

**Kazakh-British Technical University**

Faculty of Information Technology

**Laboratory Work №8**

Prepared by: Maratuly T.

**Almaty, 2021**

1. **Create a function that:**
2. **Increments given values by 1 and returns it.**

*-- 1a Create a function that: Increments given values by 1 and returns it.*CREATE OR REPLACE FUNCTION *get\_incremented\_value*(number\_a float)  
 RETURNS float  
AS  
$$  
DECLARE  
 one\_step CONSTANT float := 1.0;  
BEGIN  
 number\_a := number\_a + one\_step;  
 RETURN number\_a;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *get\_incremented\_value*(5);

1. **Returns sum of 2 numbers.**

*-- 1b Create a function that: Returns sum of 2 numbers*CREATE OR REPLACE FUNCTION *get\_two\_numbers\_sum*(number\_a float, number\_b float)  
 RETURNS float  
AS  
$$  
DECLARE  
  
BEGIN  
 RETURN number\_a + number\_b;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *get\_two\_numbers\_sum*(4.5, 5);

1. **Returns true or false if numbers are divisible by 2.**

*-- 1c Create a function that: Returns true or false if numbers are divisible by 2*CREATE OR REPLACE FUNCTION *is\_number\_divisible\_by\_two*(number\_a int)  
 RETURNS boolean  
AS  
$$  
DECLARE  
 two\_devision\_terminal CONSTANT int := 2;  
BEGIN  
 IF number\_a % two\_devision\_terminal = 0  
 THEN  
 RETURN true;  
 ELSE  
 RETURN false;  
 END IF;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *is\_number\_divisible\_by\_two*(4);  
SELECT *is\_number\_divisible\_by\_two*(5);

1. **Checks some password for validity.**

*-- 1.d Create a function that: Checks some password for validity.*CREATE OR REPLACE FUNCTION *is\_password\_valid*(password varchar)  
 RETURNS boolean  
AS  
$$  
DECLARE  
 is\_lower\_case\_letter boolean := false;  
 is\_capital\_letter\_exist boolean := false;  
 is\_figure\_exist boolean := false;  
 is\_minimum\_size\_exist boolean := false;  
 minimum\_size int := 8;  
 password\_size int := (SELECT *length*(password));  
 one\_step CONSTANT int := 1;  
 current\_char varchar;  
BEGIN  
 IF password\_size >= minimum\_size  
 THEN  
 is\_minimum\_size\_exist := true;  
 END IF;  
  
 FOR cur\_index IN 1..password\_size  
 LOOP  
 current\_char := *substring*(password, cur\_index, one\_step);  
 IF current\_char = *UPPER*(current\_char) AND  
 current\_char NOT IN ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')  
 THEN  
 is\_capital\_letter\_exist := true;  
 ELSEIF current\_char = *LOWER*(current\_char)  
 THEN  
 is\_lower\_case\_letter := true;  
 END IF;  
  
 IF current\_char IN ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')  
 THEN  
 is\_figure\_exist := true;  
 END IF;  
 END LOOP;  
  
 IF is\_lower\_case\_letter = true AND is\_capital\_letter\_exist = true AND is\_figure\_exist = true AND  
 is\_minimum\_size\_exist = true  
 THEN  
 RETURN true;  
 END IF;  
 RETURN false;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *is\_password\_valid*('temirbolat123'); *-- False because no capital letter*SELECT *is\_password\_valid*('Temirbolat123'); *-- True because all is okay*SELECT *is\_password\_valid*('TEMIRBOLAT123'); *-- False because no lower case letter*SELECT *is\_password\_valid*('TEMIRBOLAT');*-- False because no figure*

1. **Returns two outputs, but has one input**

CREATE TABLE users  
(  
 id SERIAL PRIMARY KEY,  
 username varchar(100) UNIQUE NOT NULL,  
 password varchar(100) NOT NULL,  
 age integer NOT NULL  
);

INSERT INTO users(username, password, age)  
VALUES ('TEMIRBOLAT', 'Temir1234', 20);  
INSERT INTO users(username, password, age)  
VALUES ('TAMERLAN', 'Temir1234', 21);  
INSERT INTO users(username, password, age)  
VALUES ('Superman', 'Temir1234', 21);

