НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2

з дисципліни

«Бази даних і засоби управління»

Виконав: студент групи КВ-13

Шпилька І.В.

Перевірив: Петрашенко А.В.

Київ – 2023

Мета: здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання за варіантом:

23	*GIN, Hash*	*before update, delete*

Виконання роботи

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 1.

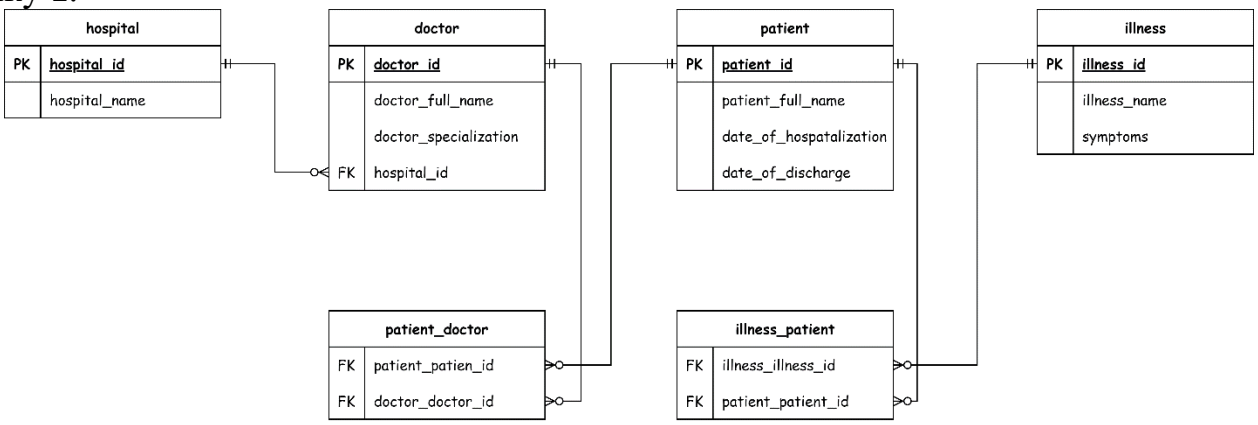


Рисунок 1 – Логічна модель

Відповідні класи ORM утворюють зв’язки що можна зобразити чином показаним на рисунку 2.

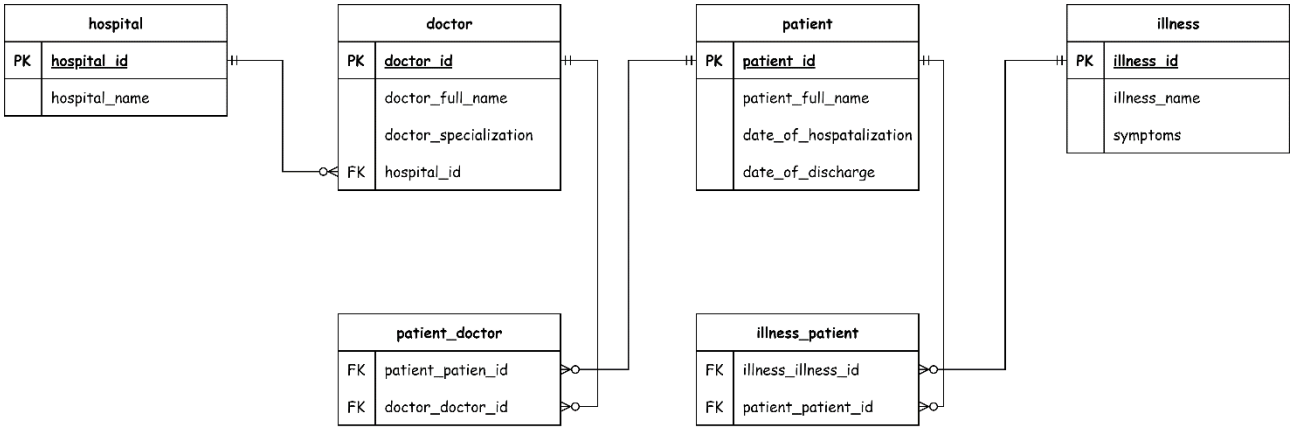


Рисунок 2 – Класи ORM

Запит на вставку має наступний вигляд:

```
public void Execute(
        string tableName,
        IEnumerable<string> tableColumns,
        IEnumerable<object> values)
    {
        _connection.Open();

        try
        {
            List<string> valueStrings = new List<string>();

            for (int i = 0; i < values.Count(); i++)
            {
                _command.Parameters.AddWithValue($"@param{i}", values.ToList()[i]);
                valueStrings.Add($"@param{i}");
            }

            string valuesString =
Utility.Utility.AggregateWithCommas(valueStrings);
            string tableColumnsString =
Utility.Utility.AggregateWithCommas(tableColumns);

            _command.CommandText = $"insert into {tableName}({tableColumnsString})
values({valuesString});";

            _command.ExecuteNonQuery();
        }
        finally { _connection.Close(); }
    }
```

