**14a) Consider table Stud(Roll, Att,Status) Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “D”. Otherwise display message “Term granted” and set the status in stud table as “nd’’**

**Ans:**

create table stud(RollNo int primary key, attendance int,status varchar(5));

insert into stud values(1,150, NULL),(2,200, NULL),(3,80, NULL),(4,70, NULL),(5,180, NULL);

select \* from stud;

delimiter //

create procedure check\_att(in roll int)

begin

declare att int;

declare total int;

declare exit handler for not found select 'Data not found!!!' message;

set total=200;

select attendance into att from stud where RollNo=roll;

if ((att/total)\*100)>=75 then

update stud set status='ND' where RollNo=roll;

select 'Term Granted' Message;

else

update stud set status='D' where RollNo=roll;

select 'Term Not Granted' Message;

end if;

end;

//

call check\_att(1);

call check\_att(2);

call check\_att(3);

select \* from stud;

**14.B Write a PL/SQL block for following requirement using user defined exception handling. The account\_master table records the current balance for an account, which is updated whenever, any deposits or withdrawals takes place. If the withdrawal attempted is more than the current balance held in the account. The user defined exception is raised, displaying an appropriate message. Write a PL/SQL block for above requirement using user defined exception handling.**

create table account\_master(ID int primary key,Current\_balance int);

insert into account\_master values(1,10000)

insert into account\_master values (2,5000)

insert into account\_master values (3,60000);

select\*from account\_master;

delimiter //

create procedure withdraw(in acc\_id int,in amt int)

begin

declare bal int;

declare sp condition for sqlstate '45000';

select Current\_balance into bal from account\_master where ID=acc\_id;

if bal<amt then

signal sqlstate '45000'

set message\_text='NotEnoughBalance';

else

set bal = bal-amt;

update account\_master set Current\_balance=bal where ID=acc\_id;

end if;

end;

//

create procedure deposit(in acc\_id int,in amt int)

begin

declare bal int;

select current\_balance into bal from account\_master where ID=acc\_id;

update account\_master set current\_balance=bal+amt where ID=acc\_id;

end;

//

call withdraw(3,40000);

select\*from account\_master;

call deposit(2,2000);

select\*from account\_master;

call withdraw(1,75000);

**15A) Write an SQL code block these raise a user defined exception where business rule is**

**voilated. BR for client\_ master table specifies when the value of bal\_due field is less than 0**

**handle the exception.**

delimiter //

create procedure check\_br(in uid int)

begin

declare temp\_bal int;

declare sp condition for sqlstate'45000';

select bal\_due into temp\_bal from client\_master where id=uid;

if temp\_bal<0 then

signal sqlstate '45000'

set message\_text='BR violated';

else

select 'BR not violated' Message;

end if;

end

//

**15B)**

**Write an SQL code block**

**Borrow(Roll\_no, Name, DateofIssue, NameofBook, Status)**

**Fine(Roll\_no,Date,Amt)**

**Accept roll\_no & name of book from user. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. After submitting the book, status**

**will change from I to R. If condition of fine is true, then details will be stored into fine table. Also handles the exception by named exception handler or user define exception handler.**

create table borrow(

roll\_no int primary key,

name varchar(50),

dateofissue date,

nameofbook varchar(50),

status varchar(50));

create table fine(

roll\_no int primary key,

dateofreturn date,

amt int);

insert into borrow values(1,'A','2022-08-15','java','I');

insert into borrow values(2,'A','2022-08-05','cns','I');

insert into borrow values(3,'A','2022-08-01','dbms','I');

insert into borrow values(4,'A','2022-08-01','spos','I');

delimiter $

create procedure fine\_calculation(in rno int,bookname varchar(20))

begin

declare issuedate date;

declare diff int;

declare fine\_amt int;

declare exit handler for sqlexception select 'Table not Found';

select dateofissue into issuedate from borrow where roll\_no=rno and nameofbook=bookname;

select datediff (curdate(),issuedate) into diff;

if (diff>15 and diff<30) then

set fine\_amt = diff\*5;

insert into fine values(rno, curdate(),fine\_amt);

elseif(diff>30) then

set fine\_amt = diff\*50;

insert into fine values(rno, curdate(),fine\_amt);

elseif(diff<15) then

set fine\_amt = 0;

insert into fine values(rno, curdate(),fine\_amt);

end if;

update borrow set status ='R' where roll\_no=rno and nameofbook=bookname;

end;

