Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 5 з дисципліни
«Бази даних»

«Основи програмування з використанням мови SQL. Збережені процедури. Курсори. Створення, програмування та керування тригерами.»

Варіант 8

Виконав студент ІП-13 Дойчев Костянтин Миколайович
(шифр, прізвище, ім'я, по батькові)

Перевірила Марченко Олена Іванівна
(прізвище, ім'я, по батькові)

Київ 2022

Лабараторна робота №5

Основи програмування з використанням мови SQL. Збережені процедури. Курсори. Створення, програмування та керування тригерами.

Постановка задачі лабораторної роботи № 5

При виконанні лабораторної роботи необхідно виконати наступні дії:
1) Збережені процедури:
a. запит для створення тимчасової таблиці через змінну типу TABLE;
b. запит з використанням умовної конструкції IF;
c. запит з використанням циклу WHILE;
d. створення процедури без параметрів;
e. створення процедури з вхідним параметром;
f. створення процедури з вхідним параметром та RETURN;
g. створення процедури оновлення даних в деякій таблиці БД;
h. створення процедури, в котрій робиться вибірка даних.
2) Функції:
a. створити функцію, котра повертає деяке скалярне значення;
b. створити функцію, котра повертає таблицю з динамічним набором стовпців;
c. створити функцію, котра повертає таблицю заданої структури.
3) Робота з курсорами:
a. створити курсор;
b. відкрити курсор;
c. вибірка даних, робота з курсорами.
4) Робота з тригерами:
a. створити тригер, котрий буде спрацьовувати при видаленні даних;
b. створити тригер, котрий буде спрацьовувати при модифікації даних;
c. створити тригер, котрий буде спрацьовувати при додаванні даних.

Індивідуальне завдання:

Програмне забезпечення «Школа». Загальноосвітня школа, в якій навчаються учні, має номер, назву, адресу, ПІБ директора. У школах є певна кількість класів, котрі мають назву, класного керівника, список учнів, певний перелік предметів. Предмети викладаються вчителями, причому один вчитель може викладати декілька предметів, а однакові предмети можуть викладати різні вчителі. Предмети викладаються згідно з розкладом у кабінетах, котрі мають номер, назву, відповідне обладнання та розкладом класів. Предмети мають назву, кількість годин вивчення, список навчальних посібників.

