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**

на тему: "Засоби оптимізації роботи СУБД PostgreSQL"

Виконав:
студент III курсу
групи КВ-23
Марінченко М.О.

Київ – 2024

Мета: здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання за варіантом:

17	*GIN, BRIN*	*before update, delete*

Виконання роботи

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

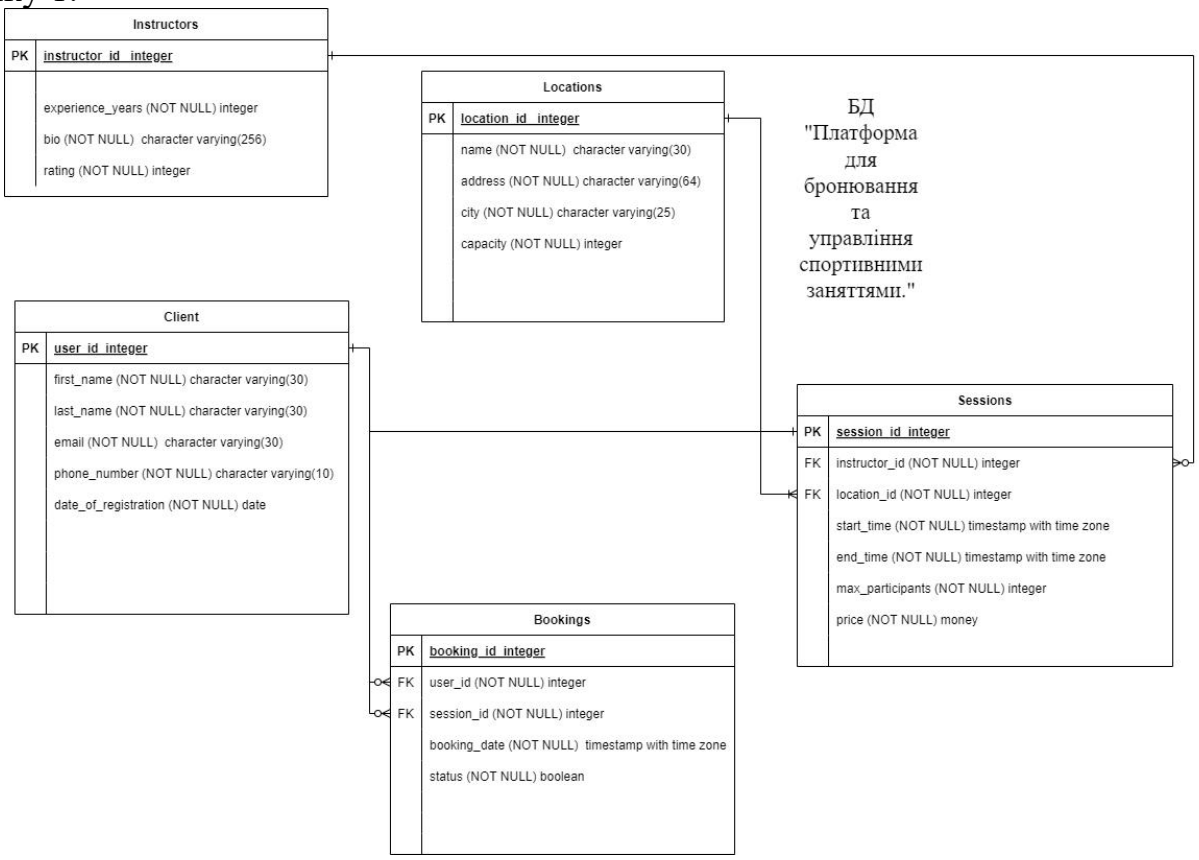


Рисунок 1 – Логічна модель

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

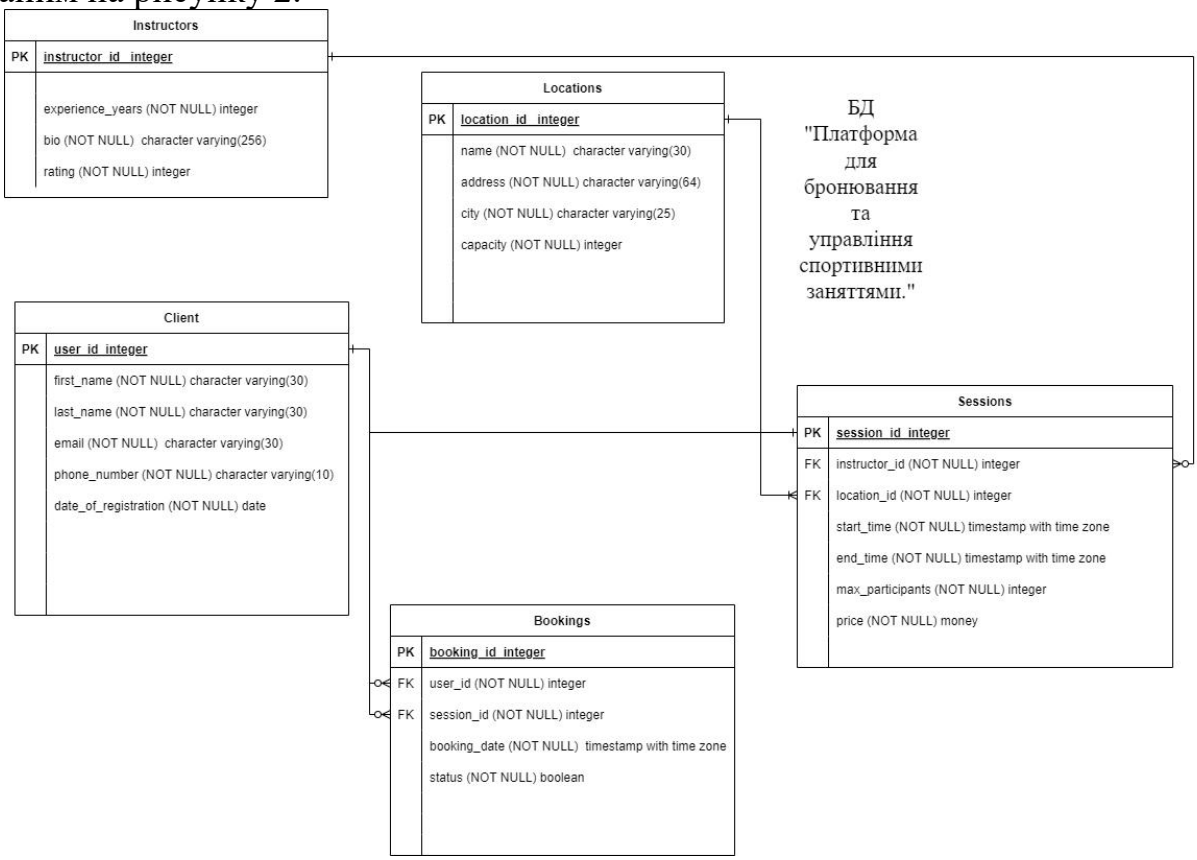


Рисунок 2 – Класи ORM

Запит на вставку має наступний вигляд:

```
using (var context = new AppDbContext())
{
    var entityType = context.Model.GetEntityTypes()
                        .FirstOrDefault(e => e.GetTableName() == table_name);
    var entity = Activator.CreateInstance(entityType.ClrType);
    foreach (var (column, value) in values)
    {
        var property = entityType.ClrType.GetProperties()
                .FirstOrDefault(p =>
                p.GetCustomAttribute<ColumnAttribute>()?.Name.Equals(column,
StringComparison.OrdinalIgnoreCase) == true);
        if (property != null)
        {
            object convertedValue = null;
            if (property.PropertyType == typeof(DateTime))
            {
                convertedValue = DateTime.Parse(value).ToUniversalTime();
            }
            else
            {
                convertedValue = Convert.ChangeType(value, property.PropertyType);
            }
            property.SetValue(entity, convertedValue);
            Console.WriteLine(convertedValue);
        }
        else
```

```
        {
            throw new Exception($"Столбец {column} не знайдено в {entityType.Name}");
        }
        var dbSetMethod = typeof(DbContext).GetMethods()
            .First(m => m.Name == "Set" && m.IsGenericMethodDefinition)
            .MakeGenericMethod(entityType.ClrType);

        var dbSet = dbSetMethod.Invoke(context, null);

        var addMethod = dbSet.GetType().GetMethod("Add");
        addMethod.Invoke(dbSet, new[] { entity });
    }
    context.SaveChanges();
                                        }
```