$

call fine\_calculation(3,'dbms');

call fine\_calculation(4,'spos');

call fine\_calculation(2,'cns');

call fine\_calculation(1,'java');

select\*from fine;

select\*from borrow;

***16. Cursor (Any Two)***

***a) The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)***

**SQL> create table bank\_manager(**

**2 id number(3) not null primary key,**

**3 inactive\_days number(3)**

**4 );**

**Table created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (01,256);**

**1 row created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (02,456);**

**1 row created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (03,545);**

**1 row created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (04,222);**

**1 row created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (05,120);**

**1 row created.**

**SQL> insert into bank\_manager (id, inactive\_days) values (06,03);**

**1 row created.**

**SQL> select \* from bank\_manager;**

**ID INACTIVE\_DAYS**

**---------- -------------**

**1 256**

**2 456**

**3 545**

**4 222**

**5 120**

**6 3**

**6 rows selected.**

**SQL> alter table bank\_manager add status number(2) ;**

**Table altered.**

**SQL> select \* from bank\_manager;**

**ID INACTIVE\_DAYS STATUS**

**---------- ------------- ----------**

**1 256**

**2 456**

**3 545**

**4 222**

**5 120**

**6 3**

**6 rows selected.**

**SQL> edit**

**Wrote file afiedt.buf**

**1 declare**

**2 total\_rows number(3);**

**3 begin**

**4 update bank\_manager set status = 1 where inactive\_days>356;**

**5 if sql%notfound then**

**6 dbms\_output.put\_line('No Record Found');**

**7 elsifsql%found then**

**8 total\_rows := sql%rowcount;**

**9 dbms\_output.put\_line('Account Updated: '||total\_rows);**

**10 end if;**

**11\* end;**

**SQL> /**

**PL/SQL procedure successfully completed.**

**SQL> set serveroutput on;**

**SQL> /**

**Account Updated: 2**

**PL/SQL procedure successfully completed.**

**SQL> select \* from bank\_manager;**

**ID INACTIVE\_DAYS STATUS**

**---------- ------------- ----------**

**1 256**

**2 456 1**

**3 545 1**

**4 222**

**5 120**

**6 3**

**6 rows selected.**

***16b)Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment\_salary table.***

**Create table employee(e\_id int,salary int);**

**insert into employee values(1,10000);**

**insert into employee values(2,11000);**

**insert into employee values(3,12500);**

**insert into employee values(4,9000);**

**insert into employee values(5,10000);**

**insert into employee values(6,7000);**

**insert into employee values(7,12000);**

**create table increament\_t(**

**e\_id int,**

**salary int**

**);**

**Declare**

**Cursor c1 is select e\_id,salary from employee where salary < (select avg(salary) from employee);**

**me\_no employee.e\_id%type;**

**msalary employee.salary%type;**

**Begin**

**open c1;**

**if(c1%isopen) then**

**loop**

**fetch c1 into me\_no,msalary;**

**exit when c1%notfound;**

**if(c1%found) then**

**update employee set salary = salary + (salary \* 0.10) where e\_id = me\_no;**

**select salary into msalary from employee where e\_id = me\_no;**

**insert into increament\_t values(me\_no,msalary);**

**end if;**

**end loop;**

**end if;**

**end;**

**-----------------🡪>**

**Code 2:**

**SQL> create table employee2(**

**2 id number not null primary key,**

**3 name varchar2(20),**

**4 salary number(10,2) not null**

**5 );**

**Table created.