НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

РГР

з дисципліни
«Бази даних і засоби управління»

Тема: «***Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL***»

Виконав: студент 3 курсу
ФПМ групи КВ-11
Рибалка Денис
Перевірив:

Київ – 2023

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

URL репозиторію з вихідним кодом - https://github.com/denisrybalka-kpi/rgr-bd

Модель «сутність-зв’язок» предметної галузі для обліку експонатів у музеї.

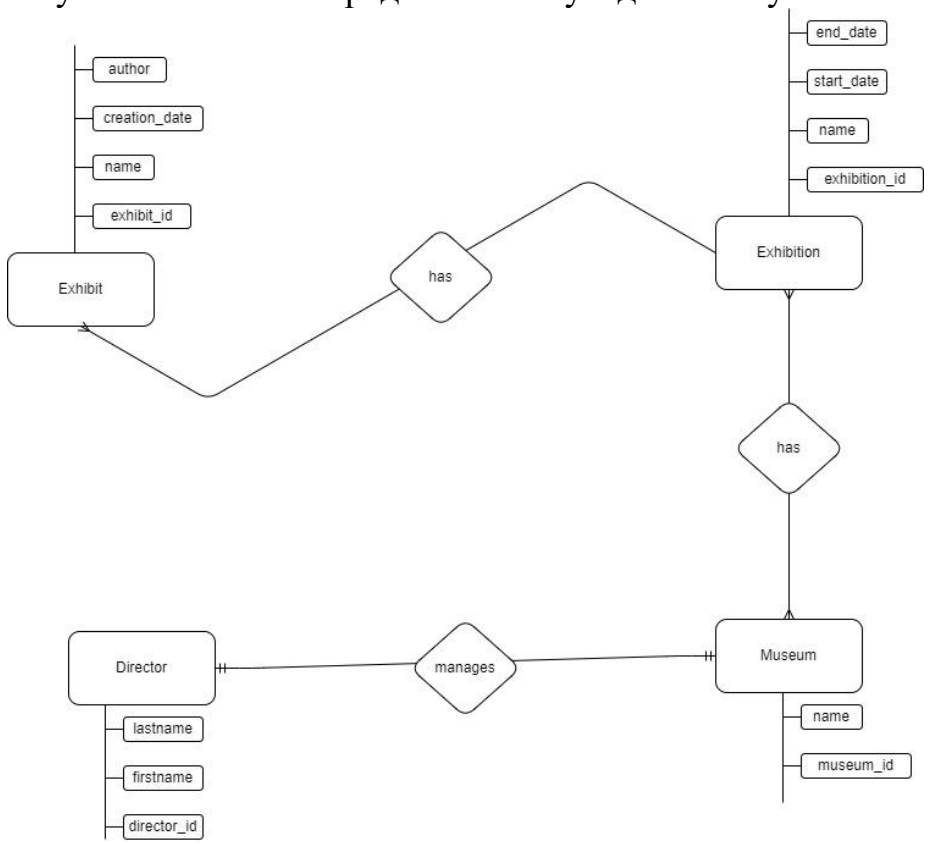


Рисунок 1. ER-діаграма побудована за нотацією «Crow`s foot»

У цій моделі кожна з сутностей має свої атрибути, а зв'язки між ними подані наступним чином:

- Зв'язок "Director-Museum" (1:1):
  Музей може мати лише одного директора, а директор може управляти лише одним музеєм. Отже це зв'язок «1:1» між **Музеєм** та **Директором.**

- Зв'язок "Museum-Exhibition" (M:N):
  Між **Музеєм** і **Виставкою** може існувати зв'язок "багато до багатьох" (M:N), оскільки виставка може проводитись у багатьох музеях, а у музеї може проводитись багато виставок.

- Зв'язок " Exhibition-Exhibit" (1:N):
  Між **Виставкою** і **Експонатом** може існувати зв'язок "один до багатьох" (1:N), оскільки експонат може належати тільки до однієї виставки, а на виставці може бути представлено багато експонатів.

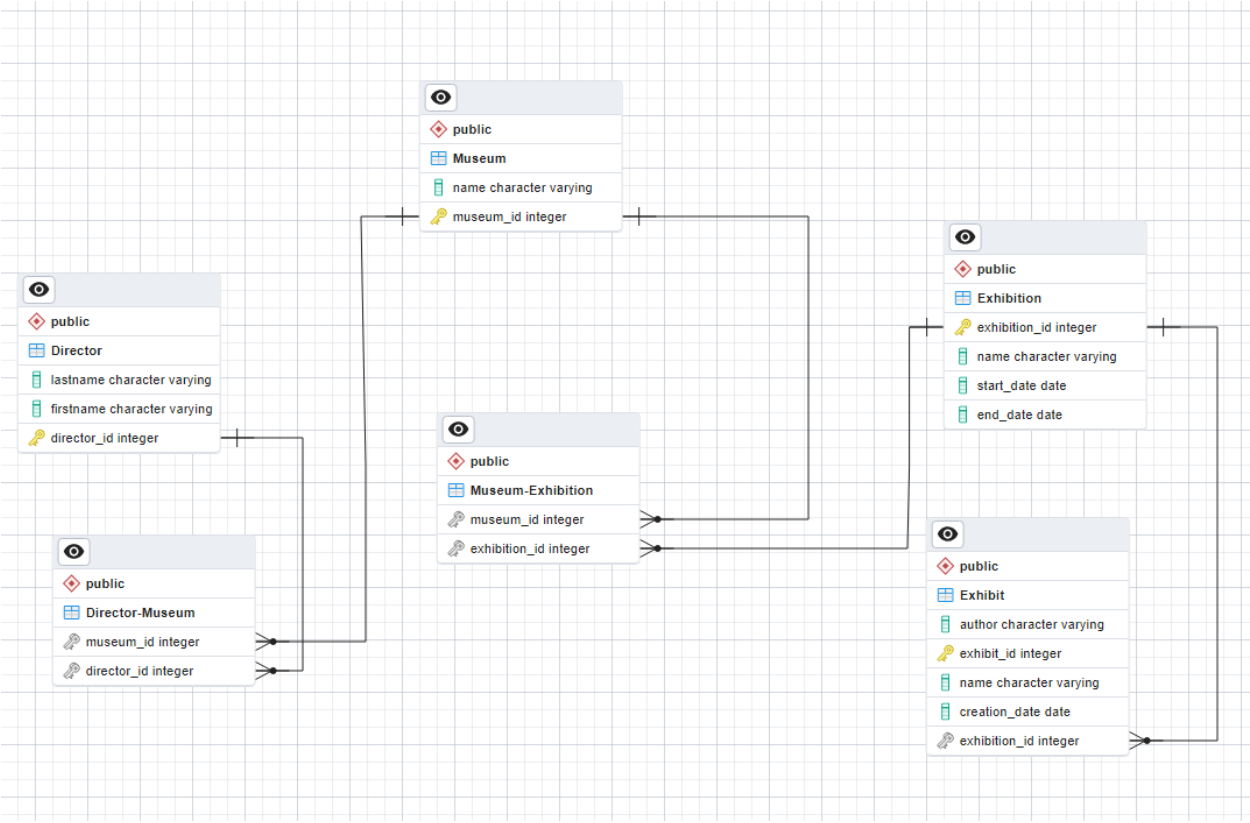


Рисунок. 2 Схема бази даних PostgreSQL на основі ER-моделі для системи обліку експонатів у музеї

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.
Мова програмування – Python 3.10.11
Середовище розробки програмного забезпечення – VS Code
Бібліотека взаємодії з PostgreSQL - Psycopg2

```
PS C:\Users\denis\OneDrive\Рабочий стол\5 SEM\БД\rgr> python main.py
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice:
```

Рисунок. 3 Головне меню програми

Головне меню для користувача складається з семи пунктів.