Запит на оновлення:

```
public void Execute(
        string tableName,
        IEnumerable<string> tableColumns,
        IEnumerable<object> values,
        string primaryKeyColumnString,
        int id)
    {
        _connection.Open();

        try
        {
            List<string> valueStrings = new List<string>();

            for (int i = 0; i < values.Count(); i++)
            {
                _command.Parameters.AddWithValue($"@param{i}", values.ToList()[i]);
                valueStrings.Add($"@param{i}");
            }

            string setString = Utility.Utility.ConvertToSetString(tableColumns,
valueStrings);

            _command.CommandText = $"update {tableName} set {setString} where
{primaryKeyColumnString} = {id};";

            _command.ExecuteNonQuery();
        }
        finally {  _connection.Close(); }
    }
```

Запит на видалення:

```
public void Execute(
        string tableName,
        string primaryKeyColumnString,
        int id)
    {
        _connection.Open();

        try
        {
            _command.CommandText = $"delete from {tableName} where
{primaryKeyColumnString} = {id};";

            _command.ExecuteNonQuery();
        }
        finally { _connection.Close(); }
    }
```

Код зміненого модуля Model:

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class DoctorModel : Model
{
    public DoctorModel(NpgsqlConnection connection) : base(
        connection,
        "doctor",
        "doctor_id",
        [
            "doctor_full_name",
            "doctor_specialiazation",
            "hospital_id"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.String(),
            GenerateCommand.ForeignKey("hospital", "hospital_id")
        ])
    {
    }

    public List<List<object>> ReadAllInfo(int firstBound, int secondBound, out
List<string> columnName)
    {
        var result = _context.Doctors
```

```
            .Where(d => d.DoctorId >= firstBound && d.DoctorId <= secondBound)
            .Select(d => new
            {
                DoctorId = d.DoctorId,
                DoctorFullName = d.DoctorFullName,
                DoctorSpecialization = d.DoctorSpecialization,
                HospitalName = d.HospitalName,
                PatientCount = d.PatientDoctors.Count
            })
            .OrderBy(d => d.DoctorId)
            .ToList();
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class HospitalModel : Model
{
    public HospitalModel(NpgsqlConnection connection) : base(
        connection,
        "hospital",
        "hospital_id",
        [
            "hospital_name"
        ],
        [
            GenerateCommand.String(),
        ])
    {
    }

    public List<List<object>> ReadAllInfo(int firstBound, int secondBound, out
List<string> columnName)
    {
        var result = _context.Hospitals
            .GroupBy(h => new { h.HospitalId, h.HospitalName })
            .Select(g => new
            {
                HospitalId = g.Key.HospitalId,
                HospitalName = g.Key.HospitalName,
                DoctorCount = g.SelectMany(h => h.Doctors).Count()
            })
            .Where(h => h.DoctorCount >= firstBound && h.DoctorCount <=
secondBound)
            .OrderBy(h => h.HospitalId)
            .ToList();
```