Запит на оновлення:

```
using (var context = new AppDbContext())
{
    var entityType = context.Model.GetEntityTypes()
                            .FirstOrDefault(e => e.GetTableName() == table_name);
    var genericSetMethod = typeof(DbContext)
                    .GetMethods()
                    .First(m => m.Name == nameof(DbContext.Set) &&
m.IsGenericMethodDefinition)
                    .MakeGenericMethod(entityType.ClrType);

    var set = (IQueryable)genericSetMethod.Invoke(context, null);
    var castedSet = set as IQueryable;

    var entity = castedSet.Cast<object>()
        .FirstOrDefault(e => EF.Property<int>(e, pk_str_column) == pk);
    if (entity != null)
    {
        foreach (var (column, value) in values_res)
        {
            var property = entityType.ClrType.GetProperties()
                .FirstOrDefault(p =>
                p.GetCustomAttribute<ColumnAttribute>()?.Name.Equals(column,
StringComparison.OrdinalIgnoreCase) == true);
            if (property != null)
            {
                object convertedValue = null;

                if(property.PropertyType == typeof(DateTime))
                {
                    convertedValue = DateTime.Parse(value).ToUniversalTime();
                }
                else
                {
                    convertedValue = Convert.ChangeType(value, property.PropertyType);
                }
                property.SetValue(entity, convertedValue);
            }
            else
            {
                throw new Exception($"Столбець {column} не знайдено в таблиці.");
            }
        }
        context.SaveChanges();
    }
    else
    {
        throw new Exception($"Запис з номером {pk} не знайдено.");
```

```
    }
                                                    }
```

Запит на видалення:

```
using (var context = new AppDbContext())
{
    var entityType = context.Model.GetEntityTypes()
                            .FirstOrDefault(e => e.GetTableName() == table_name);
    if (entityType == null)
    {
        throw new Exception($"Для таблиці {table_name} не знайдено метадані.");
    }
    var genericSetMethod = typeof(DbContext)
                    .GetMethods()
                    .First(m => m.Name == nameof(DbContext.Set) &&
m.IsGenericMethodDefinition)
                    .MakeGenericMethod(entityType.ClrType);
    var set = (IQueryable)genericSetMethod.Invoke(context, null);
    var pkProperty = entityType.FindPrimaryKey()?.Properties.FirstOrDefault(p =>
p.Name == pk_str);

    if (pkProperty == null)
    {
        throw new Exception($"Поле {pk_str} не является первичным ключом.");
    }

    var castedSet = set as IQueryable;

    var entity = castedSet.Cast<object>()
        .FirstOrDefault(e => EF.Property<int>(e, pk_str) == pk);

    if (entity != null)
    {
        context.Remove(entity);
        context.SaveChanges();
    }
    else
    {
        throw new Exception($"Запис з номером {pk} не знайдено.");
    }
}
```

Код повного зміненого модуля Model знаходиться в додатку.

Створення індексів:

uery Query History

```
CREATE INDEX client_date_brin_idx ON client USING brin (date_of_registration);
CREATE INDEX email_gin_trgm_idx ON client USING gin (email gin_trgm_ops);
```

Приклади запитів із виведенням результуючих даних, що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).
Приклад 1: Просте фільтрування

```
EXPLAIN ANALYZE SELECT * FROM client WHERE email LIKE '%vmsi%';
```

	QUERY PLAN text
1	Bitmap Heap Scan on client (cost=21.52..25.53 rows=1 width=62) (actual time=0.016..0.017 rows=1 loops=1)
2	Recheck Cond: ((email)::text ~~ '%vmsi%'::text)
3	Heap Blocks: exact=1
4	-> Bitmap Index Scan on email_gin_trgm_idx (cost=0.00..21.52 rows=1 width=0) (actual time=0.011..0.011 rows=1 loop...
5	Index Cond: ((email)::text ~~ '%vmsi%'::text)
6	Planning Time: 0.116 ms
7	Execution Time: 0.035 ms

	user_id [PK] integer	first_name character varying (30)	last_name character varying (30)	email character varying (60)	phone_number character varying (10)	date_of_registration date
1	103	FVMSIDI	CKVZBBUTTGWYJ	fvmsidi.ckvzbbuttgwyj@gmail.com	0998904967	2024-09-10

Пояснення: використовуючи GIN-індекс з триграмами, прискорює запит для пошуку відповідного запису по текстовому полю gmail, завдяки цьому ми уникаємо повного сканування таблиці.
Приклад 2: Агрегатні функції

```
EXPLAIN ANALYZE SELECT COUNT(*) AS total_clients FROM client WHERE
date_of_registration BETWEEN '2024-01-01' AND '2024-06-30';
```

QUERY PLAN text
Aggregate (cost=59.08..59.09 rows=1 width=8) (actual time=0.282..0.282 rows=1 loops=1)
-> Seq Scan on client (cost=0.00..56.44 rows=1055 width=0) (actual time=0.008..0.243 rows=1062...
Filter: ((date_of_registration >= '2024-01-01'::date) AND (date_of_registration <= '2024-06-30'::da...
Rows Removed by Filter: 1034
Planning Time: 0.121 ms
Execution Time: 0.299 ms

	total_clients bigint
1	1062

Пояснення: BRIN-індекс ефективний для великих таблиць з природно впорядкованими даними. Він повинен покращити виконання цього запиту, скануючи менше блоків.

Приклад 3: Групування та сортування

```
EXPLAIN ANALYZE
SELECT EXTRACT(YEAR FROM date_of_registration) AS registration_year,
       COUNT(*) AS client_count
FROM client
GROUP BY registration_year
ORDER BY registration_year DESC;
```

	QUERY PLAN text
1	Sort (cost=82.78..83.73 rows=381 width=40) (actual time=0.813..0.813 rows=1 loops=1)
2	Sort Key: (EXTRACT(year FROM date_of_registration)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=61.68..66.44 rows=381 width=40) (actual time=0.794..0.795 rows=1 loops=1)
5	Group Key: EXTRACT(year FROM date_of_registration)
6	Batches: 1 Memory Usage: 37kB
7	-> Seq Scan on client (cost=0.00..51.20 rows=2096 width=32) (actual time=0.017..0.438 rows=2096 loops...
8	Planning Time: 0.743 ms
9	Execution Time: 0.884 ms

	registration_year numeric	client_count bigint
1	2024	2096

Пояснення: індекс допомогає прискорити филітрацію та агрегацію, якщо дані впорядковані за датою.

Приклад 4: Операції з'єднання

```
EXPLAIN ANALYZE
SELECT c.user_id, c.first_name, c.last_name, c.email, b.booking_date
FROM client c
JOIN bookings b ON c.user_id = b.user_id
WHERE c.email LIKE '%vm%'
  AND c.date_of_registration BETWEEN '2024-01-01' AND '2024-02-20'
ORDER BY c.user_id ASC;
```

	QUERY PLAN text
1	Sort (cost=182.89..182.93 rows=17 width=55) (actual time=1.171..1.173 rows=53 loops=1)
2	Sort Key: c.user_id
3	Sort Method: quicksort Memory: 29kB
4	-> Hash Join (cost=61.76..182.54 rows=17 width=55) (actual time=0.358..1.138 rows=53 loops=1)
5	Hash Cond: (b.user_id = c.user_id)
6	-> Seq Scan on bookings b (cost=0.00..105.00 rows=6000 width=12) (actual time=0.011..0.368 rows=5982 loops=1)
7	-> Hash (cost=61.68..61.68 rows=6 width=47) (actual time=0.339..0.339 rows=11 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on client c (cost=0.00..61.68 rows=6 width=47) (actual time=0.029..0.333 rows=11 loops=1)
10	Filter: (((email)::text ~~ '%vm%'::text) AND (date_of_registration >= '2024-01-01'::date) AND (date_of_registration <= '2024-02-20'::d...
11	Rows Removed by Filter: 2085
12	Planning Time: 1.377 ms
13	Execution Time: 1.197 ms

