

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»

Виконав: студент III курсу
ФПМ групи КВ-11
Чебан М.Д.
Перевірів:

Київ – 2023

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 25

У другому завданні проаналізувати індекси BTree, GIN.

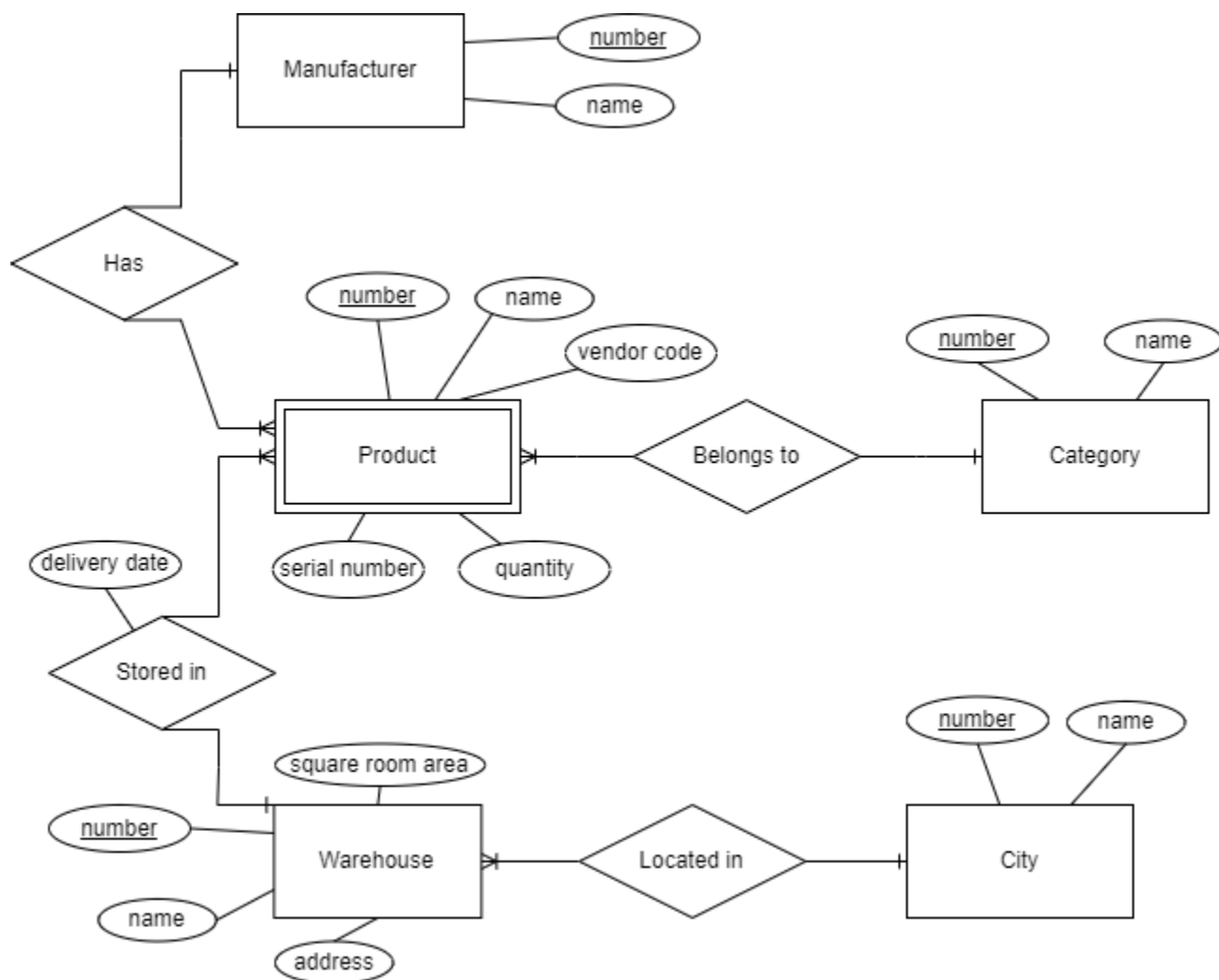
Умова для тригера – after delete, insert.

Telegram: https://t.me/M0ksem_cn

GitHub: <https://github.com/m0ksemm/Database-Theory-with-PostgreSQL>

Завдання 1

Розробка моделі «сутність-зв'язок» предметної галузі для проектування бази даних «Inventory of warehouse accounting». Предметна галузь - «Інвентаризація складського обліку».



