

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»

«Тролейбусне депо»

Варіант 2

Виконав(ла) *ІП-45 Янов. Б.Є.*

Перевірила Марченко О.І.

Київ 2025

Тролейбусне депо

Програмне забезпечення «Тролейбусне депо». База даних містить інформацію: відомості про водіїв (табельний номер; ПІБ; дата, час початку і закінчення роботи на маршруті), відомості про тролейбуси (номер; марка; кількість пасажирських місць; дата останнього технічного огляду), відомості про маршрути (номер; час початку і кінця роботи тролейбусів; початковий пункт; кінцевий пункт; список зупинок; тривалість маршруту). Кожен водій працює на одному тролейбусі та на одному маршруті. На одному маршруті працює кілька водіїв. Кількість пасажирських місць залежить тільки від марки тролейбуса.

1.a Процедура з використанням WHILE та IF.

-- List all stops for a given route

```
CREATE PROCEDURE proc_route_list_stops(p_route varchar)
LANGUAGE plpgsql
AS $$
DECLARE
    stops_count int;
    idx int := 1;
    stop_ids int[];
BEGIN
    SELECT count(*) INTO stops_count FROM route_stop WHERE
route_number = p_route;

    IF stops_count = 0 THEN
        RAISE NOTICE 'Route % has no stops', p_route;
        RETURN;
    END IF;

    SELECT array_agg(stop_id ORDER BY stop_order) INTO stop_ids
        FROM route_stop WHERE route_number = p_route;
```

```

WHILE idx <= array_length(stop_ids, 1) LOOP
    RAISE NOTICE 'stop % -> id=%', idx, stop_ids[idx];
    idx := idx + 1;
END LOOP;
END;
$$;

```

```

trolley_depot=# call proc_route_list_stops('R1');
NOTICE:  stop 1 -> id=1
NOTICE:  stop 2 -> id=4
NOTICE:  stop 3 -> id=5
CALL

```

1.b Процедура без параметрів.

-- Insert missing inspections for trolleybuses for the current date

```

CREATE PROCEDURE proc_insert_missing_inspections()
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO inspection (trolleybus_number, inspection_date,
inspector, results)
    SELECT t.number, CURRENT_DATE, 'auto_system', NULL
    FROM trolleybus t
    WHERE NOT EXISTS (
        SELECT 1 FROM inspection i
        WHERE i.trolleybus_number = t.number AND i.inspection_date =
CURRENT_DATE
    );
END;
$$;

```

```
trolley_depot=# select * from inspection where inspection_date = current_date;
 id | trolleybus_number | inspection_date | results | inspector
-----+-----+-----+-----+-----
(0 rows)

trolley_depot=# call proc_insert_missing_inspections();
CALL
trolley_depot=# select * from inspection where inspection_date = current_date;
 id | trolleybus_number | inspection_date | results | inspector
-----+-----+-----+-----+-----
 16 | TB-101           | 2026-01-11     |         | auto_system
 17 | TB-102           | 2026-01-11     |         | auto_system
 18 | TB-103           | 2026-01-11     |         | auto_system
 19 | TB-104           | 2026-01-11     |         | auto_system
 20 | TB-105           | 2026-01-11     |         | auto_system
```

1.с Процедура, котра оновлює дані та повертає значення (використано функцію, оскільки PSQL не дозволяє процедурам повертати значення).

```
-- Set the duration of a route and get the number of affected rows
(should be 0 or 1)
```

```
CREATE FUNCTION fn_set_route_duration(p_route varchar, p_minutes
int) RETURNS integer
LANGUAGE plpgsql
AS $$
DECLARE
    updated_count int;
BEGIN
    IF p_minutes <= 0 THEN
        RAISE EXCEPTION 'duration_minutes must be > 0';
    END IF;

    UPDATE route SET duration_minutes = p_minutes WHERE number =
p_route;

    GET DIAGNOSTICS updated_count = ROW_COUNT;

    RETURN updated_count;
END;
$$;
```

```

trolley_depot=# select duration_minutes from route where number = 'R1';
 duration_minutes
-----
          120
(1 row)

trolley_depot=# select fn_set_route_duration('R1', 60);
 fn_set_route_duration
-----
          1
(1 row)

trolley_depot=# select duration_minutes from route where number = 'R1';
 duration_minutes
-----
          60
(1 row)

```

2.a Функція, котра повертає скалярне значення.

```
-- Get the date of the last inspection for a given trolleybus
```

```

CREATE FUNCTION fn_last_inspection_date(p_trolley varchar) RETURNS
date
LANGUAGE sql
AS $$
    SELECT max(inspection_date) FROM inspection WHERE
trolleybus_number = p_trolley;
$$;

