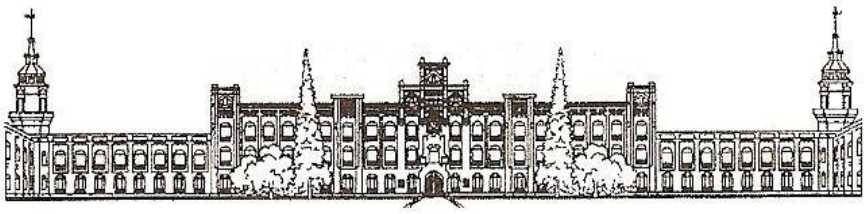


Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Розрахунково-графічна робота

з дисципліни «Бази даних»

«Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL»

Виконала студентка групи: КВ-31

ПІБ: Кузьменко О. А.

Перевірив: Петрашенко А. В.

Київ 2025

Мета роботи: здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

Виконання роботи

Сутності предметної області

Для побудови бази даних предметної області «Система інвентаризації складського облік» було виділено такі сутності:

1. **Постачання (Supply)** – представляє операції надходження товарів на склад:

   – Атрибути: ідентифікатор постачання, ідентифікатор постачальника, ідентифікатор товару, дата постачання, номер документа, кількість, ціна за одиницю.

2. **Постачальник (Supplier)** – представляє компанії, які постачають товари:

   – Атрибути: ідентифікатор постачальника, назва компанії, контактна особа, телефон, електронна пошта.

3. **Товар (Product)** – представляє номенклатуру товарів на складі:

   – Атрибути: ідентифікатор товару, назва товару, одиниця виміру, мінімальний залишок, категорія.

4. **Складський облік (Inventory)** – представляє поточний стан товарів на складі:

– Атрибути: ідентифікатор обліку, ідентифікатор товару, кількість, дата оновлення, розташування.

Зв’язки між сутностями предметної області

Зв’язки між сутностями:

1. Зв'язок "Постачальник" – "Постачання":

   Тип зв'язку: 1 до N (один постачальник може здійснити багато постачань; одне постачання належить лише одному постачальнику).

   Пояснення: Кожен постачальник може поставляти товари багато

разів, але конкретне постачання завжди пов'язане з одним постачальником.

2. Зв'язок "Постачання" – "Товар":

   Тип зв'язку: N до M (одне постачання може містити багато товарів; один товар може бути в багатьох постачаннях). Зв'язок з атрибутом: кількість товару, ціна за одиницю. Пояснення: Це зв'язок "багато-до-багатьох", який реалізується через асоціативну таблицю Supply_Product. Атрибути зв'язку зберігають конкретну інформацію про кожен товар у кожному постачанні.

3. Зв'язок "Товар" – "Складський облік":

   Тип зв'язку: 1 до 1 (кожен товар має один запис в обліку; кожен запис обліку відноситься до одного товару). Пояснення: Кожна позиція товару має унікальний запис про його поточну кількість на складі.

Концептуальна модель предметної області “Система управління процесом навчання на курсах ”

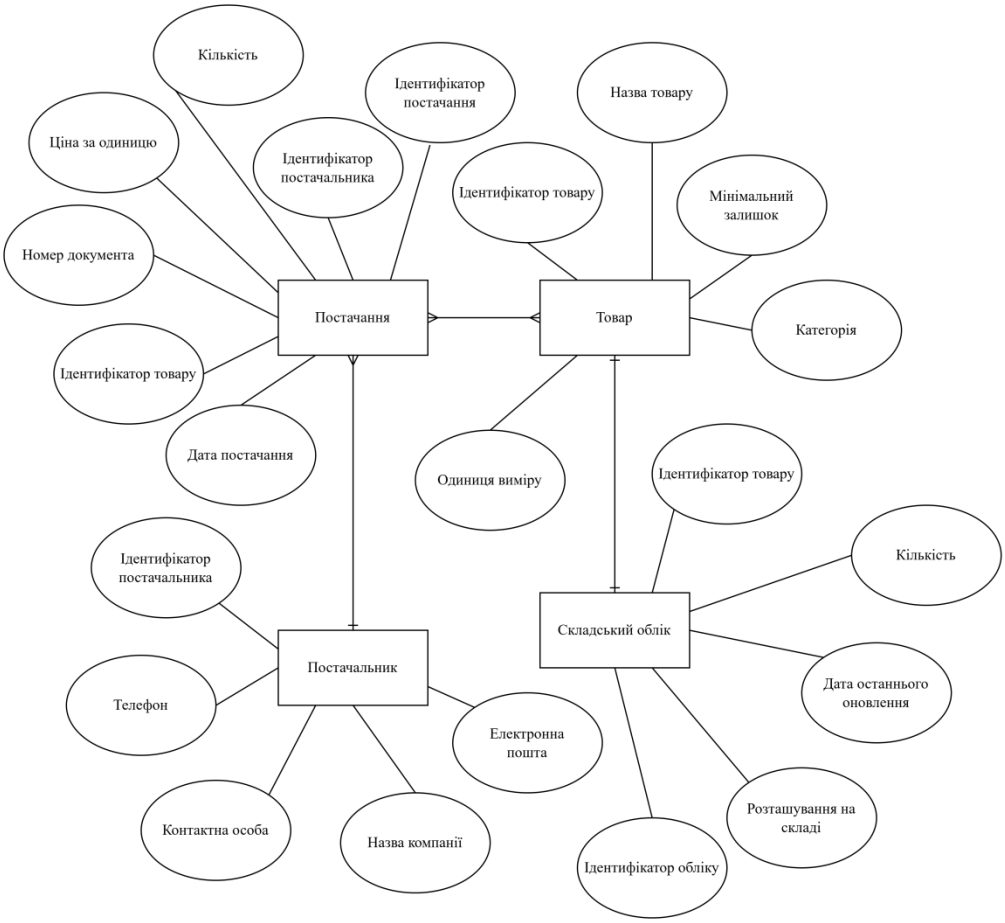


Рисунок 1 – ER-діаграма, побудована за нотацією” Пташина лапка”

Середовище та компоненти розробки

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

Шаблон проектування

MVC — це шаблон проектування, який використовується в програмі.

Model — це частина, що описує логіку обробки даних. У моїй програмі це реалізовано у файлі `models.py`, де містяться функції для підключення до бази даних, виконання SQL-запитів, а також для генерації, перегляду, редагування, додавання та видалення даних.