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

Сутності з описом призначення:

Предметна галузь «Inventory of warehouse accounting» включає в себе 5 сутностей, кожна сутність містить декілька атрибутів:

1. Product (Id, name, vendor code, quantity, serial number).
2. Manufacturer (Id, name).
3. Category (Id, name).
4. Warehouse (id, name, address, square room area).
5. City (Id, name).

Сутність Product описує продукти, які зберігаються на складі. Кожний продукт має свій ідентифікатор Id, а також містить інформацію про свою назву, код постачальника, серійний номер та кількість продукту(товару).

Сутність Manufacturer описує виробника продукту. Кожний виробник має свій ідентифікатор та назву.

Сутність Category описує категорію продукту. Кожна категорія має свій ідентифікатор та назву.

Сутність Warehouse описує склади, на яких будуть зберігатись продукти. Кожний склад має свій ідентифікатор, назву, адресу, розмір площі у м.кв.

Сутність City відповідає за міста, в яких розташовані склади. Кожне місто має свій ідентифікатор та назву.

Зв'язки між сутностями:

Зв'язок між Product та Category:

Кожний товар відноситься до певної категорії. Наприклад: «Cell phones» – мобільні телефони, «Appliances» - побутова техніка, «Furniture» - меблі і т.д. Це може бути потрібно, наприклад, для того, щоб сортувати та розподіляти товар по певних групах на складах. Зв'язок 1:N – до однієї категорії може належати багато різних товарів.

Зв'язок між Product та Manufacturer:

Кожний товар має свого виробника. Простіше кажучи, свою фірму. Зв'язок 1:N – один виробник може виготовляти багато різних товарів. Товар без виробника бути не може.

Зв'язок між Product та Warehouse:

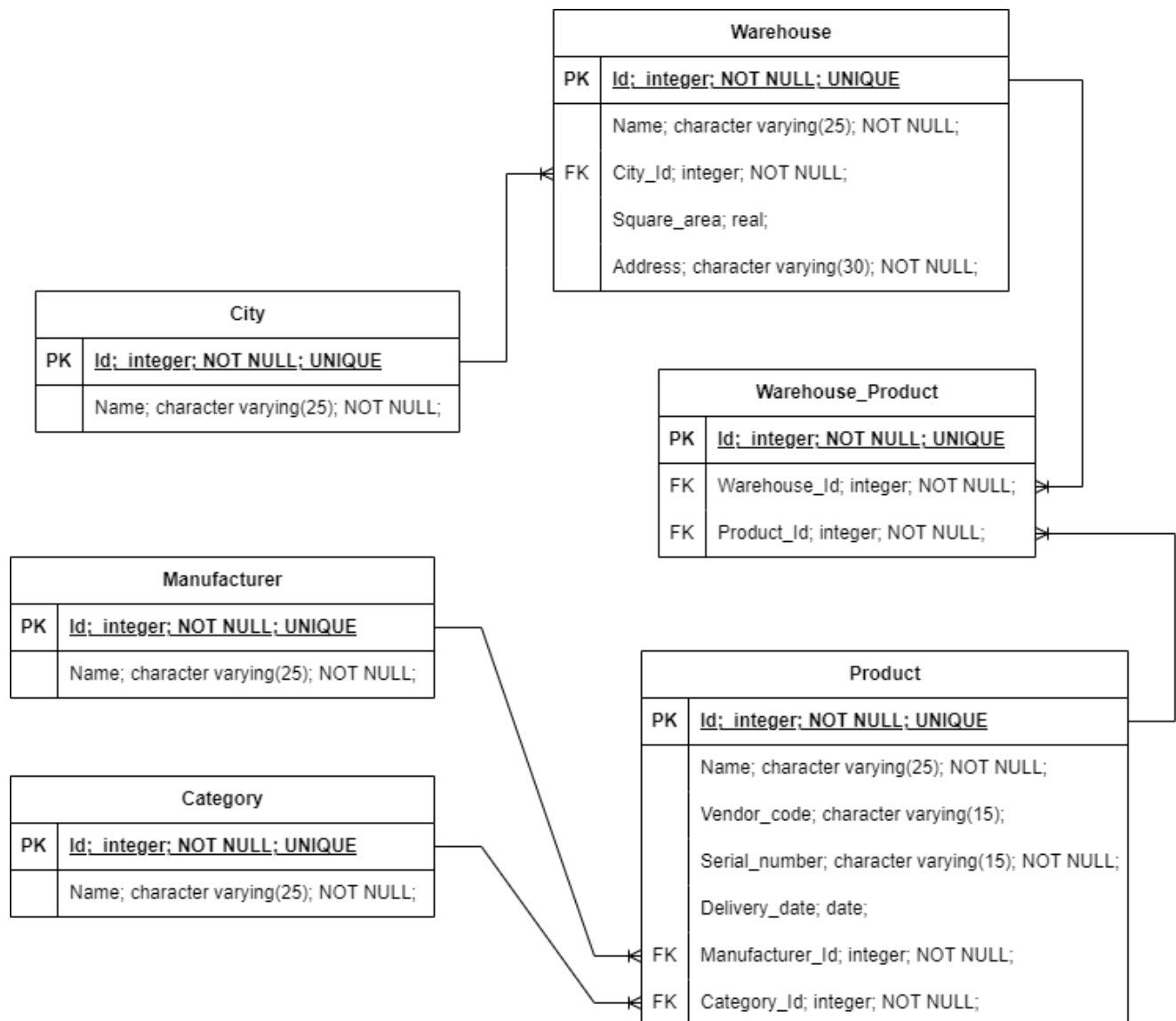
Продукти зберігаються на складі. Оскільки на одному складі зберігаються багато продуктів, зв'язок 1:N.

