Bissenova Anara / 20BD / Lab8

Task 1. Create a function that:

a) Increments given values by 1 and returns it.
b) Returns sum of 2 numbers.
c) Returns true or false if numbers are divisible by 2.
d) Checks some password for validity.
e) Returns two outputs, but has one input.

```
create function inc(val integer) returns integer as $$
begin
return val+1;
end; $$
language plpgsql;
```

```
create function calc_sum(a integer, b integer) returns integer as $$
begin
return a+b;
end;$$
language plpgsql;
```

```
create function div(val integer) returns boolean as $$
begin
    if (val % 2 = 0) then
            return true;
        else
            return false;
    end if;
end;$$
language plpgsql;
```

```
create or replace function validation_checker(pass varchar(25)) returns boolean
language plpgsql as $$
begin
    if (length(pass) < 8) then return false;
        else return true;
    end if;
end; $$;
```

```
create or replace function two_output(str varchar(25), out a varchar(25), out b
varchar(25)) as $$
begin
     a = split_part(str, ' ', 1);
     b = split_part(str, ' ', 2);
end;$$
language plpgsql;
```

Task 2. Create a trigger that:

a) Return timestamp of the occurred action within the database.
b) Computes the age of a person when persons' date of birth is inserted.
c) Adds 12% tax on the price of the inserted item.
d) Prevents deletion of any row from only one table.

e) Launches functions 1.d and 1.e

```
create table info(
    id integer primary key,
    name varchar,
    course integer,
    age integer,
    birth_date date,
    changes timestamp(6)
);

CREATE FUNCTION last_changes()
   RETURNS TRIGGER
   LANGUAGE PLPGSQL
AS $$
BEGIN
   if new.course <> old.course then
       insert into info(id, name, course, age, birth_date, changes)
       values (old.id, old.name, new.course, old.age, old.birth_date, now());
   end if;
   return new;
END; $$;

CREATE TRIGGER l_changes
   BEFORE update
   ON info
   FOR EACH ROW
       EXECUTE PROCEDURE last_changes();
```

```
create or replace function age_count() returns trigger
language plpgsql as $$
begin
    update info
    set age = round((current_date - new.birth_date)/365.25)
    where id = new.id and birth_date <> null;
    return new;
end; $$;

create trigger ag_count
    after insert
    on info
    for each row
        execute procedure age_count();
```

```
create or replace function tax_price() returns trigger
language plpgsql as $$
begin
    update product
    set price = price*(1.12)
    where id = new.id;
    return new;
end; $$;

create trigger tax_pr
    after insert
    on product
    for each row
        execute procedure tax_price();
```

```
create or replace function del_prev() returns trigger
language plpgsql as $$
    begin
        raise exception 'you cannot delete data!';
    end; $$;

create trigger dl_pr
    after delete on product
    for each row
        execute procedure del_prev();

delete from product where price = 22 ;
```

```
create table profile(
    username varchar(25),
    firstlast varchar(25),
    password varchar(25),
    val boolean
);
create or replace function create_profile() returns trigger language plpgsql
    as $$
    begin
        if validation_checker(new.password)=true then
            update profile
            set val = true, firstlast = two_output(username)
            where username=new.username;
        else
            update profile
            set val = false
            where username=new.username;
        end if;
        return new;
    end;
    $$;

create trigger checkprofile
    after insert on profile
    for each row
        execute procedure create_profile();
```

Task 3. What is the difference between procedure and function?

Function:

- the function cannot call a stored procedure
- you can call functions from a select statement
- no transactions are allowed
- only select is allowed
- must return a result or value to the caller

Procedure:

- stored procedures can call functions as needed
- there is no provision to call procedures from select/having and where statements
- transactions can be used in stored procedures
- need not return any value
- all the database operations – insert, update, delete

Task 4. Create procedures that:

a) Increases salary by 10% for every 2 years of work experience and provides 10% discount and after 5 years adds 1% to the discount.
b) After reaching 40 years, increase salary by 15%. If work experience is more than 8 years, increase salary for 15% of the already increased value for work experience and provide a constant 20% discount.

```
create table worker(
    id integer primary key,
    name varchar,
    date_of_birth date null,
    age integer,
    salary integer,
    workexperience integer,
    discount integer);

create or replace procedure inc_sal()
language plpgsql as $$
    begin
        update worker set salary = salary*(1.1)^(workexperience/2), discount =
10 where (workexperience > 2);
        update worker set discount = discount + (workexperience/5) where
(workexperience > 5);
        commit;
    end;$$;
```

```
create or replace procedure age_inc()
language plpgsql as $$
    begin
        update worker set salary = salary*(1.15) where (age = 40);
        update worker set salary = salary*(1.15), discount = 20 where
(workexperience >= 8);
        commit;
    end; $$;
```

Task 5. Produce a CTE that can return the upward recommendation chain for any member. You should be able to select recommender from recommenders where member=x. Demonstrate it by getting the chains for members 12 and 22. Results table should have member and recommender, ordered by member ascending, recommender descending.

```
create table members(
    memid integer,
    surname varchar(200),
    firstname varchar(200),
    address varchar(300),
    zipcode integer,
    telephone varchar(20),
    recommendedby integer,
    joindate timestamp
);
create table bookings(
    facid integer,
    memid integer,
    starttime timestamp,
    slots integer
);
create table facilities(
    facid integer,
    name varchar(100),
```

```
    membercost numeric,
    guestcost numeric,
    initialoutlay numeric,
    monthlymaintenance numeric
);
with recursive recommenders (member,recommender) as (
   select memid, recommendedby
       from members
   union
   select members.memid, members.recommendedby
       from recommenders
           inner join members on members.recommendedby = recommenders.member
)

select *
from recommenders where member = 22 or member = 12
order by member asc, recommender desc;
```