View — це частина, що відповідає за інтерфейс взаємодії з користувачем. В моєму випадку це консольний інтерфейс, де за допомогою бібліотеки `tabulate` відображаються таблиці, повідомлення про виконані дії та вводяться необхідні дані.

Controller — це компонент, який забезпечує зв'язок між користувачем, інтерфейсом і моделлю. У моїй програмі цей компонент реалізований у файлі `controllers.py`, де обробляються введені користувачем дані, викликаються відповідні функції моделі, контролюються помилки і виводяться результати в консоль.

Структура програми та її опис

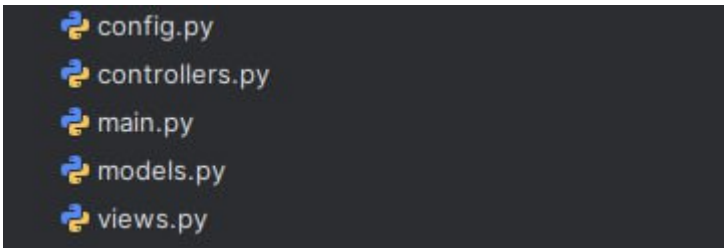


Рисунок 2 - Структура програми

Програма умовно поділена на 3 модулі:

1. файл config.py
2. файл controllers.py;
3. файл main.py
4. файл modesl.py;
5. файл view.py.

Модулі, як випливає з їхніх назв, повністю відповідають шаблону проектування MVC.

У файлі models.py описано клас моделі, який відповідає за взаємодію з базою даних PostgreSQL. Він забезпечує підключення до бази даних, виконання

SQL-запитів, а також операції з додавання, редагування, вилучення, перегляду та генерації даних.

У файлі controllers.py реалізовано контролер, який відповідає за обробку введення від користувача. Він передає запити до моделі, обробляє отримані результати та готує їх для відображення у поданні.

Компонент View реалізовано через консольний інтерфейс, який відповідає за виведення результатів роботи програми, відображення таблиць і повідомлень, а також за прийом введення від користувача.

Файл config.py містить параметри підключення до бази даних (назва БД, користувач, пароль, хост, порт), що дозволяє легко змінювати налаштування без необхідності редагування основного коду.

Файл main.py є точкою входу до програми, ініціалізує контролер та запускає головний цикл програми, забезпечуючи взаємодію між усіма компонентами.

Структура меню програми

Меню для користувача складається з 10-ти пунктів (Рисунок 4).

```
Меню:
1) Показати таблиці
2) Показати перші записи таблиці
3) Показати запис за PK
4) Додати запис
5) Редагувати запис
6) Видалити запис
7) Згенерувати дані SQL-на-сервері (generate_series)
8) Виконати складні запити (3 варіанти)
9) Перевірити наявність дітей перед видаленням (демо)
0) Вийти

Виберіть опцію:
```