Перегляд однієї таблиці (1. Show one table) дозволяє вивести вміст обраної таблиці з бази даних. Користувач обирає таблицю перед виведенням даних.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 1
 1. Exhibit
 2. Exhibition
 3. Museum
 4. Director
Enter table index: 1
# Exhibit
     :) Successfull fetch from Exhibit: [('RZU', 1, 'BGE', datetime.date(2023, 10, 19), 416), ('UEL', 2,
time.date(2023, 7, 13), 452), ('CWF', 5, 'XTJ', datetime.date(2023, 2, 26), 947), ('NOW', 6, 'VFP', datet
23, 1, 15), 738), ('URZ', 9, 'VQW', datetime.date(2023, 7, 25), 932), ('SUL', 10, 'HEH', datetime.date(20
14), 36), ('LKH', 13, 'AHS', datetime.date(2023, 6, 28), 314), ('HDX', 14, 'KAH', datetime.date(2023, 9,
, ('PZH', 17, 'ARB', datetime.date(2023, 8, 27), 370), ('TYM', 18, 'PPI', datetime.date(2023, 5, 13), 911
 21, 'RVI', datetime.date(2023, 1, 8), 603), ('PRJ', 22, 'EMT', datetime.date(2023, 6, 16), 787), ('FIA',
, datetime.date(2023, 3, 27), 855), ('RAO', 26, 'VJH', datetime.date(2023, 3, 13), 735), ('FZF', 27, 'HHQ
e.date(2023, 9, 28), 158), ('NHX', 30, 'LAP', datetime.date(2022, 12, 11), 845), ('YEM', 31, 'QFX', datet
(2023, 5, 15), 804), ('WCV', 34, 'APX', datetime.date(2023, 1, 31), 481), ('DNG', 35, 'PRZ', datetime.dat
, 18), 924), ('EFZ', 38, 'ODC', datetime.date(2022, 12, 12), 66), ('GQS', 39, 'RDN', datetime.date(2023,
528), ('KKU', 42, 'PBO', datetime.date(2023, 1, 2), 73), ('MKO', 43, 'QYC', datetime.date(2023, 9, 18), 6
', 46, 'BOT', datetime.date(2023, 7, 14), 259), ('UWE', 47, 'HKS', datetime.date(2023, 6, 23), 443), ('EQ
TEI', datetime.date(2023, 5, 1), 911), ('ZTM', 51, 'QPP', datetime.date(2023, 4, 2), 359), ('NBU', 52, 'Z
etime.date(2023, 5, 23), 853), ('SQZ', 55, 'MQW', datetime.date(2023, 1, 14), 631), ('NFT', 56, 'VQN', da
te(2023, 7, 22), 2), ('XHD', 59, 'DNX', datetime.date(2023, 4, 5), 425), ('VHP', 60, 'RAP', datetime.date
, 28), 947), ('KJC', 63, 'JWH', datetime.date(2023, 5, 2), 268), ('RPE', 64, 'BYE', datetime.date(2023, 6
), ('COO', 67, 'JZY', datetime.date(2023, 7, 25), 603), ('OXS', 68, 'JMZ', datetime.date(2023, 4, 30), 56
 71, 'UDB', datetime.date(2022, 12, 22), 63), ('YGI', 72, 'TWA', datetime.date(2023, 3, 13), 577), ('UDF'
I', datetime.date(2023, 2, 24), 595), ('VRX', 76, 'YUP', datetime.date(2022, 12, 16), 504), ('OVO', 77, '
```

Введення даних (2. Insert data) дозволяє користувачеві додавати нові записи в обрану таблицю. Користувач вибирає таблицю і вводить дані для кожного атрибуту.

```
 10, 23), 206), ( 'MSP', 999, 'JYG', datetime.date(2023,
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 2
 1. Exhibit
 2. Exhibition
 3. Museum
 4. Director
Enter table index: 1
Enter exhibit_id (int): 1111
Enter exhibition_id (int): 1
Enter name (str): QUI
Enter author (str): PIH
Enter creation_date (date): 2012-12-12
# Successfully added to Exhibit table
Navigation Menu
```

Видалення даних (3. Delete data) дозволяє користувачеві вилучати існуючі записи з обраної таблиці. Користувач обирає таблицю, потім вводить ідентифікатор рядка для видалення.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 3
 1. Exhibit
 2. Exhibition
 3. Museum
 4. Director
Enter table index: 1
Enter id: 1
# Exhibit
    :) Removed successfully from Exhibit element with id: 1
```

Редагування даних (4. Update data) дозволяє користувачеві змінювати існуючі дані в обраній таблиці. Користувач обирає таблицю, потім вказує, які саме дані редагувати та вводить нові значення.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 4
 1. Exhibit
 2. Exhibition
 3. Museum
 4. Director
Enter table index: 1
Enter id: 2
Enter name (str): ABA
Enter author (str): ABA
Enter creation_date (date): 2013-12-12
# Successfully updated Exhibit table
```

Вибірковий пошук (5. Select data) дозволяє користувачеві шукати дані за атрибутами з декількох таблиць. Користувач обирає запит і вводить параметри пошуку.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 5
1. Show Exhibit by author and Exhibitions they represented at
2. Show list of authors that are represented at specific Exhibition
Enter row index (1-2): 1
Enter Author name: ABA
# Request took 4 ms
fetched [{'exhibit_name': 'ABA', 'exhibition_name': 'ZWZ', 'author': 'ABA'}]
```

Випадкові дані (6. Randomize data) дозволяє користувачеві додавати випадкові дані до таблиць Exhibit та Exhibition.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 6
Enter how many rows do you want to add: 10
```

Вихід (7. Exit) завершує виконання програми.

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 7
# Program terminated successfully!
PS C:\Users\denis\OneDrive\Рабочий стол\5 SEM\БД\rgr>
```

Завдання 1.

Таблиця "Exhibition" у pgAdmin:

	exhibition_id [PK] integer	name character varying	start_date date	end_date date
1	1	DWD	2024-07-14	2024-05-22
2	2	WNU	2024-06-11	2024-01-25
3	3	TDM	2024-05-24	2024-05-29
4	4	FDE	2023-12-10	2024-10-18
5	5	EZS	2024-04-06	2024-04-08
6	6	ZKC	2024-06-11	2024-11-07
7	7	NDK	2023-12-28	2024-03-15
8	8	QVO	2024-05-11	2024-08-28
9	9	RSB	2024-05-16	2024-03-21
10	10	QAW	2024-04-22	2024-02-09
11	11	AGY	2024-09-17	2024-06-25
12	12	QEE	2024-03-10	2024-12-27
13	13	GCT	2024-04-22	2024-10-03
14	14	AOY	2024-02-25	2024-06-08
15	15	ZHZ	2024-09-22	2024-01-23
16	16	WGU	2024-10-06	2024-03-29
17	17	CHG	2024-02-25	2024-05-21
18	18	QIZ	2024-10-20	2024-08-18
19	19	KEM	2024-03-12	2024-09-02
20	20	OWO	2024-02-03	2024-11-03
21	21	HPX	2024-08-28	2024-04-19
22	22	EZL	2024-04-02	2024-09-30

Total rows: 1000 of 1009 Query complete 00:00:00.139

Таблиця "Exhibit" у pgAdmin:

	author character varying	exhibit_id [PK] integer	name character varying	creation_date date	exhibition_id integer
987	RTB	990	URS	2022-12-21	203
988	STO	991	JDT	2023-01-26	450
989	YTJ	992	CBH	2023-11-22	483
990	KNO	993	YAR	2023-10-27	251
991	YAW	994	YCM	2022-12-13	980
992	KFV	995	RZC	2023-08-31	575
993	QUI	996	MXE	2023-03-07	618
994	CKN	997	VED	2022-12-08	451
995	ZCH	998	WVA	2023-10-23	206
996	MSP	999	JYG	2023-06-22	433
997	DYH	1000	PIH	2023-10-24	812
998	PIH	1111	QUI	2012-12-12	1
999	MMW	1112	BHO	2023-02-23	279
1000	NCN	1113	BRI	2023-04-25	38
1001	PIX	1114	KYY	2023-02-18	659
1002	XQZ	1115	OGT	2023-11-15	190
1003	VMH	1116	NFJ	2023-05-26	882
1004	HSF	1117	EXY	2022-12-20	208
1005	PSA	1118	IJU	2023-04-15	81
1006	KIQ	1119	ZNU	2023-09-06	938
1007	FCM	1120	FDT	2023-09-14	435
1008	JPR	1121	IVD	2023-09-05	469

Total rows: 1008 of 1008 | Query complete 00:00:00.171 | Rows selected: 1

Видалимо exhibition_id = 1 через програму

```
Navigation Menu
 1. Show One Table
 2. Insert Data
 3. Delete Data
 4. Update Data
 5. Select Data
 6. Randomize Data
 7. Exit
Enter your choice: 3
 1. Exhibit
 2. Exhibition
 3. Museum
 4. Director
Enter table index: 2
Enter id: 1
# Exhibition
     :) Removed successfully from Exhibition element with id: 1
```

