20/9/21
Exp: 9

Cursors

Question:

Aim:- The HRD manager has decided to raise the salary for all employees in deptno 20 by 5%. whenever such raise is given a record having the same details is updated in emp_raise table. Write a Postgres SQL block to update the salary of each employee and insert a record in emp_raise table.

Queries:-

```
#Create table employee(empid serial primary key, empname
 varchar(20), deptno int, salary numeric(10,2));
#insert into employee values(302,'martin',13,14000);
#select * from employees;
#Create or replace function empsalary(dno int) returns text
 as $$
#declare
$# id int;
$# name text default '';
$# salary int;
$# deptn int;
$# rec_employee record;
$# cursor_employee cursor(dno int) for select * from employee
   where deptno=dno;
$# begin
$# open cursor_employee(dno);
$# loop
$# fetch cursor_employee into id,name,deptn,salary;
$# exit when not found;
$# insert into emp_raise values(id,name,deptn,salary+
   (salary*0.05));
$# update employee set salary=(employee.salary+
```

(employee.salary * 0.05)) where current of cursor_
employee;
end loop;
close cursor_employee;
return cursor_employee;
end;
$$ language plpgsql;
select empsalary(20);
select * from emp_raise;
select * from employee;

Question 7

Aim:- Create a table STUDENT having 3 fields, (regno, University marks, sessionals) if sessionals is in between 30 and 34 then give necessary moderation so that it comes upto 35. If university marks + sessionals greater than 75, then insert those tuples into a table PASSLIST. Else insert into table FAILLIST.

Queries

create table student(regno int primary key, mark int, sessionals int);
insert into student values (601, 40, 12);
:
select * from student;
create or replace function result() returns text as $$
declare
mregint;
mmarks int;
msess int;
cursor_result cursor for select * from student;
begin
open cursor_result;

```
$$ loop
$$ fetch cursor_result into mreg,mmarks,msess;
$$ exit when not found;
$$ if(msess>29 and msess<35) then update student set sessionals
   ='35' where current of cursor_result;
$$ endif;
$$ if (msess+mmarks<75) then insert into faillist
   values (mreg, (msess+mmarks));
$$ endif
$$ if(msess+mmarks>75) then insert into passlist values (mreg
   (msess+mmarks));
$$ end if;
$$ end loop;
$$ close cursor_result;
$$ return cursor_result;
$$ end;
$$ $$ language plpgsql;
## select result();
## select * from passlist;
## select * from faillist;
## select * from student;
```

Result:

output obtained successfully.

31/8/24

Experiment -10

Exception

Question-1

Aim:-

Write an exception block to illustrate division by zero.

Queries

```
#Create or replace function division (arg int) returns text as $$
$$ declare
$$ res int;
$$ begin
$$ res = 100/arg;
$$ return;
$$ exception
$$ when division_by_zero
$$ then return 404;
$$ end;
$$ $$ language plpgsql;
# select division(0);
# select division(100);
```

Question-2

Aim:-

Create an employee table that consists of fields, emp-id, department and location. Write a function to delete an employee from table when emp-id is given as argument to the function. After deletion print a message "Employee record has been deleted succesfully". The program raises an exception if there is no employee that matches the users input.

Queries

```
## create table employee(empid int primary key, dept
 varchar(15), location varchar(20));
## insert into employee values(401, 'Sales', 'Thrissur');

## select * from employee;
## Create or replace function delete_emp(id int) returns
   text as $$
$$ begin
$$ delete from employee where empid=id;
$$ if not found then raise exception '';
$$ end if;
$$ return concat('employee record has been deleted
   successfully');
$$ exception
$$ when raise_exception then
$$ return concat('empid',id,' not found');
$$ end;
$$ $$ language plpgsql;
## select delete_emp(302);
## select delete_emp(401);
## select * from employee;
```

Result:-

Output obtained successfully.

19/21 Experiment - 11

TRIGGER

a)

Aim:-

To create a trigger that can add an entry in "Employee_Audit" table if a new employee record gets inserted into the "Employee" table.

Queries:-

#Create table employee (id int primary key, firstname varchar(70), lastname varchar(70), title varchar(70), reportsto int, birthdate timestamp, hiredate timestamp, address varchar(70), city varchar(20), state varchar(20), country varchar(20), postalcode varchar(20), phone varchar(20), fax varchar(20), email varchar(20));

Create table employee_audit (id int primary key, firstname varchar(10), lastname varchar(10), username varchar(10), empadditiontime varchar(0));

create or replace function emp_insert() return trigger as $$

$# begin

$# Insert into employee_audit (id, firstname, lastname, username, empadditiontime)

$# values(new.id, new.firstname, current_user, current_date);

$# return new;

$# end;

$# $$ language plpgsql;

=# create trigger emp_insert

-# after insert

-# on employee

-# for each row

-# execute procedure emp_insert();