**

**SQL> insert into employee2(id,name,salary) values (1,'Rushikesh',20000);**

**1 row created.**

**SQL> insert into employee2(id,name,salary) values (2,'Ritul',30000);**

**1 row created.**

**SQL> insert into employee2(id,name,salary) values (3,'Sanket',35000);**

**1 row created.**

**SQL> insert into employee2(id,name,salary) values (4,'Isha',40000);**

**1 row created.**

**SQL> insert into employee2(id,name,salary) values (5,'Kunal',25000);**

**1 row created.**

**SQL> insert into employee2(id,name,salary) values (6,'Ranjit',18000);**

**1 row created.**

**SQL> select \* from employee2;**

**ID NAME SALARY**

**---------- -------------------- ----------**

**1 Rushikesh 20000**

**2 Ritul 30000**

**3 Sanket 35000**

**4 Isha 40000**

**6 Ranjit 18000**

**6 rows selected.**

**SQL> edit**

**Wrote file afiedt.buf**

**1 declare**

**2 av\_salary number(10,2);**

**3 begin**

**4 av\_salary := &av\_salary;**

**5 update employee2 set salary = salary\*0.10 where salary <av\_salary;**

**6 if sql%found then**

**7 dbms\_output.put\_line('Rows Updated: '||sql%rowcount);**

**8 elsifsql%notfound then**

**9 dbms\_output.put\_line('No Record Found');**

**10 end if;**

**11\* end;**

**SQL> /**

**Enter value for av\_salary: 28000**

**old 4: av\_salary := &av\_salary;**

**new 4: av\_salary := 28000;**

**PL/SQL procedure successfully completed.**

**SQL> set serveroutput on;**

**SQL> /**

**Enter value for av\_salary: 28000**

**old 4: av\_salary := &av\_salary;**

**new 4: av\_salary := 28000;**

**Rows Updated: 3**

**PL/SQL procedure successfully completed.**

***c) Write PL/SQL block using explicit cursor for following requirements: College has decided to mark all those students detained (D) who are having attendance less than 75%. Whenever such update takes place, a record for the same is maintained in the D\_Stud table. create table stud21(roll number(4), att number(4), status varchar(1));***

**code final:**

**create table stud1(roll\_no number(5),attendance number(5),status varchar(7));**

**insert into stud1 values(8,76,'');**

**insert into stud1 values(2,74,'');**

**insert into stud1 values(3,77,'');**

**insert into stud1 values(4,73,'');**

**insert into stud1 values(5,78,'');**

**insert into stud1 values(6,72,'');**

**declare**

**cursor c1 is select roll\_no,attendance, from employee;**

**roll stud1.roll\_no%type;**

**attend stud1.attendance%type;**

**begin**

**open c1;**

**if(c1%isopen) then**

**loop**

**fetch c1 into roll,attend;**

**exit when c1%notfound;**

**if(attend > 75) then**

**update stud1 set status = 'ND' where roll\_no = roll;**

**else**

**update stud1 set status = 'D' where roll\_no = roll;**

**insert into D\_stud values(roll);**

**end if;**

**end loop;**

**end if;**

**end;**

--------------------------------------------------------------------------------------🡪

Code 2:

**SQL> create table stud21(**

**2 roll number(4) not null primary key,**

**3 att number(4) not null,**

**4 status varchar(1)**

**5 );**

**Table created.**

**SQL> insert into stud21 (roll,att) values (1,78);**

**1 row created.**

**SQL> insert into stud21 (roll,att) values (2,58);**

**1 row created.**

**SQL> insert into stud21 (roll,att) values (3,76);**

**1 row created.**

**SQL> insert into stud21 (roll,att) values (4,66);**

**1 row created.**

**SQL> insert into stud21 (roll,att) values (5,56);**

**1 row created.**

**SQL> insert into stud21 (roll,att) values (6,88);**

**1 row created.**

**SQL> create table d\_stud(**

**2 roll number(4) not null,**

**3 att number(4) not null,**

**4 status varchar(1)**

**5 );**

**Table created.**

**SQL> set linesize 160;**

**SQL> select \* from stud21;**

**ROLL ATT S**

**---------- ---------- -**

**1 78**

**2 58**

**3 76**

**4 66**

**5 56**

**6 88**

**6 rows selected.**

**SQL> declare**

**2 cursor stu\_cursor is**

**3 select roll,att from stud21 where att<75;**

**4 stud\_recordstu\_cursor%rowtype;**

**5 begin**

**6 open stu\_cursor;**

**7 loop**

**8 fetch stu\_cursor into stud\_record;**

**9 exit when stu\_cursor%notfound;**

**10 insert into d\_stud (roll,att) values (stud\_record.roll,stud\_record.att);**

**11 update stud21 set status = 'D' where roll = stud\_record.roll;**

**12 end loop;**

**13 end;**

**14 /**

**PL/SQL procedure successfully completed.**

**SQL> select \* from stud21;**

**ROLL ATT S**

**---------- ---------- -**

**1 78**

**2 58 D**

**3 76**

**4 66 D**

**5 56 D**

**6 88**

**6 rows selected.**

**SQL> select \* from d\_stud;**

**ROLL ATT S**

**---------- ---------- -**

**2 58**

**4 66**

**5 56**

**SQL>**

***17. Cursor (Any Two) a) The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)***

SQL> create table bank\_manager(

2 id number(3) not null primary key,

3 inactive\_days number(3)

4 );

Table created.

SQL> insert into bank\_manager (id, inactive\_days) values (01,256);

1 row created.

SQL> insert into bank\_manager (id, inactive\_days) values (02,456);

1 row created.

SQL> insert into bank\_manager (id, inactive\_days) values (03,545);

1 row created.

SQL> insert into bank\_manager (id, inactive\_days) values (04,222);

1 row created.

SQL> insert into bank\_manager (id, inactive\_days) values (05,120);

1 row created.

SQL> insert into bank\_manager (id, inactive\_days) values (06,03);

1 row created.

SQL> select \* from bank\_manager;

ID INACTIVE\_DAYS

---------- -------------

1 256

2 456

3 545

4 222

5 120

6 3

6 rows selected.

SQL> alter table bank\_manager add status number(2) ;

Table altered.

SQL> select \* from bank\_manager;

ID INACTIVE\_DAYS STATUS

---------- ------------- ----------

1 256

2 456

3 545

4 222

5 120

6 3

6 rows selected.

SQL> edit

Wrote file afiedt.buf

1 declare

2 total\_rows number(3);

3 begin

4 update bank\_manager set status = 1 where inactive\_days>356;

5 if sql%notfound then

6 dbms\_output.put\_line('No Record Found');

7 elsifsql%found then

8 total\_rows := sql%rowcount;

9 dbms\_output.put\_line('Account Updated: '||total\_rows);

10 end if;

11\* end;

SQL> /

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> /

Account Updated: 2

PL/SQL procedure successfully completed.

SQL> select \* from bank\_manager;

ID INACTIVE\_DAYS STATUS

---------- ------------- ----------

1 256

2 456 1

3 545 1

4 222

5 120

6 3

6 rows selected.

**17 b) ..Write a PL/SQL block of code using parameterized Cursor, that will merge the data available**

**in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the**

**data in the first table already exist in the second table then that data should be skipped. output:**

**-- Write a PL/SQL block of code using parameterized Cursor, that will merge the data available**

**-- in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the**

**-- data in the first table already exist in the second table then that data should be skipped. output:**

**—-----------------\_???/>>>>>>>>>>>>>>>>>>**

create table n\_rollcall (roll int, name varchar(10));

insert into n\_rollcall values (2,'vishal'), (5,'pratik'), (6,'parth');

create table o\_rollcall (roll int, name varchar(10));

insert into o\_rollcall values (2,'vishal'), (4,'hettik'), (3,'kartik'), (1,'deepak'), (5,'pratik');

delimiter $

create procedure p3(in r1 int)

begin

declare r2 int;

declare exit\_loop boolean;

declare c1 cursor for select roll from o\_rollcall where roll>r1;

declare continue handler for not found set exit\_loop=true;

open c1;

loop1:loop

fetch c1 into r2;

if not exists(select \* from n\_rollcall where roll=r2)

then

insert into n\_rollcall select \* from o\_rollcall where roll=r2;

end if;

if exit\_loop

then

close c1;

leave loop1;

end if;

end loop loop1;

end;