Таблиця "Exhibition" у pgAdmin після видалення exhibition_id = 1:

	exhibition_id [PK] integer	name character varying	start_date date	end_date date
1	2	WNU	2024-06-11	2024-01-25
2	3	TDM	2024-05-24	2024-05-29
3	4	FDE	2023-12-10	2024-10-18
4	5	EZS	2024-04-06	2024-04-08
5	6	ZKC	2024-06-11	2024-11-07
6	7	NDK	2023-12-28	2024-03-15
7	8	QVO	2024-05-11	2024-08-28
8	9	RSB	2024-05-16	2024-03-21
9	10	QAW	2024-04-22	2024-02-09
10	11	AGY	2024-09-17	2024-06-25
11	12	QEE	2024-03-10	2024-12-27
12	13	GCT	2024-04-22	2024-10-03
13	14	AOY	2024-02-25	2024-06-08
14	15	ZHZ	2024-09-22	2024-01-23
15	16	WGU	2024-10-06	2024-03-29
16	17	CHG	2024-02-25	2024-05-21
17	18	QIZ	2024-10-20	2024-08-18
18	19	KEM	2024-03-12	2024-09-02
19	20	OWO	2024-02-03	2024-11-03
20	21	HPX	2024-08-28	2024-04-19
21	22	EZL	2024-04-02	2024-09-30
22	23	KPO	2024-09-09	2024-05-29

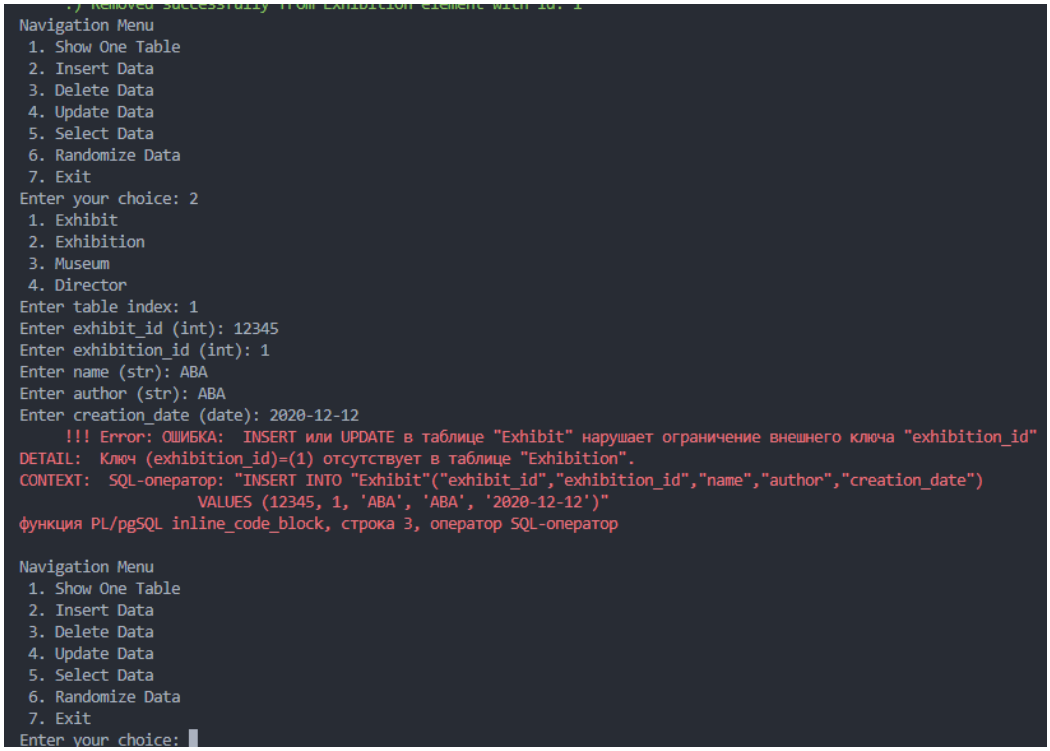
Total rows: 1000 of 1008 Query complete 00:00:00.138

Таблиця "Exhibit" у pgAdmin після видалення exhibition_id = 1:

	author character varying	exhibit_id [PK] integer	name character varying	creation_date date	exhibition_id integer
986	VLI	989	FJO	2023-09-07	659
987	RTB	990	URS	2022-12-21	203
988	STO	991	JDT	2023-01-26	450
989	YTJ	992	CBH	2023-11-22	483
990	KNO	993	YAR	2023-10-27	251
991	YAW	994	YCM	2022-12-13	980
992	KFV	995	RZC	2023-08-31	575
993	QUI	996	MXE	2023-03-07	618
994	CKN	997	VED	2022-12-08	451
995	ZCH	998	WVA	2023-10-23	206
996	MSP	999	JYG	2023-06-22	433
997	DYH	1000	PIH	2023-10-24	812
998	MMW	1112	BHO	2023-02-23	279
999	NCN	1113	BRI	2023-04-25	38
1000	PIX	1114	KYY	2023-02-18	659
1001	XQZ	1115	OGT	2023-11-15	190
1002	VMH	1116	NFJ	2023-05-26	882
1003	HSF	1117	EXY	2022-12-20	208
1004	PSA	1118	IJU	2023-04-15	81
1005	KIQ	1119	ZNU	2023-09-06	938
1006	FCM	1120	FDT	2023-09-14	435
1007	JPR	1121	IVD	2023-09-05	469

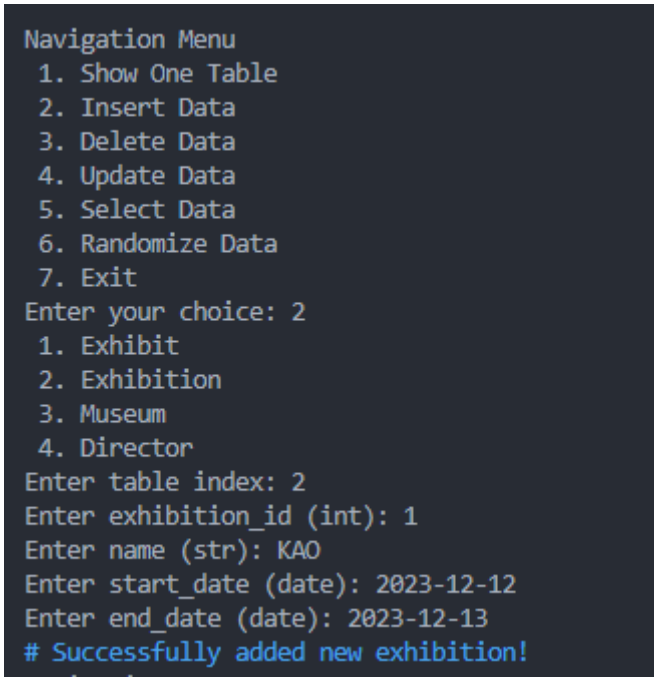
Total rows: 1007 of 1007 Query complete 00:00:00.135

Спробуємо додати новий Exhibit для exhibition_id = 1:



Отримуємо повідомлення про помилку в консоль, що exhibition_id = 1 не існує

Спробуємо додати новий елемент exhibition_id = 1



Спробуємо змінити дані для exhibition_id = 1

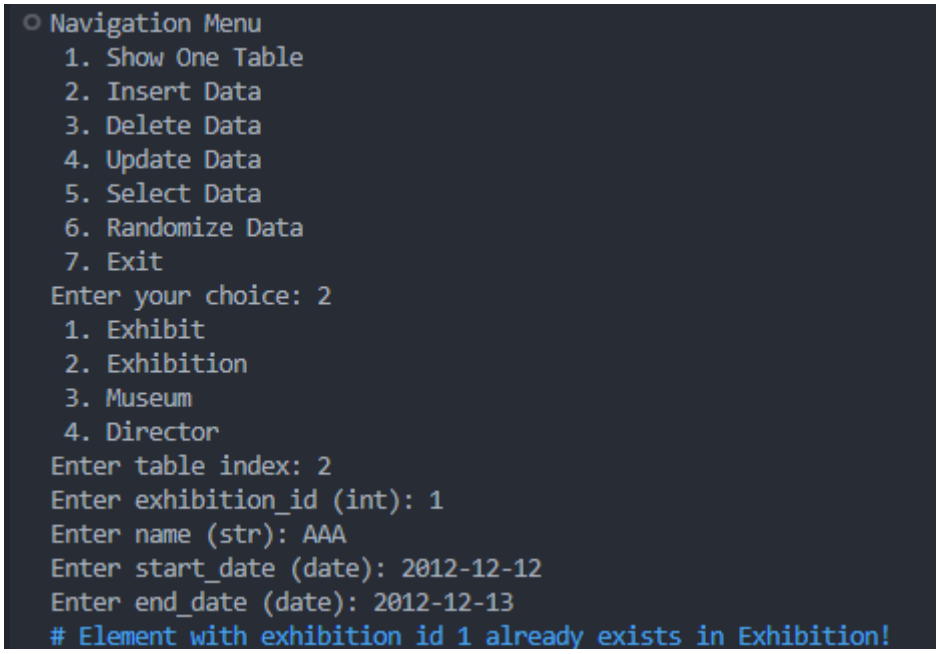
```
Navigation Menu
  1. Show One Table
  2. Insert Data
  3. Delete Data
  4. Update Data
  5. Select Data
  6. Randomize Data
  7. Exit
Enter your choice: 4
  1. Exhibit
  2. Exhibition
  3. Museum
  4. Director
Enter table index: 2
Enter id: 1
Enter name (str): KIO
Enter start_date (date): 2022-12-12
Enter end_date (date): 2023-12-12
# Successfully updated Exhibition table
```