Зв'язок між Warehouse та City:

Склади розташовані в певних містах або селищах. Зв'язок 1:N – в одному місті може бути багато складів. Склад має обов'язково бути розташованим в якомусь місті.

Завдання № 2

Перетворення розробленої моделі «сутність-зв'язок» у схему бази даних PostgreSQL



Малюнок 2. Схема бази даних у графічному вигляді

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

```

public class Cities
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Categ
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Manuf
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Prod
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Vendor_code { get; set; }
    public string Serial_number { get; set; }
    public DateOnly Delivery_date { get; set; }
    public int Quantity { get; set; }
    public int Manufacturer_Id { get; set; }
    public int Category_Id { get; set; }
}

public class Warehouse
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int City_Id { get; set; }
    public double Square_area { get; set; }
    public string Address { get; set; }
}

public class WarehouseProduct
{
    public int Id { get; set; }
    public int Warehouse_Id { get; set; }
    public int Product_Id { get; set; }
}

```

Клас Model:

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Metadata.Internal;
using Npgsql;
using RGR.DataModels;

namespace RGR
{
    public class InventoryContext : DbContext
    {
        public DbSet<Cities> City { get; set; }
    }
}

```

```

        public DbSet<Manuf> Manufacturer { get; set; }
        public DbSet<Categ> Category { get; set; }
        public DbSet<Prod> Product { get; set; }
        public DbSet<WarehouseProduct> Warehouses_Products { get; set; }

        public DbSet<Warehouse> Warehouse { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            optionsBuilder.UseNpgsql(
"Host=localhost;Port=5432;Database=Inventory;Username=postgres;Password=011427223");
        }
    }
    public class Model_
    {
        InventoryContext inventoryContext { get; set; }
        public Model_()
        {
            inventoryContext = new InventoryContext();
        }

        #region ProductMethods
        public List<Prod> GetAllProducts()
        {
            List<Prod> products = inventoryContext.Product.ToList();

            return products;
        }
        public int InputProduct(Prod prod)
        {
            try
            {
                inventoryContext.Product.Add(prod);
                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        public int DeleteProduct(int id)
        {
            try
            {
                Prod? prod = inventoryContext.Product.FirstOrDefault(o => o.Id ==
id);

                if (prod == null)
                    return 0;
                inventoryContext.Product.Remove(prod);
                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        public int EditProduct(Prod prod, int id)
        {
            var recordToUpdate = inventoryContext.Product.Find(id);

            if (recordToUpdate != null)

```

```

        {
            try
            {
                recordToUpdate.Name = prod.Name;
                recordToUpdate.Vendor_code = prod.Vendor_code;
                recordToUpdate.Serial_number = prod.Serial_number;
                recordToUpdate.Delivery_date = prod.Delivery_date;
                recordToUpdate.Quantity = prod.Quantity;
                recordToUpdate.Manufacturer_Id = prod.Manufacturer_Id;
                recordToUpdate.Category_Id = prod.Category_Id;