Рисунок 4 - Меню програми

Фрагмент коду (файл Controller.py), в якому наведено виклик функцій та значення, які їм передаються

```
      # controllers.py
from models import DBModel
import views
from typing import Dict, Any


class Controller:
    def __init__(self):
        self.model = DBModel()

        # Нові таблиці твоєї бази:
        self.tables = [
            "supplier",
            "product",
            "supply",
            "inventory",
        ]

    def close(self):
        self.model.close()

    def run(self):
        views.print_banner()
        while True:
            views.show_menu()
            choice = views.prompt("Виберіть опцію")
            if choice == "1":
                self.action_list_tables()
            elif choice == "2":
                self.action_show_table()
            elif choice == "3":
                self.action_show_by_pk()
            elif choice == "4":
                self.action_insert()
            elif choice == "5":
                self.action_update()
            elif choice == "6":
                self.action_delete()
            elif choice == "7":
                self.action_generate()
            elif choice == "8":
                self.action_complex_queries()
            elif choice == "9":
                self.action_demo_check_children()
            elif choice == "0":
                print("До побачення!")
                break
            else:
                print("Невірний вибір. Спробуйте ще.")

    # ------------------------------------------------------------
    # 1. Список таблиць
    # ------------------------------------------------------------
    def action_list_tables(self):
        try:
```

```
            tables = self.model.list_tables()
            views.print_tables(tables)
        except Exception as e:
            views.show_error(str(e))

    # ---------------------------------------------------------
    # 2. Показати таблицю
    # ---------------------------------------------------------
    def action_show_table(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця. Приклад: supplier")
            return
        try:
            rows = self.model.select_all(table, limit=500)
            views.print_rows(rows)
        except Exception as e:
            views.show_error("Не вдалося отримати записи.")

    # ---------------------------------------------------------
    # 3. Отримати запис за PK
    # ---------------------------------------------------------
    def action_show_by_pk(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        if not pk:
            views.show_error("PK не знайдено")
            return

        val = views.prompt(f"Значення PK ({pk})")

        # автоматичне визначення типу PK
        col_info = next((c for c in self.model.columns_info(table) if
c['name'] == pk), None)
        if col_info and "int" in col_info["type"]:
            try:
                val = int(val)
            except:
                views.show_error("PK має бути числом")
                return

        try:
            row = self.model.select_by_pk(table, pk, val)
            views.print_row(row)
        except:
            views.show_error("Помилка при отриманні")

    # ---------------------------------------------------------
    # Універсальний ввід+валідація
    # ---------------------------------------------------------
    def _input_and_validate_for_table(self, table: str, skip_pk=True) ->
Dict[str, Any]:
        cols = self.model.columns_info(table)
        pk = self.model.primary_key(table)
        data = {}
```

```
        for c in cols:
            name = c["name"]
            dtype = c["type"]

            if skip_pk and name == pk:
                continue

            raw = views.prompt_nullable(f"{name} ({dtype})")

            if raw is None:
                if not c["nullable"]:
                    views.show_error(f"Поле {name} не може бути пустим.")
                    return self._input_and_validate_for_table(table,
skip_pk)
                data[name] = None
                continue

            # --- типи ---
            if "int" in dtype:
                try:
                    data[name] = int(raw)
                except:
                    views.show_error(f"{name} очікується integer")
                    return self._input_and_validate_for_table(table,
skip_pk)
            elif dtype in ("numeric", "real", "double precision",
"decimal"):
                try:
                    data[name] = float(raw)
                except:
                    views.show_error(f"{name} очікується число")
                    return self._input_and_validate_for_table(table,
skip_pk)
            elif dtype == "date":
                parsed = self.model.parse_date(raw)
                if not parsed:
                    views.show_error(f"Невірний формат дати {name}")
                    return self._input_and_validate_for_table(table,
skip_pk)
                data[name] = parsed
            else:
                data[name] = raw

        return data

    # ------------------------------------------------------------
    # 4. INSERT
    # ------------------------------------------------------------
    def action_insert(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        data = self._input_and_validate_for_table(table)

        # Перевірка FK для supply і inventory
        if table == "supply":
            checks = [
                ("supplier", "supplier_id"),
```

```
                ("product", "product_id"),
            ]
        elif table == "inventory":
            checks = [
                ("product", "product_id"),
            ]
        else:
            checks = []

        for parent_table, col in checks:
            val = data.get(col)
            if val is None or not self.model.parent_exists(parent_table,
col, val):
                views.show_error(f"{col}={val} не існує у {parent_table}")
                return

        success, err = self.model.insert(table, data)
        if success:
            views.show_success("Запис додано.")
        else:
            views.show_error(f"Помилка: {err}")

    # ------------------------------------------------------------
    # 5. UPDATE
    # ------------------------------------------------------------
    def action_update(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        pk_raw = views.prompt(f"PK ({pk})")

        col_info = next((c for c in self.model.columns_info(table) if
c["name"] == pk), None)
        if col_info and "int" in col_info["type"]:
            try:
                pk_val = int(pk_raw)
            except:
                views.show_error("PK має бути числом")
                return
        else:
            pk_val = pk_raw

        row = self.model.select_by_pk(table, pk, pk_val)
        if not row:
            views.show_error("Рядок не знайдено")
            return

        updates = {}
        for c in self.model.columns_info(table):
            if c["name"] == pk:
                continue
            raw = views.prompt_nullable(f"{c['name']} (поточне:
{row[c['name']]})")
            if raw is None:
                continue
            updates[c["name"]] = raw
```

```
            if not updates:
                views.show_message("Нічого не змінено.")
                return

            success, err = self.model.update(table, pk, pk_val, updates)
            if success:
                views.show_success("Оновлено.")
            else:
                views.show_error(f"Помилка: {err}")

        # ------------------------------------------------------------
        # 6. DELETE
        # ------------------------------------------------------------
        def action_delete(self):
            table = views.prompt("Назва таблиці")
            if table not in self.tables:
                views.show_error("Невідома таблиця")
                return

            pk = self.model.primary_key(table)
            pk_raw = views.prompt(f"PK ({pk})")

            pk_col = next((c for c in self.model.columns_info(table) if
c["name"] == pk), None)
            if pk_col and "int" in pk_col["type"]:
                try:
                    pk_val = int(pk_raw)
                except:
                    views.show_error("PK має бути числом")
                    return
            else:
                pk_val = pk_raw

            try:
                if self.model.has_child_rows(table, pk, pk_val):
                    views.show_error("Не можна видалити — є дочірні записи.")
                    return
            except:
                views.show_error("Не вдалося перевірити залежності.")
                return

            if views.prompt("Підтвердити (так/ні)?").lower() not in ("так",
"y", "yes"):
                views.show_message("Скасовано.")
                return

            success, err = self.model.delete(table, pk, pk_val)
            if success:
                views.show_success("Видалено.")
            else:
                views.show_error(f"Не вдалося: {err}")

        # ------------------------------------------------------------
        # 7. Генерація даних
        # ------------------------------------------------------------
        def action_generate(self):
            count_raw = views.prompt("Скільки записів генерувати?")
            try:
                count = int(count_raw)
            except:
```

```
            views.show_error("Потрібно число.")
            return

        tasks = [
            ("supplier", self.model.generate_suppliers),
            ("product", self.model.generate_products),
            ("supply", self.model.generate_supplies),
            ("inventory", self.model.generate_inventory),
        ]

        for name, func in tasks:
            ok, err = func(count)
            if ok:
                views.show_success(f"{name}: згенеровано {count}")
            else:
                views.show_error(f"{name}: помилка — {err}")

    # ------------------------------------------------------------
    # 8. Складні SQL запити
    # ------------------------------------------------------------
    def action_complex_queries(self):
        views.show_message("1) Загальна сума постачань по постачальниках")
        views.show_message("2) Товари нижче мінімального запасу")
        views.show_message("3) Найдорожчі категорії постачань")
        views.show_message("4) ТОП-10 товарів за обсягом постачань")
        views.show_message("5) Постачання за останні 30 днів")

        choice = views.prompt("Виберіть запит (1-5)")

        query_map = {
            "1": self.model.query_supplier_totals,
            "2": self.model.query_products_below_min_stock,
            "3": self.model.query_category_supply_costs,
            "4": self.model.query_top_products_by_supply_volume,
            "5": self.model.query_last_month_supplies,
        }

        func = query_map.get(choice)
        if not func:
            views.show_error("Невірний вибір")
            return

        rows, time_ms, explain, err = func()
        if err:
            views.show_error(err)
            return

        views.show_query_result(rows, time_ms, explain)

    # ------------------------------------------------------------
    # 9. Перевірка залежностей
    # ------------------------------------------------------------
    def action_demo_check_children(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        val = views.prompt(f"PK ({pk})")
```

```
    try:
        has = self.model.has_child_rows(table, pk, val)
    except:
        views.show_error("Помилка перевірки.")
        return

    if has:
        views.show_message("Є дочірні записи.")
    else:
        views.show_message("Немає дочірніх записів.")
```

Фрагменти програм внесення, редагування, вилучення та генерації даних у базі даних

Фрагмент програми внесення даних

```
-   def insert(self, table: str, data: Dict[str, Any]) -> Tuple[bool,
    Optional[str]]:
-       cols = list(data.keys())
-       vals = [data[c] for c in cols]
-       query = sql.SQL('INSERT INTO {} ({}) VALUES ({})').format(
-           sql.Identifier(table),
-           sql.SQL(', ').join(map(sql.Identifier, cols)),
-           sql.SQL(', ').join(sql.Placeholder() * len(cols))
-       )
-       with self.conn.cursor() as cur:
-           try:
-               cur.execute(query, vals)
-               return True, None
-           except psycopg2.Error as e:
-               return False, e.pgerror or str(e)
```