*-- 1e. Create a function that: Returns two outputs, but has one input*CREATE OR REPLACE FUNCTION *return\_name\_age*(id\_user int)  
 RETURNS RECORD  
AS  
$$  
DECLARE  
 output\_record RECORD;  
BEGIN  
  
 SELECT username, age INTO output\_record FROM users WHERE users.id = id\_user;  
  
 RETURN output\_record;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *return\_name\_salary*(1);

1. **Create a trigger that:**
2. **Return timestamp of the occured action within the database.**

*--2a. Create a trigger that: Return timestamp of the occured action within the database.*CREATE OR REPLACE FUNCTION *finish\_any\_command*()  
 RETURNS event\_trigger  
 LANGUAGE plpgsql  
AS  
$$  
BEGIN  
 RAISE NOTICE 'The command % was complited at %',*tg\_tag*,*LOCALTIMESTAMP*;  
END;  
$$;  
  
CREATE EVENT TRIGGER my\_event\_trigger  
 ON ddl\_command\_end  
EXECUTE FUNCTION *finish\_any\_command*();  
  
CREATE TABLE trial\_table  
(  
 id SERIAL  
);  
SELECT *\**FROM trial\_table;

1. **Computes the age of a person when persons’ date of birth is inserted.**

*-- 2b. Computes the age of a person when persons’ date of birth is inserted*CREATE TABLE people  
(  
 id serial PRIMARY KEY,  
 name varchar(50) NOT NULL,  
 date\_of\_birth date NOT NULL,  
 age int DEFAULT NULL  
);  
  
  
CREATE OR REPLACE FUNCTION *trigger\_insert\_calculate\_age*()  
 RETURNS TRIGGER  
 LANGUAGE plpgsql  
AS  
$$  
DECLARE  
 date\_of\_birth date := (SELECT NEW.date\_of\_birth);  
 current\_date date := *CURRENT\_DATE*;  
 age\_result int;  
BEGIN  
 age\_result := *EXTRACT*(YEAR FROM *AGE*(*current\_date*, date\_of\_birth));  
 UPDATE people  
 SET age = age\_result  
 WHERE id = NEW.id;  
  
 RETURN NEW;  
END;  
$$;  
  
CREATE TRIGGER people\_insert\_trigger  
 AFTER INSERT  
 ON people  
 FOR EACH ROW  
EXECUTE PROCEDURE *trigger\_insert\_calculate\_age*();  
  
INSERT INTO people(name, date\_of\_birth)  
VALUES ('Temirbolat', '31-01-2001');  
INSERT INTO people(name, date\_of\_birth)  
VALUES ('Alexander', '28-11-2000');  
INSERT INTO people(name, date\_of\_birth)  
VALUES ('Antony', '29-11-2000');  
SELECT *\**FROM people;

1. **Adds 12% tax on the price of the inserted item.**

*-- 2c. Adds 12% tax on the price of the inserted item.*CREATE TABLE items  
(  
 id serial PRIMARY KEY,  
 name varchar(30) UNIQUE,  
 price float NOT NULL CHECK ( price > 0.0 )  
);  
  
  
CREATE OR REPLACE FUNCTION *item\_trigger\_tax\_price\_inserted*()  
 RETURNS TRIGGER  
 LANGUAGE plpgsql  
AS  
$$  
BEGIN  
 UPDATE items  
 SET price = 1.12 \* price  
 WHERE id = NEW.id;  
  
 RETURN NEW;  
END;  
$$;  
  
CREATE TRIGGER items\_insert\_trigger  
 AFTER INSERT  
 ON items  
 FOR EACH ROW  
EXECUTE PROCEDURE *item\_trigger\_tax\_price\_inserted*();  
  
INSERT INTO items(name, price)  
VALUES ('Apple', 100),  
 ('Banana', 150),  
 ('Cherry', 200);  
  
SELECT *\**FROM items;

1. **Prevents deletion of any row from only one table.**

*-- 2d. Prevents deletion of any row from only one table*CREATE OR REPLACE FUNCTION *prevent\_delete\_table*()  
 RETURNS TRIGGER  
 LANGUAGE plpgsql  
AS  
$$  
BEGIN  
 RAISE 'You can not delete row';  
END;  
$$;  
  
