Assignment 8, Svanov Zhantore.

--1.create function that:  
  
--a Increments given values by 1 and returns it.  
create function *inc*(num integer)  
 returns integer as $$  
 begin  
 return num+1;  
 end;  
 $$  
 language plpgsql;  
select *inc*(123);  
  
--b Returns cube of number  
create function *cube*(num integer)  
 returns integer as $$  
 begin  
 return num\*num\*num;  
 end;  
 $$  
 language plpgsql;  
select *cube*(3)  
  
--c Returns sum of 2 numbers.  
create function *sum*(a int, b int)  
 returns int as $$  
 begin  
 return a+b;  
 end;  
 $$  
 language plpgsql;  
select *sum*(2,3)  
  
--d Returns true or false if numbers are divisible by 2.  
create function *div2*(a int)  
 returns bool as $$  
 begin  
 return a%2=0;  
 end;  
 $$  
 language plpgsql;  
select *div2*(3);  
  
--e Average sum of numbers.  
create function *av\_sum*(variadic list int[],out total int, out av int)  
 as $$  
 begin  
 select into total *sum*(list[i])  
 from *generate\_subscripts*(list,1) g(i);  
  
 select into av *AVG*(list[i])  
 from *generate\_subscripts*(list,1) g(i);  
 end;  
 $$  
 language plpgsql;  
  
--f Returns count of numbers.  
create function *count\_num*(variadic list int[], out total int)  
 as $$  
 begin  
 select into total *count*(list[i])  
 from *generate\_subscripts*(list,1) g(i);  
 end;  
 $$  
 language plpgsql;  
  
select *count\_num*(1,2,3,4)  
-- g Checks some password for validity.  
create function *valid\_password*(password varchar)  
 returns boolean as $$  
 declare  
 size boolean = false;  
 num boolean = false;  
 symbols boolean = false;  
 up\_case boolean = false;  
 begin  
 size = *length*(password) > 8;  
 num = password ~ E'.\*\\d.\*';  
 symbols = password ~ E'.\*[!@#$%^&?]\*.';  
 up\_case = *lower*(password) != password;  
 return (size and num and symbols and up\_case);  
 end;  
 $$  
 language plpgsql;  
  
--h Returns two outputs, but has one input.  
create function *two*(a int, out aa1 int, out aa2 int)  
 as $$  
 begin  
 aa1 = a+a;  
 aa2 = a\*a;  
 end;  
 $$  
 language plpgsql;  
  
--2. Create trigger that:  
  
  
-- a. Return timestamp of the occurred action within the database.  
create table example(  
 age int,  
 b varchar  
);  
drop table example;  
create or replace function *get\_timestamp*()  
 returns trigger as $$  
 declare  
 curtime time = NULL;  
 begin  
 curtime = *current\_time*;  
 return NULL;  
 end;  
 $$  
 language plpgsql;  
  
drop trigger abc on example  
create trigger abc  
 after insert on example  
 for row execute function *get\_timestamp*();  
  
insert into example values (1,'1');  
select \* from example  
-- b. Computes the age of a person when persons’ date of birth is inserted.  
drop table person  
create table person (  
 birthDate date,  
 age integer  
);  
drop trigger age on person  
create trigger age  
 before insert on person  
 for row execute function *get\_age*();  
  
drop function *get\_age*create function *get\_age*()  
 returns trigger as $$  
 declare  
 compute\_age integer;  
 begin  
 compute\_age = *extract*(year from *age*(new.birthDate));  
 new.age = compute\_age;  
 return new;  
 end;  
 $$  
 language plpgsql;  
  
insert into person values (*now*());  
select \* from person  
-- c. Adds 12% tax on the price of the inserted item.  
create table c (  
 price int  
);  
drop trigger taxes on c  
create trigger taxes  
 before insert on c  
 for row execute function *add\_taxes*();  
  
create or replace function *add\_taxes*()  
 returns trigger as $$  
 declare new\_price int;  
 begin  
 new\_price = new.price + (new.price\*12)/100;  
 new.price = new\_price;  
 return new;  
 end;  
 $$  
 language plpgsql;  
  
insert into c values (41);  
select \* from c;  
-- d. Prevents deletion of any row from only one table.  
create table d(  
 data int  
);  
create trigger prevent\_del  
 before delete on d  
 for each row execute function *fake\_del*();  
  
create function *fake\_del*()  
 returns trigger as $$  
 begin  
 return null;  
 end;  
 $$  
 language plpgsql;  
  
-- e. Launches functions 1.d and 1.e.  
create table e(  
 data1 int,  
 data2 int,  
 data3 int  
);  
  
create trigger do\_e\_d  
 after insert on e  
 execute function *ex\_d\_e*();  
  
create or replace function *ex\_d\_e*()  
 returns trigger as $$  
 declare  
 divisable bool;  
 result record;  
 begin  
 divisable = *div2*(new.data1);  
 result = *av\_sum*(new.data1,new.data2,new.data3);  
 return NULL;  
 end;  
 $$  
 language plpgsql;  
  
insert into e values (1,2,3);  
  
--3 Create procedures that:  
drop table worker;  
create table worker(  
 id int primary key,  
 name varchar,  
 date\_of\_birth date,  
 age int,  
 discount int,  
 salary int,  
 duration int  
);  
--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.  
create or replace procedure *calc*()  
 language plpgsql  
 as $$  
 declare  
 row record;  
 new\_salary int;  
 new\_disc int;  
 begin  
 for row in select \* from worker  
 loop  
 if(row.duration%2 = 0) then  
 new\_salary = row.salary + (row.salary\*10/100);  
 new\_disc = row.discount + (row.discount\*10/100);  
 else  
 new\_salary = row.salary;  
 new\_disc = row.discount;  
 end if;  
  
 if(row.duration%5 = 0) then  
 new\_disc = row.discount + (row.discount\*1/100);  
 new\_salary = row.salary;  
 else  
 new\_salary = row.salary;  
 new\_disc = row.discount;  
 end if;  
 update worker set salary = new\_salary where id = row.id;  
 update worker set discount = new\_disc where id = row.id;  
 end loop;  
 end;  
 $$  
  
--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 constant20% discount.  
create or replace procedure *calc2*()  
 language plpgsql  
 as $$  
 declare  
 row record;  
 new\_salary int;  
 new\_disc int;  
 begin  
 for row in select \* from worker  
 loop  
 if(*extract*(year from *age*(row.date\_of\_birth)) >= 40) then  
 new\_salary = new\_salary + (new\_salary\*15/100);  
 end if;  
 if(row.duration > 8) then  
 new\_salary = new\_salary + (new\_salary\*15/100);  
 new\_disc = 20;  
 end if;  
 update worker set salary = new\_salary where id = row.id;  
 update worker set discount = new\_disc where id = row.id;  
 end loop;  
 end;  
 $$