

## **Розрахунково-графічна робота з дисципліни "Бази даних та засоби управління"**

### **Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**

Посилання на репозиторій - <https://github.com/Coltenus/BD-RGR>

Посилання на телеграм - <https://t.me/Coltenus>

*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).

У роботі була використана мова C# та бібліотека Npgsql.

## Структура бази даних з ЛР №1:

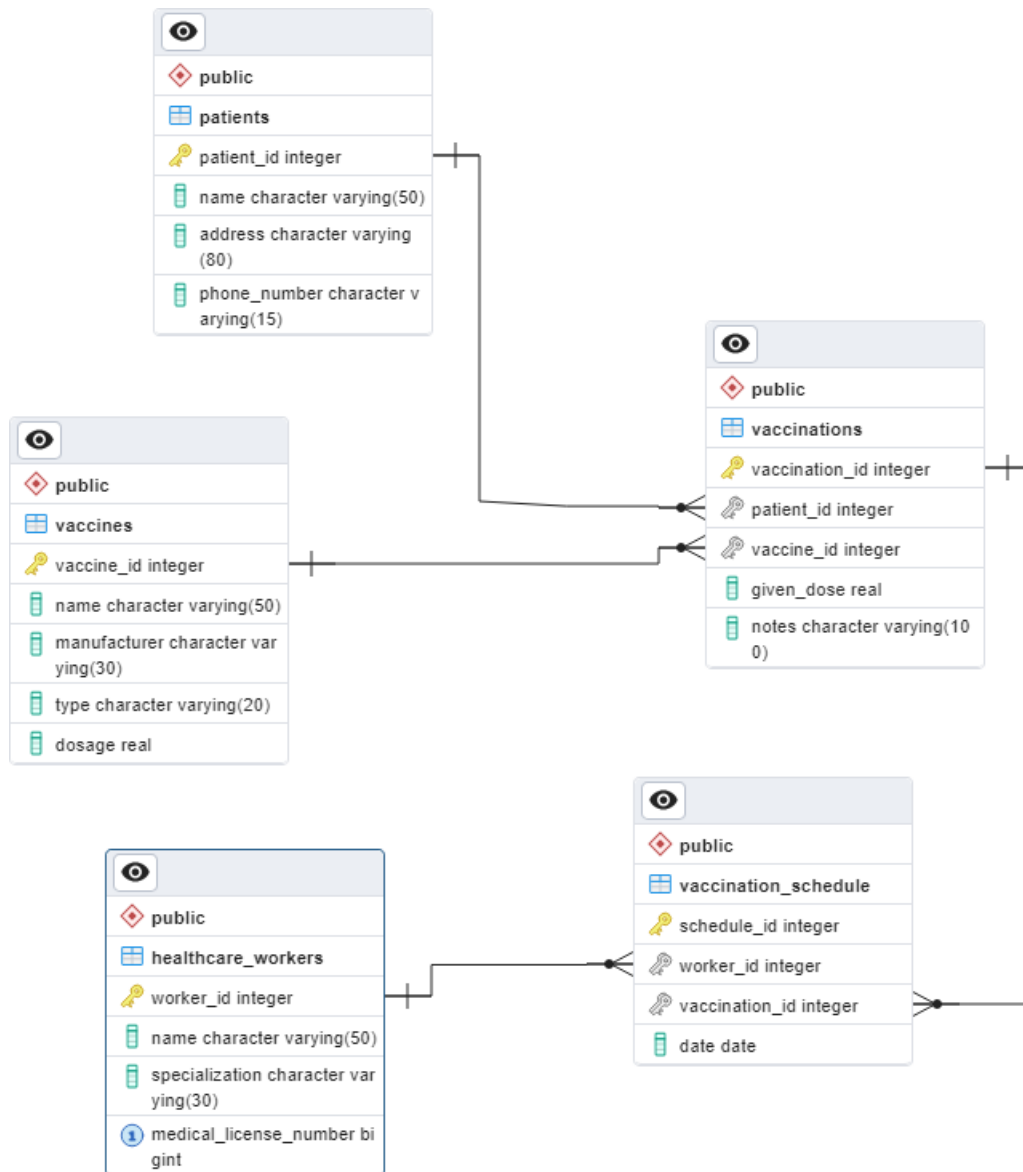











Схема меню користувача:

0. Вихід
1. Вибір таблиці
2. Пошук даних з кількох таблиць (Завдання 3). Для виконання потрібно ввести:
  - 1) рядки з даними про таблиці (назва, потрібний стовпець, стовпець спільний з попередньою таблицею)
  - 2) номер таблиці за якою відбудеться пошук
  - 3) стовпчик для пошуку
  - 4) шукані значення
3. Пошук рядка у таблиці за стовпцем і значенням
4. Пошук рядків у таблиці за стовпцем і значеннями
5. Отримання всіх рядків таблиці
6. Додавання рядка до таблиці
7. Додавання рядків до таблиці
8. Видалення рядків за значенням
9. Видалення рядків з більшим(меншим) значенням
- 10.Зміна значень рядка
- 11.Зміна значень рядків
- 12.Згенерувати задану кількість рядків