Фрагмент програми редагування даних

```
def update(self, table: str, pk: str, pk_value: Any, data: Dict[str,
Any]) -> Tuple[bool, Optional[str]]:
    cols = list(data.keys())
    vals = [data[c] for c in cols] + [pk_value]
    set_clause = sql.SQL(', ').join(
        sql.Composed([sql.Identifier(c), sql.SQL(' = '),
sql.Placeholder()]) for c in cols
    )
    query = sql.SQL('UPDATE {} SET {} WHERE {} = %s').format(
        sql.Identifier(table),
        set_clause,
        sql.Identifier(pk)
    )
    with self.conn.cursor() as cur:
```

```
            try:
                cur.execute(query, vals)
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)
```

Фрагмент програми видалення даних

```
    def delete(self, table: str, pk: str, pk_value: Any) -> Tuple[bool,
    Optional[str]]:
        query = sql.SQL('DELETE FROM {} WHERE {} = %s').format(
            sql.Identifier(table),
            sql.Identifier(pk)
        )
        with self.conn.cursor() as cur:
            try:
                cur.execute(query, (pk_value,))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)
```

Фрагмент програми генерації випадкових значень

-

```
    def generate_suppliers(self, count: int) -> Tuple[bool,
Optional[str]]:
        first_names = ["Іван", "Петро", "Ольга", "Марія", "Андрій"]
        last_names = ["Іванов", "Петренко", "Сидоренко", "Коваленко",
"Бондаренко"]
        domains = ["example.ua", "mail.ua", "suppliers.ua"]
        with self.conn.cursor() as cur:
            try:
                cur.execute("SELECT COALESCE(MAX(supplier_id),0)+1 FROM
supplier")
                start_id = cur.fetchone()[0]
                for i in range(count):
                    supplier_id = start_id + i
                    company_name = f"Компанія {supplier_id}"
                    contact_person = f"{random.choice(first_names)}
{random.choice(last_names)}"
                    phone = f"+380{random.randint(500000000, 999999999)}"
                    email = f"user{supplier_id}@{random.choice(domains)}"
                    cur.execute("""
                        INSERT INTO supplier(supplier_id, company_name,
contact_person, phone, email)
                        VALUES (%s,%s,%s,%s,%s)
                    """, (supplier_id, company_name, contact_person,
phone, email))
                return True, None
            except psycopg2.Error as e:
```

```
return False, e.pgerror or str(e)
```

Дані фрагменти програми, які наведені вище, відповідають за функціонал додавання даних, редагування та вилучення даних у базі даних.

Взаємодія відбувається через клас Model, який займається підключенням до БД, а самі функції знаходяться у файлі Controller.

Повний текст програми:

