# 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 7

Виконав студент ІП-11 Головня О.Р.

(шифр, прізвище, ім'я, по батькові)

Перевірив(-ла) Ліщук Катерина Ігорівна

(прізвище, ім'я, по батькові)

Київ 2022

**Мета:**

Вивчити правила побудови ідентифікаторів, правила визначення змінних та типів. Визначити правила роботи з циклами та умовними конструкціями, роботу зі змінними типу Table.

Вивчити синтаксис та семантику функцій та збережених процедур, способів їх ідентифікації, методів визначення та специфікації параметрів та повертаємих значень, виклик функцій та збережених процедур.

Застосування команд для створення, зміни та видалення як скалярних, так і табличних функцій, збережених процедур.

Вивчити призначення та типи курсорів, синтаксис та семантику команд мови SQL для створення курсорів, вибірки даних з курсорів, зміни даних із застосуванням курсорів.

Вивчити призначення та типи тригерів, умов їх активації, синтаксису та семантики для їх створення, модифікації, перейменування, програмування та видалення

**Постановка задачі:**

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.

1. Збережені процедури:
2. запит для створення тимчасової таблиці через змінну типу TABLE;

CREATE TEMP TABLE driver\_age(driver\_id serial, age int);

(Створення тимчасової таблиці з id та віку)

SELECT \* FROM driver\_license;

1. запит з використанням умовної конструкції IF;

DO $$

DECLARE e record;

BEGIN FOR e IN SELECT \* FROM driver

LOOP IF e.driver\_id IN (SELECT driver\_id FROM driver\_age) then

raise notice '% Age of the driver is in the range of 18-68', e.full\_name;

ELSE raise notice '% Age of the driver isnt in the range of 18-68', e.full\_name; END IF; END LOOP;

END; $$

(Якщо id з таблиці водія співпадає з id водія нашї тимчасової таблиці, то виведе що вік водія в діапазоні 18-68, інакше навпаки)

1. з запит з використанням циклу WHILE;

DO $$

BEGIN WHILE 10 > (SELECT COUNT(\*) FROM driver\_age) LOOP INSERT INTO driver\_age(age) VALUES ((random()\* 50 +18)::int); end loop; end$$;

(Заповнюємо перші 10 водіїв випадковим віком від 18 до 68)

1. створення процедури без параметрів;

CREATE OR REPLACE PROCEDURE driver()

LANGUAGE plpgsql

AS $$

DECLARE random\_id int; drivers\_num int; t\_driver\_name varchar(70);

BEGIN drivers\_num := (SELECT COUNT(\*) FROM driver);

random\_id := random()\*(drivers\_num - 1) + 1;

t\_driver\_name = (SELECT full\_name FROM driver WHERE driver\_id = random\_id);

RAISE NOTICE '% Driver: ', t\_driver\_name;

END; $$

(Вибирає випадкового водія з нашої табл. Спочатку вибираємо id від 1 до к-сті водіїв, вибираємо ім’я під цим id і виводимо хто це)

CALL driver() – Процедура нічого не повертає

1. створення процедури з вхідним параметром;

CREATE OR REPLACE PROCEDURE number\_( first\_number int, range\_n int DEFAULT 2 )

LANGUAGE plpgsql

AS $$

DECLARE random\_num int;

BEGIN

random\_num := random()\*(range\_n - 1) + 1;

IF first\_number = random\_num THEN raise notice 'first num = generated num!';

ELSE raise notice 'first num != generated num! The generated num was %', random\_num; END IF;

END; $$

(Приймаємо два параметра, range\_n – тільки додатнє, ціле. Генеруємо випадкове число у веденому діапазоні, перевіряємо чи наше перше число дорівнює згенерованому і виводимо результат)

CALL number(1, 5);

1. створення процедури з вхідним параметром та RETURN;

CREATE OR REPLACE PROCEDURE return\_procedure( input int )

LANGUAGE plpgsql

AS $$

DECLARE

BEGIN RETURN;

RAISE NOTICE 'DASDASDASASDA';

END; $$

1. створення процедури оновлення даних в деякій таблиці БД;

CREATE OR REPLACE PROCEDURE update\_driver\_age( input\_id int, new\_age int )

LANGUAGE plpgsql

AS $$

DECLARE

BEGIN

IF (new\_age) < 18 THEN

RAISE NOTICE 'age should be more then 18! !';

RETURN;

END IF;

UPDATE driver

SET driver\_age = new\_age WHERE driver\_id = input\_id;

RAISE NOTICE 'Driver age was updated!';

END; $$

(Я хочу змінити вік водія, я ввожу id і вік, якщо вік менше 18, то буде виведено помилку, якщо ні, то вік водія буде оновлено)

CALL update\_driver\_age (1, 17);

CALL update\_driver\_age (1, 27);

1. створення процедури, в котрій робиться вибірка даних.