$

call p3(2);

select\*from n\_rollcall;

**code 2:**

**Code 1:**

**create table o\_rollcall(**

**roll\_no int,**

**name varchar(20)**

**);**

**create table n\_rollcall(**

**roll\_no int,**

**name varchar(20)**

**);**

**create procedure n4(IN rno1 int)**

**begin**

**declare rno2 int;**

**declare exit\_cond boolean;**

**declare c1 cursor for select roll\_no from o\_rollcall where roll\_no>rno1;**

**open c1;**

**l1:loop**

**fetch c1 into rno2;**

**if not exists(select \* from n\_rollcall where roll\_no = rno2) then**

**insert into n\_rollcall select \* from o\_rollcall where roll\_no = rno2;**

**end if;**

**if exit\_cond then**

**close c1;**

**leave l1;**

**end if;**

**end loop l1;**

**end;**

**17cWrite the PL/SQL block for following requirements using parameterized Cursor: Consider**

**table EMP(e\_no, d\_no, Salary), department wise average salary should be inserted into new table dept\_salary(d\_no, Avg\_salary)**

mysql> delimiter //

mysql> create procedure check\_salary()

-> begin

-> declare temp\_emp int;

-> declare temp\_dno int;

-> declare temp\_salary int;

-> declare avg\_salary int;

-> declare temp\_dno\_dept\_salary int;

-> declare ec boolean;

-> declare cur1 cursor for select avg(salary),dno from emp group by dno;

-> declare continue handler for not found set ec=true;

-> open cur1;

-> l1:loop

-> fetch cur1 into temp\_salary,temp\_dno;

-> insert into dept\_salary values(temp\_salary,temp\_dno);

-> if ec then

-> close cur1;

-> leave l1;

-> end if;

-> end loop l1;

-> end

-> //

**Code 2:**

**create table EMP(e\_no int, d\_no int, Salary int);**

**create table dept\_salary(d\_no int, Avg\_salary int);**

**CREATE or REPLACE PROCEDURE avgsal**

**IS**

**CURSOR department is**

**select d\_no,AVG(salary) as avgsal from emp**

**group by d\_no;**

**BEGIN**

**FOR data in department**

**LOOP**

**insert into dept\_salary values(data.d\_no,data.avgsal);**

**end LOOP;**

**end;**

**18. TRIGGER:**

**18.A**

**Write a update, delete trigger on clientmstr table. The System should keep track of the records that ARE BEING updated or deleted. The old value of updated or deleted records should be added in audit\_trade table. (separate implementation using both row and statement triggers).**

**Code 1**

**create or replace trigger storedata**

**before delete**

**on**

**clientmstr**

**for each row**

**begin**

**insert into audit\_trade values(old.c\_id,c\_name);**

**end;**

**/**

**create or replace trigger storedata**

**before update**

**on**

**clientmstr**

**for each row**

**begin**

**insert into audit\_trade values(old.c\_id,c\_name);**

**end;**

mysql> CREATE TABLE LIB\_AUDIT(RNO INT,

-> B\_TITLE VARCHAR(20),

-> ACTION VARCHAR(20));^C

mysql> CREATE TABLE BOOKS(RNO INT,

-> B\_TITLE VARCHAR(20));

Query OK, 0 rows affected (0.04 sec)

mysql>

mysql> CREATE TABLE LIB\_AUDIT(RNO INT,

-> B\_TITLE VARCHAR(20),

-> ACTION VARCHAR(20));

mysql> INSERT INTO BOOKS VALUES(1, 'ABC');

Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO BOOKS VALUES(2, 'DEF');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO BOOKS VALUES(3, 'GHI');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO BOOKS VALUES(4, 'JKL');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO BOOKS VALUES(5, 'MNO');

Query OK, 1 row affected (0.01 sec)mysql> SELECT \*FROM BOOKS;

mysql> SELECT \*FROM LIB\_AUDIT;