Таблиця "Exhibition" у pgAdmin після додавання та редагування exhibition_id = 1

	exhibition_id [PK] integer	name character varying	start_date date	end_date date
1	1	KIO	2022-12-12	2023-12-12
2	2	WNU	2024-06-11	2024-01-25
3	3	TDM	2024-05-24	2024-05-29
4	4	FDE	2023-12-10	2024-10-18
5	5	EZS	2024-04-06	2024-04-08
6	6	ZKC	2024-06-11	2024-11-07
7	7	NDK	2023-12-28	2024-03-15
8	8	QVO	2024-05-11	2024-08-28
9	9	RSB	2024-05-16	2024-03-21
10	10	QAW	2024-04-22	2024-02-09
11	11	AGY	2024-09-17	2024-06-25
12	12	QEE	2024-03-10	2024-12-27
13	13	GCT	2024-04-22	2024-10-03
14	14	AOY	2024-02-25	2024-06-08
15	15	ZHZ	2024-09-22	2024-01-23
16	16	WGU	2024-10-06	2024-03-29
17	17	CHG	2024-02-25	2024-05-21
18	18	QIZ	2024-10-20	2024-08-18
19	19	KEM	2024-03-12	2024-09-02
20	20	OWO	2024-02-03	2024-11-03
21	21	HPX	2024-08-28	2024-04-19
22	22	EZL	2024-04-02	2024-09-30

Total rows: 1000 of 1009 Query complete 00:00:00.256

Спробуємо знову додати в таблицю «Exhibition» елемент з exhibition_id = 1. Отримуємо помилку:



Лістинг методу insert

```
def insert_data(self, table_name, data):
        values = tuple(data.values())
        columns = ', '.join(data.keys())

        c = self.conn.cursor()

        if table_name == "Exhibit":

            query = """DO $$ BEGIN IF NOT EXISTS (SELECT "exhibit_id" FROM
                    "Exhibit" WHERE "exhibit_id" = {})
                    THEN INSERT INTO
"Exhibit"("exhibit_id","exhibition_id","name","author","creation_date")
                    VALUES {}; RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;
END $$;
                    """.format(data['exhibit_id'], values,"'Successfully
added to Exhibit table'", "'Error: This exhibit_id already exists!'")

        elif  table_name == "Exhibition":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "exhibition_id" FROM "Exhibition"
                        WHERE "exhibition_id" = {}
                    ) THEN
                        INSERT INTO "Exhibition"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new exhibition!';
                    ELSE
                        RAISE NOTICE 'Element with exhibition_id {} already
exists in Exhibition!';
                    END IF;
                END $$;
```

```
            """.format(data['exhibition_id'], columns, values,
data['exhibition_id'])

        elif  table_name == "Museum":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "museum_id" FROM "Museum"
                        WHERE "museum_id" = {}
                    ) THEN
                        INSERT INTO "Museum"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new museum!';
                    ELSE
                        RAISE NOTICE 'Element with museum_id {} already
exists in Museum!';
                    END IF;
                END $$;
            """.format(data['museum_id'], columns, values, data['museum_id'])

        elif  table_name == "Director":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "director_id" FROM "Director"
                        WHERE "director_id" = {}
                    ) THEN
                        INSERT INTO "Director"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new director!';
                    ELSE
                        RAISE NOTICE 'Element with director_id {} already
exists in Director!';
                    END IF;
                END $$;
            """.format(data['director_id'], columns, values,
data['director_id'])

        try:
            c.execute(query)
            self.conn.commit()
            self.print_notices(self.conn.notices)
        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return
```

Лістинг методу get

```
def get_table_data(self, table_name):
        c = self.conn.cursor()
        Printer.print_info(table_name)
        c.execute(f'SELECT * FROM public."{table_name}"')
        result = c.fetchall()

        Printer.print_success(f"Successfull fetch from {table_name}:
{result}", 5)
```

Лістинг методу update

```
def update_data(self, table_name, id, new_data):
        c = self.conn.cursor()

        set_clause = ', '.join(f'"{key}" = \'{value}\'' for key, value in
new_data.items())

        if table_name == "Exhibit":

            query = """DO $$ BEGIN IF EXISTS (SELECT "exhibit_id" FROM
"Exhibit" WHERE "exhibit_id" = {})THEN
                    UPDATE "Exhibit" SET {} WHERE "exhibit_id" = {};
                    RAISE NOTICE 'Successfully updated Exhibit table';
                    ELSE RAISE NOTICE 'Element with exhibit_id {} does not
exists in Exhibit!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Exhibition":

            query = """DO $$ BEGIN IF EXISTS (SELECT "exhibition id" FROM
"Exhibition" WHERE "exhibition_id" = {})THEN
                    UPDATE "Exhibition" SET {} WHERE "exhibition_id" = {};
                    RAISE NOTICE 'Successfully updated Exhibition table';
                    ELSE RAISE NOTICE 'Element with exhibition_id {} does
not exists in Exhibition!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Museum":

            query = """DO $$ BEGIN IF EXISTS (SELECT "museum_id" FROM
"Museum" WHERE "museum_id" = {})THEN
                    UPDATE "Museum" SET {} WHERE "museum_id" = {};
                    RAISE NOTICE 'Successfully updated Museum table';
                    ELSE RAISE NOTICE 'Element with museum_id {} does not
exists in Museum!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Director":

            query = """DO $$ BEGIN IF EXISTS (SELECT "director_id" FROM
"Director" WHERE "director_id" = {})THEN
                    UPDATE "Director" SET {} WHERE "director_id" = {};
                    RAISE NOTICE 'Successfully updated Museum table';
                    ELSE RAISE NOTICE 'Element with director_id {} does not
exists in Director!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        try:
            c.execute(query)
            self.conn.commit()
            self.print_notices(self.conn.notices)
        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return
```

Лістинг методу delete

```
def delete_data(self, table_name, id):
        c = self.conn.cursor()

        delete = ''

        Printer.print_info(table_name)

        if table_name == "Exhibit":
            delete = f'DELETE FROM "Exhibit" WHERE "exhibit_id" = {id};'

        elif  table_name == "Exhibition":
              delete = f'DELETE FROM "Exhibit" WHERE "exhibition_id" =
{id};'\
                       f'DELETE FROM "Exhibition" WHERE "exhibition_id" =
{id};'

        elif  table_name == "Museum":
            delete = f'DELETE FROM "Museum" WHERE "museum_id" = {id};'

        elif  table_name == "Director":
            delete = f'DELETE FROM "Director" WHERE "director_id" = {id};'

        try:
            c.execute(delete)
            self.conn.commit()

            if self.conn.notices:
                self.print_notices(self.conn.notices)
            else:
                Printer.print_success(f"Removed successfully from
{table_name} element with id: {id}", 5)

        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return
```

Завдання 2

Генерування «рандомізованих» даних

Таблиця «Exhibition» до додавання «рандомізованих» даних

	exhibition_id [PK] integer	name character varying	start_date date	end_date date
998	999	YGA	2024-02-22	2024-05-06
999	1000	ZQU	2024-01-02	2024-10-25
1000	1001	FDD	2024-02-25	2024-04-17
1001	1002	YRO	2024-12-01	2024-06-18
1002	1003	XWZ	2024-10-03	2024-03-26
1003	1004	QMD	2024-11-12	2024-12-10
1004	1005	JOJ	2024-03-04	2024-07-12
1005	1006	KGF	2024-01-01	2024-12-31
1006	1007	ZSQ	2024-09-04	2024-08-12
1007	1008	LPN	2024-02-24	2024-11-17
1008	1009	SLW	2024-04-24	2024-08-03
1009	1010	XEB	2024-10-12	2024-02-01
1010	1011	EAQ	2024-02-26	2024-08-28
1011	1012	OGQ	2024-10-26	2024-05-01
1012	1013	GIW	2024-04-04	2024-03-22
1013	1014	GQF	2023-12-06	2024-04-02
1014	1015	FDX	2024-07-03	2024-08-23
1015	1016	GLO	2024-05-23	2024-06-02
1016	1017	CTH	2024-11-07	2024-10-26
1017	1018	WHS	2024-08-08	2024-08-13
1018	1019	HEB	2024-08-02	2024-09-04
1019	1020	HUJ	2024-01-19	2024-03-28