CREATE OR REPLACE PROCEDURE brand\_trolley\_( brand\_name\_ varchar(70) )

LANGUAGE plpgsql

AS $$

DECLARE our\_trolley trolley%rowtype;

BEGIN

SELECT \* FROM trolley

INTO our\_trolley

WHERE brand\_name = brand\_name\_;

IF NOT found THEN

RAISE NOTICE 'brand doesn’t exist %';

ELSE RAISE NOTICE '%, %', our\_trolley.trolley\_id, our\_trolley.brand\_name; END IF;

END; $$

(Я хочу знайти які тролейбуси їздять під певною маркою. Я ввожу назву марки, вибирається де ця назва співпадає, якщо не знайдено, то виводиться помилка, інакше – id тролейбуса та ця марка)

CALL brand\_trolley\_(‘BMW’)

1. Функції:
2. створити функцію, котра повертає деяке скалярне значення;

CREATE OR REPLACE FUNCTION route\_stops\_num( input\_id integer) RETURNS integer LANGUAGE 'plpgsql'

AS $$

DECLARE stops\_num int; e record;

BEGIN SELECT \* INTO e FROM route WHERE route\_id = input\_id;

IF NOT found THEN RAISE NOTICE 'error!!!!!' ; stops\_num := 0;

ELSE RAISE NOTICE 'Route with inputted id was found';

stops\_num := (SELECT COUNT(route\_id) AS stops\_count FROM route WHERE route\_id = input\_id GROUP BY(route\_id)); END IF;

RETURN stops\_num;

END; $$

(Вводимо id маршруту та рахуємо к-сть зупинок на даному маршруті. Якщо маршруту не знайдено – помилка та зупинок 0, інакше підзапитом рахуємо їх к-сть і в кінці повертаємо)

SELECT route\_stops\_num(1)

1. створити функцію, котра повертає таблицю з динамічним набором стовпців;

CREATE OR REPLACE FUNCTION create\_table(column\_names Text[]) RETURNS void

LANGUAGE plpgsql

AS $$

DECLARE t1 Text := ''; t2 Text := ''; i int := 1; x Text;

BEGIN t1 := 'CREATE TEMP TABLE IF NOT EXISTS my\_table ('; t2 := 'INSERT INTO my\_table VALUES (';

FOREACH x IN ARRAY column\_names LOOP t1 := t1||x||' int ,'; t2 := t2||i||','; i := i+1; END LOOP; t1 := rtrim(t1,',')||') ON COMMIT DROP'; t2 := rtrim(t2,',')||')';

EXECUTE t1; EXECUTE t2;

END; $$

(У функцію приймаємо масив тексту, на самому початку створюємо якусь табл, потім будемо додавати якісь дані, потім: для кожного х робимо цикл де в першому тексті ми робимо стовпці, а в другому заповнюємо значеннями, у моєму випадку просто лічильник. Далі корегуємо, видаляємо не потрібну кому)

BEGIN;

SELECT create\_table(ARRAY['test1', 'test2', 'test3']);

SELECT \* FROM my\_table;

1. створити функцію, котра повертає таблицю заданої структури

CREATE OR REPLACE FUNCTION driver\_info()

RETURNS TABLE (full\_name varchar(70), roll\_number int, driver\_age int)

LANGUAGE plpgsql

AS $$

DECLARE

BEGIN

RETURN QUERY

SELECT driver.full\_name, driver.roll\_number, driver.driver\_age, AS driver\_info FROM driver

END; $$

(Тут все просто, повертаємо таблицю нам потрібних даних, де усі дані подані як одне ціле)

SELECT \* FROM driver\_info();

1. Робота з курсорами::

CREATE OR REPLACE FUNCTION tech\_inspection\_update(input\_id int)

