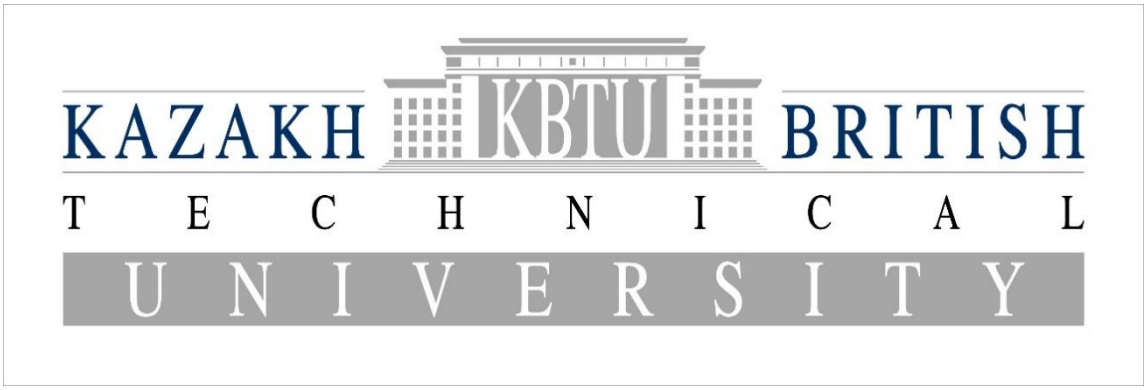

Kazakh-British Technical University
Faculty of Information Technology

Laboratory Work №8

Prepared by: Maratuly T.

Almaty, 2021

1. **Create a function that:**
   a. **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);
```

   b. **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);
```

   c. **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);
```

d. 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
```

e. 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);
```

2. **Create a trigger that:**
   a. **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;
```

b. 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;
```

c. 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;
```

d. 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;
```

e. 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;
```