                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        else return 0;
    }
    public List<Prod> ProductSearchName(string key)
    {
        var searchResults = inventoryContext.Product
            .Where(e => e.Name.Contains(key))
            .ToList();

        return searchResults.ToList();
    }
    public List<Prod> ProductSearchVendorCode(string key)
    {
        var searchResults = inventoryContext.Product
            .Where(e => e.Vendor_code.Contains(key))
            .ToList();

        return searchResults.ToList();
    }
    public List<Prod> ProductSearchSerialNumber(string key)
    {
        var searchResults = inventoryContext.Product
            .Where(e => e.Serial_number.Contains(key))
            .ToList();

        return searchResults.ToList();
    }
    public List<Prod> ProductSearchQuantity(int min, int max)
    {
        var searchResults = inventoryContext.Product
            .Where(e => e.Quantity >= min && e.Quantity <= max)
            .ToList();

        return searchResults.ToList();
    }
}

#endregion

#region ManufacturerMethods
public List<Manuf> GetAllManufacturers()
{
    List<Manuf> list = inventoryContext.Manufacturer.ToList();

    return list;
}

public int InputManufacturer(Manuf man)

```



```

{
    List<int> ids = new List<int>();
    List<Manuf> list = inventoryContext.Manufacturer.ToList();
    foreach (Manuf m in list)
    {
        if (m.Name == man.Name)
            return 0;
    }
    foreach (Manuf m in list)
    {
        ids.Add(m.Id);
    }
    man.Id = ids.Max(id => id) + 1;

    inventoryContext.Manufacturer.Add(man);
    inventoryContext.SaveChanges();

    return 1;
}
public int EditManufacturer(Manuf manuf, int id)
{
    var recordToUpdate = inventoryContext.Manufacturer.Find(id);

    if (recordToUpdate != null)
    {
        recordToUpdate.Name = manuf.Name;
        recordToUpdate.Id = id;

        inventoryContext.SaveChanges();
        return 1;
    }
    else return 0;
}
public int DeleteManufacturer(int id)
{
    var recordToDelete = inventoryContext.Manufacturer.Find(id);

    if (recordToDelete != null)
    {
        try
        {
            inventoryContext.Manufacturer.Remove(recordToDelete);
            inventoryContext.SaveChanges();
            return 1;
        }
        catch
        {
            return 2;
        }
    }
    else return 0;
}
public List<Manuf> ManufacturerSearch(string key)
{
    var searchResults = inventoryContext.Manufacturer
        .Where(e => e.Name.Contains(key))
        .ToList();

    return searchResults.ToList();
}
#endregion

#region CategoryMethods
public List<Categ> GetAllCategories()
{

```

```

        List<Categ> list = inventoryContext.Category.ToList();

        return list;
    }

    public int InputCategory(Categ cat)
    {
        List<int> ids = new List<int>();
        List<Categ> list = inventoryContext.Category.ToList();
        foreach (Categ m in list)
        {
            if (m.Name == cat.Name)
                return 0;
        }
        foreach (Categ m in list)
        {
            ids.Add(m.Id);
        }
        cat.Id = ids.Max(id => id) + 1;

        inventoryContext.Category.Add(cat);
        inventoryContext.SaveChanges();

        return 1;
    }

    public int EditCategory(Categ cat, int id)
    {
        var recordToUpdate = inventoryContext.Category.Find(id);

        if (recordToUpdate != null)
        {
            recordToUpdate.Name = cat.Name;
            recordToUpdate.Id = id;

            inventoryContext.SaveChanges();
            return 1;
        }
        else return 0;
    }

    public int DeleteCategory(int id)
    {
        var recordToDelete = inventoryContext.Category.Find(id);

        if (recordToDelete != null)
        {
            try
            {
                inventoryContext.Category.Remove(recordToDelete);
                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        else return 0;
    }

    public List<Categ> CategorySearch(string key)
    {
        var searchResults = inventoryContext.Category
            .Where(e => e.Name.Contains(key))
            .ToList();

        return searchResults.ToList();
    }

```

```

}

#endregion

#region CityMethods
public List<Cities> GetAllCities()
{
    List<Cities> list = inventoryContext.City.ToList();

    return list;
}

public int InputCity(Cities man)
{
    List<int> ids = new List<int>();
    List<Cities> list = inventoryContext.City.ToList();
    foreach (Cities m in list)
    {
        if (m.Name == man.Name)
            return 0;
    }
    foreach (Cities m in list)
    {
        ids.Add(m.Id);
    }
    man.Id = ids.Max(id => id) + 1;

    inventoryContext.City.Add(man);
    inventoryContext.SaveChanges();

    return 1;
}

public int EditCity(Cities city, int id)
{
    var recordToUpdate = inventoryContext.City.Find(id);

    if (recordToUpdate != null)
    {
        recordToUpdate.Name = city.Name;
        recordToUpdate.Id = id;

        inventoryContext.SaveChanges();
        return 1;
    }
    else return 0;
}

public int DeleteCity(int id)
{
    var recordToDelete = inventoryContext.City.Find(id);

    if (recordToDelete != null)
    {
        try
        {
            inventoryContext.City.Remove(recordToDelete);
            inventoryContext.SaveChanges();
            return 1;
        }
        catch
        {
            return 2;
        }
    }
}

```