Модуль Controller.py

```
# controllers.py
from models import DBModel
import views
from typing import Dict, Any


class Controller:
    def __init__(self):
        self.model = DBModel()

        # Нові таблиці твоєї бази:
        self.tables = [
            "supplier",
            "product",
            "supply",
            "inventory",
        ]

    def close(self):
        self.model.close()

    def run(self):
        views.print_banner()
        while True:
            views.show_menu()
            choice = views.prompt("Виберіть опцію")
            if choice == "1":
                self.action_list_tables()
            elif choice == "2":
                self.action_show_table()
            elif choice == "3":
                self.action_show_by_pk()
            elif choice == "4":
                self.action_insert()
            elif choice == "5":
                self.action_update()
            elif choice == "6":
                self.action_delete()
            elif choice == "7":
```

```
            self.action_generate()
        elif choice == "8":
            self.action_complex_queries()
        elif choice == "9":
            self.action_demo_check_children()
        elif choice == "0":
            print("До побачення!")
            break
        else:
            print("Невірний вибір. Спробуйте ще.")

# ------------------------------------------------------------
# 1. Список таблиць
# ------------------------------------------------------------
def action_list_tables(self):
    try:
        tables = self.model.list_tables()
        views.print_tables(tables)
    except Exception as e:
        views.show_error(str(e))

# ------------------------------------------------------------
# 2. Показати таблицю
# ------------------------------------------------------------
def action_show_table(self):
    table = views.prompt("Назва таблиці")
    if table not in self.tables:
        views.show_error("Невідома таблиця. Приклад: supplier")
        return
    try:
        rows = self.model.select_all(table, limit=500)
        views.print_rows(rows)
    except Exception as e:
        views.show_error("Не вдалося отримати записи.")

# ------------------------------------------------------------
# 3. Отримати запис за PK
# ------------------------------------------------------------
def action_show_by_pk(self):
    table = views.prompt("Назва таблиці")
    if table not in self.tables:
        views.show_error("Невідома таблиця")
        return

    pk = self.model.primary_key(table)
    if not pk:
        views.show_error("PK не знайдено")
        return

    val = views.prompt(f"Значення PK ({pk})")
```

```
        # автоматичне визначення типу PK
        col_info = next((c for c in self.model.columns_info(table) if c['name']
== pk), None)
        if col_info and "int" in col_info["type"]:
            try:
                val = int(val)
            except:
                views.show_error("PK має бути числом")
                return

        try:
            row = self.model.select_by_pk(table, pk, val)
            views.print_row(row)
        except:
            views.show_error("Помилка при отриманні")

    # ------------------------------------------------------------
    # Універсальний ввід+валідація
    # ------------------------------------------------------------
    def _input_and_validate_for_table(self, table: str, skip_pk=True) ->
Dict[str, Any]:
        cols = self.model.columns_info(table)
        pk = self.model.primary_key(table)
        data = {}

        for c in cols:
            name = c["name"]
            dtype = c["type"]

            if skip_pk and name == pk:
                continue

            raw = views.prompt_nullable(f"{name} ({dtype})")

            if raw is None:
                if not c["nullable"]:
                    views.show_error(f"Поле {name} не може бути пустим.")
                    return self._input_and_validate_for_table(table, skip_pk)
                data[name] = None
                continue

            # --- типи ---
            if "int" in dtype:
                try:
                    data[name] = int(raw)
                except:
                    views.show_error(f"{name} очікується integer")
                    return self._input_and_validate_for_table(table, skip_pk)
            elif dtype in ("numeric", "real", "double precision", "decimal"):
```

```
                try:
                    data[name] = float(raw)
                except:
                    views.show_error(f"{name} очікується число")
                    return self._input_and_validate_for_table(table, skip_pk)
            elif dtype == "date":
                parsed = self.model.parse_date(raw)
                if not parsed:
                    views.show_error(f"Невірний формат дати {name}")
                    return self._input_and_validate_for_table(table, skip_pk)
                data[name] = parsed
            else:
                data[name] = raw

        return data

    # ------------------------------------------------------------
    # 4. INSERT
    # ------------------------------------------------------------
    def action_insert(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        data = self._input_and_validate_for_table(table)

        # Перевірка FK для supply і inventory
        if table == "supply":
            checks = [
                ("supplier", "supplier_id"),
                ("product", "product_id"),
            ]
        elif table == "inventory":
            checks = [
                ("product", "product_id"),
            ]
        else:
            checks = []

        for parent_table, col in checks:
            val = data.get(col)
            if val is None or not self.model.parent_exists(parent_table, col,
val):
                views.show_error(f"{col}={val} не існує у {parent_table}")
                return

        success, err = self.model.insert(table, data)
        if success:
            views.show_success("Запис додано.")
```

```
        else:
            views.show_error(f"Помилка: {err}")

    # ------------------------------------------------------------
    # 5. UPDATE
    # ------------------------------------------------------------
    def action_update(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        pk_raw = views.prompt(f"PK ({pk})")

        col_info = next((c for c in self.model.columns_info(table) if c["name"]
== pk), None)
        if col_info and "int" in col_info["type"]:
            try:
                pk_val = int(pk_raw)
            except:
                views.show_error("PK має бути числом")
                return
        else:
            pk_val = pk_raw

        row = self.model.select_by_pk(table, pk, pk_val)
        if not row:
            views.show_error("Рядок не знайдено")
            return

        updates = {}
        for c in self.model.columns_info(table):
            if c["name"] == pk:
                continue
            raw = views.prompt_nullable(f"{c['name']} (поточне:
{row[c['name']]})")
            if raw is None:
                continue
            updates[c["name"]] = raw

        if not updates:
            views.show_message("Нічого не змінено.")
            return

        success, err = self.model.update(table, pk, pk_val, updates)
        if success:
            views.show_success("Оновлено.")
        else:
            views.show_error(f"Помилка: {err}")
```

```
    # ------------------------------------------------------
    # 6. DELETE
    # ------------------------------------------------------
    def action_delete(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        pk_raw = views.prompt(f"PK ({pk})")

        pk_col = next((c for c in self.model.columns_info(table) if c["name"] ==
pk), None)
        if pk_col and "int" in pk_col["type"]:
            try:
                pk_val = int(pk_raw)
            except:
                views.show_error("PK має бути числом")
                return
        else:
            pk_val = pk_raw

        try:
            if self.model.has_child_rows(table, pk, pk_val):
                views.show_error("Не можна видалити — є дочірні записи.")
                return
        except:
            views.show_error("Не вдалося перевірити залежності.")
            return

        if views.prompt("Підтвердити (так/ні)?").lower() not in ("так", "у",
"yes"):
            views.show_message("Скасовано.")
            return

        success, err = self.model.delete(table, pk, pk_val)
        if success:
            views.show_success("Видалено.")
        else:
            views.show_error(f"Не вдалося: {err}")

    # ------------------------------------------------------
    # 7. Генерація даних
    # ------------------------------------------------------
    def action_generate(self):
        count_raw = views.prompt("Скільки записів генерувати?")
        try:
            count = int(count_raw)
```

```
        except:
            views.show_error("Потрібно число.")
            return

        tasks = [
            ("supplier", self.model.generate_suppliers),
            ("product", self.model.generate_products),
            ("supply", self.model.generate_supplies),
            ("inventory", self.model.generate_inventory),
        ]

        for name, func in tasks:
            ok, err = func(count)
            if ok:
                views.show_success(f"{name}: згенеровано {count}")
            else:
                views.show_error(f"{name}: помилка — {err}")

    # ------------------------------------------------------------
    # 8. Складні SQL запити
    # ------------------------------------------------------------
    def action_complex_queries(self):
        views.show_message("1) Загальна сума постачань по постачальниках")
        views.show_message("2) Товари нижче мінімального запасу")
        views.show_message("3) Найдорожчі категорії постачань")
        views.show_message("4) ТОП-10 товарів за обсягом постачань")
        views.show_message("5) Постачання за останні 30 днів")

        choice = views.prompt("Виберіть запит (1-5)")

        query_map = {
            "1": self.model.query_supplier_totals,
            "2": self.model.query_products_below_min_stock,
            "3": self.model.query_category_supply_costs,
            "4": self.model.query_top_products_by_supply_volume,
            "5": self.model.query_last_month_supplies,
        }

        func = query_map.get(choice)
        if not func:
            views.show_error("Невірний вибір")
            return

        rows, time_ms, explain, err = func()
        if err:
            views.show_error(err)
            return

        views.show_query_result(rows, time_ms, explain)
```

```
    # ------------------------------------------------------------
    # 9. Перевірка залежностей
    # ------------------------------------------------------------
    def action_demo_check_children(self):
        table = views.prompt("Назва таблиці")
        if table not in self.tables:
            views.show_error("Невідома таблиця")
            return

        pk = self.model.primary_key(table)
        val = views.prompt(f"PK ({pk})")

        try:
            has = self.model.has_child_rows(table, pk, val)
        except:
            views.show_error("Помилка перевірки.")
            return

        if has:
            views.show_message("Є дочірні записи.")
        else:
            views.show_message("Немає дочірніх записів.")
```