#insert into employee values (13, 'Martin', 'Louis', 'Manager',

'3', '1999-07-11 00:00:00', '2020-08-01 00:00:00', '565 2nd', 'Thrissur', 'Kerala', 'India', '676 681', '7965 8435 876', '1645325', 'mlouis123@gmail.com');

-- select * from employee;

-- selat * from employee_audit;

Aim:-

(5) Crea

(6)

Aim:-

Create 2 tables student_mast and stu_log such that student_mast have 3 columns STUDENT_ID, NAME, ST_CLASS. stu_log table has 2 columns user_id and discription. Update to next class ie, 6 will be 7 7 will be 8 8 will be 9 and so on. After updating a single row in student_mast table a new row will be inserted in stu_log table where we will store the current user id and a small discription regarding the current update.

Queries

-- create table student_mast(student_id, int, name varchar(10), st_class int);

-- create table stu_log(user_id varchar(20), discription varchar(100));

-- insert into student_mast values (31, 'Martin', 6), (32, 'Vijay', 7) (33, 'Salman', 8), (34, 'Peter', 9);

-- select * from student_mast;

-- create or replace function aft_update()

-- returns trigger as $$

$ -- begin

$ -- insert into stu_log(user, concat('update student record', old.name, ' previous class: ', old.st_class, 'present class: ', new.st_class));

```
$$ return new;
$$ end;
$$ $$ language plpgsql;
# Create trigger update_log;
-# after update
-# on student_mast
-# for each row
-# excute procedure aft_update();
# Update student_mast set st_class=st_class+1;
# select * from stu_log;
```

()

Him:-

write a trigger to store some information in stu_log table after a delete operation happened on student_mast table.

Queries:-

```
# Create or replace function aft_delete() return as $$
$$ begin
$$ insert into stu_log values(user, concat('update student
    Record', old.name, 'class:', old.st_class, '-> Peleted on',
    now()));
$$ Return new;
$$ end
$$ $$ language plpgsql;
# Create trigger delet_stu
-# after delete
-# on student_mast
-# for each row
-# excute procedure after_delete();
# delete from student_mast where student_id=33;
-# select * from stu_log;
```

Result:-

Output obtained successfully

-10-21 Experiment -12

TRIGGER-2

Aim:-

Before and update

Create a table student_marks with the following fields

student_id | name | sub1 | sub2 | sub3 | sub4 | sub5 | total | per_marks | grade

We have a table students_marks with 10 columns and 4 rows. Then on data only in ~~student_id~~ STUDENT_ID and NAME Column

Now the exam is over and we have received all subject marks, now we will update the table, total marks of all subject, the percentage of total marks and grade will be automatically calculated. For this sample calculation, the following conditions are assumed:

-if PER_MARKS >= 90 -> 'EXCELLENT'

-if PER_MARKS >= 75 and PER_MARKS < 90 -> 'VERY GOOD'

-if PER_MARKS >= 60 and PER_MARKS < 75 -> 'GOOD'

-if PER_MARKS >= 40 and PER_MARK, < 60 -> 'AVERAGE'

-if PER_MARKS < 40 -> 'NOT PROMOTED'.

Create a trigger function to implement this.

Query

*Create table student_marks (student_id int primary key, name varchar(60), sub1 int, sub2 int, sub3 int, sub4 int, sub5 int, total int, percentage float, grade varchar(50));

```
# insert into student_marks(student_id, name) values(10,
  'albert'),(11, 'martin'),(12,'alan'),(13,'gokul');
# select * from student_marks;
# create or replace function before_update() returns
    trigger as $$
$# begin
$# new.total = new.sub1+new.sub2+new.sub3+new.sub4+
    new.sub5;
$# new.percentage = new.total/5;
$# if new.percentage >=90 then new.grade =
                              'excelent';
$# elseif new.percentage >=75 and new.percentage <90 then
    new.grade='very good';
$# elseif new.percentage >=60 and new.percentage <75 then
    new.grade='good';
$# elseif new.percentage >=40 and new.percentage <60 then
    new.grade='average';
$# else new.grade ='not promoted';
$# end if;
$# return new;
$# end;
$# $$ language plpgsql;
# create trigger upddt_marks
-# before update
-# on student_marks
-# for each row
-# execute procedure before_update();
# update student_marks set sub1=24, sub2=35, sub3=25
  sub4=34, sub5=24 where student_id=10;
```

update student_marks set sub1=50, sub2=44, sub3=45, sub4=55, sub5=46 where student_id=11;

update student_marks set sub1=90, sub2=94, sub3=92, sub4=94, sub5=97 where student_id=12;

update student_marks set sub1=76, sub2=80, sub3=74, sub4=79, sub5=80 where student_id=13;

select * from student_marks;

Result:-

output obtained successfully.