# Пункт №1

1)

	 vaccination_id ^	 patient_id ÷	 vaccine_id ÷	 given_dose ÷	 notes ÷
10	10	17	11	0.32003003	<null>
11	11	9	13	0.20335159	<null>
12	12	4	12	0.06800184	<null>
13	13	7	12	0.35929382	<null>
14	14	5	4	0.6214216	<null>
15	15	3	1	0.39294454	<null>
16	16	1	4	0.8881115	<null>
17	17	16	5	0.7399239	<null>
18	18	7	7	0.3227934	<null>
19	19	10	10	0.4117662	<null>
20	20	20	1	0.3046161	<null>
21	21	12	5	0.9819676	<null>
22	22	7	4	0.26508874	<null>
23	23	16	8	0.70336264	<null>
24	24	8	5	0.40868425	<null>
25	25	20	13	0.513336	<null>
26	26	19	8	0.7796658	<null>
27	27	9	11	0.47575715	<null>
28	28	7	1	0.27246863	<null>
29	29	11	10	0.7400186	<null>
					<null>
	 schedule_id ÷	 worker_id ÷	 vaccination_id ^	 date ÷	
10	4	14	0	2023-10-30	
11	1	7	8	2023-05-06	
12	8	15	8	2023-02-01	
13	25	15	10	2023-08-08	
14	21	9	10	2023-06-01	
15	27	1	11	2023-04-09	
16	16	7	12	2023-03-07	
17	28	1	13	2023-07-20	
18	30	5	14	2023-10-22	
19	3	13	14	2023-07-12	
20	9	12	16	2023-12-19	
21	22	2	16	2023-04-26	
22	18	2	17	2023-10-22	
23	13	13	19	2023-03-08	
24	29	14	19	2023-04-10	
25	12	8	19	2023-09-06	
26	19	8	22	2023-03-22	
27	6	5	22	2023-10-04	
28	2	12	24	2023-03-02	
29	23	17	26	2023-02-01	
30	10	6	27	2023-08-14	

Choose action: 1

Choose model: 3

Choose action: 9

Enter column: vaccination\_id

Enter value: 20

Greater or less(true, false): true

Choose action: 1

Choose model: 5

Choose action: 5

schedule_id	worker_id	vaccination_id	date
13	13	19	08.03.2023 0:00:00
24	2	8	13.04.2023 0:00:00
20	14	8	15.09.2023 0:00:00
7	1	5	09.06.2023 0:00:00
29	14	19	10.04.2023 0:00:00
4	14	8	30.10.2023 0:00:00
27	1	11	09.04.2023 0:00:00
22	2	16	26.04.2023 0:00:00
11	10	6	19.01.2023 0:00:00
1	7	8	06.05.2023 0:00:00
9	12	16	19.12.2023 0:00:00
26	7	4	01.06.2023 0:00:00
12	8	19	06.09.2023 0:00:00
18	2	17	22.10.2023 0:00:00

14	18	4	28.08.2023 0:00:00
8	15	8	01.02.2023 0:00:00
15	8	3	10.01.2023 0:00:00
21	9	10	01.06.2023 0:00:00
25	15	10	08.08.2023 0:00:00
16	7	12	07.03.2023 0:00:00
3	13	14	12.07.2023 0:00:00
5	16	2	30.04.2023 0:00:00
30	5	14	22.10.2023 0:00:00
28	1	13	20.07.2023 0:00:00
17	4	2	12.09.2023 0:00:00

	🔍 schedule_id ÷	🔍 worker_id ÷	🔍 vaccination_id ^	📅 date ÷
7	11	10	6	2023-01-19
8	4	14	8	2023-10-30
9	1	7	8	2023-05-06
10	20	14	8	2023-09-15
11	8	15	8	2023-02-01
12	24	2	8	2023-04-13
13	25	15	10	2023-08-08
14	21	9	10	2023-06-01
15	27	1	11	2023-04-09
16	16	7	12	2023-03-07
17	28	1	13	2023-07-20
18	3	13	14	2023-07-12
19	30	5	14	2023-10-22
20	22	2	16	2023-04-26
21	9	12	16	2023-12-19
22	18	2	17	2023-10-22
23	12	8	19	2023-09-06
24	29	14	19	2023-04-10
25	13	13	19	2023-03-08

2)

*Choose action: 6*

*Enter schedule\_id: 15*

*Enter worker\_id: 12*

*Enter vaccination\_id: 18*

*Enter date: 2023-03-10*

*повторювані значення ключа порушують обмеження унікальності  
"vaccination\_schedule\_pkey"*

*Choose action: 6*

*Enter schedule\_id: 19*

*Enter worker\_id: 12*

*Enter vaccination\_id: 18*

*Enter date: 2023-03-10*

*Choose action: 6*

*Enter schedule\_id: 23*

*Enter worker\_id: 18*

*Enter vaccination\_id: 25*

*Enter date: 2023-12-05*

*insert або update в таблиці "vaccination\_schedule" порушує обмеження  
зовнішнього ключа "vaccination\_id"*

## Пункт №2

*Choose action: 1*

*Choose model: 3*

*Choose action: 12*

*Enter count of elements: 1000*

*Need debug(true, false): true*

*INSERT INTO vaccinations*

*SELECT DISTINCT \* FROM (SELECT generate\_series AS  
vaccination\_id,TRUNC(RANDOM()\*(SELECT MAX(patient\_id) FROM patients))::int  
+ 1 AS patient\_id,TRUNC(RANDOM()\*(SELECT MAX(vaccine\_id) FROM  
vaccines))::int + 1 AS vaccine\_id,float4(RANDOM()\*0.99 + 0.01) AS dosage FROM  
GENERATE\_SERIES(1, 1000)) AS t1*

*GROUP BY t1.vaccination\_id, t1.patient\_id, t1.vaccine\_id, t1.dosage*

*Choose action: 1*

*Choose model: 5*

*Choose action: 12*

*Enter count of elements: 1000*






*Need debug(true, false): true*

*INSERT INTO vaccination\_schedule*

*SELECT DISTINCT \* FROM (SELECT generate\_series AS  
schedule\_id,TRUNC(RANDOM()\*(SELECT MAX(worker\_id) FROM  
healthcare\_workers))::int + 1 AS worker\_id,TRUNC(RANDOM()\*((SELECT  
MAX(vaccination\_id) FROM vaccinations))::int + 1 AS vaccination\_id,DATE('2023-  
01-01') + TRUNC(RANDOM()\*(DATE('2023-12-31') - DATE('2023-01-01'))::int AS  
date FROM generate\_series(1, 1000)) AS t1*

