**Київський національний університет**
**Імені Тараса Шевченка.**
Кафедра: Мережевих та інтернет-технологій.

# Лабораторна робота №4

з дисципліни: Бази даних та інформаційні системи
На тему:«Розширення можливостей PostgreSQL: користувацькі типи, функції та тригери”

Студента 3 курсу:
Групи МІТ-31
Мулико Володимира

**Київ - 2025р.**

**Мета роботи:** Закріпити знання з розширюваності PostgreSQL. Навчитися створювати користувацькі типи даних. Реалізувати власну користувацьку функцію або агрегат. Створити тригери для логування змін у базі даних. Автоматично оновлювати пов'язані таблиці чи заповнювати значення. Оновити діаграму бази даних відповідно до виконаних завдань. Перевірити коректність роботи реалізованих об'єктів через виконання тестових SQL-запитів.

## Завдання

Кожен студент має виконати завдання на основі своєї інформаційної системи, створеної в Лабораторній роботі №2. Усі зміни необхідно реалізувати у власній базі даних відповідно до обраного варіанту бізнес-процесу.

### 1. Створення користувацького типу даних

- Визначити один або кілька атрибутів у базі даних, які можна винести в окремий користувацький тип даних. Це може бути ENUM (для обмеженого набору значень) або Composite Type (якщо потрібно зберігати структуровані дані в одному полі).
- Створити користувацький тип, що відповідає логіці інформаційної системи та забезпечує розширюваність бази даних.
- Додати новий стовпець із цим типом до відповідної таблиці в базі даних, змінюючи її структуру для кращої адаптації до майбутніх оновлень.
- Переконатися, що зміни не порушують цілісність даних, підтримують взаємозв'язки між таблицями та узгоджуються з існуючою структурою бази.

### 2. Створення користувацької функції або агрегату

- Реалізувати функцію або агрегат, що підходить для аналізу даних вашої інформаційної системи.
- Функція повинна виконувати розширені обчислення або маніпуляції з даними, які неможливо здійснити стандартними SQL-запитами.
- Використати створену функцію/агрегат у тестовому запиті до таблиць бази даних та перевірити її ефективність.

- Дослідити, як використання таких функцій покращує продуктивність та зручність роботи з базою даних.

**3. Створення тригерів для логування змін та автоматичного оновлення пов’язаних таблиць**

- Створити таблицю логування змін у критично важливих таблицях вашої бази даних.
- Написати тригерну функцію на PL/pgSQL для логування операцій INSERT, UPDATE та DELETE, забезпечуючи можливість відстеження змін у базі.
- Прив’язати тригер до відповідної таблиці, щоб автоматично фіксувати кожну зміну, покращуючи аудит змін.
- Створити тригер для автоматичного оновлення пов’язаних таблиць або заповнення значень у залежних таблицях, щоб мінімізувати дублювання даних та покращити зв’язність структури бази.
- Використати тригер для обчислення загальних підсумків або створення автоматичних звітів на основі змін у таблицях.

**4. Оновлення діаграми бази даних**

- Внести зміни у структуру бази даних на основі виконаних завдань, додаючи нові сутності або зв’язки між таблицями.
- Оновити ER-діаграму (Entity-Relationship Diagram), додавши нові таблиці, стовпці, зв’язки, тригери та функції.
- Візуалізувати нову структуру для кращого розуміння взаємодії між сутностями.
- Переконатися, що всі зміни документовані та відповідають логіці проєкту, забезпечуючи майбутню розширюваність бази даних.

**5. Перевірка роботи**

- Виконати тестові SQL-запити, що демонструють роботу створених об'єктів у реальних сценаріях використання.
- Переконатися, що дані правильно логуються, функції та агрегати працюють згідно з очікуваннями, а тригери оновлюють пов’язані таблиці відповідно до встановленої логіки.
- Проаналізувати швидкість виконання запитів з використанням створених функцій та тригерів.

- Переконатися, що оновлена діаграма бази даних коректно відображає всі зміни та узгоджується з бізнес-логікою проєкту.
- Підготувати порівняльний аналіз результатів, пояснюючи, як реалізовані рішення покращують ефективність бази даних.