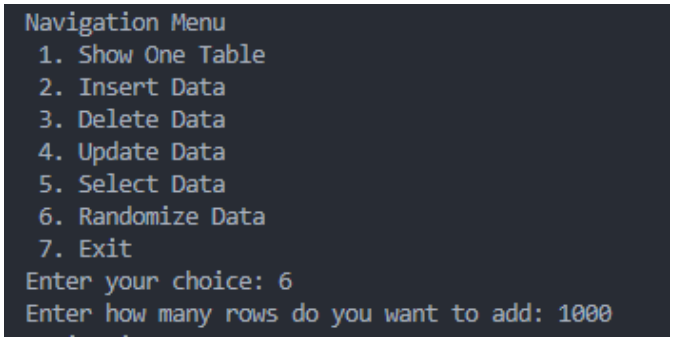
Total rows: 1019 of 1019 Query complete 00:00:00.278

Таблиця «Exhibit» до додавання «рандомізованих» даних

	author character varying	exhibit_id [PK] integer	name character varying	creation_date date	exhibition_id integer
996	MSP	999	JYG	2023-06-22	433
997	DYH	1000	PIH	2023-10-24	812
998	MMW	1112	BHO	2023-02-23	279
999	NCN	1113	BRI	2023-04-25	38
1000	PIX	1114	KYY	2023-02-18	659
1001	XQZ	1115	OGT	2023-11-15	190
1002	VMH	1116	NFJ	2023-05-26	882
1003	HSF	1117	EXY	2022-12-20	208
1004	PSA	1118	IJU	2023-04-15	81
1005	KIQ	1119	ZNU	2023-09-06	938
1006	FCM	1120	FDT	2023-09-14	435
1007	JPR	1121	IVD	2023-09-05	469
1008	NZU	1122	XVT	2023-05-30	556
1009	JZO	1123	VQS	2023-09-02	775
1010	NAX	1124	TBX	2022-12-11	315
1011	GCP	1125	YRB	2023-06-15	446
1012	VJR	1126	FCM	2023-11-13	983
1013	NKT	1127	XDN	2023-11-21	275
1014	OWT	1128	HAD	2023-03-07	39
1015	MBD	1129	APW	2023-08-17	3
1016	RYJ	1130	FKH	2023-06-24	665
1017	ZJS	1131	XIM	2023-12-04	277

Total rows: 1017 of 1017 Query complete 00:00:00.304

Додамо 1000 елементів до таблиць:



Лістинг SQL запитів:

```
DO $$
            DECLARE
                exhibition id seq INT;
                current_exhibition_id INT;
            BEGIN
                current_exhibition_id := COALESCE((SELECT max(exhibition_id) FROM "Exhibition"),
0) + 1;

                FOR exhibition id seq IN current exhibition id..current exhibition id + %s - 1
                LOOP
                    INSERT INTO "Exhibition" ("exhibition_id", "name", "start_date", "end_date")
                    VALUES (
                        exhibition id seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current_date + (random() * interval '365 days'),
                        current_date + (random() * interval '365 days') + interval '30 days'
                    );
                END LOOP;
            END $$;


DO $$
            DECLARE
                exhibit id seq INT;
                current exhibit id INT;
            BEGIN
                current_exhibit_id := COALESCE((SELECT max(exhibit_id) FROM "Exhibit"), 0) + 1;

                FOR exhibit_id_seq IN current_exhibit_id..current_exhibit_id + %s - 1
                LOOP
                    INSERT INTO "Exhibit" ("exhibit id", "author", "creation date", "name",
"exhibition id")
                    VALUES (
                        exhibit_id_seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current date - (random() * interval '365 days'),
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        (SELECT "exhibition_id" FROM "Exhibition" ORDER BY random() LIMIT 1)
                    );
                END LOOP;
            END $$;
```

Таблиця «Exhibition» після додавання «рандомізованих» даних

	exhibition_id [PK] integer	name character varying	start_date date	end_date date
1998	1999	OWN	2024-05-20	2024-09-22
1999	2000	BSV	2024-09-18	2024-12-17
2000	2001	UUN	2024-09-19	2024-10-22
2001	2002	BQY	2024-11-06	2024-01-17
2002	2003	AJM	2024-09-13	2024-03-02
2003	2004	IIG	2024-08-09	2024-06-09
2004	2005	ORD	2024-09-28	2024-10-20
2005	2006	IQM	2024-11-09	2024-11-21
2006	2007	EVM	2024-01-02	2024-07-25
2007	2008	BIY	2024-01-16	2024-04-25
2008	2009	RTD	2024-08-08	2024-05-05
2009	2010	ZIQ	2024-01-26	2024-09-28
2010	2011	NNU	2024-03-12	2024-07-12
2011	2012	QBK	2024-09-16	2024-01-14
2012	2013	UTC	2024-01-21	2024-01-23
2013	2014	HWY	2024-08-29	2024-04-22
2014	2015	OYP	2024-08-03	2024-07-05
2015	2016	FYC	2024-02-22	2024-01-08
2016	2017	LFK	2024-06-02	2024-10-12
2017	2018	DMG	2024-11-11	2024-09-05
2018	2019	CIU	2024-08-11	2024-12-19
2019	2020	DGC	2024-10-22	2024-02-26

Total rows: 2019 of 2019 Query complete 00:00:00.136

Таблиця «Exhibit» після додавання «рандомізованих» даних

	author character varying	exhibit_id [PK] integer	name character varying	creation_date date	exhibition_id integer
1996	GOP	2110	AQO	2023-03-08	1832
1997	JQY	2111	NRV	2023-11-06	621
1998	IDT	2112	VQF	2023-02-13	943
1999	EHV	2113	LRJ	2023-04-04	730
2000	DOG	2114	WQO	2023-11-15	1211
2001	NOB	2115	BRG	2022-12-07	508
2002	TSV	2116	WLW	2023-10-02	1150
2003	AQT	2117	UFP	2023-06-29	1414
2004	CWO	2118	NTI	2023-02-13	1335
2005	YAY	2119	PXO	2022-12-29	1037
2006	GFT	2120	ILB	2023-07-23	1119
2007	XXZ	2121	LFE	2023-05-11	1029
2008	ATP	2122	GNP	2023-06-16	176
2009	SUT	2123	NYF	2023-09-23	1022
2010	GYX	2124	QQA	2023-01-04	1699
2011	IRQ	2125	LNI	2023-09-22	1056
2012	AKM	2126	PKL	2023-11-14	1197
2013	NCF	2127	VKV	2023-11-09	145
2014	QZF	2128	ILI	2023-05-11	476
2015	FAF	2129	CQH	2023-07-21	1370
2016	TQU	2130	TQY	2023-01-06	1565
2017	KCW	2131	TGI	2023-02-13	939

Total rows: 2017 of 2017 Query complete 00:00:00.142

Лістинг методу randomize

```
def randomize_data(self, count):
        c = self.conn.cursor()

        c.execute("""
            DO $$
            DECLARE
                exhibition_id_seq INT;
                current_exhibition_id INT;
            BEGIN
                current_exhibition_id := COALESCE((SELECT max(exhibition_id)
FROM "Exhibition"), 0) + 1;

                FOR exhibition_id_seq IN
current_exhibition_id..current_exhibition_id + %s - 1
                LOOP
                    INSERT INTO "Exhibition" ("exhibition_id", "name",
"start_date", "end_date")
                    VALUES (
                        exhibition_id_seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current_date + (random() * interval '365 days'),
                        current_date + (random() * interval '365 days') +
interval '30 days'
                    );
                END LOOP;
            END $$;
        """, (count,))

        c.execute("""
            DO $$
            DECLARE
                exhibit_id_seq INT;
                current_exhibit_id INT;
            BEGIN
                current_exhibit_id := COALESCE((SELECT max(exhibit_id) FROM
"Exhibit"), 0) + 1;

                FOR exhibit_id_seq IN current_exhibit_id..current_exhibit_id
+ %s - 1
                LOOP
                    INSERT INTO "Exhibit" ("exhibit_id", "author",
"creation_date", "name", "exhibition_id")
                    VALUES (
                        exhibit_id_seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current_date - (random() * interval '365 days'),
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        (SELECT "exhibition_id" FROM "Exhibition" ORDER BY
random() LIMIT 1)  -- Select a random exhibition_id
                    );
                END LOOP;
            END $$;
        """, (count,))

        self.conn.commit()
```