*GROUP BY t1.schedule\_id, t1.worker\_id, t1.vaccination\_id, t1.date*



	 vaccination_id ^	 patient_id ÷	 vaccine_id ÷	 given_dose ÷	 notes
1	1	7	11	0.68115795	<null>
2	2	11	7	0.397015	<null>
3	3	1	7	0.88291603	<null>
4	4	16	3	0.6385799	<null>
5	5	5	7	0.30149752	<null>
6	6	10	2	0.95821077	<null>
7	7	7	5	0.04065869	<null>
8	8	15	9	0.100570284	<null>
9	9	15	8	0.85526323	<null>
10	10	20	1	0.37246218	<null>
11	11	3	4	0.25107664	<null>
12	12	16	6	0.9375233	<null>
13	13	7	8	0.66682774	<null>
14	14	9	3	0.7359259	<null>
15	15	20	9	0.41395208	<null>
16	16	17	13	0.33357057	<null>
17	17	7	1	0.5770015	<null>
18	18	12	10	0.75421757	<null>
19	19	2	7	0.07698974	<null>
20	20	6	7	0.37990457	<null>
21	21	15	1	0.305345	<null>
22	22	18	10	0.46919835	<null>
23	23	2	10	0.17780706	<null>
24	24	4	12	0.15107891	<null>
25	25	6	3	0.63272065	<null>
26	26	2	12	0.93333495	<null>
27	27	10	7	0.20252717	<null>
28	28	4	5	0.08141965	<null>
29	29	11	10	0.9950307	<null>
30	30	13	6	0.075835265	<null>
31	31	13	8	0.37953734	<null>
32	32	16	11	0.658689	<null>
33	33	10	4	0.42076445	<null>
34	34	14	7	0.32830587	<null>
35	35	2	3	0.98824877	<null>
36	36	3	4	0.49941692	<null>

	🔍 schedule_id ÷	🔍 worker_id ÷	🔍 vaccination_id ^	📅 date ÷
1	266	12	2	2023-02-13
2	576	5	2	2023-07-29
3	443	15	6	2023-09-20
4	724	10	8	2023-03-16
5	253	11	8	2023-10-21
6	408	3	9	2023-06-17
7	564	18	11	2023-03-30
8	352	1	12	2023-11-06
9	743	2	12	2023-04-17
10	989	12	14	2023-04-09
11	660	16	16	2023-08-13
12	752	15	18	2023-03-15
13	302	10	19	2023-02-15
14	965	17	20	2023-10-20
15	154	13	25	2023-10-11
16	838	16	26	2023-09-09
17	534	9	28	2023-04-21
18	498	17	31	2023-06-01
19	790	1	33	2023-01-05
20	474	14	34	2023-08-28
21	289	6	37	2023-05-05
22	907	4	38	2023-03-13
23	107	18	38	2023-04-05
24	260	6	39	2023-03-10
25	164	6	39	2023-03-21
26	637	5	41	2023-10-13
27	707	4	41	2023-09-23
28	520	10	41	2023-06-09
29	214	1	43	2023-12-24
30	867	4	44	2023-03-27
31	268	15	44	2023-12-15
32	881	12	45	2023-07-18
33	692	3	50	2023-04-11
34	471	8	51	2023-12-30
35	307	1	51	2023-07-16
36	628	9	52	2023-06-03

### Пункт №3

Choose action: 2

Need debug(true, false): true

Enter data(table name, needed column, column for join)

: vaccination\_schedule date nd

Enter data(table name, needed column, column for join)

: vaccinations\_given\_dose vaccination\_id

Enter data(table name, needed column, column for join)

:

Enter count of table: 1

Enter column: patient\_id

Enter value: 20

Enter value: 16

Enter value:

```
SELECT t1.date AS date, t2.given_dose AS given_dose FROM vaccination_schedule AS t1 JOIN
vaccinations AS t2 ON t2.vaccination_id = t1.vaccination_id WHERE t2.patient_id in (20,16)
GROUP BY t1.date, t2.given_dose
```

Execution Time: 194 ms

date	given_dose
14.07.2023 0:00:00	0,15531215
04.04.2023 0:00:00	0,43475035
27.01.2023 0:00:00	0,9169961
12.08.2023 0:00:00	0,28233188
27.02.2023 0:00:00	0,6101555
29.07.2023 0:00:00	0,51478064
30.05.2023 0:00:00	0,73663515
27.04.2023 0:00:00	0,46929094
08.06.2023 0:00:00	0,28233188

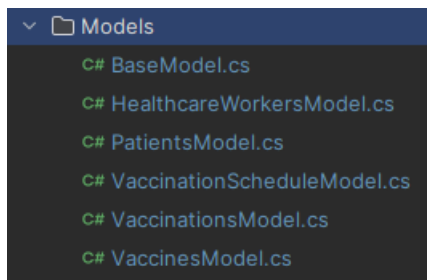
14.10.2023 0:00:00	0,26216507
08.07.2023 0:00:00	0,74469745
08.01.2023 0:00:00	0,10152554
09.07.2023 0:00:00	0,6101555
07.03.2023 0:00:00	0,31490323
23.09.2023 0:00:00	0,98138773
02.09.2023 0:00:00	0,8299163
17.01.2023 0:00:00	0,9826261
31.10.2023 0:00:00	0,048550583
25.10.2023 0:00:00	0,3063092
20.01.2023 0:00:00	0,37475297
23.02.2023 0:00:00	0,10327864
19.08.2023 0:00:00	0,80363953
09.04.2023 0:00:00	0,8712304
27.01.2023 0:00:00	0,5053786
22.04.2023 0:00:00	0,3063092
15.12.2023 0:00:00	0,74469745
12.07.2023 0:00:00	0,051271267
02.09.2023 0:00:00	0,32605198
15.09.2023 0:00:00	0,26247945
01.11.2023 0:00:00	0,98647255
09.06.2023 0:00:00	0,7705171
28.05.2023 0:00:00	0,15531215
30.01.2023 0:00:00	0,99670005
06.02.2023 0:00:00	0,85059005
04.09.2023 0:00:00	0,68810385
04.09.2023 0:00:00	0,4465328

