

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»
Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконала: студентка III курсу
ФПМ групи КВ-11
Михайліченко С.В.
Перевірів:

Київ – 2024

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Варіант 14

У другому завданні проаналізувати індекси Btree, Hash.

Умова для тригера – after insert, update.

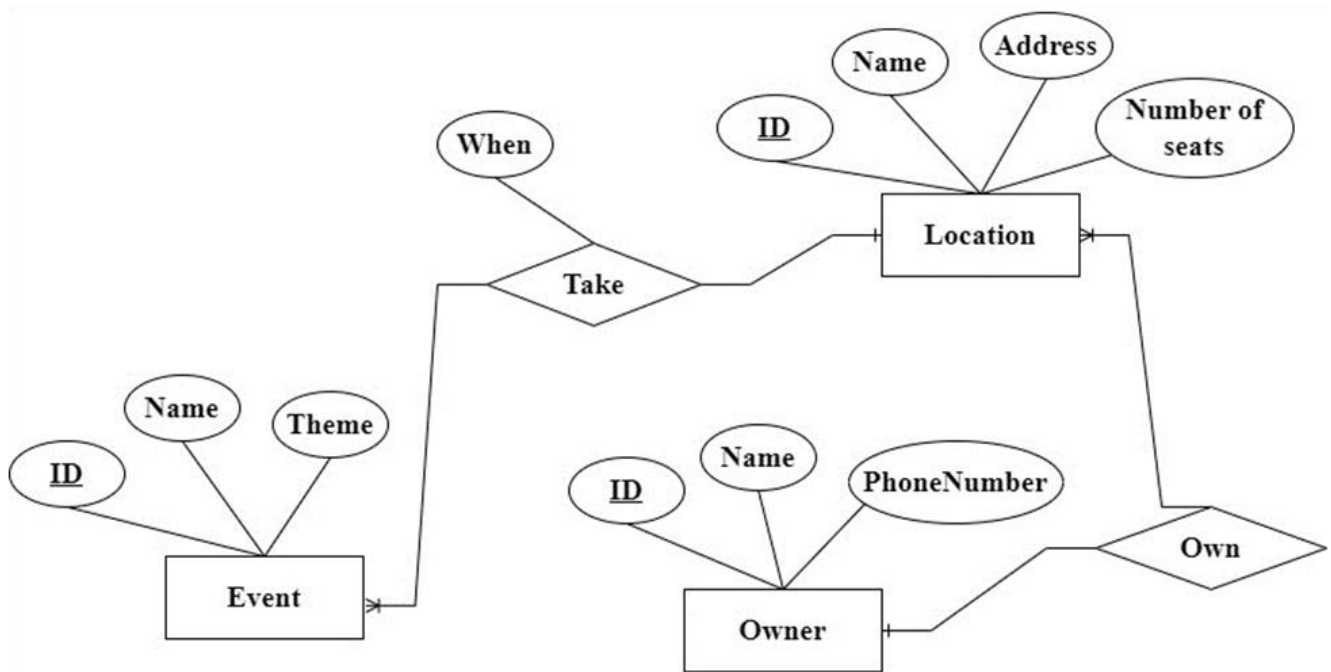
Telegram:

GitHub: https://github.com/Sonneetta/database/tree/lab_2

Завдання 1

Інформація про базу даних

Розробка моделі «сутність-зв'язок» предметної галузі для проектування бази даних «A platform for booking and managing venues for events». Предметна галузь – 65 «Платформа для бронювання та управління майданчиками для подій.».



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

Сутності з описом призначення:

Предметна галузь «A platform for booking and managing venues for events» включає в себе 3 сутності, кожна сутність містить декілька атрибутів:

1. Event (ID, Name, Theme).
2. Location (ID, Name, Address, Number of seats).
3. Owner (ID, Name, PhoneNumber).

Сутність Event описує подію, яка відбудеться на певній локації. Кожна подія має свій ідентифікатор ID, а також має назву та тематику.

Сутність Location описує місце, де певна подія відбуватиметься. Кожна локація має свій ідентифікатор, назву, адресу та кількість можливих відвідувачів за один раз.

Сутність Owner описує власника місця, де відбувається певна подія. Кожен власник має свій ідентифікатор, ім'я та номер телефону.

Зв'язки між сутностями:

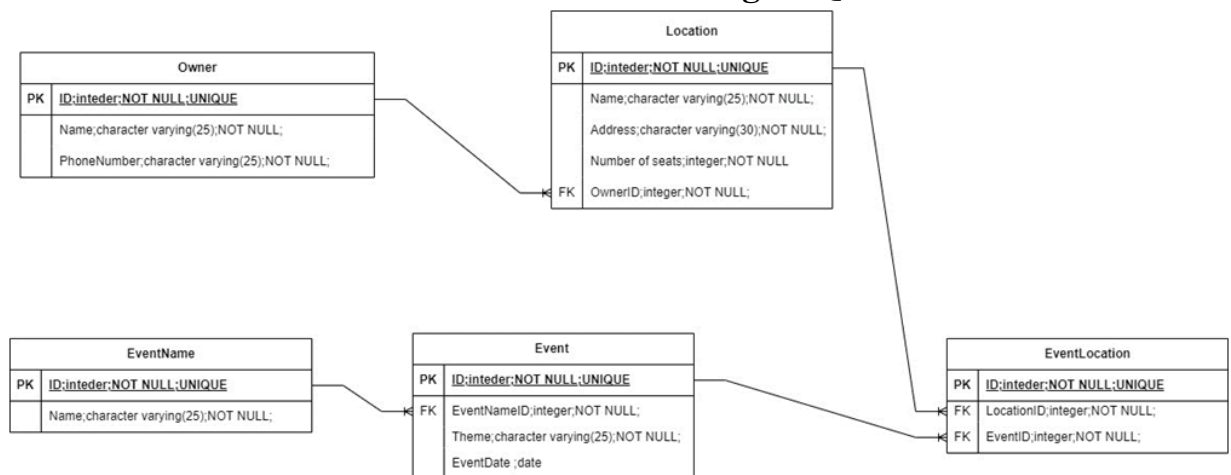
Зв'язок між Event та Location:

Кожна подія має своє місце проведення. Зв'язок 1:N – на одній локації можуть відбуватися декілька подій (наприклад в ресторані можуть замовити столики під святкування різних подій).

Зв'язок між Location та Owner:

Кожна локація має свого власника. Оскільки один власник може мати декілька локацій, зв'язок 1:N.

