```sql
-- create schema lab_8;
-- 1.Create a function that:
-- a. Increments given values by 1 and returns it.
create function increment(x numeric) returns integer as
    $$
    begin
        return x + 1;
    end;
    $$
language plpgsql;

-- select increment(67);

-- b. Returns sum of 2 numbers.
create function sum(x numeric, y numeric) returns integer as
    $$
    begin
        return x + y;
    end;
    $$
language plpgsql;
-- select sum(3 + 5);

-- c. returns true or false if numbers are divisible by 2.
create function parity (x integer) returns boolean as
    $$
    begin
        return (x % 2 = 0);
    end;
    $$
language plpgsql;

-- select parity(7632436);

-- d. Checks some password for validity.
create function validity(password varchar(20)) returns boolean as
    $$
    begin
        return length(password) > 12;
    end;
    $$
language plpgsql;

-- select validity('sdasd');

-- e. Returns two outputs, but has one input.
create or replace function split(location varchar(50), out country varchar(50), out
capital varchar(50)) as
    $$
    begin
        country := split_part(location, '-', 1);
        capital := split_part(location, '-', 2);
    end;
    $$
language plpgsql;

-- select split('Kazakhstan-Astana');

-- 2. Create a trigger that:
create table orders(
    client_name varchar(50),
    birthday date,
    age interval,
    product_id int,
    quantity int,
    price float,
    total float,
    address varchar(50)
);

insert into orders values ('Aisha', null, null, 1, 5, null, null, 'Kazakhstan-Almaty'),
                          ('Aru', null, null, 2, 1, null, null, 'Japan-Tokyo'),
                          ('Ainur', null, null, 5, 0, null, null, 'Russia-Moscow');
-- drop table orders;
create table modified(
    client_name varchar(50),
    name varchar(50),
    last_updated timestamp
);

-- a.Return timestamp of the occured action within the database
create function t_stamp() returns trigger as
    $$
    begin
        if new <> old then
            insert into modified values (old.client_name, new.quantity, now());
        end if;
        return new;
    end;
    $$
    language plpgsql;

create trigger refresh
    before update on orders
    for each row execute procedure t_stamp();

update orders set quantity = 6 where client_name = 'Aru';
update orders set quantity = 7 where client_name = 'Aisha';

select * from modified;

-- b.Computes the age of a person when persons’ date of birth is inserted
create or replace function compute() returns trigger as
    $$
    begin
        update orders set age = age(now(), birthday) where client_name =
new.client_name;
        return new;
    end;
    $$
    language plpgsql;

create trigger add_age
    after update of birthday on orders
    for each row execute procedure compute();


update orders set birthday = '12-05-2003' where client_name = 'Aisha';
update orders set birthday = '05-05-2003' where client_name = 'Aru';
update orders set birthday = '22-05-2002' where client_name = 'Ainur';

-- c. Adds 12% tax on the price of the inserted item
create or replace function tax() returns trigger as $$
    begin
        update orders set price = price * 1.12 where client_name = new.client_name;
        return new;
    end;
$$ language plpgsql;

create trigger compute_tax after update of price on orders
    for each row execute function tax();

update orders set price = 3.80 where client_name = 'Aisha';
```

```
delete from orders;
create or replace function prevent_del() returns trigger as
    $$
   begin
       raise exception 'data cannot be deleted';
   end;
   $$
language plpgsql;

create trigger prev_del before delete on orders
   for each row execute function prevent_del();




-- e.Launches functions  1.d and 1.e.
CREATE table account(
    id int,
    name varchar(50),
    password varchar(50),
    address text
);

CREATE table changes(
    id int,
    name varchar(50),
    valid bool,
    country varchar(50),
    city varchar(50)
);

create or replace function func() returns trigger as
    $$
    begin
        if validity(new.password) then
            insert into changes values (new.id, new.name, true,
(split(new.address)).country , (split(new.address)).capital);
        elsif not validity(new.password) then
            insert into changes values (new.id, new.name, false, null, null);
        end if;
        return new;
    end;
    $$
    language plpgsql;

create trigger trig
    after insert on account
    for each row execute procedure func();

insert into account values (1, 'Aisha', 'qwertyugfgfi', 'Kazakhstan-Almaty');
insert into account values (2, 'Aru', 'asdfghjfgfgfgkl', 'Japan-Tokyo');
insert into account values (3, 'Ainur', 'zxccgfgfgfggx', 'Russia-Moscow');

-- 4a. 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.
create table workers(
    id integer primary key,
    name varchar,
    date_of_birth date,
    age integer,
    salary integer,
    workexperience integer,
    discount integer
);
```

```
insert into workers(id, name, date_of_birth, age, salary, workexperience, discount)
values (1, 'David', '1993-06-07', 28, 100, 10, 0);

select * from workers;

create or replace procedure procedure_a() as
    $$
    begin
        update workers
        set salary = salary * 1.1 * workexperience / 2, discount = discount + 10;
        update workers
        set discount = (discount * 1.01) * workexperience / 5;
        commit;
    end;
    $$
language plpgsql;
call procedure_a();


-- 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.
insert into workers(id, name, date_of_birth, age, salary, workexperience, discount)
values (3, 'Carla', '1970-06-07', 40, 100, 10, 0);

create or replace procedure procedure_b() as
    $$
    begin
        update workers
        set salary = salary * 1.15 where age >= 40;
        update workers
        set salary = salary * 1.15 * workers.workexperience / 8, discount = discount +
20 where workexperience >= 8;
        commit;
    end;
    $$
language plpgsql;
call procedure_b();

-- 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 rec_recommendedby(recommender, member) as(
    select recommendedby, memid from members
    union all
    select members.recommendedby, rec.member from rec_recommendedby rec
    inner join members on members.memid = rec.recommender
)
select rec.member, rec.recommender, members.firstname, members.surname
from rec_recommendedby rec
inner join members on rec.recommender=members.memid
where rec.member = 12 or rec.member = 22
order by rec.member asc, rec.recommender desc
```