21.03.2023 0:00:00	0,68810385
01.03.2023 0:00:00	0,32363883
24.12.2023 0:00:00	0,39435914
30.08.2023 0:00:00	0,80363953
26.02.2023 0:00:00	0,3771932
11.09.2023 0:00:00	0,99158573
21.05.2023 0:00:00	0,98138773
20.04.2023 0:00:00	0,77825445
24.01.2023 0:00:00	0,5821133
01.09.2023 0:00:00	0,470914
14.04.2023 0:00:00	0,4465328
02.09.2023 0:00:00	0,9169961
10.12.2023 0:00:00	0,5357832
09.09.2023 0:00:00	0,75403786
27.02.2023 0:00:00	0,1822983
01.07.2023 0:00:00	0,09160606
18.06.2023 0:00:00	0,46929094
14.02.2023 0:00:00	0,56803876
20.01.2023 0:00:00	0,99158573
07.06.2023 0:00:00	0,64415747
24.08.2023 0:00:00	0,6471622
21.12.2023 0:00:00	0,048550583
22.06.2023 0:00:00	0,0684005
17.10.2023 0:00:00	0,6471622
31.05.2023 0:00:00	0,07641386
02.07.2023 0:00:00	0,3259418
11.04.2023 0:00:00	0,5021166

12.05.2023 0:00:00	0,77825445
02.02.2023 0:00:00	0,5127494
29.12.2023 0:00:00	0,31490323
21.06.2023 0:00:00	0,4629884
11.01.2023 0:00:00	0,51478064
07.07.2023 0:00:00	0,26651016
10.07.2023 0:00:00	0,5708448
17.12.2023 0:00:00	0,43475035
18.11.2023 0:00:00	0,98647255
12.05.2023 0:00:00	0,7850739
07.09.2023 0:00:00	0,07641386
23.02.2023 0:00:00	0,6101555
24.08.2023 0:00:00	0,3384408
24.04.2023 0:00:00	0,17394559
27.06.2023 0:00:00	0,6471622
18.12.2023 0:00:00	0,74469745
18.07.2023 0:00:00	0,09160606
22.02.2023 0:00:00	0,1870466
02.01.2023 0:00:00	0,0684005
05.12.2023 0:00:00	0,64415747
24.04.2023 0:00:00	0,8662361
11.12.2023 0:00:00	0,98647255
15.01.2023 0:00:00	0,012270244
18.05.2023 0:00:00	0,26216507
17.04.2023 0:00:00	0,9375233
29.08.2023 0:00:00	0,31490323
13.12.2023 0:00:00	0,79720986

05.09.2023 0:00:00	0,98138773
31.10.2023 0:00:00	0,28233188
13.05.2023 0:00:00	0,73663515
05.09.2023 0:00:00	0,46929094
15.05.2023 0:00:00	0,72112906
29.07.2023 0:00:00	0,5023419
13.03.2023 0:00:00	0,011778697
08.08.2023 0:00:00	0,4257487
10.02.2023 0:00:00	0,9169961
30.01.2023 0:00:00	0,5821133
15.02.2023 0:00:00	0,1127428
06.11.2023 0:00:00	0,9375233
15.09.2023 0:00:00	0,72273886

## Пункт №4



### Опис модулів

- **BaseModel** – базова модель, яка має методи реалізацію методів дочірніх моделей. Має методи для пошуку, додавання, видалення та редагування даних таблиці.  
GetFormatValues – форматує дані у рядок потрібного вигляду.  
GetMaxId – надає найбільше значення унікального ключа.  
Find – знаходить рядок(рядки) за значенням(значеннями) стовпчика.  
Find(без параметрів) – знаходить всі рядки.  
Add – додає значення до таблиці.  
Remove – видаляє рядок за значенням.  
Remove(з булевою змінною) – видаляє рядки більші(менші) за значення.  
EditOne – редагує значення першого знайденого рядка.  
EditSome – редагує значення знайдених рядка.  
FindInTables – шукає рядки різних таблиць, об'єднуючи їх за схожими стовпчиками.
- **PatientsModel, VaccinesModel, VaccinationsModel, HealthcareWorkersModel, VaccinationScheduleModel** – дочірні моделі, які мають дані про власні таблиці та змінені методи генерування та видалення даних таблиць.  
GenerateSeries – генерує задану кількість рядків.  
Remove – змінені методи видалення базової моделі.