Схема бази даних PostgreSQL



Малюнок 2. Схема бази даних у графічному вигляді

Функціональні залежності:

1. Event (ID, Name, Theme).

ID→Name

ID→Theme

ID→ EventID

2. Location (ID, Name, Address, Number of seats).

ID→Name

ID→ Address

ID→ Number of seats

ID→ OwnerID

3. Owner (ID, Name, PhoneNumber).

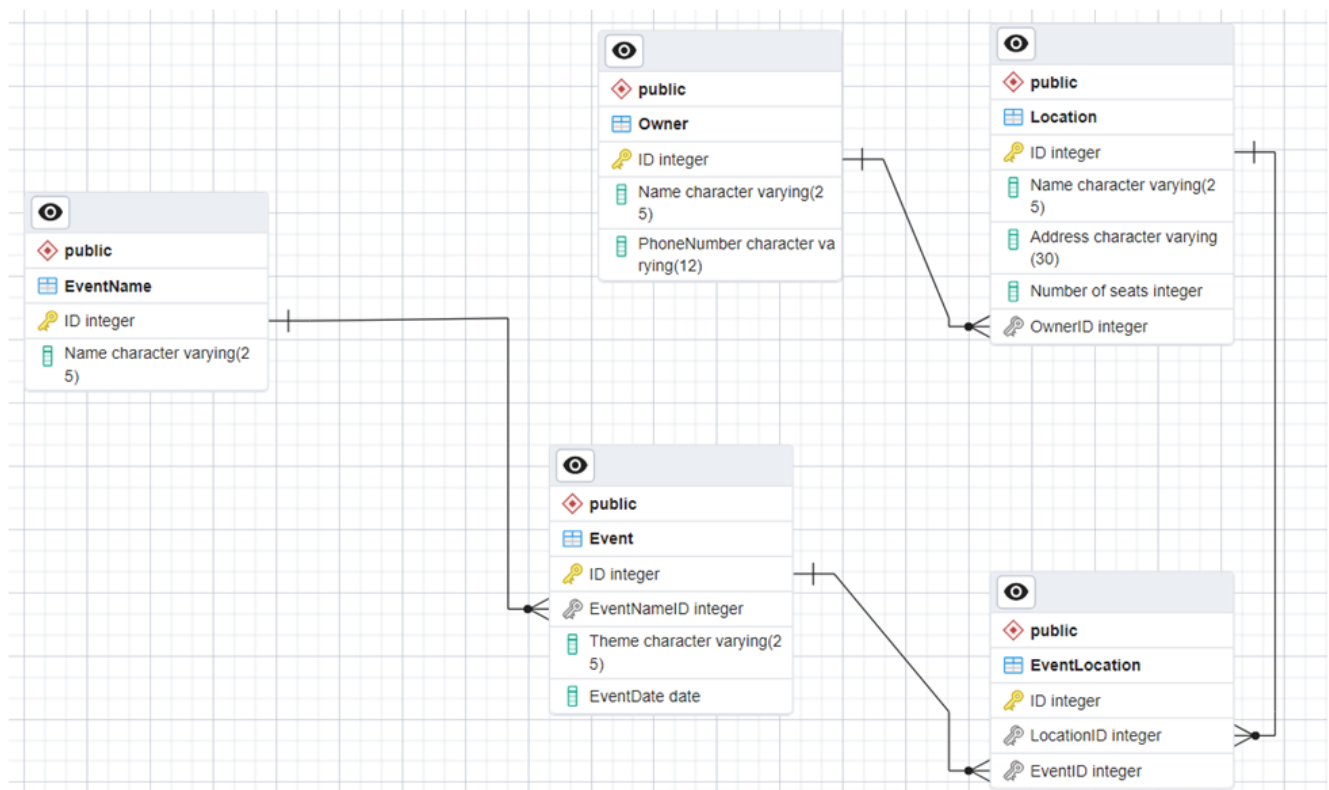
ID→Name

- ID→ PhoneNumber
4. EventName (ID, Name).
ID→Name
5. EventLocation (ID).
ID→LocationID
ID→EventID

Схема бази даних відповідає 1НФ, тому що значення в кожній комірці таблиці є атомарними, кожне поле таблиці є неподільним, кожен рядок є унікальним, немає повторень рядків.

Схема бази даних відповідає 2НФ, бо вона відповідає 1НФ та кожен неключовий атрибут залежить від первинного повного ключа, отже первинний ключ одразу визначає запис та не є надмірним.

Схема бази даних відповідає 3НФ, тому що вона відповідає 2НФ та кожен неключовий атрибут не є транзитивно залежним від кожного кандидатного ключа. В таблицях нема не ключового поля, яке залежить від значення іншого не ключового поля.



Малюнок 3. Схема бази даних у pgAdmin4

Для перетворення модулів моделей програми, створених в розрахунковій роботі, у вигляд об'єктно-реляційної моделі було використано Entity Framework Core.

Необхідні класи для таблиць бази даних та клас моделі:

```

public class TEvent
{
    public int ID { get; set; }
    public int EventNameId { get; set; }
    public string Theme { get; set; }
    public DateTime EventDate { get; set; }
}

public class TEventLocation
{
    public int ID { get; set; }
    public int EventId { get; set; }
    public int LocationId { get; set; }
}

public class TEventLocation
{
    public int ID { get; set; }
    public int EventId { get; set; }
    public int LocationId { get; set; }
}

public class TLocation
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public int NumberOfSeats { get; set; }
    public int OwnerId { get; set; }
}

public class TOwner
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string PhoneNumber { get; set; }
}

public class ModelClass
{
    private ContextClass context;
    public ModelClass() {
        context = new ContextClass();
    }
    public List<TEvent> GetAllEvent()
    {
        var evs = context.Event.ToList();

        return evs;
    } // Отримання всіх івентів
    static string GenerateRandomString(int length)
    {
        const string chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

        Random random = new Random();

        char[] randomArray = new char[length];
        for (int i = 0; i < length; i++)
        {
            randomArray[i] = chars[random.Next(chars.Length)];
        }

        string randomString = new string(randomArray);
        return randomString;
    }
    public int AddEvent(TEvent event_)
    {
        var names = context.EventName.ToList();
        List<string> nm = new List<string>();
    }
}

```

```

        List<int> ids = new List<int>();
        int eventNameId = -1;
        foreach (TEventName t in names)
        {
            if (t.ID == event_.EventNameId)
                eventNameId = t.ID;
        }
        if (eventNameId != -1)
        {
            var evs = context.Event.ToList();
            foreach (TEvent ev in evs)
            {
                ids.Add(ev.ID);
            }
            event_.ID = ids.Max() + 1;

            context.Event.Add(event_);
            context.SaveChanges();
            return 1;
        }
        else return 0;
    } // Додавання нового івенту
    public int DeleteEvent(int index)
    {
        try
        {
            List<int> ids = new List<int>();
            var evs = context.Event.ToList();
            foreach (TEvent ev in evs)
            {
                ids.Add(ev.ID);
            }
            if (!ids.Contains(index))
            {
                return -1;
            }

            TEvent? even = context.Event.Find(index);
            context.Event.Remove(even);
            context.SaveChanges();
            return 1;
        }
        catch
        {
            return 0;
        }
    } // Видалення івенту
    public int UpdateEvent(int id, TEvent event_)
    {
        List<int> ids_ = new List<int>();
        var evs = context.Event.ToList();
        foreach (TEvent ev in evs)
        {
            ids_.Add(ev.ID);
        }
        if (!ids_.Contains(id))
        {
            return 0;
        }

        var names = context.EventName.ToList();
        List<string> nm = new List<string>();
        List<int> ids = new List<int>();
        int eventNameId = -1;
        foreach (TEventName t in names)