SQL скрипти

Усі скрипти можна знайти у репозиторії на платформі GitHub.

```
-- Task 1

-- a
create or replace procedure get_temp_table()
    language plpgsql
as
$$

begin
    drop table if exists temp_table;
    create temporary table temp_table as (select *
                                          from author
                                          limit 10);

    raise notice 'temp_table created';
    commit;
end;
$$;

call get_temp_table();

-- b
create or replace procedure is_user_exist (
```

```
user_id uuid
) language plpgsql as
    $$

    begin
        if exists (select * from public.user where "id" =
user_id) then
            raise notice 'user exists';
        else
            raise notice 'user does not exist';
        end if;
    end;
    $$;

call is_user_exist('b0acbec0-4d97-4cc7-920b-86da341aabe8');

-- c
create or replace procedure fibonacci (
    iteration int
) language plpgsql as
    $$
    declare
        a int := 0;
        b int := 1;
        result int := 0;
    begin
        while iteration > 0 loop
            result := a + b;
            a := b;
            b := result;
            iteration := iteration - 1;
        end loop;
        raise notice 'fibonacci number is %', result;
    end;
    $$;
```

```
call fibonacci(3);

-- d
create or replace procedure select_all_users()
    language plpgsql
as
$$
    declare
        u user%rowtype;
begin
    for u in select * from public.user
        loop
            raise notice 'user id: %, full name: %', u.id,
concat(u.first_name, ' ', u.last_name);
        end loop;
end;
$$;

call select_all_users();

-- e
create or replace procedure select_users(
    lim int default 10,
    offs int default 0
)
    language plpgsql
as
$$
    declare
        query text;
        rec record;
begin
        query := 'select * from public.user limit $1 offset $2';
    for rec in execute query using lim, offs
        loop
            raise notice 'user id: %, full name: %', rec.id,
```

```
concat(rec.first_name, ' ', rec.last_name);
        end loop;
end;
$$;

call select_users(5);

-- f
create or replace procedure has_role (
    usr_id uuid,
    role_name role_type
)
    language plpgsql
as
$$
    declare

begin
    -- we use 'perform' here because we don't care about the
result
    perform * from public."user" u
    inner join public."role" r on u.id = r.user_id
    where u.id = usr_id and r.type = role_name;

    if not found then
        raise exception 'User does not have role';
        return;
    end if;

    raise notice 'User has role';
end;
$$;

call has_role('dc37e6d8-173c-418b-a306-52146839bf52',
'student');
```

```sql
-- g
create or replace procedure change_subject_credits (
subj_id uuid,
new_credits real
) language plpgsql as
    $$
    begin
        update public.subject
        set credits = new_credits
        where id = subj_id;
    end;
$$;

call
change_subject_credits('f31dce9e-36f8-4184-b692-dfad03c83816',
95);

-- h
create or replace procedure select_all_books()
    language plpgsql
as
$$
    declare
        book public.book%rowtype;
begin
    for book in select * from public.book
        loop
            raise notice 'Book id: %, name: %', book.id,
book.name;
        end loop;
end;
$$;

call select_all_books();

-- Task 2
```

```sql
-- a
create or replace function get_number_of_students_in_class (
    chosen_class_id uuid
) returns integer
    language plpgsql
as
    $$
    declare
        number_of_students integer;
    begin
        select count("role".id) into number_of_students
        from "role"
        inner join "class" on "class".id = "role".class_id
        where "class".id = chosen_class_id
        and "role".type = 'student';

        return number_of_students;
    end;
    $$;

select
get_number_of_students_in_class('96a42758-13b9-42fc-a307-f4e4a1b
90a93');

-- query to validate the answer
-- get all roles connected to the class including the teacher
select "class".id, "class"."name", count("role".id)
from "role"
inner join "class" on "class".id = "role".class_id
group by "class"."name", "class".id;

-- b
drop function get_dynamic_books();
create or replace function get_dynamic_books(
)
```

```
    returns record
    language plpgsql
as
$$
    declare
        rec record;
begin
    select * from public.book into rec;
    return rec;
end ;
$$;

select get_dynamic_books();

-- c
drop function get_detailed_books();
create or replace function get_detailed_books(
)
    returns table
            (
                "id"           uuid,
                "name"         varchar(120),
                "author_name"  varchar(150),
                "published_at" smallint
            )
    language plpgsql
as
$$
begin
    return query select "b".id
as "id",
                        "b".name::varchar(120)
as "name",
                        concat("a".first_name, ' ',
"a".last_name)::varchar(150) as "author_name",
                        "b".published_at
```

```
as "published_at"
                from public.book as "b"
                         inner join public."author" as "a" on
"b".author_id = "a".id;
end ;
$$;

select *
from get_detailed_books();

-- Task 3
create or replace function get_student_full_names(
) returns text
as
$$
declare
    full_names text default '';
    usr        record;
    student_names cursor for
        select concat("u".first_name, ' ', "u".last_name) as
"full_name"
        from public.role
                 inner join "user" u on u.id = role.user_id
        where role.type = 'student';
begin
    open student_names;
    loop
        -- unlike in for loop, we do not fetch all the records
and then process them, we process them one by one by "lazy"
loading
        fetch student_names into usr;
        exit when not found;
        if (full_names = '') then
            full_names := usr.full_name;
        else
            full_names := full_names || ', ' || usr.full_name;
```

```
        end if;
    end loop;

    close student_names;

    return full_names;
end ;
$$
    language plpgsql;

select get_student_full_names();

-- Task 4
create type action_type as ENUM ('delete', 'update', 'insert');

create table if not exists "log"
(
    "id"          uuid primary key default uuid_generate_v4(),
    "user_id"     uuid         not null,
    "action_type" action_type not null,
    "time" timestamp    not null
);

create or replace function handle_user_change(
) returns trigger
    language plpgsql
as
$$
begin
    -- tg_op is a special variable from trigger context
https://www.postgresql.org/docs/current/plpgsql-trigger.html
    insert into public."log" ("user_id", "action_type", "time")
values (old.id, lower(TG_OP)::action_type, now());
    return new;
end;
$$;
```

```
create trigger user_trigger
    after insert or update or delete
    on public."user"
    for each row
    execute procedure handle_user_change();

insert into public."user" ("first_name", "last_name") values
('John', 'Doe');
update public."user" set "first_name" = 'Jane' where
"first_name" = 'John';
delete from public."user" where "id" =
'f8bb79c9-a01e-4363-a224-c2f791279a04';

select * from public.log;
```