Модуль Model.py

```
# models.py
from typing import List, Dict, Any, Optional, Tuple
import psycopg2
import psycopg2.extras
from psycopg2 import sql
from config import DB
from dateutil import parser as date_parser
import random
from datetime import datetime, timedelta


class DBModel:
    def __init__(self):
        try:
            self.conn = psycopg2.connect(**DB)
            self.conn.autocommit = True
        except Exception as e:
            raise RuntimeError("Не вдалося підключитися до бази даних. Перевірте
налаштування в config.py") from e

    def close(self):
        self.conn.close()

    # ----------------- Generic CRUD -----------------
    def list_tables(self) -> List[str]:
        q = """
        SELECT table_name
        FROM information_schema.tables
        WHERE table_schema='public' AND table_type='BASE TABLE'
        ORDER BY table_name;
        """
        with self.conn.cursor() as cur:
            cur.execute(q)
            return [r[0] for r in cur.fetchall()]

    def columns_info(self, table: str) -> List[Dict[str, Any]]:
        q = """
        SELECT column_name, data_type, is_nullable
        FROM information_schema.columns
        WHERE table_schema='public' AND table_name=%s
        ORDER BY ordinal_position;
        """
        with self.conn.cursor() as cur:
            cur.execute(q, (table,))
            res = []
            for r in cur.fetchall():
                res.append({"name": r[0], "type": r[1], "nullable": (r[2] ==
'YES')})
            return res

    def primary_key(self, table: str) -> Optional[str]:
        q = """
        SELECT a.attname
```

```
        FROM pg_index i
        JOIN pg_attribute a ON a.attrelid = i.indrelid AND a.attnum =
ANY(i.indkey)
        WHERE i.indrelid = quote_ident(%s)::regclass AND i.indisprimary;
        """
        with self.conn.cursor() as cur:
            cur.execute(q, (table,))
            row = cur.fetchone()
            return row[0] if row else None

    def select_all(self, table: str, limit: int = 200) -> List[Dict[str, Any]]:
        with self.conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as
cur:
            cur.execute(sql.SQL('SELECT * FROM {} ORDER BY 1 LIMIT
%s').format(sql.Identifier(table)), (limit,))
            return cur.fetchall()

    def select_by_pk(self, table: str, pk: str, pk_value: Any) ->
Optional[Dict[str, Any]]:
        with self.conn.cursor(cursor_factory=psycopg2.extras.RealDictCursor) as
cur:
            cur.execute(sql.SQL('SELECT * FROM {} WHERE
{}=%s').format(sql.Identifier(table), sql.Identifier(pk)), (pk_value,))
            return cur.fetchone()

    def insert(self, table: str, data: Dict[str, Any]) -> Tuple[bool,
Optional[str]]:
        cols = list(data.keys())
        vals = [data[c] for c in cols]
        query = sql.SQL('INSERT INTO {} ({}) VALUES ({})').format(
            sql.Identifier(table),
            sql.SQL(', ').join(map(sql.Identifier, cols)),
            sql.SQL(', ').join(sql.Placeholder() * len(cols))
        )
        with self.conn.cursor() as cur:
            try:
                cur.execute(query, vals)
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    def update(self, table: str, pk: str, pk_value: Any, data: Dict[str, Any]) ->
Tuple[bool, Optional[str]]:
        cols = list(data.keys())
        vals = [data[c] for c in cols] + [pk_value]
        set_clause = sql.SQL(', ').join(
            sql.Composed([sql.Identifier(c), sql.SQL(' = '), sql.Placeholder()])
for c in cols
        )
        query = sql.SQL('UPDATE {} SET {} WHERE {} = %s').format(
            sql.Identifier(table),
            set_clause,
            sql.Identifier(pk)
        )
```

```
        with self.conn.cursor() as cur:
            try:
                cur.execute(query, vals)
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    def delete(self, table: str, pk: str, pk_value: Any) -> Tuple[bool,
Optional[str]]:
        query = sql.SQL('DELETE FROM {} WHERE {} =
%s').format(sql.Identifier(table), sql.Identifier(pk))
        with self.conn.cursor() as cur:
            try:
                cur.execute(query, (pk_value,))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    # ----------------- Helpers -----------------
    def has_child_rows(self, parent_table: str, parent_pk: str, pk_value: Any) ->
bool:
        q = """
        SELECT c.table_name, kcu.column_name
        FROM information_schema.table_constraints c
        JOIN information_schema.key_column_usage kcu
          ON c.constraint_name = kcu.constraint_name AND c.table_schema =
kcu.table_schema
        JOIN information_schema.constraint_column_usage ccu
          ON c.constraint_name = ccu.constraint_name AND c.table_schema =
ccu.constraint_schema
        WHERE c.constraint_type = 'FOREIGN KEY' AND ccu.table_name = %s AND
ccu.column_name = %s;
        """
        with self.conn.cursor() as cur:
            cur.execute(q, (parent_table, parent_pk))
            fks = cur.fetchall()
            for fk_table, fk_col in fks:
                check_q = sql.SQL('SELECT EXISTS (SELECT 1 FROM {} WHERE {} = %s
LIMIT 1)').format(
                    sql.Identifier(fk_table), sql.Identifier(fk_col)
                )
                cur.execute(check_q, (pk_value,))
                if cur.fetchone()[0]:
                    return True
        return False

    def parent_exists(self, parent_table: str, parent_pk: str, pk_value: Any) ->
bool:
        with self.conn.cursor() as cur:
            cur.execute(sql.SQL('SELECT EXISTS (SELECT 1 FROM {} WHERE {} = %s
LIMIT 1)').format(
                sql.Identifier(parent_table), sql.Identifier(parent_pk)
            ), (pk_value,))
            return cur.fetchone()[0]
```

```
    def parse_date(self, value: str) -> Optional[str]:
        try:
            d = date_parser.parse(value)
            return d.date().isoformat()
        except Exception:
            return None

    # ---------------- Генерація даних ----------------
    def generate_suppliers(self, count: int) -> Tuple[bool, Optional[str]]:
        first_names = ["Іван", "Петро", "Ольга", "Марія", "Андрій"]
        last_names = ["Іванов", "Петренко", "Сидоренко", "Коваленко",
"Бондаренко"]
        domains = ["example.ua", "mail.ua", "suppliers.ua"]
        with self.conn.cursor() as cur:
            try:
                cur.execute("SELECT COALESCE(MAX(supplier_id),0)+1 FROM
supplier")
                start_id = cur.fetchone()[0]
                for i in range(count):
                    supplier_id = start_id + i
                    company_name = f"Компанія {supplier_id}"
                    contact_person = f"{random.choice(first_names)}
{random.choice(last_names)}"
                    phone = f"+380{random.randint(500000000, 999999999)}"
                    email = f"user{supplier_id}@{random.choice(domains)}"
                    cur.execute("""
                        INSERT INTO supplier(supplier_id, company_name,
contact_person, phone, email)
                        VALUES (%s,%s,%s,%s,%s)
                    """, (supplier_id, company_name, contact_person, phone,
email))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    def generate_products(self, count: int) -> Tuple[bool, Optional[str]]:
        units = ["шт", "уп", "кг", "л"]
        categories = ["Комп'ютерна техніка", "Оргтехніка", "Канцтовари",
"Витратні матеріали"]
        with self.conn.cursor() as cur:
            try:
                cur.execute("SELECT COALESCE(MAX(product_id),0)+1 FROM product")
                start_id = cur.fetchone()[0]
                for i in range(count):
                    product_id = start_id + i
                    product_name = f"Товар {product_id}"
                    unit_measure = random.choice(units)
                    min_stock = random.randint(1, 100)
                    category = random.choice(categories)
                    cur.execute("""
                        INSERT INTO product(product_id, product_name,
unit_measure, min_stock, category)
                        VALUES (%s,%s,%s,%s,%s)
```

```
                    """, (product_id, product_name, unit_measure, min_stock,
category))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    def generate_supplies(self, count: int) -> Tuple[bool, Optional[str]]:
        with self.conn.cursor() as cur:
            try:
                # FK: get existing supplier_ids and product_ids
                cur.execute("SELECT supplier_id FROM supplier")
                supplier_ids = [r[0] for r in cur.fetchall()]
                cur.execute("SELECT product_id FROM product")
                product_ids = [r[0] for r in cur.fetchall()]
                if not supplier_ids or not product_ids:
                    return False, "Відсутні дані для FK"

                cur.execute("SELECT COALESCE(MAX(supply_id),0)+1 FROM supply")
                start_id = cur.fetchone()[0]

                for i in range(count):
                    supply_id = start_id + i
                    supplier_id = random.choice(supplier_ids)
                    product_id = random.choice(product_ids)
                    supply_date = datetime.now() -
timedelta(days=random.randint(0, 365))
                    document_number = f"ПН-{supply_id:05d}"
                    quantity = round(random.uniform(1, 100), 2)
                    unit_price = round(random.uniform(10, 5000), 2)
                    cur.execute("""
                        INSERT INTO supply(supply_id, supplier_id, product_id,
supply_date, document_number, quantity, unit_price)
                        VALUES (%s,%s,%s,%s,%s,%s,%s)
                    """, (supply_id, supplier_id, product_id, supply_date,
document_number, quantity, unit_price))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)

    def generate_inventory(self, count: int) -> Tuple[bool, Optional[str]]:
        locations = [
            "Секція A, полиця 1", "Секція A, полиця 2", "Секція A, полиця 3",
            "Секція B, полиця 1", "Секція B, полиця 2", "Секція B, полиця 3",
            "Секція C, полиця 1", "Секція C, полиця 2", "Секція C, полиця 3",
            "Секція D, полиця 1", "Секція D, полиця 2"
        ]
        with self.conn.cursor() as cur:
            try:
                cur.execute("SELECT COALESCE(MAX(inventory_id),0)+1 FROM
inventory")
                start_id = cur.fetchone()[0]
                cur.execute("SELECT product_id FROM product")
                product_ids = [r[0] for r in cur.fetchall()]
                if not product_ids:
```

```
                    return False, "Відсутні продукти для FK"

                used_products = set()
                for i in range(count):
                    inventory_id = start_id + i
                    product_id = random.choice(product_ids)
                    # не дублюємо inventory для одного product_id
                    while product_id in used_products:
                        product_id = random.choice(product_ids)
                    used_products.add(product_id)
                    quantity = round(random.uniform(0, 200), 2)
                    last_updated = datetime.now() -
timedelta(days=random.randint(0, 365))
                    location = random.choice(locations)
                    cur.execute("""
                        INSERT INTO inventory(inventory_id, product_id, quantity,
last_updated, location)
                        VALUES (%s,%s,%s,%s,%s)
                    """, (inventory_id, product_id, quantity, last_updated,
location))
                return True, None
            except psycopg2.Error as e:
                return False, e.pgerror or str(e)
```