```

```

        {
            if (t.ID == event_.EventNameId)
                eventNameId = t.ID;
        }
        if (eventNameId != -1)
        {
            TEvent? updatedEvent = context.Event.Find(id);
            updatedEvent.Theme = event_.Theme;
            updatedEvent.EventNameId = event_.EventNameId;
            updatedEvent.EventDate = event_.EventDate;

            context.SaveChangesAsync();
            context = new ContextClass();
            return 1;
        }
        return -1;
    } // Оновлення івенту
    public List<TEvent> SearchEventByName(int id)
    {
        var evs = context.Event.Where(e => e.EventNameId == id)
            .ToList();
        return evs;
    } // Пошук івенту за іменем
    public List<TEvent> SearchEventByTheme(string tm)
    {
        var evs = context.Event.Where(e => e.Theme == tm)
            .ToList();
        return evs;
    } // Пошук івенту за темою
    public List<TEvent> SearchEventByDate(DateOnly dt)
    {
        var evs = context.Event.Where(e => e.EventDate == dt)
            .ToList();
        return evs;
    } // Пошук івенту за датою
    public void GenerateEvents(int n)
    {
        for (int i = 0; i < n; i++)
        {
            var names = context.EventName.ToList();
            List<int> ids = new List<int>();
            List<int> hosps = new List<int>();
            foreach (TEventName d in names)
            {
                ids.Add(d.ID);
            }
            Random rnd = new Random();
            int nameid = ids[rnd.Next(0, ids.Count)];

            List<int> ids_ = new List<int>();
            var evs = context.Event.ToList();
            foreach (TEvent ev in evs)
            {
                ids_.Add(ev.ID);
            }
            int id = ids_.Max() + 1;
            TEvent event_ = new TEvent();
            event_.ID = id;
            event_.EventNameId = nameid;
            event_.EventDate =
                DateOnly.FromDateTime(Convert.ToDateTime("01.01.2024"));
            event_.Theme = GenerateRandomString(15);

            context.Event.Add(event_);
        }
    }

```



```

        context.SaveChanges();
    }
} //Генерація нових івентів

////////////////////////////////////
////////////////////////////////////
public List<TOwner> GetAllOwner()
{
    var ow = context.Owner.ToList();
    return ow;
} // Отримання всіх власників
public int AddOwner(TOwner owner)
{
    List<int> ids = new List<int>();
    var ow = context.Owner.ToList();
    foreach (TOwner o in ow)
    {
        ids.Add(o.ID);
    }
    owner.ID = ids.Max() + 1;

    context.Owner.Add(owner);
    context.SaveChanges();
    return 1;
} // Додавання нового власника
public int DeleteOwner(int index)
{
    try
    {
        List<int> ids = new List<int>();
        var ows = context.Owner.ToList();
        foreach (TOwner o in ows)
        {
            ids.Add(o.ID);
        }
        if (!ids.Contains(index))
        {
            return -1;
        }
        TOwner? owner = context.Owner.Find(index);
        context.Owner.Remove(owner);

        context.SaveChanges();
        return 1;
    }
    catch
    {
        return 0;
    }
} // Видалення власника
public int UpdateOwner(int id, TOwner owner)
{
    List<int> ids_ = new List<int>();
    var ows = context.Owner.ToList();
    foreach (TOwner o in ows)
    {
        ids_.Add(o.ID);
    }
    if (!ids_.Contains(id))
    {
        return 0;
    }
    TOwner own = context.Owner.Find(id);
    own.Name = owner.Name;
}

```

```

        own.PhoneNumber = owner.PhoneNumber;

        context.SaveChangesAsync();
        context = new ContextClass();
        return 1;
    } // Оновлення власника
    public List<TOwner> SearchOwnerByName(string nm)
    {
        var ows = context.Owner.Where(e => e.Name.Contains(nm))
            .ToList();
        return ows;
    } // Пошук власника за іменем
    public List<TOwner> SearchOwnerByPhone(string ph)
    {
        var ows = context.Owner.Where(e => e.PhoneNumber.Contains(ph))
            .ToList();
        return ows;
    } // Пошук власника за телефоном
    public void GenerateOwner(int n)
    {
        for (int i = 0; i < n; i++)
        {
            List<int> ids = new List<int>();
            var ow = context.Owner.ToList();
            foreach (TOwner o in ow)
            {
                ids.Add(o.ID);
            }
            int Id = ids.Max() + 1;

            TOwner owner = new TOwner();
            owner.ID = Id;
            owner.Name = GenerateRandomString(5) +
GenerateRandomString(5);
            owner.PhoneNumber = GenerateRandomString(10);

            context.Add(owner);
            context.SaveChanges();
        }
    } // Генерація нових власників

////////////////////////////////////
////////////////////////////////////
    public List<TLocation> GetAllLocation()
    {
        var locations = context.Location.ToList();

        return locations;
    } // Отримання всіх локацій
    public int AddLocation(TLocation location)
    {
        List<int> ids = new List<int>();
        var locations = context.Location.ToList();
        foreach (TLocation l in locations)
        {
            ids.Add(l.ID);
        }
        location.ID = ids.Max() + 1;
        int ownerId = -1;
        var ows = context.Owner.ToList();
        foreach (TOwner o in ows)
        {
            if (o.ID == location.OwnerId)
            {

```

```

        ownerId = o.ID;
        break;
    }
}
if (ownerId == -1)
    return 0;

context.Location.Add(location);
context.SaveChanges();
return 1;
} // Додавання нової локації
public int DeleteLocation(int index)
{
    try
    {
        List<int> ids = new List<int>();
        var locations = context.Location.ToList();
        foreach (TLocation l in locations)
        {
            ids.Add(l.ID);
        }
        if (!ids.Contains(index))
        {
            return -1;
        }
        TLocation? loc = context.Location.Find(index);
        context.Location.Remove(loc);
        context.SaveChanges();

        return 1;
    }
    catch
    {
        return 0;
    }
} // Видалення локації
public int UpdateLocation(int id, TLocation location)
{
    List<int> ids_ = new List<int>();
    var locations = context.Location.ToList();
    foreach (TLocation l in locations)
    {
        ids_.Add(l.ID);
    }
    if (!ids_.Contains(id))
    {
        return 0;
    }
    int ownerId = -1;
    var ows = context.Owner.ToList();
    foreach (TOwner o in ows)
    {
        if (o.ID == location.OwnerId)
        {
            ownerId = o.ID;
            break;
        }
    }
    if (ownerId == -1)
        return 0;
    TLocation loc = context.Location.Find(id);
    loc.Address = location.Address;
    loc.OwnerId = location.OwnerId;
    loc.NumberOfSeats = location.NumberOfSeats;
}