	user_id integer	first_name character varying (30)	last_name character varying (30)	email character varying (60)	booking_date timestamp with time zone
1	157	HALRVMVO	FFAV	halrvmvo.ffav@gmail.com	2024-11-03 21:19:21.489092+02
2	157	HALRVMVO	FFAV	halrvmvo.ffav@gmail.com	2024-10-06 16:58:30.742489+03
3	157	HALRVMVO	FFAV	halrvmvo.ffav@gmail.com	2024-02-01 15:59:18.781297+02
4	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-08-10 15:31:12.149448+03
5	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-05-14 16:02:53.539719+03
6	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-01-24 01:37:41.343807+02
7	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-03-02 05:45:35.808096+02
8	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-06-11 10:17:18.015849+03
9	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-06-28 14:36:53.713394+03
10	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-02-24 02:17:39.749059+02
11	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-08-17 05:22:13.36094+03
12	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-06-12 22:04:56.955604+03
13	251	BVMRAAJ	PJKPCOOGIUZN	bvmraaj.pjkpcoogiuzn@gmail.com	2024-04-08 04:20:29.667511+03
14	270	IEMCJU	NIKNJEQEBSVMXT	iemcju.niknjeqebsvmxt@gmail.com	2024-10-31 14:16:46.989228+02
15	270	IEMCJU	NIKNJEQEBSVMXT	iemcju.niknjeqebsvmxt@gmail.com	2024-11-14 21:34:09.596106+02
16	270	IEMCJU	NIKNJEQEBSVMXT	iemcju.niknjeqebsvmxt@gmail.com	2024-06-19 01:55:43.997048+03
17	270	IEMCJU	NIKNJEQEBSVMXT	iemcju.niknjeqebsvmxt@gmail.com	2024-01-14 20:59:11.146533+02
18	270	IEMCJU	NIKNJEQEBSVMXT	iemcju.niknjeqebsvmxt@gmail.com	2024-07-09 19:30:34.987476+03
19	279	DSLKHT	FLYUFEVPZVMW	dslkht.flyufevpzvmw@gmail.com	2024-02-25 12:08:29.432562+02
20	279	DSLKHT	FLYUFEVPZVMW	dslkht.flyufevpzvmw@gmail.com	2024-05-29 21:53:50.362153+03
21	279	DSLKHT	FLYUFEVPZVMW	dslkht.flyufevpzvmw@gmail.com	2024-08-23 04:23:44.762164+03

Total rows: 53 of 53 Query complete 00:00:00.073 Ln 6, Col 24

Пояснення: Індекси на умовах з'єднання можуть значно покращити продуктивність, особливо для великих наборів даних. В нашому прикладі GIN-індекс приксорює пошук підстрок в gmail, а BRIN-індекс фільтрацію по даті реєстрації.

Коли не варто використовувати індексацію:

1. **Малі таблиці:** Індексація може не бути корисною для малих таблиць, де повне сканування таблиці швидше.
2. **Часті записи:** Висока активність вставки, оновлення або видалення може зробити індексацію менш ефективною через навантаження на підтримку індексу.
3. **Колонки з низькою кардинальністю:** Колонки з малою кількістю унікальних значень (наприклад, стать) є поганими кандидатами для індексації.

Розробка тригерів

Код тригера:

```
DECLARE
    rec RECORD;
BEGIN
    IF TG_OP = 'UPDATE' THEN
        RAISE NOTICE 'Оновлення для user_id = %', OLD.user_id;
        RETURN NEW;
    END IF;

    IF TG_OP = 'DELETE' THEN
        RAISE NOTICE 'Видалення клієнта user_id = % з email = "%"',
                     OLD.user_id, OLD.email;
        IF EXISTS (SELECT 1 FROM bookings WHERE user_id = OLD.user_id) THEN
            FOR rec IN
                SELECT * FROM bookings WHERE user_id = OLD.user_id
            LOOP
                RAISE NOTICE 'Видалення запису booking_id: % для user_id = %',
                             rec.booking_id, OLD.user_id;
            END LOOP;
        END IF;
        RETURN OLD;
    END IF;
    RETURN NULL;
    EXCEPTION
        WHEN OTHERS THEN
            RAISE NOTICE 'Помилка тригера: %', SQLERRM;
            RETURN NULL;
END;
```

Даний тригер викликає повідомлення про видалення або оновлення клієнта.

Прикріплення тригеру до відповідної таблиці:

CREATE TRIGGER client_before_update_delete
BEFORE UPDATE OR DELETE ON client
FOR EACH ROW
EXECUTE FUNCTION client_before_update_delete_trigger();

Тестування тригеру:

- Оновлення запису про клієнта

UPDATE client
set first_name = 'Alekse', date_of_registration = '2024-04-15'
WHERE user_id = 23;

```
UPDATE client
set first_name = 'Alekse', date_of_registration = '2024-04-15'
WHERE user_id = 23;
```

Data Output Messages Notifications

```
ЗАМЕЧАНИЕ:  Оновлення для user_id = 23
UPDATE 1

Query returned successfully in 1 secs 683 msec.
```

Отримано відповідне повідомлення про оновлення від тригера

- Видалення запису про студента

DELETE from client where user_id = 135

Query Query History

```
DELETE from client where user_id = 135
```

Data Output Messages Notifications

```
ЗАМЕЧАНИЕ:  Видалення клієнта user_id = 135 з email = "fcbcnnln.yytr@gmail.com"
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 188 для user_id = 135
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 1281 для user_id = 135
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 1580 для user_id = 135
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 3300 для user_id = 135
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 3813 для user_id = 135
ЗАМЕЧАНИЕ:  Видалення запису booking_id: 4331 для user_id = 135
DELETE 1
```

Отримано відповідне повідомлення тригеру.

Використання рівнів ізоляції

1. READ COMMITTED

На цьому рівні кожна команда в транзакції бачить лише дані, зафіксовані до початку команди; вона не бачить змін від паралельних незафіксованих транзакцій. Однак результати запиту можуть змінюватися під час транзакції.

Вікно 1:

Query Query History

```
1  BEGIN;
2  UPDATE client SET last_name = 'Temp_3' WHERE user_id = 13;
3
```

Data Output Messages Notifications

```
ЗАМЕЧАНИЕ:  Оновлення для user_id = 13
UPDATE 1

Query returned successfully in 44 msec.
```

Вікно 2:

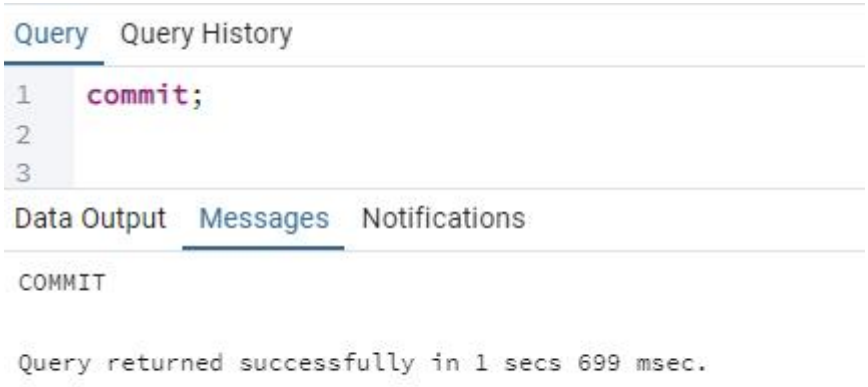
Query Query History

```
1  begin;
2  select last_name from client where user_id = 13;
3
```

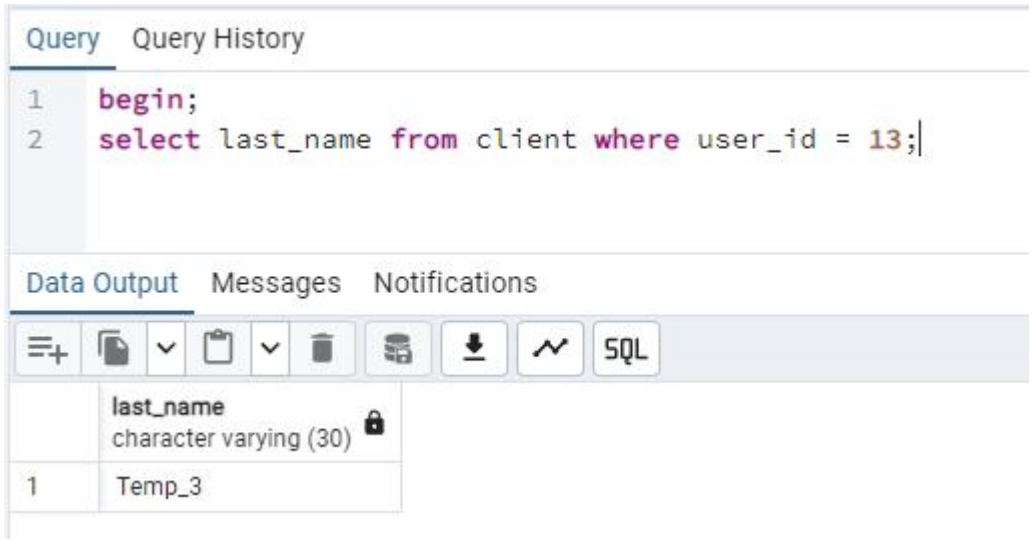
Data Output Messages Notifications

	last_name character varying (30)
1	Temp_2

Вікно 1:



Вікно 2:

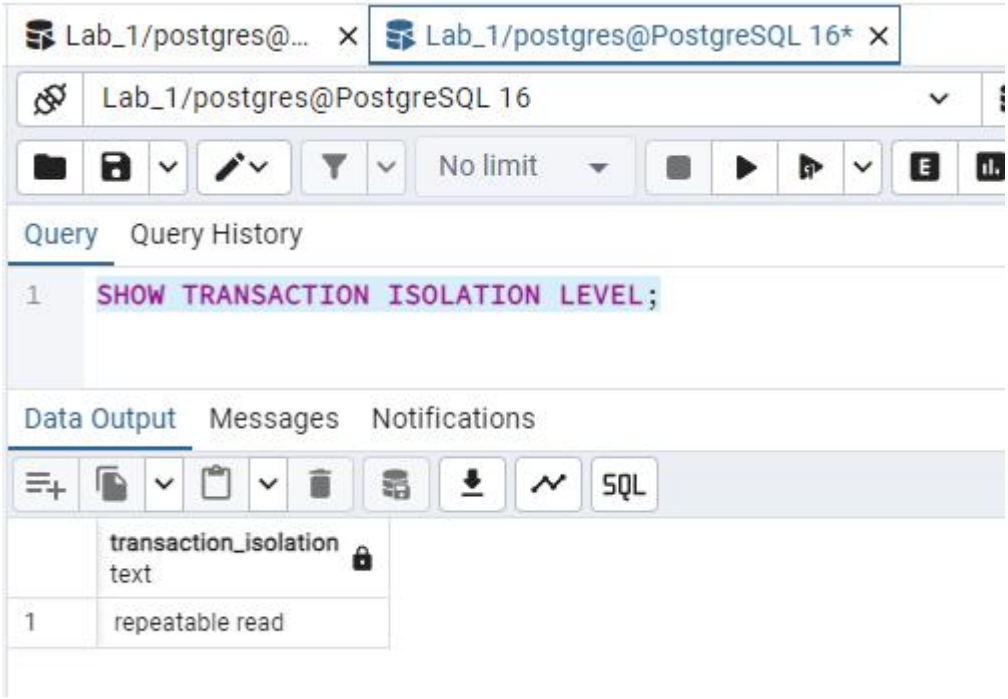


Зміна, зроблена у Вікні 1, не бачиться запитом у Вікні 2, доки вона не буде зафіксована. Це демонстрація рівня ізоляції READ COMMITTED.

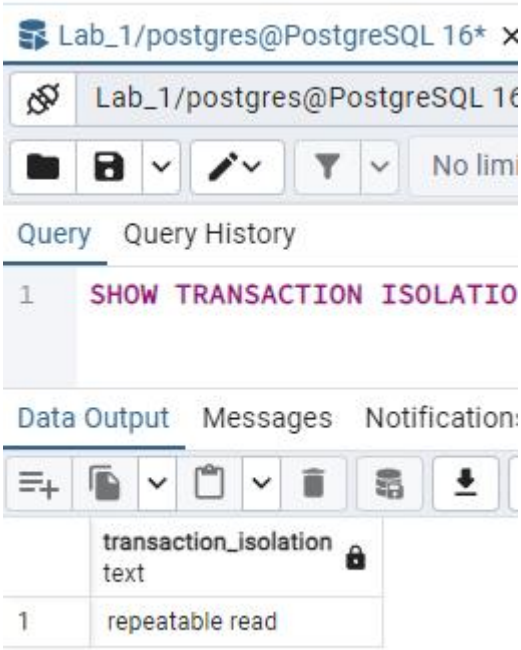
2. REPEATABLE READ

Цей рівень гарантує, що будь-які дані, прочитані під час транзакції, не будуть змінені іншими транзакціями до завершення першої транзакції.

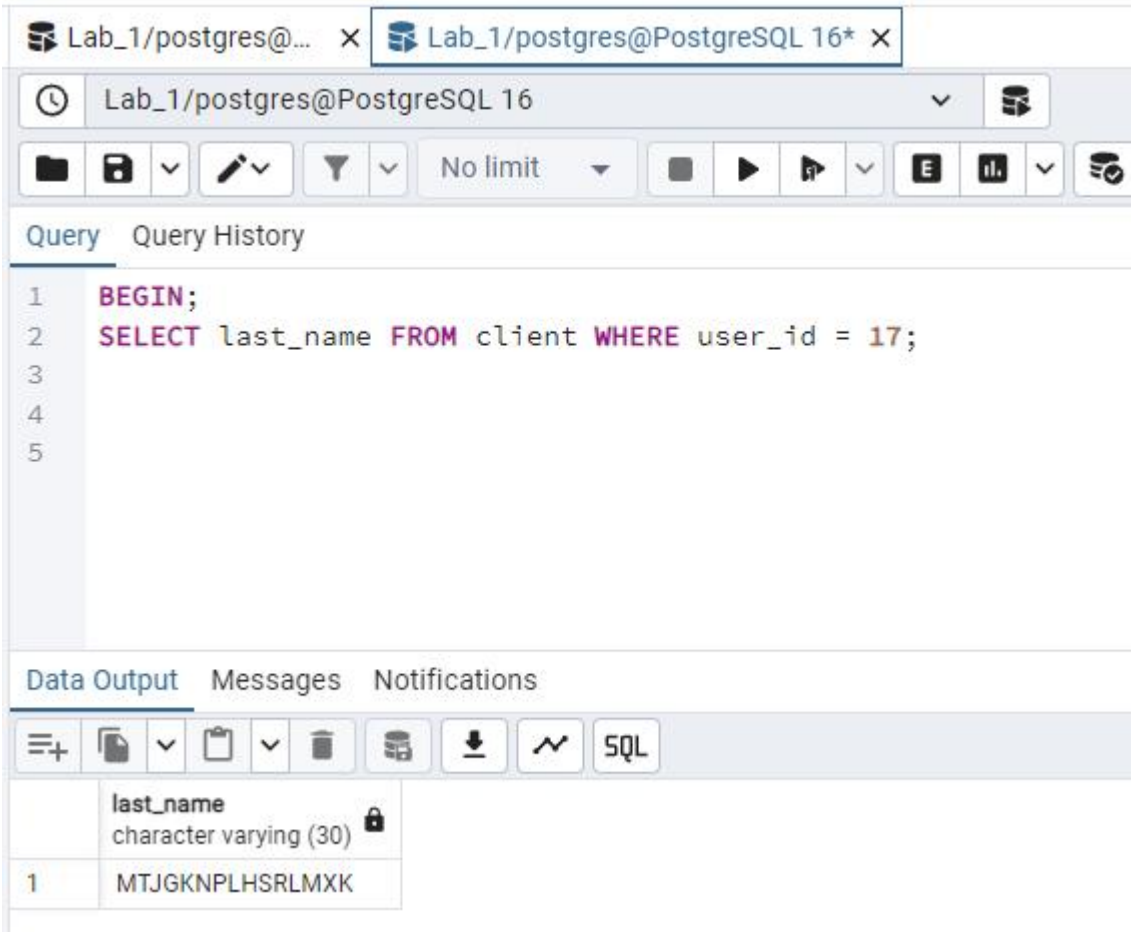
Вікно 2:



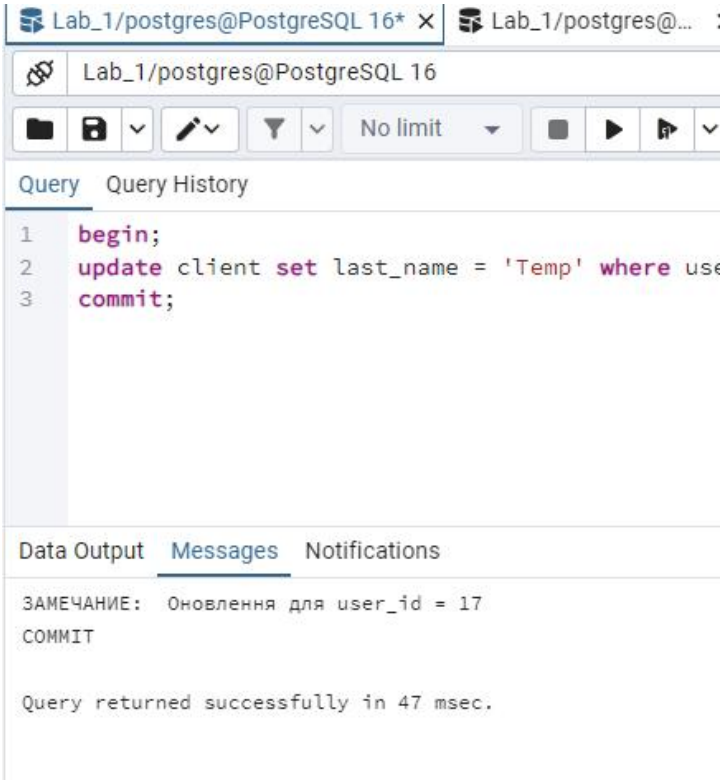
Вікно 1:



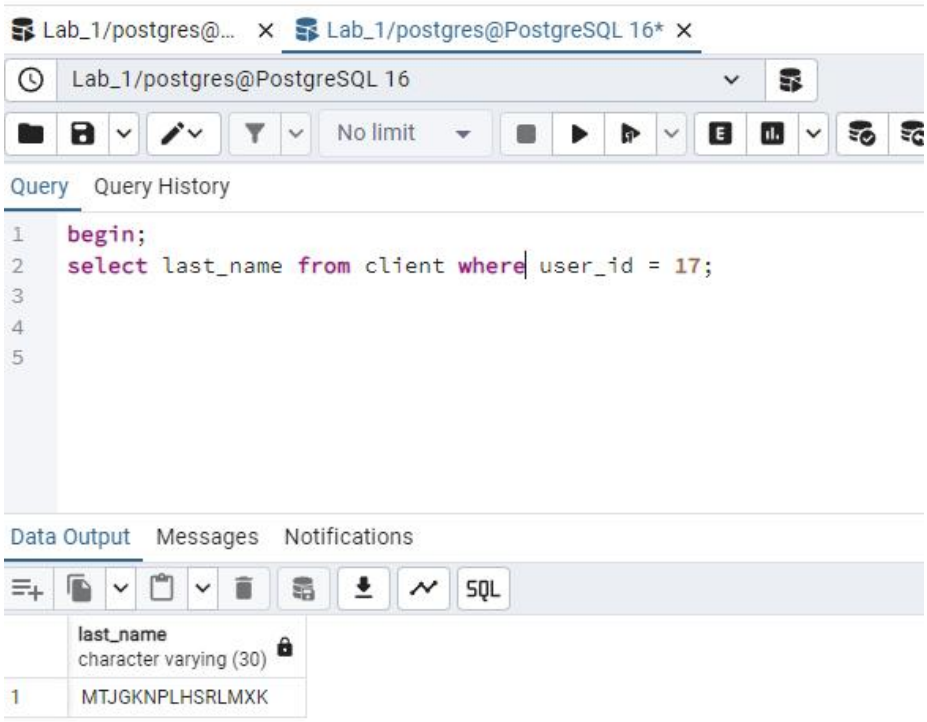
Вікно 1:



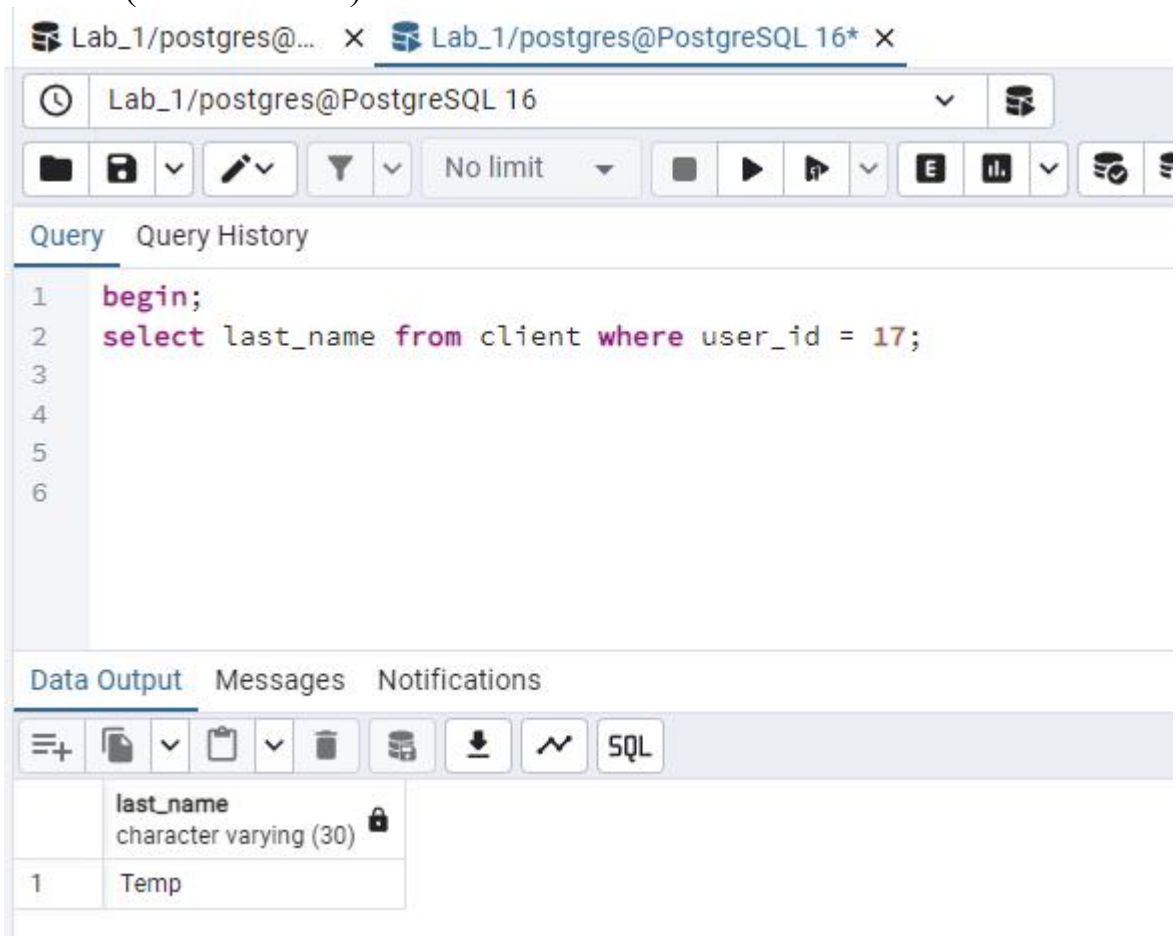
Вікно 2:



Вікно 1:



Вікно 1(Після commit):

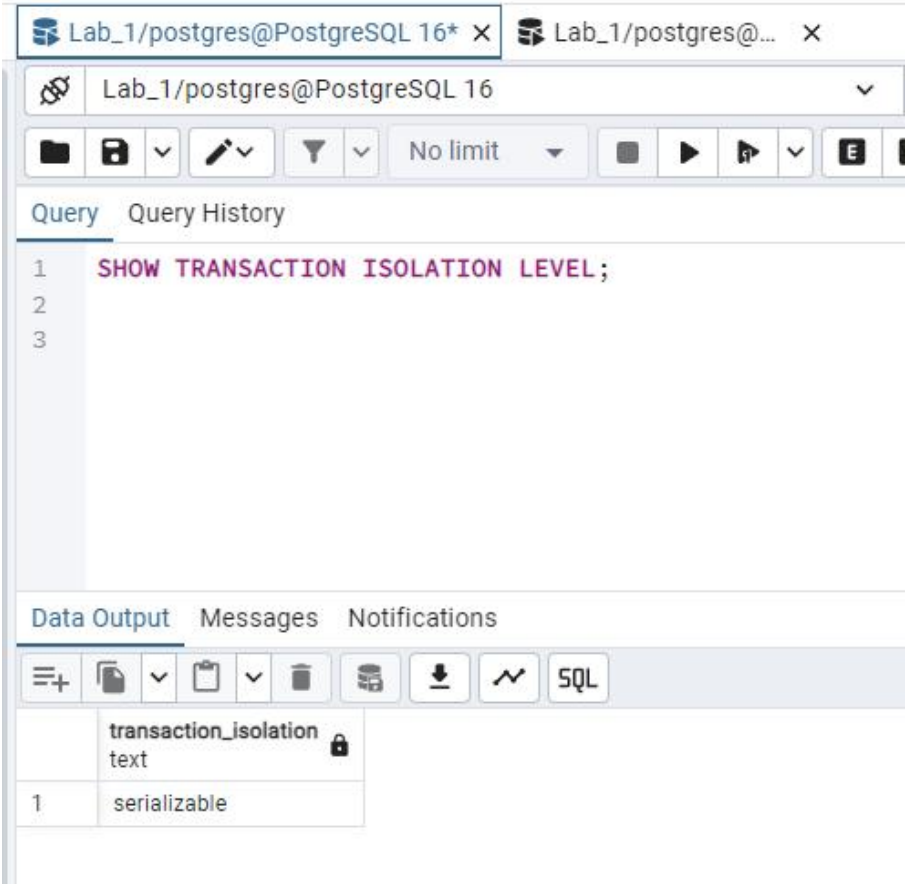


Незважаючи на оновлення у Вікні 2, другий запит SELECT у Вікні 1 все ще показує прізвище до оновлення, демонструючи захист від неповторюваних читань на рівні REPEATABLE READ