Модуль View.py

```
# views.py
from typing import List, Dict, Any, Optional

def print_banner():
    print("==================================")
    print("   Система обліку постачань склад")
    print("==================================\n")

def show_menu():
    print("\nМеню:")
    print("1. Список таблиць")
    print("2. Показати таблицю")
    print("3. Показати запис за PK")
    print("4. Додати запис")
    print("5. Оновити запис")
    print("6. Видалити запис")
    print("7. Генерація випадкових даних")
    print("8. Складні SQL запити")
    print("9. Перевірка дочірніх записів")
    print("0. Вийти")

def prompt(msg: str) -> str:
    return input(f"{msg}: ").strip()

def prompt_nullable(msg: str) -> Optional[str]:
    raw = input(f"{msg} (залиште пустим для NULL): ").strip()
    return raw if raw != "" else None
```

```
def print_tables(tables: List[str]):
    print("\nТаблиці бази:")
    for t in tables:
        print(" -", t)

def print_rows(rows: List[Dict[str, Any]]):
    if not rows:
        print("Немає записів.")
        return
    print("\nРядки таблиці:")
    for r in rows:
        print(r)

def print_row(row: Optional[Dict[str, Any]]):
    if not row:
        print("Запис не знайдено.")
    else:
        print("\nЗапис:")
        print(row)

def show_error(msg: str):
    print(f"[ПОМИЛКА] {msg}")

def show_success(msg: str):
    print(f"[УСПІХ] {msg}")

def show_message(msg: str):
    print(f"[INFO] {msg}")

def show_query_result(rows: List[Dict[str, Any]], exec_time_ms: Optional[float],
explain: str):
    print("\n=== Результат запиту ===")
    for r in rows:
        print(r)
    if exec_time_ms is not None:
        print(f"\nЧас виконання: {exec_time_ms:.2f} ms")
    if explain:
        print("\n--- EXPLAIN ANALYZE ---")
        print(explain)
```

Модуль Config.py

```
DB = {
    "host": "localhost",
    "port": 5432,
    "dbname": "sasha",   # використовуємо системну БД
    "user": "postgres",
```

```
    "password": ""
}
```

Модуль Main.py

```
# main.py
from controllers import Controller

def main():
    ctrl = Controller()
    try:
        ctrl.run()
    finally:
        ctrl.close()

if __name__ == "__main__":
    main()
```

Скріншоти результатів і вигляду меню

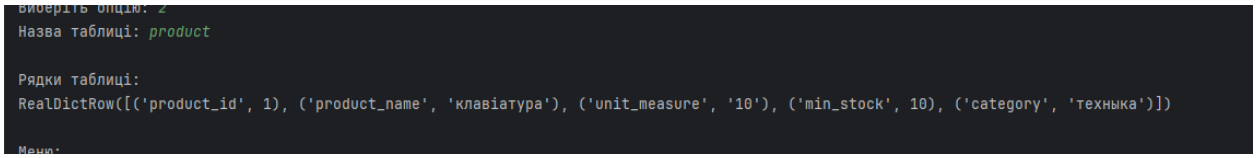
```
Меню:
1. Список таблиць
2. Показати таблицю
3. Показати запис за PK
4. Додати запис
5. Оновити запис
6. Видалити запис
7. Генерація випадкових даних
8. Складні SQL запити
9. Перевірка дочірніх записів
0. Вийти
Виберіть опцію:
```