```

```

        context.SaveChangesAsync();
        context = new ContextClass();
        return 1;
    } // Оновлення локації
    public List<TLocation> SearchLocationByName(string nm)
    {
        var locations = context.Location.Where(e => e.Name.Contains(nm))
            .ToList();

        return locations;
    } // Пошук локації за назвою
    public List<TLocation> SearchLocationByAddress(string address)
    {
        var locations = context.Location.Where(e =>
e.Address.Contains(address))
            .ToList();

        return locations;
    } // Пошук локації за адресою
    public List<TLocation> SearchLocationByNumberOfSeats(int minNos, int
maxNos)
    {
        var locations = context.Location
            .Where(e => e.NumberOfSeats >= minNos && e.NumberOfSeats <=
maxNos)
            .ToList();

        return locations;
    } // Пошук локації за кількістю місць
    public List<TLocation> SearchLocationByOwner(string owner)
    {
        int ownerId = -1;
        var ows = context.Owner.ToList();
        foreach (TOwner o in ows)
        {
            if (o.Name == owner)
            {
                ownerId = o.ID;
                break;
            }
        }
        var locations = context.Location
            .Where(e => e.OwnerId == ownerId)
            .ToList();
        return locations.ToList();
    } // Пошук локації за власником
    public void GenerateLocation(int n)
    {
        for (int i = 0; i < n; i++)
        {
            List<int> ids = new List<int>();
            var ow = context.Owner.ToList();
            foreach (TOwner o in ow)
            {
                ids.Add(o.ID);
            }
            Random rnd = new Random();
            int ownerId = ids[rnd.Next(0, ids.Count)];

            List<int> ids_ = new List<int>();
            var locations = context.Location.ToList();
            foreach (TLocation l in locations)

```

```

        {
            ids_.Add(l.ID);
        }
        int Id = ids_.Max() + 1;

        TLocation location = new TLocation();
        location.ID = Id;
        location.Name = GenerateRandomString(5) +
GenerateRandomString(5);
        location.Address = GenerateRandomString(15);
        location.OwnerId = ownerId;
        location.NumberOfSeats = rnd.Next(30, 1000);

        context.Location.Add(location);
        context.SaveChanges();
    }
} // Генерація нових локацій

////////////////////////////////////
////////////////////////////////////
public List<TEventName> GetAllEventName()
{
    var ow = context.EventName.ToList();

    return ow.ToList();
} // Отримання назв івентів
public int AddEventName(TEventName eventName)
{
    List<int> ids = new List<int>();
    var eventNames = context.EventName.ToList();
    foreach (TEventName en in eventNames)
    {
        ids.Add(en.ID);
    }
    eventName.ID = ids.Max() + 1;

    context.EventName.Add(eventName);
    context.SaveChanges();

    return 1;
} // Додавання нової назви івенту
public int DeleteEventName(int index)
{
    try
    {
        List<int> ids = new List<int>();
        var eventNames = context.EventName.ToList();
        foreach (TEventName en in eventNames)
        {
            ids.Add(en.ID);
        }
        if (!ids.Contains(index))
        {
            return -1;
        }
        TEventName? eventName = context.EventName.Find(index);
        context.EventName.Remove(eventName);
        context.SaveChanges();

        return 1;
    }
    catch
    {
        return 0;
    }
}

```

```

    }
} // Видалення назви івенту
public int UpdateEventName(int id, TEventName eventName)
{
    List<int> ids_ = new List<int>();
    var eventNames = context.EventName.ToList();
    foreach (TEventName en in eventNames)
    {
        ids_.Add(en.ID);
    }
    if (!ids_.Contains(id))
    {
        return 0;
    }
    TEventName? updatedEventName = context.EventName.Find(id);
    updatedEventName.Name = eventName.Name;
    context.SaveChanges();

    context = new ContextClass();
    return 1;
} // Оновлення назви івенту
public List<TEventName> SearchEventName(string nm)
{
    var eventNames = context.EventName.Where(e =>
e.Name.Contains(nm))
        .ToList();
    return eventNames.ToList();
} // Пошук назви івенту
public void GenerateEventName(int n)
{
    for (int i = 0; i < n; i++)
    {
        List<int> ids = new List<int>();
        var eventName = context.EventName.ToList();
        foreach (TEventName en in eventName)
        {
            ids.Add(en.ID);
        }
        int Id = ids.Max() + 1;

        TEventName eventName1 = new TEventName();
        eventName1.Name = GenerateRandomString(5) +
GenerateRandomString(5);
        eventName1.ID = Id;
        context.EventName.Add(eventName1);

        context.SaveChanges();
    }
} // Генерація нових назв івентів

////////////////////////////////////
////////////////////////////////////
public List<TEventLocation> GetAllEventLocation()
{
    var pd = context.EventLocation.ToList();

    return pd.ToList();
} // Отримання всіх пар івентів та локацій
public int AddEventLocation(int eventId, int locationId)
{
    int eventId_ = -1;
    int locationId_ = -1;
    List<int> idsEvent = new List<int>();
    List<int> idsLocation = new List<int>();

```

```

var evs = context.Event.ToList();
foreach (TEvent ev in evs)
{
    if (ev.ID == eventId)
        eventId_ = eventId;
}
var locations = context.Location.ToList();

foreach (TLocation loc in locations)
{
    if (loc.ID == locationId)
        locationId_ = locationId;
}

if (eventId_ == -1)
    return -1;
if (locationId_ == -1)
    return 0;

List<int> ids = new List<int>();
var eventLocations = context.EventLocation.ToList();

foreach (TEventLocation el in eventLocations)
{
    ids.Add(el.ID);
}
int Id = ids.Max(id => id) + 1;

TEventLocation eventLocation = new TEventLocation();
eventLocation.ID = Id;
eventLocation.LocationId = locationId_;
eventLocation.EventId = eventId_;

context.EventLocation.Add(eventLocation);
context.SaveChanges();
return 1;
} // Додавання нової пари
public int DeleteEventLocation(int index)
{
    List<int> ids = new List<int>();
    var eventLocations = context.EventLocation.ToList();
    foreach (TEventLocation el in eventLocations)
    {
        ids.Add(el.ID);
    }
    if (!ids.Contains(index))
    {
        return -1;
    }
    TEventLocation? eventLocation =
context.EventLocation.Find(index);
    context.EventLocation.Remove(eventLocation);
    context.SaveChanges();
    return 1;
} // Видалення пари
public int UpdateEventLocation(int eventId, int locationId, int id)
{
    int eventId_ = -1;
    int locationId_ = -1;
    List<int> idsEvent = new List<int>();
    List<int> idsLocation = new List<int>();
    var evs = context.Event.ToList();
    foreach (TEvent ev in evs)