```

    }
    else return 0;
}
public List<Cities> CitiesSearch(string key)
{
    var searchResults = inventoryContext.City
        .Where(e => e.Name.Contains(key))
        .ToList();

    return searchResults.ToList();
}

#endregion

#region WarehouseMethods
public List<Warehouse> GetAllWarehouses()
{
    List<Warehouse> warehouses = inventoryContext.Warehouse.ToList();
    return warehouses;
}
public int InputWarehouse(Warehouse warehouse)
{
    List<int> ids = new List<int>();
    List<Warehouse> warehouses = inventoryContext.Warehouse.ToList();
    foreach (Warehouse w in warehouses)
    {
        ids.Add(w.Id);
    }
    warehouse.Id = ids.Max(id => id) + 1;
    try
    {
        inventoryContext.Warehouse.Add(warehouse);
        inventoryContext.SaveChanges();
        return 1;
    }
    catch
    {
        return 2;
    }
}
public int DeleteWarehouse(int id)
{
    var recordToDelete = inventoryContext.Warehouse.Find(id);

    if (recordToDelete != null)
    {
        try
        {
            inventoryContext.Warehouse.Remove(recordToDelete);
            inventoryContext.SaveChanges();
            return 1;
        }
        catch
        {
            return 2;
        }
    }
    else return 0;
}
public int EditWarehouse(Warehouse warehouse, int id)
{
    var recordToUpdate = inventoryContext.Warehouse.Find(id);

    if (recordToUpdate != null)
    {

```

```

        try
        {
            recordToUpdate.Name = warehouse.Name;
            recordToUpdate.City_Id = warehouse.City_Id;
            recordToUpdate.Square_area = warehouse.Square_area;
            recordToUpdate.Address = warehouse.Address;

            inventoryContext.SaveChanges();
            return 1;
        }
        catch
        {
            return 2;
        }
    }
    else return 0;
}

public List<Warehouse> WarehouseSearchName(string key)
{
    var searchResults = inventoryContext.Warehouse
        .Where(e => e.Name.Contains(key))
        .ToList();

    return searchResults.ToList();
}

public List<Warehouse> WarehouseSearchAddress(string key)
{
    var searchResults = inventoryContext.Warehouse
        .Where(e => e.Name.Contains(key))
        .ToList();

    return searchResults.ToList();
}

public List<Warehouse> WarehouseSearchSquareArea(double min, double max)
{
    var searchResults = inventoryContext.Warehouse
        .Where(e => e.Square_area >= min && e.Square_area <= max)
        .ToList();

    return searchResults.ToList();
}
#endregion

#region WarehouseProductMethods
public List<WarehouseProduct> GetAllWarehousesProducts()
{
    List<WarehouseProduct> list =
inventoryContext.Warehouses_Products.ToList();

    return list;
}

public int InputWarehouseProduct(WarehouseProduct prod)
{
    List<int> ids = new List<int>();
    List<WarehouseProduct> list =
inventoryContext.Warehouses_Products.ToList();
    foreach (WarehouseProduct wp in list)
    {
        ids.Add(wp.Id);
    }
    try

```

```

        {
            prod.Id = ids.Max(id => id) + 1;
            inventoryContext.Warehouses_Products.Add(prod);
            inventoryContext.SaveChanges();
            return 1;
        }
        catch
        {
            return 0;
        }
    }

    public int DeleteWarehouseProduct(int id)
    {
        var recordToDelete = inventoryContext.Warehouses_Products.Find(id);

        if (recordToDelete != null)
        {
            try
            {
                inventoryContext.Warehouses_Products.Remove(recordToDelete);
                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        else return 0;
    }

    public int EditWarehouseProduct(WarehouseProduct warprod, int id)
    {
        var recordToUpdate = inventoryContext.Warehouses_Products.Find(id);