Скріншоти результатів виконання операції вставки запису в дочірню таблицю

```
Назва таблиці: product
product_name (character varying) (залиште пустим для NULL): клавіатура
unit_measure (character varying) (залиште пустим для NULL): 10
min_stock (integer) (залиште пустим для NULL): 10
category (character varying) (залиште пустим для NULL): техныка
[УСПІХ] Запис додано.
```

product_id [PK] integer	product_name character varying (100)	unit_measure character varying (20)	min_stock integer	category character varying (50)
1	клавіатура	10	10	техныка

Скріншоти результатів виконання операції редагування таблиці

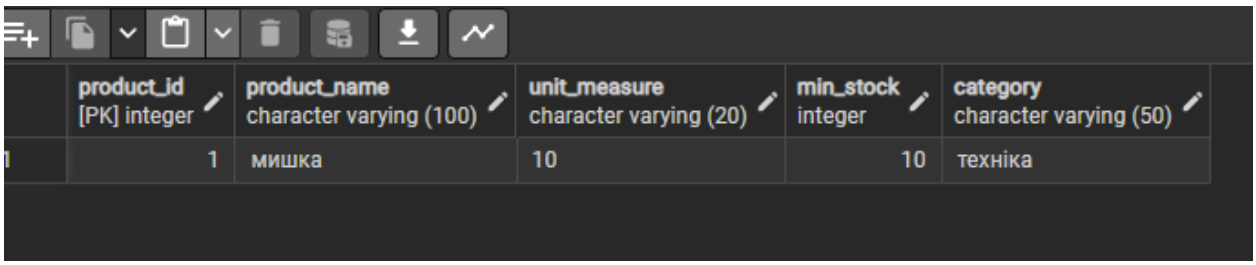


```
Виберіть опцію: 2
Назва таблиці: product

Рядки таблиці:
RealDictRow([('product_id', 1), ('product_name', 'клавіатура'), ('unit_measure', '10'), ('min_stock', 10), ('category', 'техныка')])

Меню:
```

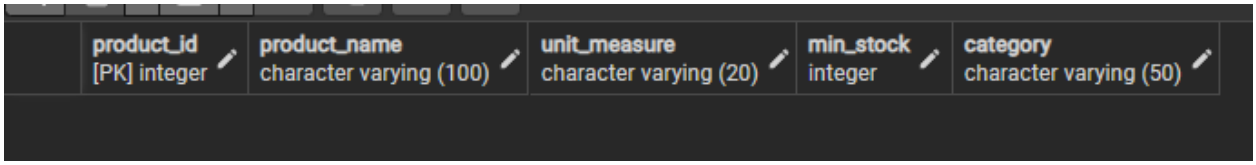
```
Виберіть опцію: 3
Назва таблиці: product
PK (product_id): 1
product_name (поточне: клавіатура) (залиште пустим для NULL): мишка
unit_measure (поточне: 10) (залиште пустим для NULL): 10
min_stock (поточне: 10) (залиште пустим для NULL): 10
category (поточне: техныка) (залиште пустим для NULL): техніка
[УСПІХ] Оновлено.
```



product_id [PK] integer	product_name character varying (100)	unit_measure character varying (20)	min_stock integer	category character varying (50)
1	мишка	10	10	техніка

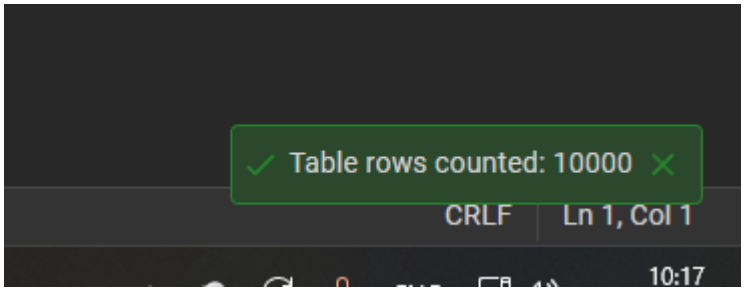
Скріншоти результатів виконання операції видалення

```
0. Вийти
Виберіть опцію: 6
Назва таблиці: product
PK (product_id): 1
Підтвердити (так/ні)?: y
[УСПІХ] Видалено.
```



Скріншоти результатів виконання операції генерування

```
Виберіть опцію: 7
Скільки записів генерувати?: 10000
[УСПІХ] supplier: згенеровано 10000
[УСПІХ] product: згенеровано 10000
[УСПІХ] supply: згенеровано 10000
[УСПІХ] inventory: згенеровано 10000
```



Висновок
У процесі виконання розрахунково-графічної роботи було розроблено консольний застосунок, який реалізує взаємодію з базою даних PostgreSQL, використовуючи шаблон проектування MVC (Model-View-Controller). У рамках проєкту було налаштовано підключення до бази даних і реалізовано функціонал для перегляду, додавання, редагування, видалення та генерації даних у таблицях. Забезпечено перевірку коректності типів даних під час введення, а також обробку помилок за допомогою конструкцій try-except, що запобігає виникненню системних помилок під час роботи з програмою.

Особлива увага була приділена реалізації зв'язків типу 1:N між таблицями. Було забезпечено контроль цілісності даних при видаленні батьківських записів, на які є посилання з підлеглих таблиць, а також перевірка наявності відповідних батьківських записів під час додавання нових підлеглих.

У ході роботи застосовувався модульний підхід:

-**Model** відповідає за взаємодію з базою даних та виконання SQL-запитів, **Controller** реалізує обробку дій користувача, **View** забезпечує зручне відображення інформації у консольному інтерфейсі.

В результаті виконаної роботи була отримана повнофункціональна система керування базою даних через консоль, яка демонструє основні CRUD-операції, генерацію випадкових даних і базові можливості оптимізації SQL-запитів з використанням EXPLAIN ANALYZE.

Посилання на GitHub:https://github.com/sashastarwort/DB_1lab

Нік в Telegram: @sashastarwort