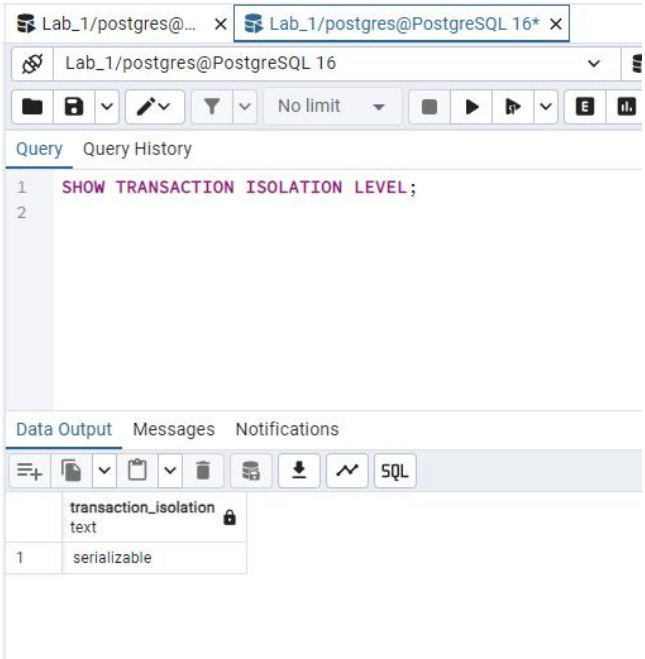
3. SERIALIZABLE

Це найстрогіший рівень, який повністю ізолює транзакцію від будь-якої іншої паралельної транзакції

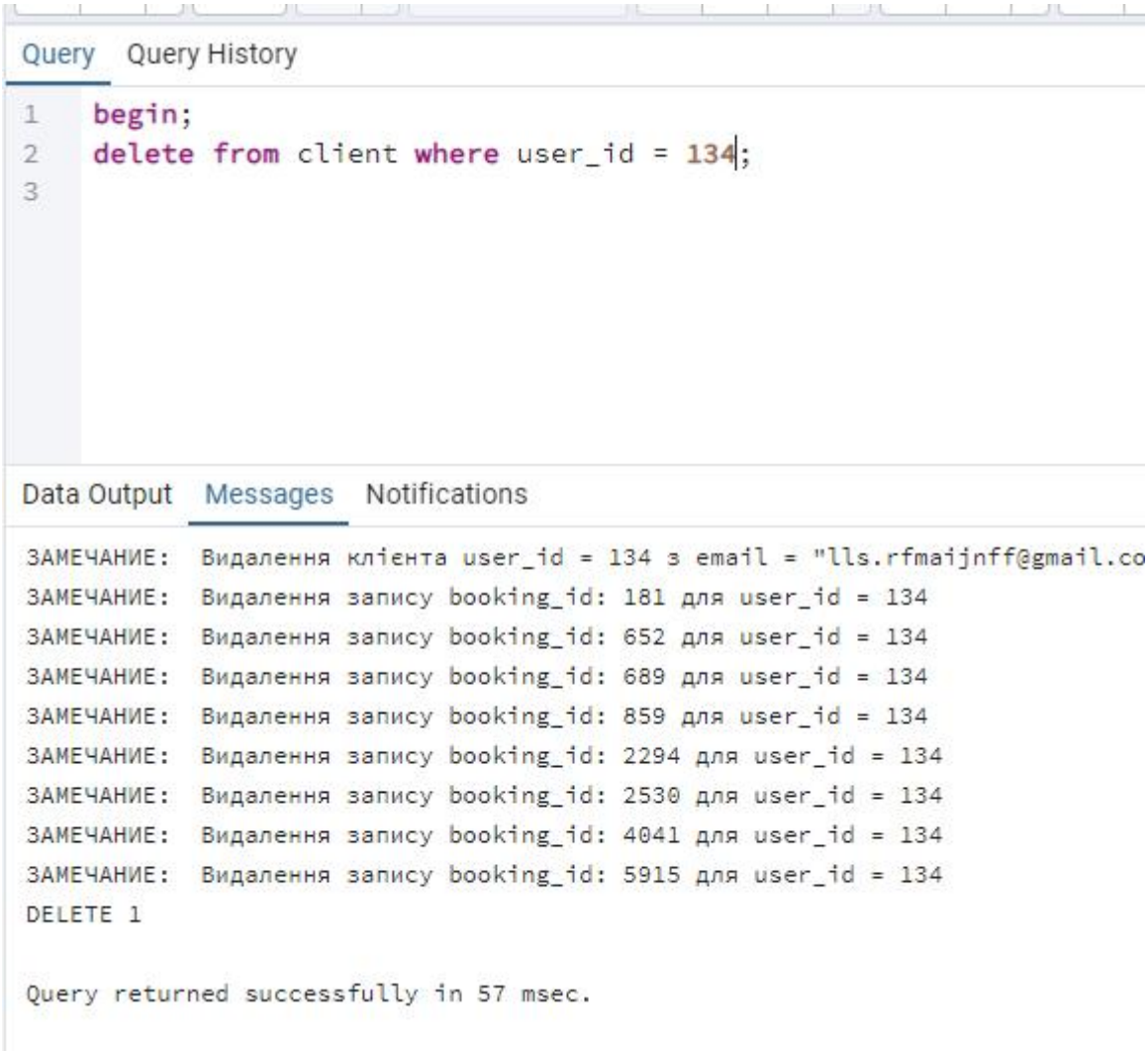
Вікно 1:



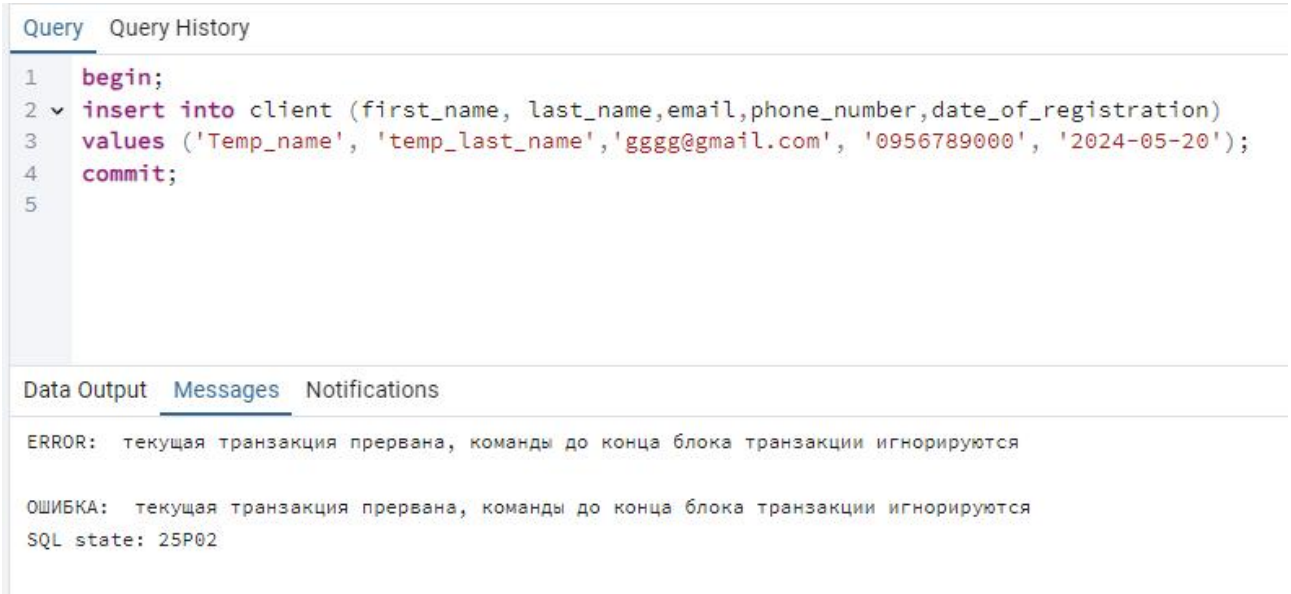
Вікно 2:



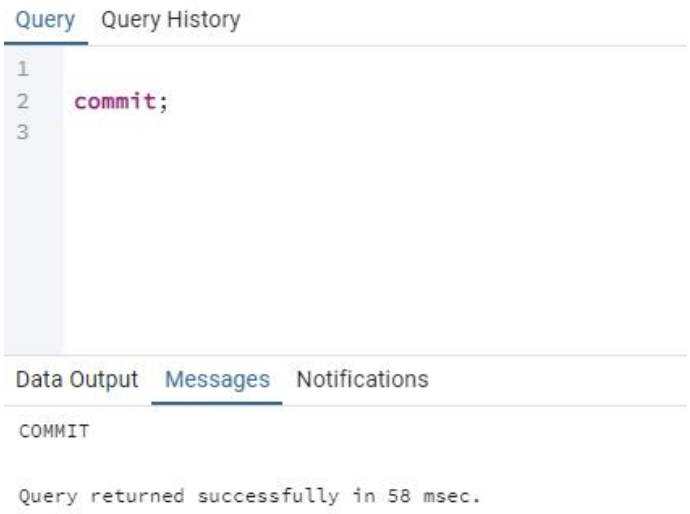
Вікно 2:



Вікно 1:



Вікно 2:



INSERT у Вікні 2 буде заблокованим або відхилено через конфлікт з незафіксованим DELETE у Вікні 1. Це показує строгу ізоляцію, яку забезпечує рівень SERIALIZABLE, уникаючи "брудних" читань, неповторюваних читань та "фантомних" читань

Отож

READ COMMITTED:

- Підходить для застосунків, де потрібно збалансувати **узгодженість** з **продуктивністю**.
- Призначений для більшості звичайних випадків. Забезпечує видимість лише зафіксованих змін з інших транзакцій.
- Може стати проблемою в умовах високої конкуренції, коли одна транзакція може бачити проміжні дані іншої.

REPEATABLE READ:

- Підходить для застосунків, яким потрібні **узгоджені** дані протягом всієї транзакції.
- Забезпечує, що всі запити в рамках транзакції будуть бачити ті самі дані, навіть якщо інші транзакції змінюють ці дані. Може бути менш конкурентоспроможним порівняно з **READ COMMITTED**.

SERIALIZABLE:

- Ідеальний для застосунків, яким потрібна **повна ізоляція** і **узгодженість** даних, зазвичай за рахунок зниження конкурентоспроможності.
- Усі транзакції виглядають як виконані послідовно, одна за одною, що запобігає будь-яким конфліктам, але може значно знизити продуктивність через блокування та інші обмеження.

Додаток

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using Npgsql;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data;
using System.Diagnostics;
using System.Linq.Dynamic.Core;
using System.Reflection;

namespace RGR_BD
{
   public class Model
   {
      public Model()
      {
      }
      public bool AddDataToTableModel(List<(string Column, string Value)> values,string table_name)
      {
         try
         {
            using (var context = new AppDbContext())
            {
               var entityType = context.Model.GetEntityTypes()
                                 .FirstOrDefault(e => e.GetTableName() == table_name);
               var entity = Activator.CreateInstance(entityType.ClrType);
               foreach (var (column, value) in values)
               {
                  var property = entityType.ClrType.GetProperties()
                        .FirstOrDefault(p =>
                        p.GetCustomAttribute<ColumnAttribute>()?.Name.Equals(column, StringComparison.OrdinalIgnoreCase) ==
true);
                  if (property != null)
                  {
                     object convertedValue = null;
                     if (property.PropertyType == typeof(DateTime))
                     {
                        convertedValue = DateTime.Parse(value).ToUniversalTime();
                     }
                     else
                     {
                        convertedValue = Convert.ChangeType(value, property.PropertyType);
                     }
                     property.SetValue(entity, convertedValue);
                     Console.WriteLine(convertedValue);
                  }
                  else
                  {
                     throw new Exception($"Столбец {column} не знайдено в {entityType.Name}");
                  }
                  var dbSetMethod = typeof(DbContext).GetMethods()
                     .First(m => m.Name == "Set" && m.IsGenericMethodDefinition)
                     .MakeGenericMethod(entityType.ClrType);

                  var dbSet = dbSetMethod.Invoke(context, null);

                  var addMethod = dbSet.GetType().GetMethod("Add");
                  addMethod.Invoke(dbSet, new[] { entity });
               }
               context.SaveChanges();
            }
         }
```

```
            catch (Exception ex)
            {
                Console.WriteLine("Помилка при додаванні данних " + ex.Message + " " + ex.InnerException.Message);
                return true;
            }

            return false;
        }
        public string GetPrimaryKeyColumn(string table_name)
        {
            string pk_str = string.Empty;
            using (var context = new AppDbContext()){
                try
                {
                    var entityType = context.Model.GetEntityTypes()
                                    .FirstOrDefault(e => e.GetTableName() == table_name);
                    if (entityType == null)
                    {
                        throw new Exception($"Для таблиці {table_name} не знайдено метадані.");
                    }
                    var pk = entityType.FindPrimaryKey();
                    if (pk == null)
                    {
                        throw new Exception($"Для таблиці {table_name} не знайдено pk.");
                    }
                    pk_str = pk.Properties.FirstOrDefault()?.Name;
                }
                catch (Exception ex) {
                    Console.WriteLine("Помилка при отриманні pk " + ex.Message);
                    return pk_str;
                }
            }
            return pk_str;
        }
        public bool UpdateDataInTable(List<(string Column, string Value)> values_res, string table_name, int pk)
        {
            string pk_str_column = GetPrimaryKeyColumn(table_name);
            try
            {
                using (var context = new AppDbContext())
                {
                    var entityType = context.Model.GetEntityTypes()
                                    .FirstOrDefault(e => e.GetTableName() == table_name);
                    var genericSetMethod = typeof(DbContext)
                                .GetMethods()
                                .First(m => m.Name == nameof(DbContext.Set) && m.IsGenericMethodDefinition)
                                .MakeGenericMethod(entityType.ClrType);

                    var set = (IQueryable)genericSetMethod.Invoke(context, null);
                    var castedSet = set as IQueryable;

                    var entity = castedSet.Cast<object>()
                        .FirstOrDefault(e => EF.Property<int>(e, pk_str_column) == pk);
                    if (entity != null)
                    {
                        foreach (var (column, value) in values_res)
                        {
                            var property = entityType.ClrType.GetProperties()
                                .FirstOrDefault(p =>
                                p.GetCustomAttribute<ColumnAttribute>()?.Name.Equals(column, StringComparison.OrdinalIgnoreCase) ==
true);
                            if (property != null)
                            {
                                object convertedValue = null;

                                if(property.PropertyType == typeof(DateTime))
                                {
                                    convertedValue = DateTime.Parse(value).ToUniversalTime();
                                }
                                else
```

```
                {
                    convertedValue = Convert.ChangeType(value, property.PropertyType);
                }
                property.SetValue(entity, convertedValue);
            }
            else
            {
                throw new Exception($"Столбець {column} не знайдено в таблиці.");
            }
        }
        context.SaveChanges();
    }
    else
    {
        throw new Exception($"Запис з номером {pk} не знайдено.");
    }
  }
 }
 catch (Exception ex)
 {
  Console.WriteLine("Помилка при зміненні данних " + ex.Message);
  return true;
 }
 return false;
}
public (bool error, List<string> ColumnsName) GetColumnNameOfTable(string table_name)
{
 List<string> columnsname = new List<string>();
 try
 {
  using (var context = new AppDbContext())
  {
   var designTimeModel = context.GetService<IDesignTimeModel>();
   var model = designTimeModel.Model;
   var entityType = model.GetEntityTypes()
                .FirstOrDefault(e => e.GetTableName() == table_name);

   if (entityType == null)
   {
    throw new Exception($"Таблица {table_name} не найдена в модели.");
   }

   columnsname = entityType.GetProperties()
                .OrderBy(p => p.GetColumnOrder() ?? int.MaxValue)
                .Select(p => p.GetColumnName())
                .ToList();
  }
 }
 catch (Exception ex)
 {
  Console.WriteLine("Помилка при отриманні назв столбців "+ex.Message);
  return (true, columnsname);
 }
 return (false, columnsname);
}
public (bool error, List<List<string>> rows) GetRowsOfTable(string table_name, int page_num)
{
 List<List<string>> rows = new List<List<string>>();
 int pageSize = 50;
 int startRow = (page_num - 1) * pageSize;
 string pkPropertyName = GetPrimaryKeyColumn(table_name);
 try {
  using (var context = new AppDbContext())
  {
   var entityClrType = context.Model.GetEntityTypes()
                .FirstOrDefault(e => e.GetTableName() == table_name).ClrType;
   var genericSetMethod = typeof(DbContext)
            .GetMethods()
            .First(m => m.Name == nameof(DbContext.Set) && m.IsGenericMethodDefinition)
            .MakeGenericMethod(entityClrType);
```