Завдання 3

Пошук даних

Спробуємо отримати всі роботи і виставки на яких представлений певний автор:

```
Navigation Menu
  1. Show One Table
  2. Insert Data
  3. Delete Data
  4. Update Data
  5. Select Data
  6. Randomize Data
  7. Exit
Enter your choice: 5
1. Show Exhibit by author and Exhibitions they represented at
2. Show list of authors that are represented at specific Exhibition
Enter row index (1-2): 1
Enter Author name: ABA
# Request took 5 ms
fetched [{'exhibit_name': 'ABA', 'exhibition_name': 'ZWZ', 'author': 'ABA'}]
```

Спробуємо отримати список всіх авторів роботи яких представлені на певній виставці:

```
Enter your choice: 5
1. Show Exhibit by author and Exhibitions they represented at
2. Show list of authors that are represented at specific Exhibition
Enter row index (1-2): 2
Enter Exhibition id: 3
# Request took 0 ms
fetched authors: [{'author_name': 'HJQ'}, {'author_name': 'MBD'}, {'author_name': 'ZZJ'}]
```

Лістинг методу select

```
def select_data(self, selected_options):
        option_index = selected_options['option_index']
        data = selected_options['data']

        query = ''

        c = self.conn.cursor()


        if (option_index == 1):
            author_name = data['author_name']

            query = """
                SELECT ex.name as exhibit_name, e.name as exhibition_name,
ex.author
                FROM "Exhibition" e
                LEFT JOIN "Exhibit" ex ON e.exhibition_id = ex.exhibition_id
```

```
                WHERE ex.author = '{}';
                """.format(author_name)

            try:
                begin = int(time.time() * 1000)
                c.execute(query)
                end = int(time.time() * 1000) - begin

                Printer.print_info(f"Request took {end} ms")

                records = c.fetchall()

                result_objects = []

                for record in records:
                    exhibit_name, exhibition_name, author = record
                    result_objects.append({
                        'exhibit_name': exhibit_name,
                        'exhibition_name': exhibition_name,
                        'author': author
                    })

                Printer.print_text('fetched {}'.format(result_objects))

            except psycopg2.Error as e:
                self.conn.rollback()
                Printer.print_error(f"Error: {e}", 5)
                return


        elif (option_index == 2):
            exhibition_id = data['exhibition_id']

            query = """
                    SELECT ex.author
                    FROM "Exhibition" e
                    LEFT JOIN "Exhibit" ex ON e.exhibition_id =
ex.exhibition_id
                    WHERE e.exhibition_id = {}
                    GROUP BY ex.author;
                    """.format(exhibition_id)

            try:
                begin = int(time.time() * 1000)
                c.execute(query)
                end = int(time.time() * 1000) - begin

                Printer.print_info(f"Request took {end} ms")

                authors = c.fetchall()

                result_objects = []

                for author in authors:
                    author_name, = author
                    result objects.append({
                        'author_name': author_name
                    })

                Printer.print_text('fetched authors:
{}'.format(result_objects))

            except psycopg2.Error as e:
                self.conn.rollback()
```

```
            Printer.print_error(f"Error: {e}", 5)
            return
```

Завдання 4

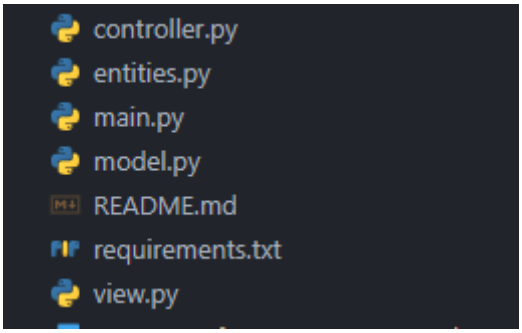
Шаблон MVC

У даній роботі був використаний шаблон проектування MVC (Model-View-Controller).

Model представляє собою клас, що визначає логіку обробки даних. Його реалізація розташована у файлі model.py, де відбуваються складні операції, такі як вставка, видалення, оновлення, пошук, рандомізація даних, а після виконання події результат передається до View.

View представляє консольний інтерфейс для взаємодії з користувачем та відповідає за введення/виведення даних. Реалізація знаходиться у файлі view.py, де присутній клас View.

Controller забезпечує комунікацію між користувачем і системою, а також між поданням і сховищем даних. Він отримує введені користувачем дані та обробляє їх. Реалізація цього компонента розташована у файлі controller.py.



main.py – точка входу в програму, запускає початковий інтерфейс. model.py – виконує операції з базою даних.
view.py – файл, що відповідає за функціонал виведення даних, повідомлень для користувача та реалізовує меню для взаємодії з користувачем, приймає введені дані від користувача і передає їх у контролер.
controller.py – виконує підключення до бази даних, обробляє введені користувачем дані, подає відповідну команду до model.py.

також було створено додаткові класи-утиліти:
printer.py – для зручного виведення повідомлень у консоль
entitites.py – для зручної взаємодії з сутностями у базі даних

Код програми

main.py

```
from controller import Controller

if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

controller.py

```
from model import Model
from view import View
from utils.printer import Printer
import sys

class Controller:
    def __init__(self):
        self.model = Model()
        self.view = View()

    def run(self):
        while True:
            choice = self.view.print_menu(indent=1)

            if choice == "1":
                self.show_one_table()
            elif choice == "2":
                self.insert_data()
            elif choice == "3":
                self.delete_data_from_table()
            elif choice == "4":
                self.update_data()
            elif choice == "5":
                self.select_data()
            elif choice == "6":
                self.randomize_data()
            elif choice == "7":
                self.exit_program()
                break
            else:
                Printer.print_error("Invalid choice. Please try again.",
indent=1)

    def get_table_index(self):
        self.view.show_all_tables_names()
        table_index = self.view.get_table_index()

        return table_index

    def get_table_row_index(self, table_name):
        tableData = self.model.get_table_data(table_name)
        self.view.print_table_content(tableData)
        row_index = self.view.get_row_index()

        return row_index

    def show_one_table(self):
        tables = self.view.entities
        table_index = self.get_table_index() - 1
        table_name = tables[table_index].getName()
```

```
        tableData = self.model.get_table_data(table_name)
        self.view.print_table_content(tableData)

    def show_all_tables(self):
        self.model.get_all_tables_data()

    def insert_data(self):
        tables = self.view.entities
        table_index = self.get_table_index() - 1
        input_data = self.view.get_input_data_for_table(table_index)
        table_name = tables[table_index].getName()

        self.model.insert_data(table_name, input_data)

    def delete_data_from_table(self):
        tables = self.view.entities
        table_index = self.get_table_index() - 1
        table_name = tables[table_index].getName()
        id = self.view.get_row_id()

        self.model.delete_data(table_name, id)

    def update_data(self):
        tables = self.view.entities
        table_index = self.get_table_index() - 1
        table_name = tables[table_index].getName()
        id = self.view.get_row_id()
        new_data = self.view.get_input_data(table_index, isUpdate=True)


        self.model.update_data(table_name, id, new_data)

    def select_data(self):
        selected_options = self.view.print_select_options()

        self.model.select_data(selected_options)


    def randomize_data(self):
        count = self.view.get_rows_count()

        self.model.randomize_data(count)

    def exit_program(self):
        self.view.exit_program()
        sys.exit()
```

view.py

```
# View.py
from utils.printer import Printer
from entities import Exhibit, Exhibition, Museum, Director

class View:
    def print_menu(self, indent=0):
        Printer.print_text("Navigation Menu", indent=0)
        Printer.print_text("1. Show One Table", indent)
        Printer.print_text("2. Insert Data", indent)
        Printer.print_text("3. Delete Data", indent)
        Printer.print_text("4. Update Data", indent)
        Printer.print_text("5. Select Data", indent)
```