DROP FUNCTION *prevent\_delete\_table*;  
  
CREATE TRIGGER items\_delete\_trigger  
 BEFORE DELETE  
 ON items  
 FOR EACH STATEMENT  
EXECUTE PROCEDURE *prevent\_delete\_table*();  
  
DROP TRIGGER items\_delete\_trigger ON items;  
  
DELETE  
FROM items;

1. **Launches functions 1.d and 1.e**

*-- 2e. Launches functions 1.d and 1.e*CREATE TABLE users  
(  
 id SERIAL PRIMARY KEY,  
 username varchar(100) UNIQUE NOT NULL,  
 password varchar(100) NOT NULL,  
 age integer NOT NULL  
);  
  
CREATE OR REPLACE FUNCTION *user\_function*()  
 RETURNS TRIGGER  
AS  
$$  
BEGIN  
 RAISE NOTICE '%; %',NEW.id,NEW.username;  
 IF *is\_password\_valid*(NEW.password) = false  
 THEN  
 RAISE EXCEPTION 'The password is invalid';  
 ELSE  
 RAISE NOTICE 'The user with age: % has been successfully added',*return\_name\_age*(NEW.id);  
 END IF;  
  
 RETURN NEW;  
END  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER my\_trigger  
 AFTER INSERT  
 ON users  
 FOR EACH ROW  
EXECUTE PROCEDURE *user\_function*();  
  
INSERT INTO users(username, password, age)  
VALUES ('TEMIRBOLAT', 'Temir1234', 20);  
INSERT INTO users(username, password, age)  
VALUES ('TAMERLAN', 'Temir1234', 21);  
INSERT INTO users(username, password, age)  
VALUES ('Superman', 'Temir1234', 21);  
  
SELECT *\**FROM users;

**3. What is the difference between procedure and function**

|  |  |
| --- | --- |
| **Function** | **Procedure** |
| Called in a request as part of it | Called by the CALL keyword and cannot be part of the request |
| Can return values | Returns no values, but this can be bypassed through the "out" parameters |
| It is impossible to manage transactions inside the function (start and rollback) | If the procedure is not called inside another explicit transaction, then transactions can be managed inside the procedure (start and rollback completely or to a recovery point) |
| Created by the CREATE FUNCTION command | Created by the CREATE PROCEDURE command |

**4. Create procedures that:**

CREATE TABLE emplyees  
(  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(50) NOT NULL,  
 date\_of\_birth DATE NOT NULL,  
 age INTEGER NOT NULL,  
 salary INTEGER NOT NULL,  
 workexperience INTEGER NOT NULL,  
 discont INTEGER DEFAULT 0  
);  
  
INSERT INTO emplyees(name, date\_of\_birth, age, salary, workexperience)  
VALUES ('Temirbolat', '31.01.2001', 20, 35000, 3),  
 ('Assanali', '10.05.1951', 70, 40000, 10),  
 ('Alexander', '04.02.1959', 62, 55000, 2),  
 ('Antony', '31.03.2000', 21, 100000, 12),  
 ('Tamerlan', '06.08.1955', 66, 42000, 1),  
 ('Karlygash', '15.07.1952', 69, 30000, 1);

**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.**

*-- 4a. 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 FUNCTION *get\_salary\_discount\_multiplyer*(work\_experience int)  
 RETURNS float  
AS  
$$  
DECLARE  
 two\_year\_terminal CONSTANT int := 2;  
BEGIN  
 RETURN (work\_experience / two\_year\_terminal)::float;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE OR REPLACE FUNCTION *get\_one\_percent\_discount*(work\_experience int)  
 RETURNS INT  
AS  
$$  
DECLARE  
 one\_step CONSTANT int = 1;  
BEGIN  
 IF work\_experience >= 5  
 THEN  
 RETURN one\_step;  
 ELSE  
 RETURN 0;  
 END IF;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE OR REPLACE PROCEDURE *calculate\_salary\_a*()  
AS  
$$  
DECLARE  
 ten\_percent CONSTANT float := 0.1;  
BEGIN  
  