```
            var set = (IQueryable)genericSetMethod.Invoke(context, null);
            var query = set
                .OrderBy($"{pkPropertyName} ASC")
                .Skip(startRow)
                .Take(pageSize);

            foreach (var entity in query)
            {
                var row = new List<string>();

                var properties = entity.GetType().GetProperties();
                foreach (var property in properties)
                {
                    var value = property.GetValue(entity)?.ToString() ?? string.Empty;
                    row.Add(value);
                }

                rows.Add(row);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка при отриманні строк {table_name} {ex.Message}");
        return (true, rows);
    }
    return (false, rows);
}
public (bool error, List<string> tables) GetAllTables()
{
    List<string> tables = new List<string>();
    try
    {
        using (var context = new AppDbContext())
        {
            var tableNames = context.Model.GetEntityTypes()
                .Select(t => t.GetTableName())
                .ToList();

            tables.AddRange(tableNames);
        }
    }
    catch (Exception ex) {
        Console.WriteLine("Помилка при отриманні списку таблиць " + ex.Message );
        return (true, tables);
    }
    return (false, tables);
}
public bool DeleteDataOfTable(string table_name, int pk, string pk_str)
{
    try
    {
        using (var context = new AppDbContext())
        {
            var entityType = context.Model.GetEntityTypes()
                            .FirstOrDefault(e => e.GetTableName() == table_name);
            if (entityType == null)
            {
                throw new Exception($"Для таблиці {table_name} не знайдено метадані.");
            }
            var genericSetMethod = typeof(DbContext)
                        .GetMethods()
                        .First(m => m.Name == nameof(DbContext.Set) && m.IsGenericMethodDefinition)
                        .MakeGenericMethod(entityType.ClrType);
            var set = (IQueryable)genericSetMethod.Invoke(context, null);
            var pkProperty = entityType.FindPrimaryKey()?.Properties.FirstOrDefault(p => p.Name == pk_str);

            if (pkProperty == null)
            {
```

```
                throw new Exception($"Поле {pk_str} не является первичным ключом.");
            }

            var castedSet = set as IQueryable;

            var entity = castedSet.Cast<object>()
                .FirstOrDefault(e => EF.Property<int>(e, pk_str) == pk);

            if (entity != null)
            {
                context.Remove(entity);
                context.SaveChanges();
            }
            else
            {
                throw new Exception($"Запис з номером {pk} не знайдено.");
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка при видаленні данних {ex.Message}");
        return (true);
    }
    return false;
}
public bool GenerateDataToCurrentTable(string proc_name, int count_rows)
{
    try
    {
        using (var context = new AppDbContext())
        {
            var query = $"CALL {proc_name}({count_rows})";
            context.Database.ExecuteSqlRaw(query);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Помилка при генерації випадкових данних" + ex.Message);
        return true;
    }
    return false;
}

public (bool error, List<List<string>> str_res, long time) SearchFirst(string status)
{
    List<List<string>> rows = new List<List<string>>();
    long executionTimeMs = 0;
        //string query = $@"SELECT
        //                    b.status,
        //                    COUNT(b.booking_id) AS total_bookings,
        //                    SUM(s.price) AS total_cost
        //                FROM
        //                    bookings b
        //                JOIN
        //                    session s ON b.session_id = s.session_id
        //                WHERE
        //                    b.status = {status}
        //                GROUP BY
        //                    b.status
        //                ORDER BY
        //                    total_bookings DESC;";
        Stopwatch stopwatch = Stopwatch.StartNew();
    try {
        using (var context = new AppDbContext())
        {
            var result = context.Bookings
                .Where(b => b.Status == bool.Parse(status))
                .Join(
                    context.Sessions,
```

```
                b => b.SessionId,
                s => s.SessionId,
                (b, s) => new { b.Status, BookingId = b.BookingId, Price = (decimal)s.PriceMoney }
            )
            .GroupBy(
                bs => bs.Status,
                bs => new { bs.BookingId, bs.Price }
            )
            .Select(g => new
            {
                Status = g.Key,
                TotalBookings = g.Count(),
                TotalCost = g.Sum(x => x.Price)
            })
            .OrderByDescending(x => x.TotalBookings)
            .ToList();
        rows = result.Select(r => new List<string>
        {
            r.Status.ToString(),
            r.TotalBookings.ToString(),
            r.TotalCost.ToString()
        }).ToList();
        stopwatch.Stop();
        executionTimeMs = stopwatch.ElapsedMilliseconds;
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("Помилка при пошукову запиті №1" + ex.Message);
    return (true, rows, 0);
  }
  return (false, rows, executionTimeMs);
}
public (bool error, List<List<string>> str_res, long time) SearchSecond(string start_time)
{
  List<List<string>> rows = new List<List<string>>();
  long executionTimeMs = 0;
  try
  {
    //string query = $@"SELECT
    //                SUM(s.max_participants) AS count_max_participants,
    //                l.city,
    //               count(b.booking_id) as count_total_bookings
    //              FROM
    //                session s
    //              JOIN
    //                locations l ON s.location_id = l.location_id
    //              JOIN
    //                bookings b ON s.session_id = b.session_id
    //              WHERE
    //                s.start_time >= '{start_time}'
    //              GROUP BY
    //                l.city
    //              ORDER BY
    //                count_max_participants DESC;";
    Stopwatch stopwatch = Stopwatch.StartNew();
    using (var context = new AppDbContext())
    {
      DateTime parsedStartTime;
      if (!DateTime.TryParse(start_time, out parsedStartTime))
      {
        throw new Exception("Невірний формат дати.");

      }
      var result = context.Sessions
        .Where(s => s.StartTime >= parsedStartTime.ToUniversalTime())
        .Join(
          context.Locations,
          s => s.LocationId,
          l => l.LocationId,
```

```
                (s, l) => new { s, l }
            )
            .Join(
                context.Bookings,
                sl => sl.s.SessionId,
                b => b.SessionId,
                (sl, b) => new { sl.s, sl.l, b }
            )
            .GroupBy(
                x => x.l.City,
                x => new { x.s.MaxParticipants, x.b.BookingId }
            )
            .Select(g => new
            {
                City = g.Key,
                CountMaxParticipants = g.Sum(x => x.MaxParticipants),
                CountTotalBookings = g.Count()
            })
            .OrderByDescending(x => x.CountMaxParticipants)
            .ToList();
        rows = result.Select(r => new List<string>
        {
            r.City,
            r.CountMaxParticipants.ToString(),
            r.CountTotalBookings.ToString()
        }).ToList();
      }
      stopwatch.Stop();
      executionTimeMs = stopwatch.ElapsedMilliseconds;
    }
    catch (Exception ex)
    {
      Console.WriteLine("Помилка при пошукову запиті №2" + ex.Message);
      return (true, rows, 0);
    }
    return (false, rows, executionTimeMs);
}
public (bool error, List<List<string>> str_res, long time) SearchThird(string city, string exp_years)
{
    long executionTimeMs = 0;
    List<List<string>> rows = new List<List<string>>();
    try
    {
      Stopwatch stopwatch = Stopwatch.StartNew();
      using (var context = new AppDbContext())
      {
        string query = $@"
              SELECT
                l.city,
                i.rating,
                AVG(i.experiance_years) AS av_exp_yars,
                AVG(s.price::numeric) AS avg_price
              FROM
                instructors i
              JOIN
                session s ON i.instructor_id = s.instructor_id
              JOIN
                locations l ON s.location_id = l.location_id
              WHERE
                l.city = '{city}'
                AND i.experiance_years >= {exp_years}
              GROUP BY
                l.city, i.rating
              ORDER BY
                i.rating DESC";
        using (var command = context.Database.GetDbConnection().CreateCommand())
        {
          command.CommandText = query;
          command.CommandType = System.Data.CommandType.Text;
```

```
                context.Database.OpenConnection();

                using (var reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        var row = new List<string>();
                        for (int i = 0; i < reader.FieldCount; i++)
                        {
                            row.Add(reader[i]?.ToString() ?? string.Empty);
                        }
                        rows.Add(row);
                    }
                }
            }

            stopwatch.Stop();
            executionTimeMs = stopwatch.ElapsedMilliseconds;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Помилка при пошукову запиті №3" + ex.Message);
        return (true, rows, 0);
    }
    return (false, rows, executionTimeMs);
  }
 }
}
```