## BaseModel.cs

```
using System;
using System.Collections.Generic;
using Npgsql;

namespace bd_rgr
{
    public class Connection
    {
        private NpgsqlConnection connection;
        public NpgsqlCommand Cmd;

        public Connection(string host, string db, string user, string password)
        {
            connection = new NpgsqlConnection(
                connectionString: $"Server={host};Port=5432;User
Id={user};Password={password};Database={db};"
            );
            Cmd = new NpgsqlCommand();
            Cmd.Connection = connection;
        }
    }

    public abstract class BaseModel
    {
        protected readonly Connection Connection;
        public string TableName;
        public List<string> TableFields;

        public delegate void SetDateAction(ref Dictionary<string, object>
item1, ref NpgsqlDataReader item2);

        public abstract void GenerateSeries(uint count, bool debug);

        private void SetParameters(ref Dictionary<string, object> row, ref
NpgsqlDataReader reader)
        {
            for (int i = 0; i < TableFields.Count; i++)
            {
                row[TableFields[i]] = reader.GetFieldValue<object>(i);
            }
        }

        public BaseModel(Connection connection)
        {
            Connection = connection;
        }

        public static string GetFormatValues<T>(in List<T> values, string
format = ",%s", string format_start = "%s")
        {
            if (values.Count == 0) return string.Empty;
            string result = format_start.Replace("%s", values[0].ToString());
        }
    }
}
```

```

        for (int i = 1; i < values.Count; i++)
        {
            result += format.Replace("%s", values[i].ToString());
        }

        return result;
    }

    public static string GetFormatValues<T1, T2>(in List<Tuple<T1, T2>>
values, string format = ",%s1=%s2", string format_start = "%s1=%s2")
    {
        if (values.Count == 0) return string.Empty;
        string result = format_start.Replace("%s1",
values[0].Item1.ToString())
            .Replace("%s2", values[0].Item2.ToString());

        for (int i = 1; i < values.Count; i++)
        {
            result += format.Replace("%s1", values[i].Item1.ToString())
                .Replace("%s2", values[i].Item2.ToString());
        }

        return result;
    }

    public static string GetFormatValues<T1, T2, T3>(in List<Tuple<T1,
T2, T3>> values, string format = ",%s1.%s2=%s3",
        string format_start = "%s1.%s2=%s3")
    {
        if (values.Count == 0) return string.Empty;
        string result = format_start.Replace("%s1",
values[0].Item1.ToString())
            .Replace("%s2", values[0].Item2.ToString())
            .Replace("%s3", values[0].Item3.ToString());

        for (int i = 1; i < values.Count; i++)
        {
            result += format.Replace("%s1", values[i].Item1.ToString())
                .Replace("%s2", values[i].Item2.ToString())
                .Replace("%s3", values[i].Item3.ToString());
        }

        return result;
    }

    public static string GetFormatValues<T1, T2, T3, T4>(in
List<Tuple<T1, T2, T3, T4>> values, string format = ",%s1.%s2=%s3.%s4",
        string format_start = "%s1.%s2=%s3.%s4")
    {
        if (values.Count == 0) return string.Empty;
        string result = format_start.Replace("%s1",
values[0].Item1.ToString())
            .Replace("%s2", values[0].Item2.ToString())
            .Replace("%s3", values[0].Item3.ToString())
            .Replace("%s4", values[0].Item4.ToString());

        for (int i = 1; i < values.Count; i++)

```

```

        {
            result += format.Replace("%s1", values[i].Item1.ToString())
                          .Replace("%s2", values[i].Item2.ToString())
                          .Replace("%s3", values[i].Item3.ToString())
                          .Replace("%s4", values[i].Item4.ToString());
        }

        return result;
    }

    public static string GetFormatValues(in List<DateTime> values, string
format = ",%s", string format_start = "%s")
    {
        if (values.Count == 0) return string.Empty;
        string result = format_start.Replace("%s",
values[0].ToString("yyyy-MM-dd"));

        for (int i = 1; i < values.Count; i++)
        {
            result += format.Replace("%s", values[i].ToString("yyyy-MM-
dd"));
        }

        return result;
    }

    public static string GetFieldsFormat(in Dictionary<string, object>
data, in List<string> fields,
        string format = ",%s", string format_start = "%s")
    {
        if (fields.Count == 0) return string.Empty;
        string result = format_start.Replace("%s",
data[fields[0]].ToString());

        for (int i = 1; i < fields.Count; i++)
        {
            result += format.Replace("%s", data[fields[i]].ToString());
        }

        return result;
    }

    protected int GetMaxId()
    {
        int id = 0;
        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"SELECT MAX({TableFields[0]})
FROM {TableName}";
            var reader = Connection.Cmd.ExecuteReader();
            while (reader.Read())
            {
                id = reader.GetFieldValue<int>(0);
            }
        }
        catch (Npgsql.PostgresException er)

```

```

        {
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        catch (InvalidCastException er)
        {
            // Console.WriteLine($"{er.Message}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }

        return id;
    }

    public Dictionary<string, object> FindOne<T>(string column, T value)
    {
        Dictionary<string, object> result = new Dictionary<string,
object>();

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {column}='{value}'";
            var reader = Connection.Cmd.ExecuteReader();
            while (reader.Read())
            {
                SetParameters(ref result, ref reader);
            }
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"Given data: column({column})
value({value})");
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }

        return result;
    }

    public List<Dictionary<string, object>> FindSome<T>(string column, in
List<T> values)
    {
        List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();
        string str_val = GetFormatValues(in values);

        try
        {
            Connection.Cmd.Connection.Open();

```

```

        Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {column} IN ({str_val})";
        var reader = Connection.Cmd.ExecuteReader();
        while (reader.Read())
        {
            Dictionary<string, object> patient = new
Dictionary<string, object>();
            SetParameters(ref patient, ref reader);
            result.Add(patient);
        }
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"Given data: column({column})
values({str_val})");
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }

    return result;
}

public List<Dictionary<string, object>> FindSome<T>(string column, T
value, bool greater)
{
    List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();
    string sign = "<";
    if (greater) sign = ">";
    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {column} {sign} {value.ToString()}";
        var reader = Connection.Cmd.ExecuteReader();
        while (reader.Read())
        {
            Dictionary<string, object> patient = new
Dictionary<string, object>();
            SetParameters(ref patient, ref reader);
            result.Add(patient);
        }
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}

```

```

        return result;
    }

    public List<Dictionary<string, object>> FindSome(string column, in
List<string> values)
    {
        List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();
        string str_val = GetFormatValues(in values, $" OR {column} LIKE
'%s'", $"{column} LIKE '%s'");

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {str_val}";
            var reader = Connection.Cmd.ExecuteReader();
            while (reader.Read())
            {
                Dictionary<string, object> patient = new
Dictionary<string, object>();
                SetParameters(ref patient, ref reader);
                result.Add(patient);
            }
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"Given data: column({column})
values({str_val})");
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }

        return result;
    }

    public List<Dictionary<string, object>> FindSome(string column, bool
value)
    {
        List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {column}='{value}'";
            var reader = Connection.Cmd.ExecuteReader();
            while (reader.Read())
            {
                Dictionary<string, object> patient = new
Dictionary<string, object>();
                SetParameters(ref patient, ref reader);