        if (recordToUpdate != null)
        {
            try
            {
                recordToUpdate.Warehouse_Id = warprod.Warehouse_Id;
                recordToUpdate.Product_Id = warprod.Product_Id;

                inventoryContext.SaveChanges();
                return 1;
            }
            catch
            {
                return 2;
            }
        }
        else return 0;
    }

    public List<WarehouseProduct> WarehouseProductSearchW(int warId)
    {
        var searchResults = inventoryContext.Warehouses_Products
            .Where(e => e.Warehouse_Id == warId)
            .ToList();

        return searchResults.ToList();
    }

    public List<WarehouseProduct> WarehouseProductSearchP(int prodId)
    {

```

```

        var searchResults = inventoryContext.Warehouses_Products
            .Where(e => e.Product_Id == prodId)
            .ToList();

        return searchResults.ToList();
    }
    #endregion
}
}

```

Завдання 2

Створення таблиці:

```

create table text1(
    body text
);

```

Вставка та генерація даних:

```

insert into text1

```

```

Select

```

```

md5(random()::text)

```

```

from(select * from generate_series(1, 100000) as id) as x;

```



Inventory/postgres@PostgreSQL 15

Query Query History

```

1 insert into text1
2 Select
3 md5(random()::text)
4 from(select * from generate_series(1, 100000) as id) as x;
5

```

Data Output Messages Notifications

INSERT 0 100000

Query returned successfully in 7 secs 45 msec.

BTREE

Час виконання без індексу:

Query Query History

```

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

```

Data Output Messages Notifications

QUERY PLAN	
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
<div> <div>+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>🗑️</div> <div>🔄</div> <div>⬇️</div> <div>📈</div> </div>			
	QUERY PLAN text <div>🔒</div>		
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: 0.198 ms		

BTREE (Binary Tree) - це особливий тип індексу в PostgreSQL, спроектований для ефективного управління та пошуку даних. В основі його функціонування лежить структура бінарного дерева, де кожен вузол містить ключ та вказівники на лівого та правого нащадка.

Дані в таблиці організовані в бінарне дерево, де кожен вузол представляє собою ключ і вказівники на два піддерева. В результаті операцій вставки, оновлення та видалення дані розташовуються у відсортованому порядку, що дозволяє ефективно використовувати бінарне дерево для швидкого пошуку конкретних значень.

Кожен вузол бінарного дерева має властивість бути меншим за всі ключі в його правому піддереві та більшим за всі ключі в його лівому піддереві, забезпечуючи швидкий доступ до конкретних значень через порівняння ключів.

GIN

Час виконання без індексу:

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=1)
5	Index Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6':text)
6	Planning Time: 3.232 ms
7	Execution Time: 2.921 ms

Час виконання з використанням індексу:

FitnessTracker/postgres@PostgreSQL 16

No limit

Query
Query History