```

```

trolley_depot=#
trolley_depot=# select fn_last_inspection_date('TB-101');
 fn_last_inspection_date
-----
    2026-01-11
(1 row)

```

2.b Функція, котра повертає таблицю.

```
-- Retrieve stops for a given route
```

```

CREATE FUNCTION fn_route_stops(p_route varchar)
RETURNS TABLE(stop_order int, stop_id int, stop_name varchar)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
        SELECT rs.stop_order, rs.stop_id, s.name
        FROM route_stop rs
        JOIN stop s ON s.id = rs.stop_id
        WHERE rs.route_number = p_route
        ORDER BY rs.stop_order;
END;
$$;

```

```

trolley_depot=# select * from fn_route_stops('R1');
 stop_order | stop_id | stop_name
-----+-----+-----
          1 |        1 | Central Station
          2 |        4 | Market
          3 |        5 | Airport
(3 rows)

```

3. Робота з курсорами (створення та відкриття курсору, вибірка даних, робота з курсорами).

```
-- Work with cursors: open, fetch, log, close
```

```

CREATE PROCEDURE proc_cursor_demo()
LANGUAGE plpgsql
AS $$
DECLARE
    cur refcursor;

```

```

rec record;

BEGIN

OPEN cur FOR SELECT id, name, location FROM stop ORDER BY id;

LOOP

    FETCH cur INTO rec;

    EXIT WHEN NOT FOUND;

    RAISE NOTICE 'Stop %: % (%)', rec.id, rec.name, rec.location;

END LOOP;

CLOSE cur;

END;

$$;

```

```

trolley_depot=# call proc_cursor_demo();
NOTICE:  Stop 1: Central Station (Main St 1)
NOTICE:  Stop 2: University (Campus Rd 5)
NOTICE:  Stop 3: City Hall (Liberty Sq 2)
NOTICE:  Stop 4: Market (Market St 8)
NOTICE:  Stop 5: Airport (Aeroporto Ave 3)
NOTICE:  Stop 6: Depot (Industrial Zone)
NOTICE:  Stop 7: Park (Green Rd 10)
NOTICE:  Stop 8: Hospital (Health Blvd 6)
NOTICE:  Stop 9: Theatre (Art St 11)

```

4. Створення таблиць для тригерів.

```

CREATE TABLE inspection_audit (
    id serial PRIMARY KEY,
    inspection_id int,
    trolleybus_number varchar(20),
    inspection_date date,
    inspector varchar,

```

```
results text,  
action varchar(10),  
changed_at timestamptz DEFAULT now()  
);
```

```
CREATE TABLE shift_audit (  
    id serial PRIMARY KEY,  
    shift_id int,  
    driver_id int,  
    work_date date,  
    start_time time,  
    end_time time,  
    action varchar(10),  
    changed_at timestamptz DEFAULT now()  
);
```

4.a Тригер, котрий спрацьовує при видаленні даних.

```
-- Log inspection deletions
```

```
CREATE FUNCTION trg_inspection_delete() RETURNS trigger  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    INSERT INTO inspection_audit (  
        inspection_id, trolleybus_number, inspection_date, inspector,  
        results, action  
    ) VALUES (OLD.id, OLD.trolleybus_number, OLD.inspection_date,  
        OLD.inspector, OLD.results, 'DELETE');  
    RETURN OLD;  
END;  
$$;
```

```
CREATE TRIGGER inspection_after_delete
AFTER DELETE ON inspection
FOR EACH ROW
EXECUTE FUNCTION trg_inspection_delete();
```

```
trolley_depot=# select * from inspection_audit;
 id | inspection_id | trolleybus_number | inspection_date | inspector | results | action | changed_at
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)

trolley_depot=# delete from inspection;
DELETE 15
trolley_depot=# select * from inspection_audit;
 id | inspection_id | trolleybus_number | inspection_date | inspector | results | action | changed_at
-----+-----+-----+-----+-----+-----+-----+-----
 1 |          1 | TB-101           | 2025-10-01      | Andrii Kovalenko |         | DELETE | 2026-01-11 18:36:38.505507+02
 2 |          2 | TB-102           | 2025-10-02      | Andrii Kovalenko | Brake pads replaced | DELETE | 2026-01-11 18:36:38.505507+02
 3 |          3 | TB-103           | 2025-10-03      | Andrii Kovalenko |         | DELETE | 2026-01-11 18:36:38.505507+02
 4 |          4 | TB-104           | 2025-10-04      | Oksana Bilyk | Minor oil leak fixed | DELETE | 2026-01-11 18:36:38.505507+02
 5 |          5 | TB-105           | 2025-10-05      | Oksana Bilyk |         | DELETE | 2026-01-11 18:36:38.505507+02
 6 |          6 | TB-106           | 2025-10-06      | Oksana Bilyk | Replaced mirrors | DELETE | 2026-01-11 18:36:38.505507+02
 7 |          7 | TB-107           | 2025-10-07      | Dmytro Hlushko |         | DELETE | 2026-01-11 18:36:38.505507+02
 8 |          8 | TB-108           | 2025-10-08      | Dmytro Hlushko |         | DELETE | 2026-01-11 18:36:38.505507+02
 9 |          9 | TB-109           | 2025-10-09      | Dmytro Hlushko | Electrical issue fixed | DELETE | 2026-01-11 18:36:38.505507+02
```