```

```

        result.Add(patient);
    }
}
catch (Npgsql.PostgresException er)
{
    Console.WriteLine($"Given data: column({column})
value({value})");
    Console.WriteLine($"{er.MessageText}");
    Console.WriteLine($"{er.Hint}");
}
finally
{
    Connection.Cmd.Connection.Close();
}

return result;
}

public List<Dictionary<string, object>> FindSome(string column, in
List<DateTime> values)
{
    List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();
    string str_val = GetFormatValues(in values);

    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = $"SELECT * FROM {TableName}
WHERE {column} IN ({str_val})";
        var reader = Connection.Cmd.ExecuteReader();
        while (reader.Read())
        {
            Dictionary<string, object> patient = new
Dictionary<string, object>();
            SetParameters(ref patient, ref reader);
            result.Add(patient);
        }
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"Given data: column({column})
values({str_val})");
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }

    return result;
}

public List<Dictionary<string, object>> FindAll()
{
    List<Dictionary<string, object>> result = new

```

```

List<Dictionary<string, object>>();

    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = $"SELECT * FROM {TableName}";
        var reader = Connection.Cmd.ExecuteReader();
        while (reader.Read())
        {
            Dictionary<string, object> patient = new
Dictionary<string, object>();
            SetParameters(ref patient, ref reader);
            result.Add(patient);
        }
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }

    return result;
}

public void AddOne(Dictionary<string, object> data)
{
    var fields = GetFormatValues(TableFields);
    var values = GetFieldsFormat(in data, in TableFields, ", 's'",
"'%s'");

    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = $"INSERT INTO
{TableName}({fields})" +
                                $" VALUES ({values})";
        Connection.Cmd.ExecuteNonQuery();
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}

public void AddSome(List<Dictionary<string, object>> data)
{
    var fields = GetFormatValues(TableFields);

```



```

        foreach (var patient in data)
        {
            var values = GetFieldsFormat(in patient, in TableFields, ",
'%s'", "%s'");

            try
            {
                Connection.Cmd.Connection.Open();
                Connection.Cmd.CommandText = $"INSERT INTO
{TableName}({fields})" +
                                $" VALUES ({fields})";
                Connection.Cmd.ExecuteNonQuery();
            }
            catch (Npgsql.PostgresException er)
            {
                Console.WriteLine($"{er.MessageText}");
                Console.WriteLine($"{er.Hint}");
            }
            finally
            {
                Connection.Cmd.Connection.Close();
            }
        }
    }

    public virtual void RemoveSome<T>(string column, T value)
    {
        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"DELETE FROM {TableName} WHERE
{column}='{value}'";
            Connection.Cmd.ExecuteNonQuery();
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"Given data: column({column})
value({value})");
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }
    }

    public virtual void RemoveSome<T>(string column, T value, bool
greater)
    {
        string sign = "<";
        if (greater) sign = ">";
        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"DELETE FROM {TableName} WHERE
{column}{sign}'{value}'";

```

```

        Connection.Cmd.ExecuteNonQuery();
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"Given data: column({column})
value({value})");
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}

public void EditOne<T1, T2>(string column, T1 value, List<string>
eColumn, List<T2> newValue)
{
    var patient = FindOne(column, value);

    if(!(eColumn.Count > 0 && eColumn.Count == newValue.Count))
        return;
    string sets = "";
    int i;
    for(i = 0; i < eColumn.Count-1; i++)
    {
        sets += $"{eColumn[i]}='{newValue[i].ToString()}', ";
    }
    sets += $"{eColumn[i]}='{newValue[i].ToString()}'";

    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = $"UPDATE {TableName} " +
            $"SET {sets} " +
            $"WHERE
{TableFields[0]}={patient[TableFields[0]]}";
        Connection.Cmd.ExecuteNonQuery();
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"column({column}) value({value})");
        Console.WriteLine($"sets({sets})");
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}

public void EditSome<T1, T2>(string column, T1 value, List<string>
eColumn, List<T2> newValue)
{
    if(!(eColumn.Count > 0 && eColumn.Count == newValue.Count))
        return;

```

```

        string sets = "";
        int i;
        for(i = 0; i < eColumn.Count-1; i++)
        {
            sets += $"{eColumn[i]}='{newValue[i].ToString()}', ";
        }
        sets += $"{eColumn[i]}='{newValue[i].ToString()}'";

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = $"UPDATE {TableName} " +
                $"SET {sets} " +
                $"WHERE {column}='{value}'";
            Connection.Cmd.ExecuteNonQuery();
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"Given data: column({column})
value({value})");
            Console.WriteLine($"sets({sets})");
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="data">List of data of tables (table name, needed
column, column for join)</param>
    /// <param name="where">List of data for WHERE (count of table in
data, column name, list of values)</param>
    /// <typeparam name="T">The type of values for WHERE</typeparam>
    /// <returns></returns>
    public List<Dictionary<string, object>>
FindSeveralTables<T>(List<Tuple<string, string, string>> data,
    Tuple<int, string, List<T>> where, bool debug = false)
    {
        var watch = new System.Diagnostics.Stopwatch();
        watch.Start();

        var select = new List<Tuple<string, string>>();
        var joins = new List<Tuple<string, string, string>>();
        var group_by = new List<Tuple<string, string>>();

        {
            int count = 0;
            foreach (var item in data)
            {
                select.Add(new Tuple<string, string>($"t{count+1}",
item.Item2));
                var joins_t = $"t{count}";