```
        Printer.print_text("6. Randomize Data", indent)
        Printer.print_text("7. Exit", indent)
        return input("Enter your choice: ")

    entities = [Exhibit, Exhibition, Museum, Director];

    def get_table_index(self):
        return int(input("Enter table index: "))

    def get_row_id(self):
        return int(input("Enter id: "))

    def get_rows_count(self):
        return int(input("Enter how many rows do you want to add: "))

    def get_row_index(self):
        return int(input("Enter row index: "))

    def show_table_name(self, table_name):
        Printer.print_text(table_name, indent=1)

    def show_all_tables_names(self):
        tables = self.entities

        for i, table in enumerate(tables, start=1):
            table_name = table.getName()
            self.show_table_name(f"{i}. {table_name.capitalize()}")

    def print_table_content(self, tableRows):
        if tableRows:
            for i, row in enumerate(tableRows, start=1):
                Printer.print_text(f"{i}. {row}")

    def exit_program(self):
        Printer.print_info("Program terminated successfully!")

    def get_input_data_for_table(self, table_index):
        tables = self.entities

        if 0 <= table_index <= len(tables):
            return self.get_input_data(table_index)
        else:
            Printer.print_error("Invalid table index.")

    def get_input_data(self, table_index, isUpdate=False):
        tables = self.entities

        if 0 <= table_index <= len(tables):
            return tables[table_index].get_input_data(isUpdate)
        else:
            Printer.print_error("Invalid table index.")

    def get_table_properties(self, table_index):
        tables = self.entities

        if 0 <= table index <= len(tables):
            return tables[table_index].print_properties()
        else:
            Printer.print_error("Invalid table index.")

    def print_select_options(self):
        Printer.print_text("1. Show Exhibit by author and Exhibitions they
represented at")
```

```
        Printer.print_text("2. Show list of authors that are represented at
specific Exhibition")

        option_index = int(input("Enter row index (1-2): "))
        data = {}

        while (option_index < 1 or option_index > 3):
           option_index = int(input("Enter row index (1-2): "))

        if (option_index == 1):
            data['author_name'] = input("Enter Author name: ")
        elif (option_index == 2):
            data['exhibition_id'] = int(input("Enter Exhibition id: "))

        return {"option_index": option_index, "data": data}
```

model.py

```
import os
from dotenv import load_dotenv
import psycopg2
from utils.printer import Printer
import time

# Load environment variables from .env
load_dotenv()

DB_PASSWORD = os.getenv("DB_PASSWORD")

class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='postgres',
            user='postgres',
            password=DB_PASSWORD,
            host='localhost',
            port=5432
        )

    def get_table_data(self, table_name):
        c = self.conn.cursor()
        Printer.print_info(table_name)
        c.execute(f'SELECT * FROM public."{table_name}"')
        result = c.fetchall()

        Printer.print_success(f"Successfull fetch from {table_name}:
{result}", 5)

    def get_all_tables_data(self):
        Printer.print_success(f"get_all_tables_data tables", 5)

    def print_notices(self, notices):
        for notice in notices:
            _, _, notice_text = notice.partition(':')
            clean_notice = notice_text.strip()
            Printer.print_info(clean_notice)

    def insert_data(self, table_name, data):
        values = tuple(data.values())
        columns = ', '.join(data.keys())
```

```
        c = self.conn.cursor()

        if table_name == "Exhibit":

            query = """DO $$ BEGIN IF NOT EXISTS (SELECT "exhibit_id" FROM
                    "Exhibit" WHERE "exhibit_id" = {})
                    THEN INSERT INTO
"Exhibit"("exhibit_id","exhibition_id","name","author","creation_date")
                    VALUES {}; RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;
END $$;
                    """.format(data['exhibit_id'], values,"'Successfully
added to Exhibit table'", "'Error: This exhibit_id already exists!'")

        elif  table_name == "Exhibition":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "exhibition_id" FROM "Exhibition"
                        WHERE "exhibition_id" = {}
                    ) THEN
                        INSERT INTO "Exhibition"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new exhibition!';
                    ELSE
                        RAISE NOTICE 'Element with exhibition_id {} already
exists in Exhibition!';
                    END IF;
                END $$;
            """.format(data['exhibition_id'], columns, values,
data['exhibition_id'])

        elif  table_name == "Museum":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "museum_id" FROM "Museum"
                        WHERE "museum_id" = {}
                    ) THEN
                        INSERT INTO "Museum"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new museum!';
                    ELSE
                        RAISE NOTICE 'Element with museum_id {} already
exists in Museum!';
                    END IF;
                END $$;
            """.format(data['museum_id'], columns, values, data['museum_id'])

        elif  table_name == "Director":

            query = """
                DO $$ BEGIN
                    IF NOT EXISTS (
                        SELECT "director id" FROM "Director"
                        WHERE "director_id" = {}
                    ) THEN
                        INSERT INTO "Director"({})
                        VALUES {};
                        RAISE NOTICE 'Successfully added new director!';
                    ELSE
                        RAISE NOTICE 'Element with director_id {} already
exists in Director!';
```

```
                    END IF;
                END $$;
            """.format(data['director_id'], columns, values,
data['director_id'])

        try:
            c.execute(query)
            self.conn.commit()
            self.print_notices(self.conn.notices)
        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return

    def delete_data(self, table_name, id):
        c = self.conn.cursor()

        delete = ''

        if table_name == "Exhibit":
            delete = f'DELETE FROM "Exhibit" WHERE "exhibit_id" = {id};'

        elif  table_name == "Exhibition":
              delete = f'DELETE FROM "Exhibit" WHERE "exhibition_id" =
{id};'\
                       f'DELETE FROM "Exhibition" WHERE "exhibition_id" =
{id};'

        elif  table_name == "Museum":
            delete = f'DELETE FROM "Museum" WHERE "museum_id" = {id};'

        elif  table_name == "Director":
            delete = f'DELETE FROM "Director" WHERE "director_id" = {id};'

        try:
            c.execute(delete)
            self.conn.commit()

            if self.conn.notices:
                self.print_notices(self.conn.notices)
            else:
                Printer.print_success(f"Removed successfully from
{table_name} element with id: {id}", 5)

        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return


    def update_data(self, table_name, id, new_data):
        c = self.conn.cursor()

        set_clause = ', '.join(f'"{key}" = \'{value}\'' for key, value in
new_data.items())

        if table_name == "Exhibit":

            query = """DO $$ BEGIN IF EXISTS (SELECT "exhibit_id" FROM
"Exhibit" WHERE "exhibit_id" = {})THEN
                    UPDATE "Exhibit" SET {} WHERE "exhibit_id" = {};
                    RAISE NOTICE 'Successfully updated Exhibit table';
                    ELSE RAISE NOTICE 'Element with exhibit_id {} does not
exists in Exhibit!';
```

```
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Exhibition":

            query = """DO $$ BEGIN IF EXISTS (SELECT "exhibition_id" FROM
"Exhibition" WHERE "exhibition_id" = {})THEN
                    UPDATE "Exhibition" SET {} WHERE "exhibition_id" = {};
                    RAISE NOTICE 'Successfully updated Exhibition table';
                    ELSE RAISE NOTICE 'Element with exhibition_id {} does
not exists in Exhibition!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Museum":

            query = """DO $$ BEGIN IF EXISTS (SELECT "museum_id" FROM
"Museum" WHERE "museum_id" = {})THEN
                    UPDATE "Museum" SET {} WHERE "museum_id" = {};
                    RAISE NOTICE 'Successfully updated Museum table';
                    ELSE RAISE NOTICE 'Element with museum_id {} does not
exists in Museum!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        elif  table_name == "Director":

            query = """DO $$ BEGIN IF EXISTS (SELECT "director_id" FROM
"Director" WHERE "director_id" = {})THEN
                    UPDATE "Director" SET {} WHERE "director_id" = {};
                    RAISE NOTICE 'Successfully updated Museum table';
                    ELSE RAISE NOTICE 'Element with director_id {} does not
exists in Director!';
                    END IF; END $$;
                    """.format(id, set_clause, id, id)

        try:
            c.execute(query)
            self.conn.commit()
            self.print_notices(self.conn.notices)
        except psycopg2.Error as e:
            self.conn.rollback()
            Printer.print_error(f"Error: {e}", 5)
            return

    def select_data(self, selected_options):
        option_index = selected_options['option_index']
        data = selected_options['data']

        query = ''

        c = self.conn.cursor()


        if (option_index == 1):
            author name = data['author name']

            query = """
                SELECT ex.name as exhibit_name, e.name as exhibition_name,
ex.author
                FROM "Exhibition" e
                LEFT JOIN "Exhibit" ex ON e.exhibition_id = ex.exhibition_id
                WHERE ex.author = '{}';
                """.format(author_name)
```

```
            try:
                begin = int(time.time() * 1000)
                c.execute(query)
                end = int(time.time() * 1000) - begin

