NAME = 'SQUARENUMBER';
```

```
SET LINESIZE 132
SET LONG 4000
SELECT TRIGGER_BODY from user_triggers where trigger_name = 'FOO_TRIGGER'
```

```
declare
a number(2);
begin
a:=1;
while a<=10
loop
dbms_output.put_line(a);
a:=a+1;
end loop;
end;
/
```