```

```

        if (count == 0) joins_t = "";
        joins.Add(new Tuple<string, string, string,
string>(item.Item1, $"t{count+1}", item.Item3, joins_t));
        group_by.Add(new Tuple<string, string>($"t{count+1}",
item.Item2));
        count++;
    }
}

List<Dictionary<string, object>> result = new
List<Dictionary<string, object>>();
var select_str = GetFormatValues(select, ", %s1.%s2 AS %s2",
"%s1.%s2 AS %s2");
var from = GetFormatValues(joins, " JOIN %s1 AS %s2 ON %s2.%s3 =
%s4.%s3", "%s1 AS %s2");
var where_values = GetFormatValues(where.Item3);
var str_groups = GetFormatValues(group_by, ", %s1.%s2",
"%s1.%s2");

var request = $"SELECT {select_str} " +
    $"FROM {from} " +
    $"WHERE {joins[where.Item1].Item2}.{where.Item2} in
({where_values}) " +
    $"GROUP BY {str_groups}";

if (debug)
    Console.WriteLine(request);

try
{
    Connection.Cmd.Connection.Open();
    Connection.Cmd.CommandText = request;
    var reader = Connection.Cmd.ExecuteReader();
    while (reader.Read())
    {
        Dictionary<string, object> row = new Dictionary<string,
object>();

        int i = 0;
        foreach (var el in select)
        {
            row[el.Item2] = reader.GetFieldValue<object>(i);
            i++;
        }
        result.Add(row);
    }
}
catch (Npgsql.PostgresException er)
{
    Console.WriteLine($"{{er.MessageText}}");
    Console.WriteLine($"{{er.Hint}}");
}
finally
{
    Connection.Cmd.Connection.Close();
}

watch.Stop();

```

```

        Console.WriteLine($"Execution Time: {watch.ElapsedMilliseconds}
ms");
        return result;
    }
}

```

## PatientsModel.cs

```

using System;
using System.Collections.Generic;

namespace bd_rgr
{
    public class PatientsModel : BaseModel
    {
        public PatientsModel(Connection connection) : base(connection)
        {
            TableName = "patients";
            TableFields = new List<string>()
            {
                "patient_id", "name", "address", "phone_number"
            };
        }

        public override void GenerateSeries(uint count, bool debug)
        {
            int max_id = GetMaxId();

            var command = $"INSERT INTO patients\n" +
                $"SELECT DISTINCT * FROM (SELECT generate_series AS
patient_id, CHR(TRUNC(65+RANDOM()*25)::int)" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"AS name, CHR(TRUNC(65+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"AS address, '+' ||

```

```

CHR(TRUNC(48+RANDOM()*10)::int)\n" +
        $"|| CHR(TRUNC(48+RANDOM()*10)::int) ||
CHR(TRUNC(48+RANDOM()*10)::int)\n" +
        $"|| CHR(TRUNC(48+RANDOM()*10)::int) ||
CHR(TRUNC(48+RANDOM()*10)::int)\n" +
        $"|| CHR(TRUNC(48+RANDOM()*10)::int)\n" +
        $"AS phone_number FROM generate_series({max_id+1},
{max_id + count})) AS t1\n" +
        $"GROUP BY t1.patient_id, t1.name, t1.address,
t1.phone_number";

        if(debug)
            Console.WriteLine(command);

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = command;
            Connection.Cmd.ExecuteNonQuery();
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }
    }

    public override void RemoveSome<T>(string column, T value)
    {
        var list = FindSome(column, new List<T>() { value });
        var model = new VaccinationsModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value);
    }

    public override void RemoveSome<T>(string column, T value, bool
greater)
    {
        var list = FindSome(column, value, greater);
        var model = new VaccinationsModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value, greater);
    }
}
}

```

## VaccinesModel.cs

```
using System;
using System.Collections.Generic;

namespace bd_rgr
{
    public class VaccinesModel : BaseModel
    {
        public VaccinesModel(Connection connection) : base(connection)
        {
            TableName = "vaccines";
            TableFields = new List<string>()
            {
                "vaccine_id", "name", "manufacturer", "type", "dosage"
            };
        }

        public override void GenerateSeries(uint count, bool debug)
        {
            int max_id = GetMaxId();

            var command = $"INSERT INTO vaccines\n" +
                $"SELECT DISTINCT * FROM (SELECT generate_series AS  

vaccine_id, CHR(TRUNC(65+RANDOM()*25)::int) +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"AS name, CHR(TRUNC(65+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"AS manufact, CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||  

CHR(TRUNC(97+RANDOM()*25)::int)\n" +  

                $"AS type, TRUNC(RANDOM()*99.99 + 0.01)::float4 AS
```

```

dosage " +
                                $"FROM GENERATE_SERIES({max_id + 1}, {max_id +
count})) AS t1\n" +
                                $"GROUP BY t1.vaccine_id, t1.name, t1.manufact,
t1.type, t1.dosage";

        if (debug)
            Console.WriteLine (command);

        try
        {
            Connection.Cmd.Connection.Open();
            Connection.Cmd.CommandText = command;
            Connection.Cmd.ExecuteNonQuery();
        }
        catch (Npgsql.PostgresException er)
        {
            Console.WriteLine($"{er.MessageText}");
            Console.WriteLine($"{er.Hint}");
        }
        finally
        {
            Connection.Cmd.Connection.Close();
        }
    }

    public override void RemoveSome<T>(string column, T value)
    {
        var list = FindSome(column, new List<T>() { value });
        var model = new VaccinationsModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value);
    }

    public override void RemoveSome<T>(string column, T value, bool
greater)
    {
        var list = FindSome(column, value, greater);
        var model = new VaccinationsModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value, greater);
    }
}