```
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

```
using Npgsql;
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class IllnessModel : Model
{
    public IllnessModel(NpgsqlConnection connection) : base(
        connection,
        "illness",
        "illness_id",
        [
            "illness_name",
            "symptoms"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.String(),
        ])
    {
    }

    public List<List<object>> ReadFullInfo(string likeExp, out List<string>
columnName)
    {
        var result = _context.Illnesses
            .Where(i => EF.Functions.Like(i.IllnessName, likeExp))
            .GroupBy(i => new { i.IllnessId, i.IllnessName, i.Symptoms })
            .Select(g => new
            {
                IllnessId = g.Key.IllnessId,
                IllnessName = g.Key.IllnessName,
                Symptoms = g.Key.Symptoms,
                PatientCount = g.SelectMany(i => i.IllnessPatients).Count()
            })
            .OrderBy(i => i.IllnessId)
            .ToList();
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

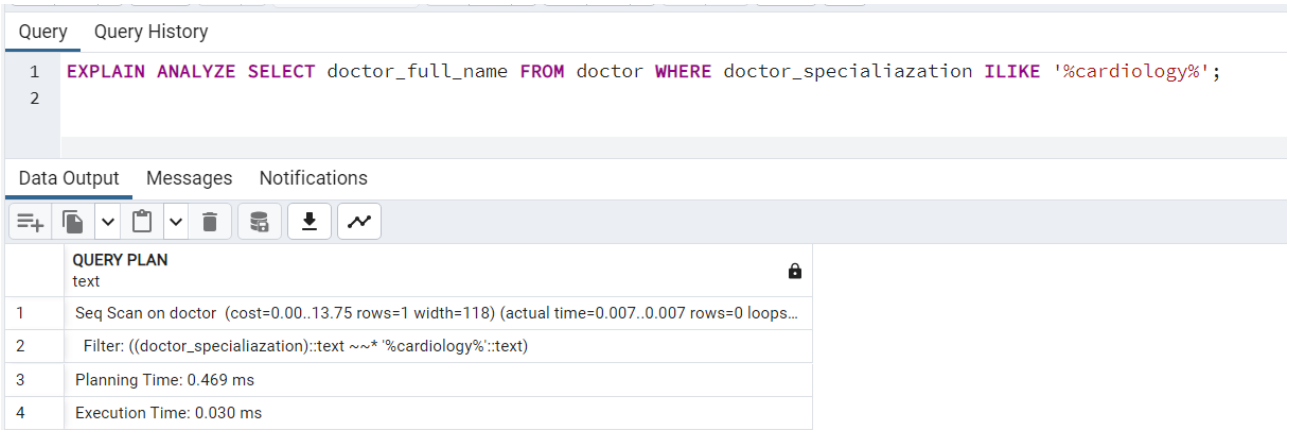
```
using Npgsql;
```

```
using Rgr.Models.Commands;

namespace Rgr.Models.Models;

public sealed class PatientModel : Model
{
    public PatientModel(NpgsqlConnection connection) : base(
        connection,
        "patient",
        "patient_id",
        [
            "patient_full_name",
            "date_of_hospitalization",
            "date_of_discharge"
        ],
        [
            GenerateCommand.String(),
            GenerateCommand.DateOnly(),
            GenerateCommand.DateOnly(),
        ])
    {
    }

    public List<List<object>> ReadFullInfo(string likeExp, out List<string>
columnName)
    {
        var result = _context.Illness
            .Where(i => EF.Functions.ILike(i.IllnessName, $"%{likeExp}%"))
            .GroupJoin(_context.IllnessPatient,
                illness => illness.IllnessId,
                illnessPatient => illnessPatient.IllnessId,
                (illness, illnessPatient) => new { Illness = illness, PatientCount
= illnessPatient.Count() })
            .Select(i => new
            {
                IllnessId = i.Illness.IllnessId,
                IllnessName = i.Illness.IllnessName,
                Symptoms = i.Illness.Symptoms,
                PatientCount = i.PatientCount
            })
            .OrderBy(i => i.IllnessId)
            .ToList();
        columnName = _customCommand.ColumnNames;
        return result;
    }
}
```

Створення індексів:

GIN

Запит без індексу



Так як GIN використовується для масивів – створимо нову колонку яка буде підходити для індексу

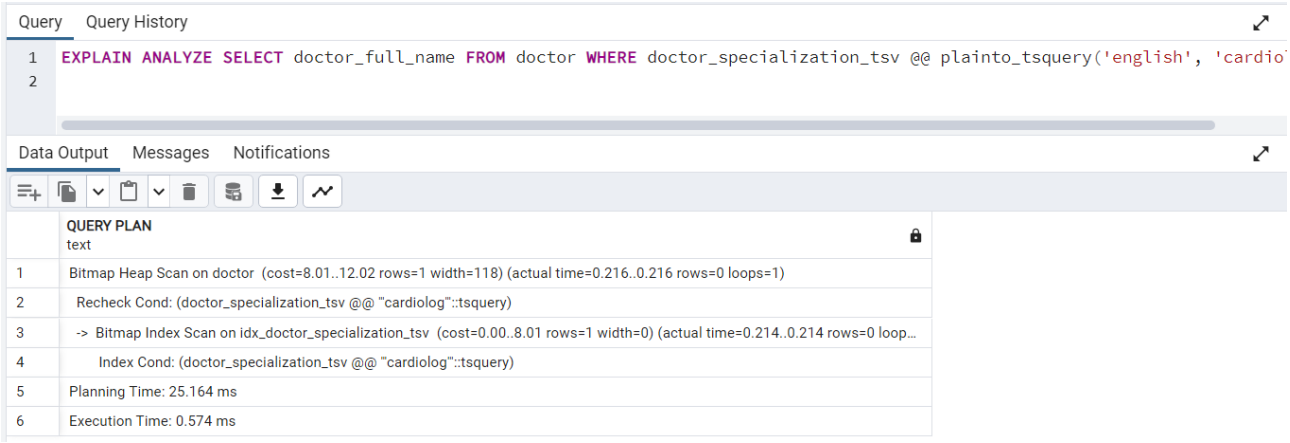
ALTER TABLE doctor ADD COLUMN doctor_specialization_tsv tsvector;

UPDATE doctor SET doctor_specialization_tsv = to_tsvector('english', doctor_specialization);

Створимо індекс

CREATE INDEX idx_doctor_specialization_tsv ON doctor USING GIN (doctor_specialization_tsv);

Запит з індексом



Hash

Запит без індексу

Query Query History