```

```

        {
            if (ev.ID == eventId)
                eventId_ = eventId;
        }
        var locations = context.Location.ToList();
        foreach (TLocation loc in locations)
        {
            if (loc.ID == locationId)
                locationId_ = locationId;
        }
        if (eventId_ == -1)
            return -1;
        if (locationId_ == -1)
            return 0;
        List<int> ids = new List<int>();
        var eventLocations = context.EventLocation.ToList();
        foreach (TEventLocation el in eventLocations)
        {
            ids.Add(el.ID);
        }
        if (!ids.Contains(id))
            return -2;
        TEventLocation? eventLocation = context.EventLocation.Find(id);
        eventLocation.LocationId = locationId;
        eventLocation.EventId = eventId;

        context.SaveChangesAsync();
        context = new ContextClass();
        return 1;
    } // Оновлення пари
    public List<TEventLocation> SearchEventLocationByEvent(int id)
    {
        var pd = context.EventLocation.Where(o => o.EventId ==
id).ToList();

        return pd.ToList();
    } // Пошук пари за івентом
    public List<TEventLocation> SearchEventLocationByLocation(int id)
    {
        var pd = context.EventLocation.Where(e => e.LocationId == id)
            .ToList();
        return pd.ToList();
    } // пошук пари за локацією
    public void GenerateEventLocation(int n)
    {
        for (int i = 0; i < n; i++)
        {
            List<int> idsEvent = new List<int>();
            List<int> idsLocation = new List<int>();
            var evs = context.Event.ToList();
            foreach (TEvent ev in evs)
            {
                idsEvent.Add(ev.ID);
            }
            var locations = context.Location.ToList();
            foreach (TLocation loc in locations)
            {
                idsLocation.Add(loc.ID);
            }
            List<int> ids = new List<int>();
            var eventLocations = context.EventLocation.ToList();
            foreach (TEventLocation el in eventLocations)
            {
                ids.Add(el.ID);
            }
        }
    }

```



```

    }
    Random rnd = new Random();
    int Id = ids.Max(id => id) + 1;
    int eventId_ = idsEvent[rnd.Next(0, idsEvent.Count)];
    int locationId_ = idsLocation[rnd.Next(0,
idsLocation.Count)];

    TEventLocation eventLocation = new TEventLocation();
    eventLocation.ID = Id;
    eventLocation.EventId = eventId_;
    eventLocation.LocationId = locationId_;

    context.EventLocation.Add(eventLocation);
    context.SaveChanges();
}
} // Генерація нових пар івентів та локацій
}

```

Завдання 2

Створення таблиці:

```

create table text1(
body text
);

```

Вставка та генерація даних в таблиці:

```

insert into text1
Select
md5(random()::text)
from(select * from generate_series(1, 100000) as id) as x;

```

The screenshot shows a PostgreSQL query editor interface. At the top, there is a toolbar with icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, displaying a single SQL statement: `1 create table text1(body text);`. The 'Messages' tab is also visible, showing the output: `CREATE TABLE` and `Query returned successfully in 100 msec.`. The 'Data Output' and 'Notifications' tabs are also present but empty.

The screenshot shows a PostgreSQL query editor interface. At the top, the connection is set to 'Inventory/postgres@PostgreSQL 15'. The toolbar is identical to the first screenshot. The 'Query' tab is active, displaying a multi-line SQL statement: `1 insert into text1`, `2 Select`, `3 md5(random())::text)`, `4 from(select * from generate_series(1, 100000) as id) as x;`, and `5`. The 'Messages' tab is active, showing the output: `INSERT 0 100000` and `Query returned successfully in 7 secs 45 msec.`. The 'Data Output' and 'Notifications' tabs are also present but empty.

BTREE

Час виконання без індексу:

No limit

Query Query History

1

EXPLAIN analyze select count(*) from text1 where body = 'bec7cbfda722c3cab4f6b75dace22db6';

Data Output Messages Notifications

QUERY PLAN

text

1

Aggregate (cost=404.02..404.03 rows=1 width=8) (actual time=2.047..2.048 rows=1 loops=1)

2

-> Bitmap Heap Scan on text1 (cost=400.01..404.02 rows=1 width=0) (actual time=2.040..2.040 rows=0 loops=1)

3

Recheck Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)

4

-> Bitmap Index Scan on textgin (cost=0.00..400.01 rows=1 width=0) (actual time=2.035..2.035 rows=0 loop...

5

Index Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)

6

Planning Time: 3.232 ms

7

Execution Time: 2.921 ms

Час виконання з використанням індексу:

Query Query History

1

CREATE INDEX textbtree

2

ON text1

3

USING btree

4

(body);

5

explain analyze select count(*) from text1 where body = 'bec7cbfda722c3cab4f6b75dace22db6';

Data Output Messages Notifications

QUERY PLAN

text

1

Aggregate (cost=4.44..4.45 rows=1 width=8) (actual time=0.160..0.161 rows=1 loops=1)

2

-> Index Only Scan using textbtree on text1 (cost=0.42..4.44 rows=1 width=0) (actual time=0.155..0.155 rows=0 loop...

3

Index Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)

4

Heap Fetches: 0

5

Planning Time: 4.937 ms

6

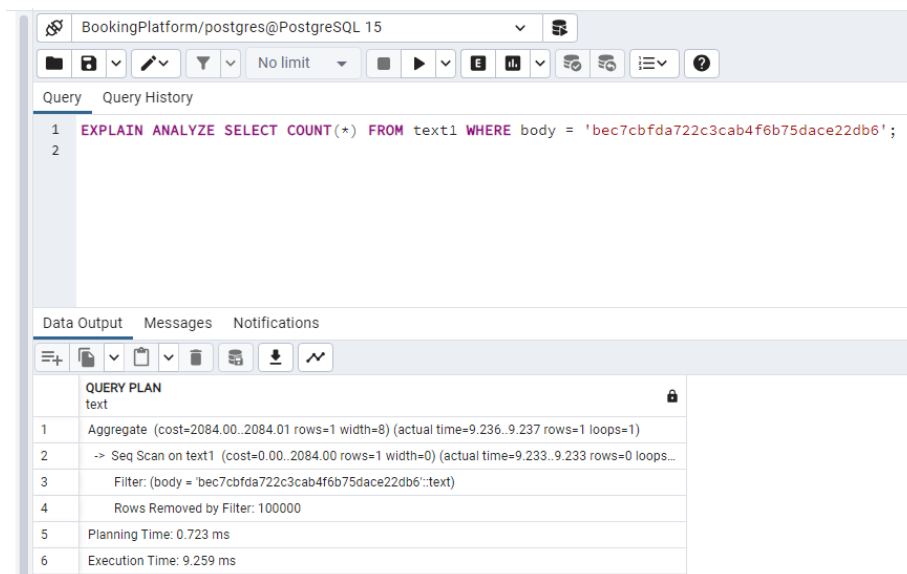
Execution Time: 0.148 ms

Індекс В-дерево (B-tree) — це структура даних, яка використовується в базах даних для швидкого пошуку, вставки і видалення даних. В-дерева є ефективними для операцій здебільшого в невідсортованих масивах або наборах даних, таких як сховища ключ-значення, бази даних та файлові системи. Основні характеристики В-дерева: Балансованість: В-дерево завжди

залишається балансованим, що означає, що відстань від кореня до будь-якого листа в дереві має бути приблизною однаковою. Це досягається шляхом перерозподілу ключів при вставці або видаленні. Висока ефективність пошуку: Благодаря балансованості і структурі В-дерева, час пошуку в середньому $O(\log n)$, де n - кількість ключів в дереві. Підтримка діапазонних запитів: В-дерева дозволяють ефективно виконувати діапазонні запити, так як діапазон ключів може бути легко знайдений в порівнянні з бінарними деревами пошуку (BST). Підтримка вставки та видалення: Операції вставки і видалення можуть бути виконані досить ефективно, оскільки дерево може легко адаптуватися до змін. В-дерева широко використовуються в базах даних для індексації даних і покращення швидкодії операцій пошуку. Їх використання дозволяє прискорити доступ до даних, зменшити кількість блокувань і покращити загальну продуктивність бази даних.