```

VaccinationsModel.cs



```

using System;
using System.Collections.Generic;

namespace bd_rgr
{
    public class VaccinationsModel : BaseModel
    {
        public VaccinationsModel(Connection connection) : base(connection)
        {
            TableName = "vaccinations";
            TableFields = new List<string>()
            {
                "vaccination_id", "patient_id", "vaccine_id", "given_dose",
"notes"
            };
        }

        public override void GenerateSeries(uint count, bool debug)
        {
            int max_id = GetMaxId();

            var command = $"INSERT INTO vaccinations\n" +
                $"SELECT DISTINCT * FROM (SELECT generate_series AS
vaccination_id," +
                $"TRUNC(RANDOM() * (SELECT MAX(patient_id) FROM
patients))::int + 1 AS patient_id," +
                $"TRUNC(RANDOM() * (SELECT MAX(vaccine_id) FROM
vaccines))::int + 1 AS vaccine_id," +
                $"float4(RANDOM() * 0.99 + 0.01) AS dosage " +
                $"FROM GENERATE_SERIES({max_id + 1}, {max_id +
count})) AS t1\n" +
                $"GROUP BY t1.vaccination_id, t1.patient_id,
t1.vaccine_id, t1.dosage";

            if (debug)
            {
                Console.WriteLine(command);
            }

            try
            {
                Connection.Cmd.Connection.Open();
                Connection.Cmd.CommandText = command;
                Connection.Cmd.ExecuteNonQuery();
            }
            catch (Npgsql.PostgresException er)
            {
                Console.WriteLine($"{er.MessageText}");
                Console.WriteLine($"{er.Hint}");
            }
            finally
            {
                Connection.Cmd.Connection.Close();
            }
        }

        public override void RemoveSome<T>(string column, T value)
        {
            var list = FindSome(column, new List<T>() { value });

```

```

        var model = new VaccinationScheduleModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value);
    }

    public override void RemoveSome<T>(string column, T value, bool
greater)
    {
        var list = FindSome(column, value, greater);
        var model = new VaccinationScheduleModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value, greater);
    }
}
}

```

## HealthcareWorkersModel.cs

```

using System;
using System.Collections.Generic;

namespace bd_rgr
{
    public class HealthcareWorkersModel : BaseModel
    {
        public HealthcareWorkersModel(Connection connection) :
base(connection)
        {
            TableName = "healthcare_workers";
            TableFields = new List<string>()
            {
                "worker_id", "name", "specialization",
"medical_license_number"
            };
        }

        public override void GenerateSeries(uint count, bool debug)
        {
            int max_id = GetMaxId();

            var command = $"INSERT INTO healthcare_workers\n" +
                $"SELECT DISTINCT * FROM (SELECT generate_series AS
worker_id, CHR(TRUNC(65+RANDOM()*25)::int)" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
                $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||

```

```

CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"AS name, CHR(TRUNC(65+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"|| CHR(TRUNC(97+RANDOM()*25)::int) ||
CHR(TRUNC(97+RANDOM()*25)::int)\n" +
    $"AS spec, TRUNC(RANDOM()*999999999999999)::int8 AS
license" +
    $"FROM generate_series({max_id + 1}, {max_id +
count})) AS t1\n" +
    $"GROUP BY t1.worker_id, t1.name, t1.spec,
t1.license";

    if(debug)
        Console.WriteLine(command);

    try
    {
        Connection.Cmd.Connection.Open();
        Connection.Cmd.CommandText = command;
        Connection.Cmd.ExecuteNonQuery();
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}

public override void RemoveSome<T>(string column, T value)
{
    var list = FindSome(column, new List<T>() { value });
    var model = new VaccinationScheduleModel(Connection);
    foreach (var item in list)
    {
        model.RemoveSome(TableFields[0], item[TableFields[0]]);
    }
    base.RemoveSome(column, value);
}

public override void RemoveSome<T>(string column, T value, bool
greater)
{
    var list = FindSome(column, value, greater);

```

```

        var model = new VaccinationScheduleModel(Connection);
        foreach (var item in list)
        {
            model.RemoveSome(TableFields[0], item[TableFields[0]]);
        }
        base.RemoveSome(column, value, greater);
    }
}

```

## VaccinationScheduleMode.cs

```

using System;
using System.Collections.Generic;

namespace bd_rgr
{
    public class VaccinationScheduleModel : BaseModel
    {
        public VaccinationScheduleModel(Connection connection) :
        base(connection)
        {
            TableName = "vaccination_schedule";
            TableFields = new List<string>()
            {
                "schedule_id", "worker_id", "vaccination_id", "date"
            };
        }

        public override void GenerateSeries(uint count, bool debug)
        {
            int max_id = GetMaxId();

            var command = $"INSERT INTO vaccination_schedule\n" +
                $"SELECT DISTINCT * FROM (SELECT generate_series AS
schedule_id," +
                $"TRUNC(RANDOM()*(SELECT MAX(worker_id) FROM
healthcare_workers))::int + 1 AS worker_id," +
                $"TRUNC(RANDOM()*(SELECT MAX(vaccination_id) FROM
vaccinations))::int + 1 AS vaccination_id," +
                $"DATE('2023-01-01') + TRUNC(RANDOM()*(DATE('2023-
12-31') - DATE('2023-01-01'))::int AS date " +
                $"FROM generate_series({max_id + 1}, {max_id +
count})) AS t1\n" +
                $"GROUP BY t1.schedule_id, t1.worker_id,
t1.vaccination_id, t1.date";

            if(debug)
                Console.WriteLine(command);

            try
            {
                Connection.Cmd.Connection.Open();
            }
        }
    }
}

```

```
        Connection.Cmd.CommandText = command;
        Connection.Cmd.ExecuteNonQuery();
    }
    catch (Npgsql.PostgresException er)
    {
        Console.WriteLine($"{er.MessageText}");
        Console.WriteLine($"{er.Hint}");
    }
    finally
    {
        Connection.Cmd.Connection.Close();
    }
}
}
```