Код для завдання: 1-3, 5.

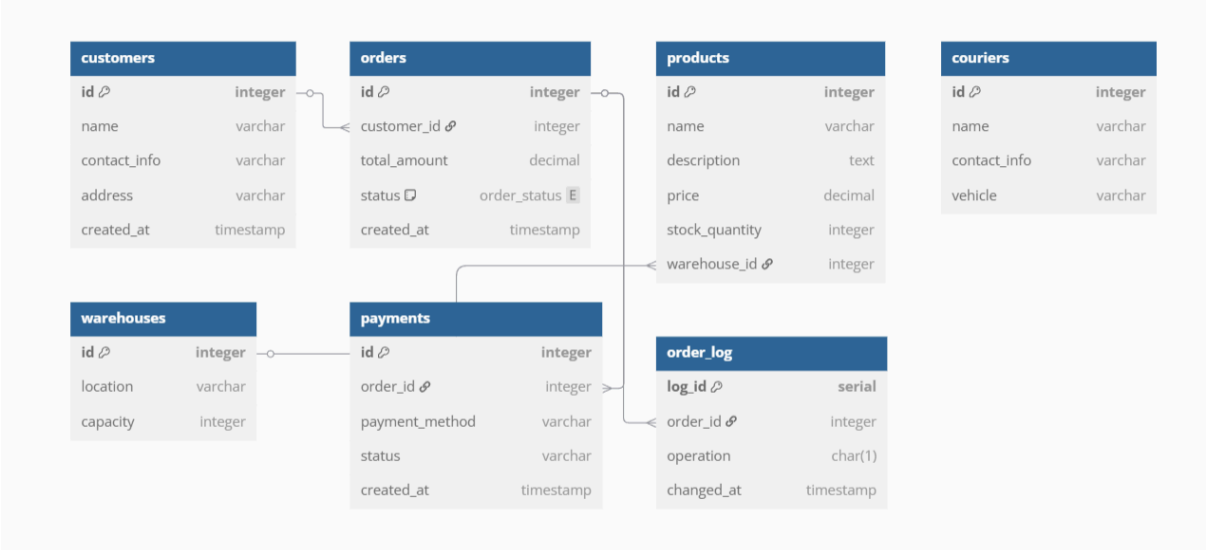
```
-- Завдання 1 --
-- Створення ENUM для статусів замовлення (якщо не існує)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'order_status')
THEN
      CREATE TYPE order_status AS ENUM ('Очікується', 'Оплачено', 'Скасовано',
'Доставлено');
  END IF;
END $$;

-- Оновлення типу колонки status в таблиці orders
ALTER TABLE orders
  ALTER COLUMN status TYPE order_status USING status::order_status;

-- Створення композитного типу contact_info (якщо не існує)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'contact_info')
THEN
     CREATE TYPE contact_info AS (
       phone TEXT,
       email TEXT
     );
  END IF;
END $$;

-- Додавання стовпця contact_info до таблиці customers, якщо він ще не існує
ALTER TABLE customers
        ADD COLUMN IF NOT EXISTS contact_info contact_info DEFAULT
((",")::contact_info);

-- Оновлення записів, де contact_info має NULL
UPDATE customers
  SET contact_info = ((",")::contact_info)
  WHERE contact_info IS NULL;

-- Встановлення обмеження NOT NULL для contact_info
ALTER TABLE customers
```

```
   ALTER COLUMN contact_info SET NOT NULL;

-- Оновлення записів, де addres є NULL (за умовчанням "Невідома адреса")
UPDATE customers
   SET addres = 'Невідома адреса'
   WHERE addres IS NULL;

-- Додавання стовпця total_spent до таблиці customers, якщо він ще не існує
ALTER TABLE customers
   ADD COLUMN IF NOT EXISTS total_spent NUMERIC DEFAULT 0;

-- Завдання 2 --
CREATE OR REPLACE FUNCTION total_spent_by_customer(p_customer_id INT)
RETURNS NUMERIC AS $$
DECLARE
   total NUMERIC;
BEGIN
   SELECT COALESCE(SUM(total_amount), 0)
     INTO total
     FROM orders
    WHERE customer_id = p_customer_id;
   RETURN total;
END;
$$ LANGUAGE plpgsql;

-- Тест функції
SELECT total_spent_by_customer(1);

-- Завдання 3 --
-- Видалення таблиці логування, якщо вона існує
DROP TABLE IF EXISTS orders_log CASCADE;

-- Створення таблиці логування
CREATE TABLE orders_log (
   log_id SERIAL PRIMARY KEY,
   order_id INT,
   operation CHAR(1) CHECK (operation IN ('I', 'U', 'D')),
   changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Функція логування змін (UPDATE/DELETE)
CREATE OR REPLACE FUNCTION log_order_changes() RETURNS TRIGGER AS
$$
BEGIN
   INSERT INTO orders_log (order_id, operation, changed_at)
   VALUES (OLD.id,
         CASE
            WHEN TG_OP = 'UPDATE' THEN 'U'
```

```
            WHEN TG_OP = 'DELETE' THEN 'D'
        END,
        NOW());
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Функція логування вставки (INSERT)