HASH

Час виконання без індексу:



The screenshot shows a PostgreSQL query execution interface. The query is: `EXPLAIN ANALYZE SELECT COUNT(*) FROM text1 WHERE body = 'bec7cbfda722c3cab4f6b75dace22db6';`. The query plan shows a sequential scan on text1 with a filter for the specific body value. The execution time is 9.259 ms.

	QUERY PLAN
1	Aggregate (cost=2084.00..2084.01 rows=1 width=8) (actual time=9.236..9.237 rows=1 loops=1)
2	-> Seq Scan on text1 (cost=0.00..2084.00 rows=1 width=0) (actual time=9.233..9.233 rows=0 loops=1)
3	Filter: (body = 'bec7cbfda722c3cab4f6b75dace22db6':text)
4	Rows Removed by Filter: 100000
5	Planning Time: 0.723 ms
6	Execution Time: 9.259 ms

Час виконання з використанням індексу:

The screenshot shows a database query interface with a toolbar at the top containing icons for file operations, filters, and execution. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query:

```
1 CREATE INDEX hashindex ON text1 USING hash(body);
2 EXPLAIN ANALYZE SELECT COUNT(*) FROM text1 WHERE body = 'bec7cbfda722c3cab4f6b75dace22db6';
3
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the query plan results:

	QUERY PLAN
1	Aggregate (cost=8.02..8.03 rows=1 width=8) (actual time=0.021..0.021 rows=1 loops=1)
2	-> Index Scan using hashindex on text1 (cost=0.00..8.02 rows=1 width=0) (actual time=0.018..0.018 rows=0 loop...)
3	Index Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)
4	Planning Time: 0.865 ms
5	Execution Time: 0.039 ms

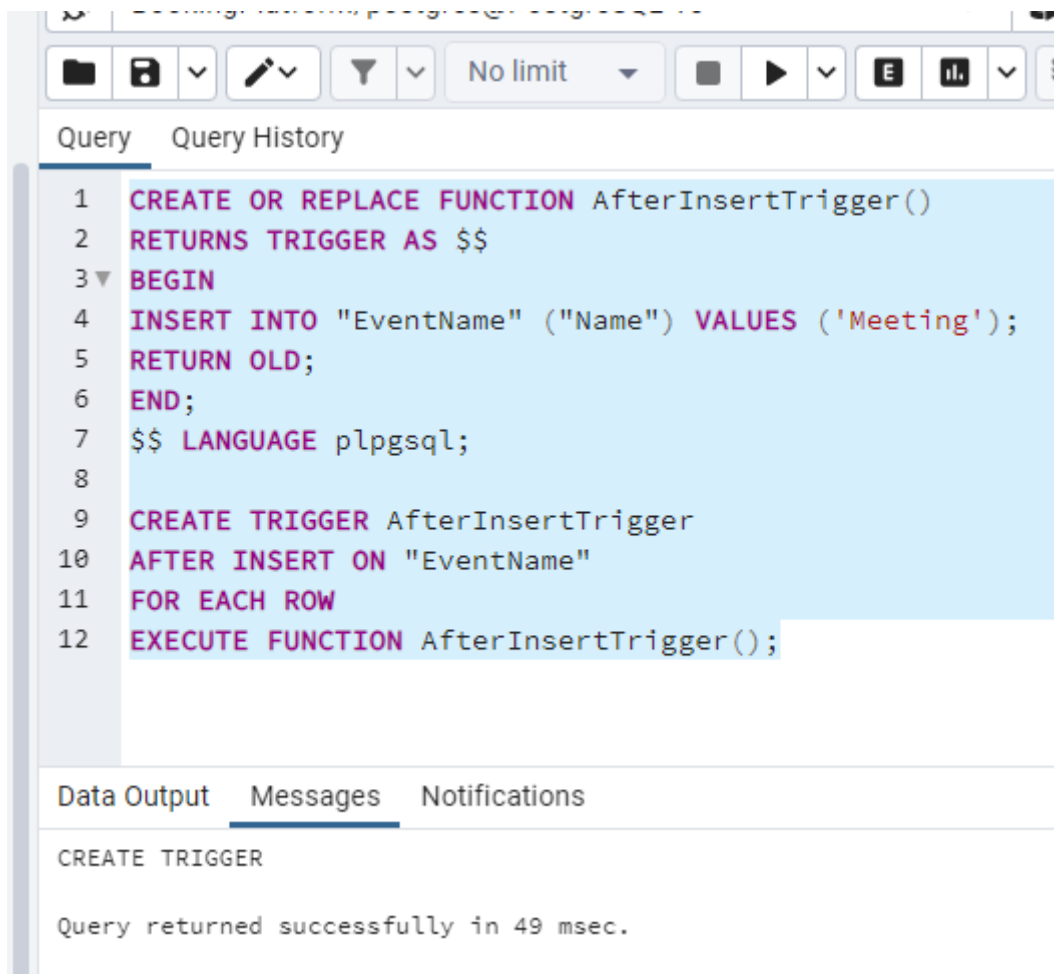
Хеш-індекс — це тип індексу в базі даних, який використовує хеш-функції для швидкого пошуку конкретного ключа в таблиці. Основна ідея полягає в тому, що величина (хеш-код), отримана застосуванням хеш-функції до ключа, використовується для визначення позиції, де має знаходитися відповідний запис в таблиці. Основні характеристики хеш-індексів: Хеш-функція: Хеш-індекс використовує хеш-функцію для генерації унікального коду для кожного ключа. Цей хеш-код служить величиною, яка визначає позицію запису в індексі або основній таблиці. Швидкий пошук: Основною перевагою хеш-індексу є швидкість пошуку, оскільки, в середньому, час пошуку є константним ($O(1)$). Однак в реальних умовах може бути необхідно враховувати випадки колізій (два різних ключі мають однаковий хеш-код). Колізії: Колізії виникають, коли два різних ключі мають однаковий хеш-код. Для вирішення колізій використовуються різні техніки, такі як відкрите адресування (всі ключі, які мають колізію, розташовуються в одній таблиці), ланцюжки (створення списків для розташування всіх ключів з однаковим хешем) та інші. Використання пам'яті: Хеш-індекси можуть вимагати значно менше пам'яті порівняно з іншими видами індексів, оскільки вони не зберігають велику кількість додаткової інформації. Непідтримка діапазонних запитів: Однак хеш-індекс не підтримує ефективний пошук для діапазонів

ключів, оскільки вони розташовані випадковим чином. Хеш-індекси є ефективними для оперативного пошуку точних значень великої кількості даних, але важливо уникати ситуацій, коли колізії стають частими, так як це може суттєво вплинути на швидкість виконання операцій.

Завдання 3

after insert

Код тригера:



```
1 CREATE OR REPLACE FUNCTION AfterInsertTrigger()
2 RETURNS TRIGGER AS $$
3 BEGIN
4 INSERT INTO "EventName" ("Name") VALUES ('Meeting');
5 RETURN OLD;
6 END;
7 $$ LANGUAGE plpgsql;
8
9 CREATE TRIGGER AfterInsertTrigger
10 AFTER INSERT ON "EventName"
11 FOR EACH ROW
12 EXECUTE FUNCTION AfterInsertTrigger();
```

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query' and 'Query History'. The main editor area displays the SQL code for creating a trigger. The code is as follows:

```
1 CREATE OR REPLACE FUNCTION AfterInsertTrigger()
2 RETURNS TRIGGER AS $$
3 BEGIN
4 INSERT INTO "EventName" ("Name") VALUES ('Meeting');
5 RETURN OLD;
6 END;
7 $$ LANGUAGE plpgsql;
8
9 CREATE TRIGGER AfterInsertTrigger
10 AFTER INSERT ON "EventName"
11 FOR EACH ROW
12 EXECUTE FUNCTION AfterInsertTrigger();
```

Below the code editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the following output:

```
CREATE TRIGGER

Query returned successfully in 49 msec.
```

Тестування роботи тригера:

Query Query History

```

1 SELECT * FROM public."EventName"
2 ORDER BY "ID" ASC

```

Data Output Messages Notifications

	ID [PK] integer	Name character varying (25)
1	1	Party
2	2	New Name
3	3	Wedding
4	4	Charity evening
5	5	Fest
6	6	XCR TNY
7	7	JBG LGQ
8	8	VCY PXF
9	9	mLuFy9m5kb
10	10	D5mNE43zY2

Data Output Messages Notifications

	ID [PK] integer	Name character varying (25)
1	1	Party
2	2	New Name
3	3	Wedding
4	4	Charity evening
5	5	Fest
6	6	XCR TNY
7	7	JBG LGQ
8	8	VCY PXF
9	9	mLuFy9m5kb
10	10	D5mNE43zY2
11	11	New Event
12	12	Meeting

Trigger Functions (2)

- afterinserttrigger()
- afterupdatetrigger()

Types

afret update

Код триггеру:

BookingPlatform/postgres@PostgreSQL 15

Query Query History

```
1 CREATE OR REPLACE FUNCTION AfterUpdateTrigger()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     UPDATE "EventName" SET "Name" = 'Fest' WHERE "ID" = 5;
5     RETURN OLD;
6 END;
7 $$ LANGUAGE plpgsql;
8
9 CREATE TRIGGER AfterUpdateTrigger
10 BEFORE UPDATE ON "EventName"
11 FOR EACH ROW
12 EXECUTE FUNCTION AfterUpdateTrigger();
```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 53 msec.

Тестування роботи триггеру:

BookingPlatform/postgres@PostgreSQL 15

Query Query History

```
1 UPDATE "EventName" SET "Name" = 'New Name' WHERE "ID" = 2;
2
```


Query		Query History
1	SELECT * FROM public."EventName"	
2	ORDER BY "ID" ASC	

Data Output		Messages	Notifications
<div> <div>+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>📦</div> <div>⬇️</div> <div>📈</div> </div>			
	ID [PK] integer	Name character varying (25)	
1	1	Party	
2	2	New Name	
3	3	Wedding	
4	4	Charity evening	
5	5	Fest	
6	6	XCR TNY	
7	7	JBG LGQ	
8	8	VCY PXF	
9	9	mLuFy9m5kb	
10	10	D5mNE43zY2	

Завдання 4

REPEATABLE READ

базами даних (СУБД). Цей рівень забезпечує високий рівень ізоляції транзакцій, забезпечуючи виключність читань, але водночас дозволяючи "фантомні оновлення".

Основні характеристики рівня ізоляції REPEATABLE READ:

Виділений час життя транзакції: Транзакція на рівні REPEATABLE READ має свій власний виділений час життя, і весь час цієї транзакції транзакційний простір даних залишається консистентним.

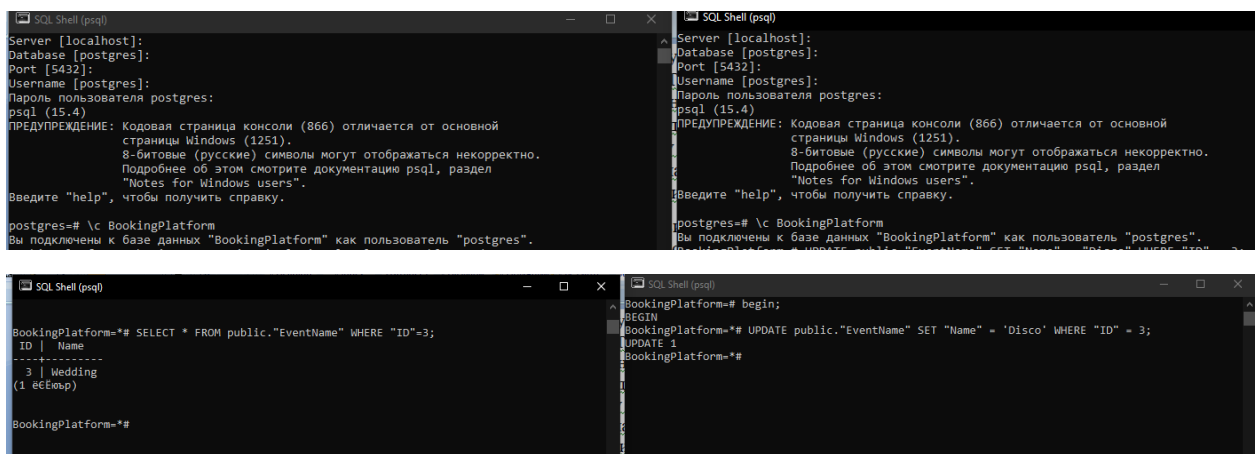
1. Блокування для читання і запису: Транзакція, яка працює на рівні REPEATABLE READ, блокує рядки даних, які вона читає, а також ті, що задіяні у виразі WHERE у SELECT-запитах. Це має на меті

уникнення "грязних читань", "неповторних читань" і "фантомних вставок".

2. Запобігання неповторним читанням: Транзакція на рівні REPEATABLE READ гарантує, що якщо вона читає дані певного рядка, цей рядок залишатиметься незмінним протягом усього часу життя транзакції, навіть якщо інші транзакції модифікують цей рядок.
3. Фантомні оновлення: Однак REPEATABLE READ не гарантує відсутності "фантомних оновлень". Іншими словами, хоча транзакція блокує рядки, які вона читає, нові рядки можуть бути додані в результат виконання SELECT-запиту протягом транзакції. Це може призвести до того, що одна і та ж сама транзакція при повторному виконанні одного і того ж самого запиту побачить нові рядки.

REPEATABLE READ є компромісом між високим рівнем ізоляції та продуктивністю, і він підходить для багатьох випадків використання, де важлива стійкість до "грязних читань" і "неповторних читань", але можливі "фантомні оновлення".

Спочатку треба під'єднатись до БД.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (15.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \c BookingPlatform
Вы подключены к базе данных "BookingPlatform" как пользователь "postgres".

SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (15.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \c BookingPlatform
Вы подключены к базе данных "BookingPlatform" как пользователь "postgres".

BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=#

BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----+-----
  3 | Wedding
(1 row)

BookingPlatform=#
```