```
1  EXPLAIN ANALYZE SELECT * FROM hospital WHERE hospital_name = 'City Hospital';
2
```

Data Output Messages Notifications

	QUERY PLAN text
1	Seq Scan on hospital (cost=0.00..16.75 rows=3 width=122) (actual time=0.008..0.008 rows=0 loops...
2	Filter: ((hospital_name)::text = 'City Hospital'::text)
3	Planning Time: 6.397 ms
4	Execution Time: 0.021 ms

Створимо індекс

CREATE INDEX idx_hospital_name ON public.hospital USING HASH (hospital_name);

Запит з індексом

Data Output Messages Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on hospital (cost=4.02..11.13 rows=3 width=122) (actual time=0.023..0.024 rows=0 loops=1)
2	Recheck Cond: ((hospital_name)::text = 'City Hospital'::text)
3	-> Bitmap Index Scan on idx_hospital_name (cost=0.00..4.02 rows=3 width=0) (actual time=0.021..0.021 rows=0 loop...
4	Index Cond: ((hospital_name)::text = 'City Hospital'::text)
5	Planning Time: 0.420 ms
6	Execution Time: 0.054 ms

Розробка тригерів

Код тригера:

```
CREATE OR REPLACE FUNCTION patient_update_delete_trigger()
RETURNS TRIGGER AS $$
DECLARE
    cur_patient CURSOR FOR SELECT date_of_hospitalization FROM patient WHERE
patient_id = NEW.patient_id;
    patient_row RECORD;
BEGIN
    -- Cursor loop to iterate over patient records
    FOR patient_row IN cur_patient LOOP
        -- Conditional statement to check if date_of_hospitalization is in the
future
        IF patient_row.date_of_hospitalization > CURRENT_DATE THEN
            RAISE EXCEPTION 'Hospitalization date cannot be in the future.';
        END IF;
    END LOOP;

    -- Proceed with the update or delete
    RETURN NEW;
EXCEPTION
    WHEN OTHERS THEN
        -- Exception handling
        RAISE NOTICE 'An error occurred: %', SQLERRM;
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

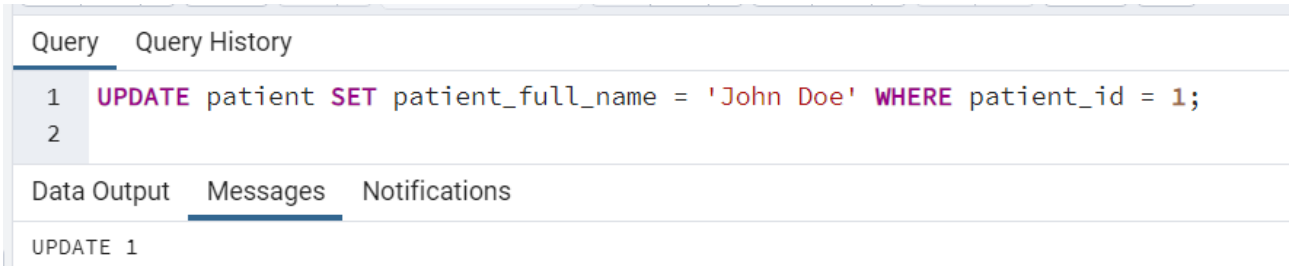
Даний тригер викликає повідомлення про некоректну дату госпіталізації при спробі оновлення чи видалення відповідного запису

Прикріплення тригеру до відповідної таблиці:

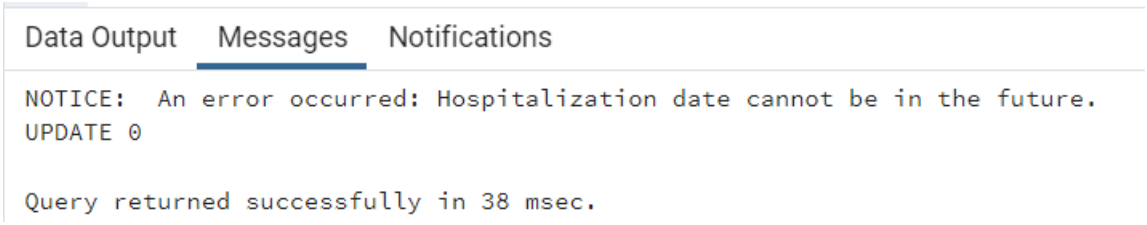
```
CREATE TRIGGER patient_before_update_delete
BEFORE UPDATE OR DELETE ON patient
FOR EACH ROW EXECUTE FUNCTION patient_update_delete_trigger();
```

Тестування тригеру:

- Зміна запису що не відповідає умовам тригеру не має викликати помилку, що і спостерігається



- Зміна запису що задовольняє умови тригеру



Використання рівнів ізоляції

1. READ COMMITTED

Вікно А

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
Lab1=*# SELECT * FROM patient WHERE patient_id = 1;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | John Doe          | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# SELECT * FROM patient WHERE patient_id = 1;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | John Hamster      | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# COMMIT;
COMMIT
Lab1=#
```

Вікно Б

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
Lab1=*# UPDATE patient SET patient_full_name = 'John Hamster' WHERE
patient_id = 1;
UPDATE 1
Lab1=*# COMMIT;
COMMIT
Lab1=#
```

2. REPEATABLE READ

Вікно А

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
Lab1=*# SELECT * FROM patient WHERE patient_id = 1;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | John Hamster      | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# SELECT * FROM patient WHERE patient_id = 1;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | John Hamster      | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# COMMIT;
COMMIT
Lab1=#
```

Вікно Б

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
Lab1=*# UPDATE patient SET patient_full_name = 'Jane Doe' WHERE patient_id =
1;
UPDATE 1
Lab1=*# COMMIT;
COMMIT
Lab1=#
```

3. SERIALIZABLE

Вікно А

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
Lab1=*# SELECT * FROM patient;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | Jane Doe          | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# SELECT * FROM patient;
 patient_id | patient_full_name | date_of_hospitalization | date_of_discharge
------------+-------------------+-------------------------+-------------------
          1 | Jane Doe          | 2023-01-01              | 2023-12-23
(1 row)


Lab1=*# COMMIT;
COMMIT
Lab1=#
```

Вікно Б

```
Lab1=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
Lab1=*# INSERT INTO patient (patient_id, patient_full_name,
date_of_hospitalization, date_of_discharge) VALUES (123, 'New
 Patient', '2023-01-01', '2023-01-10');
INSERT 0 1
Lab1=*# COMMIT;
COMMIT
Lab1=#
```

Опис всіх рівнів ізоляції відносно наведених прикладів

1. READ COMMITTED

- **Вікно А**: Почалася транзакція з рівнем ізоляції READ COMMITTED. Перший запит SELECT показав ім'я пацієнта як "John Doe".
- **Вікно Б**: Почалася транзакція, і було виконано оновлення імені пацієнта на "John Hamster", після чого транзакція була завершена.
- **Вікно А**: Другий запит SELECT показав змінене ім'я "John Hamster", оскільки в рівні READ COMMITTED нові дані, що з'явилися після початку транзакції, видимі іншим транзакціям. Це приклад явища "неповторюване читання" (non-repeatable read).

2. REPEATABLE READ

- **Вікно А**: Почалася транзакція з рівнем ізоляції REPEATABLE READ. Запит **SELECT** показав ім'я пацієнта як "John Hamster".
- **Вікно Б**: Почалася транзакція, і було виконано оновлення імені пацієнта на "Jane Doe", після чого транзакція була завершена.
- **Вікно А**: Другий запит SELECT знову показав ім'я "John Hamster", не дивлячись на зміну в іншій транзакції. Це тому, що REPEATABLE READ гарантує стабільність даних протягом усієї транзакції і запобігає явищу "неповторюване читання".

3. SERIALIZABLE

- **Вікно А**: Почалася транзакція з рівнем ізоляції SERIALIZABLE. Запит **SELECT** показав ім'я пацієнта як "Jane Doe".
- **Вікно Б**: Почалася транзакція, в якій було вставлено новий запис про пацієнта, після чого транзакція була завершена.
- **Вікно А**: Другий запит SELECT знову показав ім'я "Jane Doe". На рівні SERIALIZABLE нові записи, що з'явилися після початку транзакції, не видимі іншим транзакціям до їх завершення. Цей рівень ізоляції запобігає явищам "неповторюване читання" та "фантомне читання" (phantom read).

Контакти:

1. Github: https://github.com/IhorShpilka/BD_labs
2. Telegram: https://t.me/ihorshpilka