4.b Тригер, котрий спрацьовує при модифікації даних.

```
-- Ensure correct shift times and log updates
```

```
CREATE FUNCTION trg_shift_before_update() RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.end_time <= NEW.start_time THEN
        RAISE EXCEPTION 'end_time must be > start_time';
    END IF;

    INSERT INTO shift_audit (shift_id, driver_id, work_date,
start_time, end_time, action)
VALUES (OLD.id, OLD.driver_id, OLD.work_date, OLD.start_time,
OLD.end_time, 'UPDATE');

    RETURN NEW;
END;
$$;
```

```

CREATE TRIGGER shift_before_update
BEFORE UPDATE ON shift
FOR EACH ROW
EXECUTE FUNCTION trg_shift_before_update();

```

```

trolley_depot=# select * from shift limit 1;
 id | driver_id | work_date | start_time | end_time | trolleybus_number | route_number
-----+-----+-----+-----+-----+-----+-----
  1 |         1 | 2025-11-01 | 06:00:00  | 08:00:00 | TB-101            | R1
(1 row)

trolley_depot=# update shift set end_time = '05:00' where route_number = 'R1';
ERROR:  end_time must be > start_time
CONTEXT:  PL/pgSQL function trg_shift_before_update() line 4 at RAISE
trolley_depot=# update shift set end_time = '09:00' where route_number = 'R1';
UPDATE 1
trolley_depot=# select * from shift_audit;
 id | shift_id | driver_id | work_date | start_time | end_time | action | changed_at
-----+-----+-----+-----+-----+-----+-----+-----
  1 |         1 |         1 | 2025-11-01 | 06:00:00  | 08:00:00 | UPDATE | 2026-01-11 18:39:06.44775+02
(1 row)

```

4.c Тригер, котрий спрацьовує при додаванні даних.

-- Ensure no overlapping shifts for the same driver on the same day

```

CREATE FUNCTION trg_shift_before_insert() RETURNS trigger
LANGUAGE plpgsql
AS $$
DECLARE
    overlap_count int;
BEGIN
    SELECT count(*) INTO overlap_count
    FROM shift
    WHERE driver_id = NEW.driver_id
        AND work_date = NEW.work_date
        AND (NEW.start_time < end_time AND NEW.end_time > start_time);

    IF overlap_count > 0 THEN
        RAISE EXCEPTION 'Driver % has overlapping shift on %',
NEW.driver_id, NEW.work_date;
    END IF;

```

```

RETURN NEW;

END;

$$;

CREATE TRIGGER shift_before_insert
BEFORE INSERT ON shift
FOR EACH ROW
EXECUTE FUNCTION trg_shift_before_insert();

```

```

trolley_depot=# select * from shift limit 1;
 id | driver_id | work_date | start_time | end_time | trolleybus_number | route_number
-----+-----+-----+-----+-----+-----+-----
  2 |         2 | 2025-11-01 | 08:00:00  | 10:00:00 | TB-102            | R2
(1 row)

trolley_depot=# insert into shift (driver_id, work_date, start_time, end_time, trolleybus_number, route_number)
values (2, '2025-11-01', '08:00:00', '10:00:00', 'TB-102', 'R2');
ERROR:  Driver 2 has overlapping shift on 2025-11-01
CONTEXT:  PL/pgSQL function trg_shift_before_insert() line 13 at RAISE
trolley_depot=# insert into shift (driver_id, work_date, start_time, end_time, trolleybus_number, route_number)
values (2, '2025-11-01', '12:00:00', '14:00:00', 'TB-102', 'R2');
INSERT 0 1
trolley_depot=# select * from shift_audit;
 id | shift_id | driver_id | work_date | start_time | end_time | action | changed_at
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)

```

Висновок

У ході виконання лабораторної роботи було створено збережені процедури та функції для обробки й аналізу даних, зокрема з використанням умовних конструкцій, циклів та параметрів. Реалізовано роботу з курсорами для поетапної обробки вибірок даних. Також були розроблені тригери, що забезпечують контроль цілісності даних та автоматичну реакцію системи на операції додавання, зміни й видалення записів. Отримані рішення демонструють можливості автоматизації бізнес-логіки на рівні бази даних та підвищують надійність і узгодженість збереженої інформації.