```

1 CREATE EXTENSION IF NOT EXISTS pg_trgm;
2 CREATE INDEX textgin ON text1 USING gin(body gin_trgm_ops);
3 explain analyze select count(*) from text1 where body = 'bec7cbfda722c3cab4f6b75dace22db6';
4

```

Data Output
Messages
Notifications

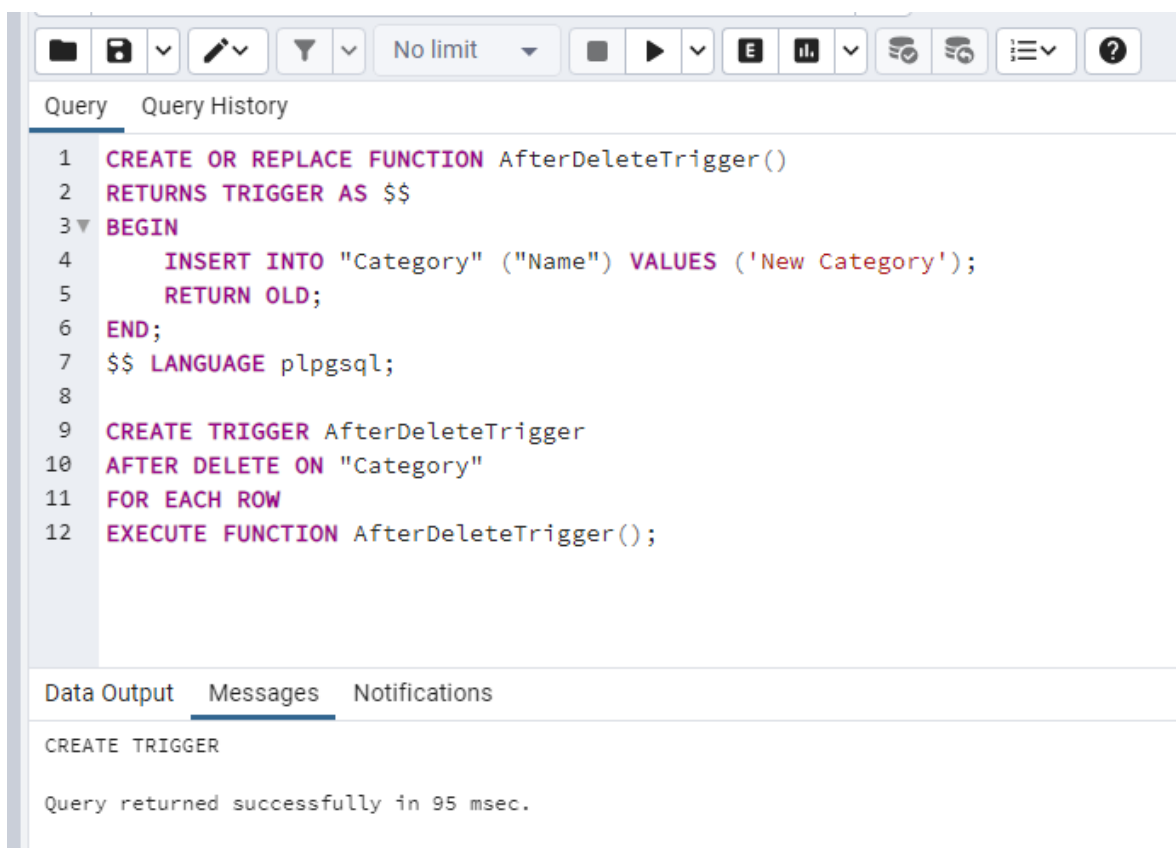
	QUERY PLAN	
	text	
1	Aggregate (cost=435.25..435.26 rows=1 width=8) (actual time=0.680..0.681 rows=1 loops=1)	
2	-> Bitmap Heap Scan on text1 (cost=431.23..435.25 rows=1 width=0) (actual time=0.675..0.676 rows=1 loops=1)	
3	Recheck Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)	
4	Heap Blocks: exact=1	
5	-> Bitmap Index Scan on textgin (cost=0.00..431.23 rows=1 width=0) (actual time=0.664..0.664 rows=1 loop...	
6	Index Cond: (body = 'bec7cbfda722c3cab4f6b75dace22db6'::text)	
7	Planning Time: 1.595 ms	
8	Execution Time: 0.790 ms	

Індекс GIN призначений для швидкого пошуку, але його ефективність може бути обмеженою при великому об'ємі однотипних даних або запитах, які не використовують індексовані стовпці. Неправильна конфігурація запитів або часті зміни даних можуть призвести до швидкого застаріння індексу, що зменшить його ефективність. Для оптимізації використання індексів слід ретельно переглянути структуру даних та оптимізувати запити для використання індексованих полів.

Завдання 3

after delete

Код триггеру:



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

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 95 msec.

Тестування роботи триггеру:

```

1 SELECT * FROM public."Category"
2 ORDER BY "Id" ASC

```

Data Output Messages Notifications

	Id [PK] integer	Name character varying (25)
1	1	Furniture
2	2	Appliances
3	3	Cell phones
4	5	New Category

after insert

Код триггеру:

```

1 CREATE OR REPLACE FUNCTION AfterInsertTrigger()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     DELETE FROM "Category" WHERE "Id" = OLD."Id";
5     RETURN OLD;
6 END;
7 $$ LANGUAGE plpgsql;
8
9 CREATE TRIGGER AfterInsertTrigger
10 AFTER DELETE ON "Category"
11 FOR EACH ROW
12 EXECUTE FUNCTION AfterInsertTrigger();

```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 127 msec.

Тестування роботи триггеру:

1 SELECT * FROM public."Category"
2 ORDER BY "Id" ASC

Data Output Messages Notifications

	Id [PK] integer	Name character varying (25)
1	6	Category A
2	1	Furniture
3	2	Appliances
4	3	Cell phones
5	5	New Category

Query Query History

1 SELECT * FROM public."Category"
2 ORDER BY "Id" ASC

Data Output Messages Notifications

	Id [PK] integer	Name character varying (25)
1	1	Furniture
2	2	Appliances
3	3	Cell phones
4	5	New Category

Завдання 4

REPEATABLE READ