 UPDATE emplyees  
 SET salary = (salary::float + *get\_salary\_discount\_multiplyer*(workexperience) \* ten\_percent \* salary::float)::int,  
 discont = (*get\_salary\_discount\_multiplyer*(workexperience) \* ten\_percent \* 100.0)::int;  
  
 UPDATE emplyees  
 SET discont = discont + *get\_one\_percent\_discount*(workexperience);  
  
END;  
$$  
 LANGUAGE plpgsql;  
  
SELECT *\**FROM emplyees;  
  
CALL *calculate\_salary\_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**

CREATE OR REPLACE FUNCTION *get\_updated\_salary*(age int, salary int, case\_salary boolean, work\_experience int)  
 RETURNS INT  
AS  
$$  
DECLARE  
 fifteen\_percent CONSTANT float := 1.15;  
BEGIN  
  
 IF (age >= 40 AND case\_salary = false) OR (work\_experience > 8 AND case\_salary = true)  
 THEN  
 RETURN (salary::FLOAT \* fifteen\_percent)::INT;  
 ELSE  
 RETURN salary;  
 END IF;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE OR REPLACE FUNCTION *get\_updated\_discount*(work\_experience int, discount int)  
 RETURNS INT  
AS  
$$  
DECLARE  
 twenty\_percent CONSTANT INT := 20;  
BEGIN  
  
 IF work\_experience > 8  
 THEN  
 RETURN twenty\_percent;  
 ELSE  
 RETURN discount;  
 END IF;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE OR REPLACE PROCEDURE *calculate\_salary\_b*()  
AS  
$$  
DECLARE  
BEGIN  
 UPDATE emplyees  
 SET salary = *get\_updated\_salary*(age, salary, false, workexperience);  
  
 UPDATE emplyees  
 SET salary = *get\_updated\_salary*(age, salary, true, workexperience);  
 *-- Спросить: Нужно прибавить к текущей скидки 20 проц или установить 20 проц* UPDATE emplyees  
 SET discont = *get\_updated\_discount*(workexperience, discont);  
  
END;  
$$  
 LANGUAGE plpgsql;  
  
SELECT *\**FROM emplyees;  
CALL *calculate\_salary\_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 int PRIMARY KEY,  
 surname varchar(200) NOT NULL,  
 firstname varchar(200) NOT NULL,  
 address varchar(300) NOT NULL,  
 zipcode int,  
 telephone varchar(20),  
 recommendedby int,  
 joindate timestamp,  
 CONSTRAINT fk\_recommendedby\_id FOREIGN KEY (recommendedby) REFERENCES members (memid)  
);  
  
CREATE TABLE facilities  
(  
 facid int PRIMARY KEY,  
 name varchar(100) NOT NULL,  
 membercost numeric,  
 guestcost numeric,  
 initialoutlay numeric,  
 monthlymaintenance numeric  
);  
  
CREATE TABLE bookings  
(  
 facid int,  
 memid int,  
 starttime timestamp,  
 slots int,  
 CONSTRAINT fk\_facilities\_id FOREIGN KEY (facid) REFERENCES facilities (facid),  
 CONSTRAINT fk\_members\_id FOREIGN KEY (memid) REFERENCES members (memid)  
);  
  
SELECT *\**FROM members;  
  
*-- For 12*WITH RECURSIVE members\_recommenders AS (  
 SELECT memid, recommendedby  
 FROM members  
 WHERE memid = 12  
 UNION  
 SELECT m.memid, m.recommendedby  
 FROM members m  
 INNER JOIN members\_recommenders m\_r ON m\_r.memid = m.recommendedby  
)  
SELECT *\**FROM members\_recommenders ORDER BY memid ASC, recommendedby DESC;  
  
*-- For 22 member*WITH RECURSIVE members\_recommenders AS (  
 SELECT memid, recommendedby  
 FROM members  
 WHERE memid = 22  
 UNION  
 SELECT m.memid, m.recommendedby  
 FROM members m  
 INNER JOIN members\_recommenders m\_r ON m\_r.memid = m.recommendedby  
)  
SELECT *\**FROM members\_recommenders ORDER BY memid ASC, recommendedby DESC;