SERIALIZABLE

SERIALIZABLE - це найвищий рівень ізоляції транзакцій в базах даних. Цей рівень забезпечує повну ізоляцію транзакцій, що означає, що жодна транзакція не може бачити змін, внесених іншими транзакціями, поки вони не будуть закінчені. Рівень ізоляції SERIALIZABLE запобігає всім видам аномалій, таких як "грязне читання", "неповторне читання", "фантомні читання", "грязні оновлення" і "фантомні оновлення".

Основні характеристики рівня ізоляції SERIALIZABLE:

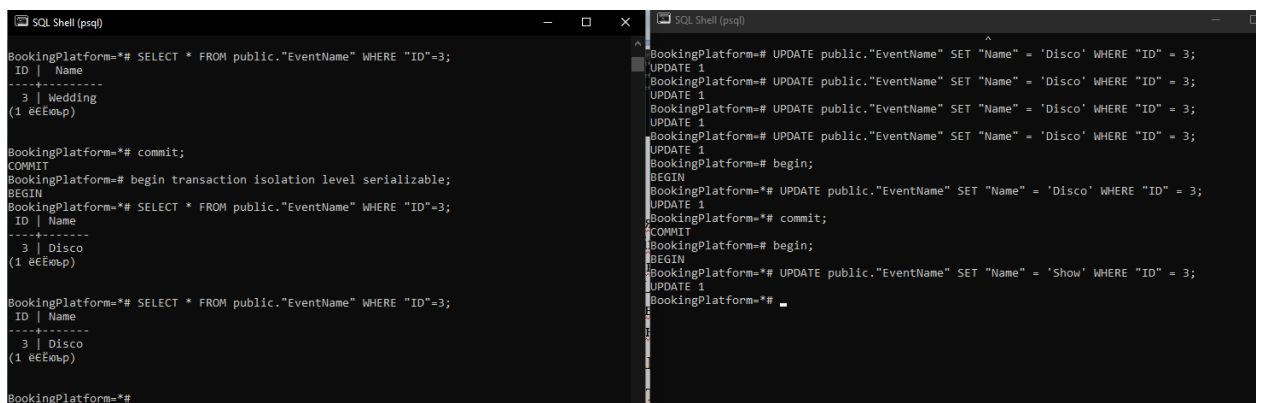
1. Повна блокада для читання і запису: Транзакція, яка працює на рівні SERIALIZABLE, блокує всі рядки даних, з якими вона взаємодіє, для

читання і запису. Це забезпечує повну ізоляцію від інших транзакцій і уникнення всіх видів конфліктів.

2. Запобігання фантомним читанням і вставкам: **SERIALIZABLE** гарантує, що транзакції не побачать нових рядків, які були вставлені іншими транзакціями, і не зможуть вставити нові рядки в той самий набір даних після читання. Це включає в себе запобігання "фантомним читанням" і "фантомним вставкам".
3. Недопущення фантомних оновлень: **SERIALIZABLE** також запобігає "фантомним оновленням", гарантуючи, що транзакції не зможуть оновлювати дані, які були змінені іншими транзакціями після початкового читання.

Стабільність порядку читання: Цей рівень також гарантує стабільність порядку читання, тобто результати **SELECT**-запитів залишаються незмінними в межах транзакції.

Хоча **SERIALIZABLE** надає максимальний рівень ізоляції, він може також призвести до значного блокування ресурсів і впливати на продуктивність системи. Тому обирайте його тільки в тих випадках, коли висока ізоляція є критичною, і ви готові пожертвувати частиною продуктивності.



```
SQL Shell (psql)
BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Wedding
(1 row)

BookingPlatform=# commit;
COMMIT
BookingPlatform=# begin transaction isolation level serializable;
BEGIN
BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Disco
(1 row)

BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Disco
(1 row)

BookingPlatform=#

SQL Shell (psql)
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# commit;
COMMIT
BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Show' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=#
```

READ COMMITTED

READ COMMITTED - це рівень ізоляції транзакцій в системах управління базами даних (СУБД). Цей рівень забезпечує менший рівень ізоляції порівняно з **SERIALIZABLE**, але при цьому він є більш продуктивним і менше схильним до блокування ресурсів.

Основні характеристики рівня ізоляції READ COMMITTED:

1. Блокування для запису, але не для читання: Транзакція, яка працює на рівні **READ COMMITTED**, блокує рядки даних для запису, коли вона їх

- змінює, але не блокує їх для читання. Це означає, що інші транзакції можуть читати дані, навіть якщо вони ще не були фіксовані (збережені).
2. Недопущення "грязних читань": Транзакція READ COMMITTED не допускає "грязних читань". Це означає, що транзакція не може читати незафіксовані зміни інших транзакцій.
 3. Допущення "неповторних читань" і "фантомних вставок": Однак, READ COMMITTED дозволяє "неповторні читання", тобто можливість читання одного і того ж рядка, який був змінений іншою транзакцією після початку поточної транзакції. Також можливі "фантомні вставки" або "фантомні оновлення", тобто вставка або оновлення нових рядків в поточній транзакції.
 4. Читання фіксованих змінених даних: Транзакція може читати тільки фіксовані (збережені) зміни інших транзакцій. Якщо транзакція А змінює дані, транзакція В може прочитати тільки фіксовані дані А, а не їхні зміни.

READ COMMITTED є рівнем ізоляції, який часто використовується в багатьох системах, оскільки він надає гарний баланс між ізоляцією і продуктивністю, але слід враховувати його особливості при розробці додатків, особливо у випадках, коли критична стійкість до "грязних читань".

```
SQL Shell (psql)
3 | Disco
(1 row)

BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Disco
(1 row)

BookingPlatform=# commit;
COMMIT

BookingPlatform=# begin transaction isolation level read committed;
BEGIN
BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Show
(1 row)

BookingPlatform=# SELECT * FROM public."EventName" WHERE "ID"=3;
 ID | Name
-----
  3 | Show
(1 row)

BookingPlatform=#

SQL Shell (psql)
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Disco' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# commit;
COMMIT
BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Show' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=# commit;
COMMIT
BookingPlatform=# begin;
BEGIN
BookingPlatform=# UPDATE public."EventName" SET "Name" = 'Debates' WHERE "ID" = 3;
UPDATE 1
BookingPlatform=#
```