```
SQL Shell (psql)
Введите "help", чтобы получить справку.

postgres=# begin transaction isolation repeatable read;
ОШИБКА: ошибка синтаксиса (примерное положение: "repeatable")
СТРОКА 1: begin transaction isolation repeatable read;
      ^
postgres=# begin transaction isolation level repeatable read;
BEGIN
postgres=# \c Inventory
postgres=# \c Inventory
Вы подключены к базе данных "Inventory" как пользователь "postgres".
Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
ОШИБКА: ошибка синтаксиса (примерное положение: "/")
СТРОКА 1: \c Inventory
      ^
Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Furniture
(1 record)

Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Flowers
(1 record)

Inventory=#
```

```
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 Inventory
Вы подключены к базе данных "Inventory" как пользователь "postgres".
Inventory=# UPDATE public."Category" SET
Inventory=# UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: ошибка синтаксиса (примерное положение: "public")
СТРОКА 2: UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = ...
      ^
Inventory=# UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: столбец "Flowers" не существует
СТРОКА 1: UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
      ^
Inventory=# UPDATE "Category" SET "Name" = 'Flowers' WHERE "Id" = 1;
UPDATE 1
Inventory=#
```

SERIALIZABLE

```
SQL Shell (psql)
1 | Flowers
(1 record)

Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Flowers
(1 record)

Inventory=# commit;
COMMIT
Inventory=# begin transaction isolation level serializable;
BEGIN
Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Carpets
(1 record)

Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Carpets
(1 record)

Inventory=#
```

```
SQL Shell (psql)
Введите "help", чтобы получить справку.

postgres=# \c Inventory
Вы подключены к базе данных "Inventory" как пользователь "postgres".
Inventory=# UPDATE public."Category" SET
Inventory=# UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: ошибка синтаксиса (примерное положение: "public")
СТРОКА 2: UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = ...
      ^
Inventory=# UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: столбец "Flowers" не существует
СТРОКА 1: UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
      ^
Inventory=# UPDATE "Category" SET "Name" = 'Flowers' WHERE "Id" = 1;
UPDATE 1
Inventory=# commit;
COMMIT
ПРЕДУПРЕЖДЕНИЕ: нет незавершенной транзакции
COMMIT
Inventory=# begin;
BEGIN
Inventory=# UPDATE "Category" SET "Name" = 'Carpets' WHERE "Id" = 1;
UPDATE 1
Inventory=# commit;
COMMIT
Inventory=# begin;
BEGIN
Inventory=# UPDATE "Category" SET "Name" = 'Floor' WHERE "Id" = 1;
UPDATE 1
Inventory=#
```

READ COMMITTED

```
SQL Shell (psql)
 Id | Name
-----+-----
  1 | Flowers
(1 record)

Inventory=# commit;
ПРЕДУПРЕЖДЕНИЕ: нет незавершенной транзакции
COMMIT
Inventory=# begin transaction isolation level read committed;
ОШИБКА: ошибка синтаксиса (примерное положение: "committed")
СТРОКА 1: begin transaction isolation level read committed;
      ^
Inventory=# begin transaction isolation level read committed;
BEGIN
Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Flowers
(1 record)

Inventory=# SELECT * FROM public."Category" WHERE "Id" = 1;
 Id | Name
-----+-----
  1 | Flowers
(1 record)

Inventory=#
```

```
SQL Shell (psql)
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 Inventory
Вы подключены к базе данных "Inventory" как пользователь "postgres".
Inventory=# UPDATE public."Category" SET
Inventory=# UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: ошибка синтаксиса (примерное положение: "public")
СТРОКА 2: UPDATE public."Category" SET "Name" = "Flowers" WHERE "Id" = ...
      ^
Inventory=# UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
ОШИБКА: столбец "Flowers" не существует
СТРОКА 1: UPDATE "Category" SET "Name" = "Flowers" WHERE "Id" = 1;
      ^
Inventory=# UPDATE "Category" SET "Name" = 'Flowers' WHERE "Id" = 1;
UPDATE 1
Inventory=# commit;
ПРЕДУПРЕЖДЕНИЕ: нет незавершенной транзакции
COMMIT
Inventory=# begin;
BEGIN
Inventory=# UPDATE "Category" SET "Name" = 'Carpets' WHERE "Id" = 1;
UPDATE 1
Inventory=#
```