RETURNS Text

LANGUAGE plpgsql

AS $$

DECLARE text\_ trolley\_ record; tech\_inspection\_update NO SCROLL CURSOR (input\_parking\_id int) FOR SELECT \* FROM trolley WHERE parking\_id = input\_parking\_id;

BEGIN

OPEN tech\_inspection\_update (input\_id);

LOOP

FETCH tech\_inspection\_update INTO trolley\_;

EXIT WHEN NOT found;

UPDATE trolley

SET inspection\_date = now() END LOOP;

CLOSE tech\_inspection\_update;

END; $$

(Курсор приймає значення id парку і складається з тролейбусів, що перебувають під цим id парку. Далі: відкриваємо курсор, оновлюємо тролейбус, тобто дату його останнього тех-обсл.

Вивід id тролейбуса, що

SELECT tech\_inspection\_update(1);

1. Робота з тригерами:
2. створити тригер, котрий буде спрацьовувати при видаленні даних;

CREATE OR REPLACE FUNCTION update\_parking\_id()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS $$

BEGIN

RAISE NOTICE 'trigger is working rn';

UPDATE trolley

SET parking\_id = 1 WHERE trolley.parking\_id = OLD.parking\_id;

RETURN NEW;

END; $$

(Суть цього тригера в тому, що я хочу видалити id якоїсь парковки і тоді тригер замінить цей іd на одиницю)

CREATE OR REPLACE TRIGGER update\_parking\_id

BEFORE DELETE

ON parking

FOR EACH ROW

EXECUTE PROCEDURE update\_parking\_id ();

DELETE FROM parking WHERE parking\_id = 5;

SELECT \* FROM trolley;

1. створити тригер, котрий буде спрацьовувати при модифікації даних;

CREATE OR REPLACE FUNCTION update\_name()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS $$

BEGIN

RAISE NOTICE 'trigger is working rn';

IF NOT OLD.full\_name = NEW.full\_name THEN

RAISE NOTICE 'name updated ';

END IF;

RETURN NEW;

END; $$

(Суть сама по собі зрозуміла: старе значення замінюємо на нове)

CREATE OR REPLACE TRIGGER update\_name

BEFORE UPDATE

ON driver

FOR EACH ROW

EXECUTE PROCEDURE update\_name();

UPDATE driver SET full\_name = 'Asdasd Ssasas'

WHERE driver\_id = 1;

(Змінює ім’я водія, заданого id, та виводить що тригер працює та ім’я оновлено)

SELECT \* FROM driver;

1. створити тригер, котрий буде спрацьовувати при додаванні даних.

CREATE OR REPLACE FUNCTION add\_route()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS $$

DECLARE result Record;

BEGIN

RAISE NOTICE 'trigger is working';

SELECT \* INTO result FROM route WHERE trolley\_id = NEW.trolley\_id OR driver\_id = NEW.driver\_id;

IF NOT found THEN RETURN NEW;

ELSE RAISE NOTICE 'returned old'; RETURN OLD;

END IF;

END; $$

(Водій працює на одному тролейбусі і на одному маршруті, тому якщо id тролейбуса або id водія дорівнюють встановленому, то ми повернемо old, запит не виконається)

CREATE OR REPLACE TRIGGER add\_route

BEFORE INSERT

ON route

FOR EACH ROW

EXECUTE PROCEDURE add\_route();

INSERT INTO route(driver\_id, trolley\_id, route\_id) VALUES(1, 3, 3)

(В даному випадку тригер спрацює, але запит не виконається, тому що id тролейбуса і id водія не співпали, тому і поміняти ми нічого не можемо

**Висновок:** виконуючи лабораторну роботу, я вивчив правила побудови ідентифікаторів, правила визначення змінних та типів, визначив правила роботи з циклами та умовними конструкціями, роботу зі змінними типу Table. Вивчив синтаксис та семантику функцій та збережених процедур, способів їх ідентифікації, методів визначення та специфікації параметрів та повертаємих значень, виклик функцій та збережених процедур, дослідив призначення та типи курсорів, синтаксис та семантику команд мови SQL для створення курсорів, вибірки даних з курсорів, зміни даних із застосуванням курсорів. Також, вивчив призначення та типи тригерів, умов їх активації, синтаксису та семантики для їх створення, модифікації, перейменування, програмування та видалення