-- 1.Create a function that:

-- a. Increments given values by 1 and returns it.

CREATE FUNCTION *inc*(val integer) RETURNS integer AS $$ BEGIN

RETURN val + 1;

END; $$

LANGUAGE PLPGSQL;

SELECT *inc*(5);

-- b. Returns sum of 2 numbers.

CREATE FUNCTION *sum*(val1 integer, val2 integer) RETURNS integer AS $$ BEGIN

RETURN val1 + val2;

END; $$

LANGUAGE PLPGSQL;

SELECT sum(5, 3);

-- c. Returns true or false if numbers are divisible by 2.

CREATE FUNCTION *div\_2*(val integer) RETURNS bool AS $$ BEGIN

RETURN val % 2 = 0;

END; $$

LANGUAGE PLPGSQL;

SELECT *div\_2*(6);

-- SELECT div\_2(5);

-- d. Checks some password for validity.e. Returns two outputs, but has one input.

CREATE FUNCTION *check\_password*(password char(9)) RETURNS bool AS $$ BEGIN

RETURN password = 'diana';

END; $$

LANGUAGE PLPGSQL;

SELECT *check\_password*('diana');

-- e. Returns two outputs, but has one input.

CREATE FUNCTION *area\_and\_perimeter*(in val integer, out area integer, out perimeter integer) AS $$ BEGIN

area := val \* val;

perimeter := val \* 4;

END; $$

LANGUAGE PLPGSQL;

SELECT *area\_and\_perimeter*(5);

-- 2.

-- а. Create a trigger that:a.Return timestamp of the occured action within the database.

CREATE TABLE employees(

id INT GENERATED ALWAYS AS IDENTITY,

first\_name VARCHAR(40) NOT NULL,

last\_name VARCHAR(40) NOT NULL,

PRIMARY KEY(id)

);

CREATE TABLE employee\_audits (

id INT GENERATED ALWAYS AS IDENTITY,

employee\_id INT NOT NULL,

last\_name VARCHAR(40) NOT NULL,

new\_last\_name VARCHAR(40) NOT NULL,

changed\_on TIMESTAMP(6) NOT NULL

);

CREATE OR REPLACE FUNCTION *log\_last\_name\_changes*()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

BEGIN

IF NEW.last\_name <> OLD.last\_name THEN

INSERT INTO employee\_audits(employee\_id,last\_name,new\_last\_name,changed\_on)

VALUES(OLD.id,OLD.last\_name,NEW.last\_name,now());

END IF;

RETURN NEW;

END;

$$

CREATE TRIGGER last\_name\_changes

BEFORE UPDATE

ON employees

FOR EACH ROW

EXECUTE PROCEDURE *log\_last\_name\_changes*();

INSERT INTO employees (first\_name, last\_name)

VALUES ('Diana', 'Ruzi');

INSERT INTO employees (first\_name, last\_name)

VALUES ('Mariyam', 'Baigissiyeva');

UPDATE employees

SET last\_name = 'Kurenbekova'

WHERE ID = 2;

SELECT \* FROM employee\_audits;

-- b.Computes the age of a person when persons’ date of birth is inserted.

CREATE TABLE patient (

id INT GENERATED ALWAYS AS IDENTITY,

date\_of\_birth date,

age integer,

PRIMARY KEY(id)

);

CREATE OR REPLACE FUNCTION *ages*()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

BEGIN

NEW.age = extract(years from age(current\_date, new.date\_of\_birth ));

RETURN NEW;

END;

$$

CREATE TRIGGER compute\_age

Before INSERT

ON patient

FOR EACH ROW

EXECUTE PROCEDURE *ages*();

INSERT INTO patient (date\_of\_birth)

VALUES ('2002-11-20');

INSERT INTO patient (date\_of\_birth)

VALUES ('1980-04-01');

INSERT INTO patient (date\_of\_birth)

VALUES ('2008-12-05');

-- c.Adds 12% tax on the price of the inserted item.

CREATE TABLE items (

item\_id INT GENERATED ALWAYS AS IDENTITY,

price INT NOT NULL,

PRIMARY KEY(item\_id)

);

CREATE OR REPLACE FUNCTION *tax*() RETURNS TRIGGER AS $$

BEGIN

new.price = new.price + new.price \* 0.12;

RETURN new;

END; $$

LANGUAGE PLPGSQL;

CREATE TRIGGER set\_tax

BEFORE INSERT ON items

FOR EACH ROW

EXECUTE PROCEDURE *tax*();

INSERT INTO items(price)

VALUES (100);

-- d.Prevents deletion of any row from only one table.

CREATE TABLE students(

id INT GENERATED ALWAYS AS IDENTITY,

first\_name VARCHAR(40) NOT NULL,

last\_name VARCHAR(40) NOT NULL,

PRIMARY KEY(id)

);

CREATE OR REPLACE FUNCTION *delete\_off*() RETURNS TRIGGER AS $$

BEGIN

RETURN null;

END; $$

LANGUAGE PLPGSQL;

CREATE TRIGGER del\_off

BEFORE delete ON students

FOR EACH ROW

EXECUTE PROCEDURE *delete\_off*();

INSERT INTO students (first\_name, last\_name)

VALUES ('Mariyam', 'Kurenbekova');

delete from students where id=1;

-- e.Launches functions 1.d and 1.e.

-- 3

-- What is the difference between procedure and function

-- Function is used to calculate something from a given input.

-- While procedure is the set of commands, which are executed in a order.

-- 4.Create procedures that:

CREATE TABLE employee

(

id INTEGER,

name VARCHAR,

date\_of\_birth DATE,

age INTEGER,

salary INTEGER,

workexperience INTEGER,

discount INTEGER

);

drop table employee

-- 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 *bonus*()

language plpgsql

as $$

begin

update employee

set salary = employee.salary\*(employee.workexperience/2+1),

discount = 10

where employee.workexperience >= 2;

update employee set discount = employee.discount+(employee.workexperience/5)

where employee.workexperience >= 5;

commit;

end;$$;

call *bonus*();

insert into employee values (1,'Diana', '2002-11-20', 19, 100000,5, 0);

-- 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 or replace procedure *bonus2*()

language plpgsql

as $$

begin

update employee

set salary = employee.salary\*1.15

where employee.age >= 40;

update employee set discount = 20,

salary = salary\*1.15

where employee.workexperience >= 8 and employee.age >= 40;

commit;

end;$$;

call *bonus2*();

insert into employee values ( 2, 'Ayana', '2006-01-15', 15, 40000, 10,0);

UPDATE employee

SET age = 18

WHERE ID = 1;

-- 5

with recursive recommenders(recommender, member) as (

select recommendedby, memid

from members

union all

select mems.recommendedby, recs.member

from recommenders recs

inner join members mems

on mems.memid = recs.recommender

)

select recs.member member, recs.recommender, mems.firstname, mems.surname

from recommenders recs

inner join members mems

on recs.recommender = mems.memid

where recs.member = 22 or recs.member = 12

order by recs.member asc, recs.recommender desc