CREATE OR REPLACE FUNCTION log_order_insert() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO orders_log (order_id, operation, changed_at)
    VALUES (NEW.id, 'I', NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Видалення існуючих тригерів та створення нових для таблиці orders
DROP TRIGGER IF EXISTS track_order_changes ON orders;
CREATE TRIGGER track_order_changes
    AFTER UPDATE OR DELETE ON orders
    FOR EACH ROW EXECUTE FUNCTION log_order_changes();

DROP TRIGGER IF EXISTS track_order_insert ON orders;
CREATE TRIGGER track_order_insert
    AFTER INSERT ON orders
    FOR EACH ROW EXECUTE FUNCTION log_order_insert();

-- Функція та тригер для автоматичного оновлення загальної суми витрат клієнта
DROP TRIGGER IF EXISTS update_customer_spending ON orders;
CREATE OR REPLACE FUNCTION update_customer_total_spent() RETURNS
TRIGGER AS $$
BEGIN
    UPDATE customers
       SET total_spent = total_spent_by_customer(NEW.customer_id)
     WHERE id = NEW.customer_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_customer_spending
    AFTER INSERT OR UPDATE OR DELETE ON orders
    FOR EACH ROW EXECUTE FUNCTION update_customer_total_spent();

-- Завдання 5 --
-- Вставка нового клієнта із зазначенням усіх необхідних полів:
-- id, name, addres, contact_info, created_at
INSERT INTO customers (id, name, addres, contact_info, created_at)
VALUES (1,
```

```
        'Іван Петренко',
        'Невідома адреса',
        (('+380501234567','ivan@example.com')::contact_info),
        NOW())
ON CONFLICT (id) DO NOTHING;

-- Вставка нового замовлення
INSERT INTO orders (id, customer_id, total_amount, status, created_at)
VALUES (5, 1, 500.00, 'Оплачено', NOW());

-- Перевірка логу вставки
SELECT * FROM orders_log WHERE order_id = 5;

-- Оновлення замовлення
UPDATE orders SET total_amount = 600.00 WHERE id = 5;

-- Перевірка логу оновлення
SELECT * FROM orders_log WHERE order_id = 5;

-- Видалення замовлення
DELETE FROM orders WHERE id = 5;

-- Перевірка логу видалення
SELECT * FROM orders_log WHERE order_id = 5;

-- Перевірка оновленої загальної суми витрат клієнта
SELECT total_spent_by_customer(1);
```

Завдання 4 (розширення існуючої таблиці):



Висновки: В цій лабораторній роботі ми роздивлялися покращення вже існуючого функціоналу з таблиць. Ми розширили функціонал нашої бази даних, та провели загальний тест працюючих тригерів. Ми повністю роздивилися як працює наша система з покращенням функціоналу. Створили користувацький агрегат та функціонал, написали логування для покращення виводу нашої таблиці. Зрозуміли що таблицю можна запрограмувати по будь-якій системі, для будь-яких запитів та прохань.