                Printer.print_info(f"Request took {end} ms")

                records = c.fetchall()

                result_objects = []

                for record in records:
                    exhibit_name, exhibition_name, author = record
                    result_objects.append({
                        'exhibit_name': exhibit_name,
                        'exhibition_name': exhibition_name,
                        'author': author
                    })

                Printer.print_text('fetched {}'.format(result_objects))

            except psycopg2.Error as e:
                self.conn.rollback()
                Printer.print_error(f"Error: {e}", 5)
                return


        elif (option_index == 2):
            exhibition_id = data['exhibition_id']

            query = """
                    SELECT ex.author
                    FROM "Exhibition" e
                    LEFT JOIN "Exhibit" ex ON e.exhibition_id =
ex.exhibition_id
                    WHERE e.exhibition_id = {}
                    GROUP BY ex.author;
                    """.format(exhibition_id)

            try:
                begin = int(time.time() * 1000)
                c.execute(query)
                end = int(time.time() * 1000) - begin

                Printer.print_info(f"Request took {end} ms")

                authors = c.fetchall()

                result_objects = []

                for author in authors:
                    author_name, = author
                    result_objects.append({
                        'author_name': author_name
                    })

                Printer.print_text('fetched authors:
{}'.format(result_objects))

            except psycopg2.Error as e:
                self.conn.rollback()
                Printer.print_error(f"Error: {e}", 5)
                return
```

```
    def randomize_data(self, count):
        c = self.conn.cursor()

        c.execute("""
            DO $$
            DECLARE
                exhibition_id_seq INT;
                current_exhibition_id INT;
            BEGIN
                current_exhibition_id := COALESCE((SELECT max(exhibition_id)
FROM "Exhibition"), 0) + 1;

                FOR exhibition_id_seq IN
current_exhibition_id..current_exhibition_id + %s - 1
                LOOP
                    INSERT INTO "Exhibition" ("exhibition_id", "name",
"start_date", "end_date")
                    VALUES (
                        exhibition_id_seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current_date + (random() * interval '365 days'),
                        current_date + (random() * interval '365 days') +
interval '30 days'
                    );
                END LOOP;
            END $$;
        """, (count,))

        c.execute("""
            DO $$
            DECLARE
                exhibit_id_seq INT;
                current_exhibit_id INT;
            BEGIN
                current_exhibit_id := COALESCE((SELECT max(exhibit_id) FROM
"Exhibit"), 0) + 1;

                FOR exhibit_id_seq IN current_exhibit_id..current_exhibit_id
+ %s - 1
                LOOP
                    INSERT INTO "Exhibit" ("exhibit_id", "author",
"creation_date", "name", "exhibition_id")
                    VALUES (
                        exhibit_id_seq,
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        current_date - (random() * interval '365 days'),
                        chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int),
                        (SELECT "exhibition_id" FROM "Exhibition" ORDER BY
random() LIMIT 1)  -- Select a random exhibition_id
                    );
                END LOOP;
            END $$;
        """, (count,))

        self.conn.commit()
```

entities.py

```
from utils.printer import Printer
from datetime import date, datetime

class Entity:
    def __init__(self, **kwargs):
        for key, value in kwargs.items():
            setattr(self, key, value)

    @classmethod
    def get_input_data(cls, isUpdate):
        input_data = {}
        for prop, prop_type in cls.__init__.__annotations__.items():
            if not (isUpdate and ('_id' in prop)):
                value = cls.get_valid_input(f"Enter {prop}
({prop_type.__name__}): ", prop_type)
                input_data[prop] = value
        return input_data

    @classmethod
    def print_properties(self):
        for i, prop in self.__init__.__annotations__.items():
            Printer.print_text(f"{i}. {prop}")

    @classmethod
    def getName(cls):
        return cls.__name__

    @staticmethod
    def get_valid_input(prompt: str, data_type: type):
        while True:
            try:
                if data_type == date:
                    user_input = date.fromisoformat(input(prompt))

                    user_input = user_input.strftime('%Y-%m-%d')
                else:
                    user_input = data_type(input(prompt))

                return user_input
            except ValueError:
                Printer.print_error(f"Invalid input. Please enter a valid
{data_type.__name__}.")



class Exhibit(Entity):
    def __init__(self, exhibit_id: int, exhibition_id: int, name: str,
author: str, creation_date: date):
        super().__init__(exhibit_id=exhibit_id, exhibition_id=exhibition_id,
name=name, author=author, creation_date=creation_date)

class Exhibition(Entity):
    def __init__(self, exhibition_id: int, name: str, start_date: date,
end_date: date):
        super().__init__(exhibition_id=exhibition_id, name=name,
start_date=start_date, end_date=end_date)


class Museum(Entity):
    def __init__(self, museum_id: int, name: str):
        super().__init__(museum_id=museum_id, name=name)
```

```
class Director(Entity):
    def __init__(self, director_id: int, firstname: str, lastname: str):
        super().__init__(director_id=director_id, firstname=firstname,
lastname=lastname)
```

printer.py

```
from colorama import init, Fore, Style

init(autoreset=True)

class Printer:
    @staticmethod
    def print_colored_text(text, color=Fore.WHITE, indent=0):
        indentation = " " * indent
        print(f"{indentation}{color}{text}{Style.RESET_ALL}")

    @staticmethod
    def print_success(text, indent=0):
        Printer.print_colored_text(f":) {text}", color=Fore.GREEN,
indent=indent)

    @staticmethod
    def print_error(text, indent=0):
        Printer.print_colored_text(f"!!! {text}", color=Fore.RED,
indent=indent)

    @staticmethod
    def print_info(text, indent=0):
        Printer.print_colored_text(f"# {text}", color=Fore.BLUE,
indent=indent)

    @staticmethod
    def print_text(text, indent=0):
        Printer.print_colored_text(f"{text}", color=Fore.WHITE,
indent=indent)
```

Опис функцій модуля «Controller»:

- *get_table_index()* - Виводить на екран назви всіх таблиць та отримує від користувача індекс вибраної таблиці.
- *get_table_row_index(table_name)* - Отримує від користувача індекс рядка для вибраної таблиці.
- *show_one_table()* - Виводить вміст вибраної таблиці.
- *show_all_tables()* - Виводить вміст всіх таблиць.
- *insert_data()* - Додає нові дані в вибрану таблицю.

  delete_data_from_table() - Видаляє дані з вибраної таблиці.
- *update_data()* - Оновлює дані в вибраній таблиці.

- *select_data()* - Здійснює вибірковий вивід даних відповідно до вибору користувача.
- *randomize_data()* - Заповнює вибрані таблиці випадковими даними. *exit_program()* - Завершує роботу програми.

Опис функцій модуля «View»:

- *print_menu(indent)* - Виводить на екран головне меню та отримує від користувача вибір опції.
- *get_table_index()* - Отримує від користувача індекс вибраної таблиці.
- *get_row_id()* - Отримує від користувача ідентифікатор рядка.
- *get_rows_count()* - Отримує від користувача кількість рядків для додавання.
- *get_row_index()* - Отримує від користувача індекс рядка таблиці.
- *show_table_name(table_name)* - Виводить на екран назву таблиці.
- *show_all_tables_names()* - Виводить на екран назви всіх таблиць.
- *print_table_content(tableRows)* - Виводить на екран вміст таблиці.
- *exit_program()* - Виводить повідомлення про завершення роботи програми.
- *get_input_data_for_table(table_index)* - Отримує від користувача дані для додавання в таблицю.
- *get_input_data(table_index, isUpdate=False)* - Отримує від користувача дані для таблиці.
- *get_table_properties(table_index)* - Отримує від користувача властивості таблиці.
- *print_select_options()* - Виводить опції для вибіркового виводу даних та отримує від користувача вибір.

Опис функцій модуля «Model»:

- *get_table_data(table_name)* - Отримує дані з вказаної таблиці та виводить їх на екран.
- *get_all_tables_data()* - Виводить на екран дані з усіх таблиць.
- *print_notices(notices)* - Виводить повідомлення.
- *insert_data(table_name, data)* - Додає нові дані в вказану таблицю.
- *delete_data(table_name, id)* - Видаляє дані з вказаної таблиці за ідентифікатором.
- *update_data(table_name, id, new_data)* - Оновлює дані в вказаній таблиці за ідентифікатором.
- *select_data(selected_options)* - Виконує вибірковий вивід даних відповідно до вибору користувача.
- *randomize_data(count)* - Заповнює вказану кількість рядків у вибраних таблицях випадков