Empty set (0.00 sec)

mysql> DELIMITER $

mysql> CREATE TRIGGER before\_book\_delete

-> AFTER DELETE

-> ON books

-> FOR EACH ROW

-> BEGIN

-> INSERT INTO LIB\_AUDIT

-> SET action ='DELETE',

-> RNO=OLD.RNO,

-> B\_TITLE=OLD.B\_TITLE;

-> END;

-> $

Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> DELETE FROM BOOKS WHERE RNO = 1;

Query OK, 1 row affected (0.01 sec)

mysql> SELECT \*FROM BOOKS;

1 row in set (0.00 sec)

mysql> DELIMITER $

mysql> CREATE TRIGGER before\_book\_update

-> BEFORE UPDATE

-> ON BOOKS

-> FOR EACH ROW

-> BEGIN

-> INSERT INTO LIB\_AUDIT

-> SET action ='UPDATE',

-> RNO=NEW.RNO,

-> B\_TITLE=NEW.B\_TITLE;

-> END;

-> $

Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;

mysql> UPDATE BOOKS SET B\_TITLE = 'XYZ' WHERE RNO = 2;

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT\* FROM BOOKS;

**18 a or —->**

18A

delimiter //

create trigger after\_delete

after delete on client\_master

for each row

begin

insert into audit\_table

set action='DELETE',

id=old.id,

data=old.data;

end

//

delimiter //

create trigger after\_update

after update on client\_master

for each row

begin

insert into audit\_table

set action='UPDATE',

id=old.id,

data=old.data;

end

//

18B

**B)Write a before trigger for Insert, update event considering following requirement: Emp(e\_no, e\_name, salary) I) Trigger action should be initiated when salary is tried to be inserted is less than Rs. 50,000/- II) Trigger action should be initiated when salary is tried to be updated for value less than Rs. 50,000/- Action should be rejection of update or Insert operation by displaying appropriate error message. Also the new values expected to be inserted will be stored in new table Tracking(e\_no, salary).**

delimiter //

create trigger after\_insert

after insert

on emp

for each row

begin

if(new.salary<50000) then

signal sqlstate '45000' set message\_text='Rejected!!!';

end if;

insert into tracking

set eno=new.eno,

salary=new.salary;

end

//

**18b).\_\_\_\_\_\_\_>Write a before trigger for Insert, update event considering following requirement:**

**Emp(e\_no, e\_name, salary) I) Trigger action should be initiated when salary is tried to be**

**inserted is less than Rs. 50,000/- II) Trigger action should be initiated when salary is tried to be**

**updated for value less than Rs. 50,000/- Action should be rejection of update or Insert**

**operation by displaying appropriate error message. Also the new values expected to be inserted**

**will be stored in new table Tracking(e\_no, salary)**

**code :**

**create or replace trigger emp\_sal\_trigger**

**before insert or update on emp**

**for each row**

**begin**

**if:NEW.salary < 50000 then**

**RAISE\_APPLICATION\_ERROR(-20001,'MINIMUM SALARY SHOULD BE 50000');**

**end if;**

**end;**

CREATE TABLE Employee

(

Id INT PRIMARY KEY,

Name VARCHAR(45),

Salary INT,

Gender VARCHAR(12),

DepartmentId INT

)

CREATE TABLE Audit2

(

Salary INT

) ;

INSERT INTO Employee VALUES (1,'Steffan', 82000, 'Male', 3);

INSERT INTO Employee VALUES (2,'XYZ', 79000, 'Female', 4);

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON Employee

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

sal\_diff:= :OLD.salary;

dbms\_output.put\_line('New salary: ' || :NEW.salary);

insert into Audit2 values(sal\_diff);

END;

update Employee set salary=85080 where id=2;

select \* from Audit2;

**create or replace trigger emp\_sal\_trigger**

**before insert or update on emp**

**for each row**

**begin**

**if:NEW.salary < 50000 then**

**RAISE\_APPLICATION\_ERROR(-20001,'MINIMUM SALARY SHOULD BE 50000');**

**end if;**

**end;**