НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота № 2

з дисципліни
«Бази даних і засоби управління»

Виконав: студент III курсу

ФПМ групи КВ-01

Калитенко Максим Петрович

Перевірив: _____

Київ – 2022

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Деталізоване завдання:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Приклад генерації 100 псевдовипадкових чисел:

```
select trunc(random()*1000)::int
from generate_series(1,100)
```

Data Output | Explain | Messages | Notific

	trunc integer
1	368
2	773
3	29
4	66
5	497
6	956

Приклад генерації 5 псевдовипадкових рядків:

```
select chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int)
from generate_series(1,5)
```

Data Output | Explain | Messages | Notifications

	?column? text
1	NE
2	MQ
3	RN
4	DW
5	DA

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

3. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

4. Програмний код організувати згідно шаблону Model-View-Controller(MVC). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Вимоги до інтерфейсу користувача

Використовувати консольний інтерфейс користувача.

Вимоги до інструментарію

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.6-3.9

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Вимоги до оформлення звіту лабораторної роботи у електронному вигляді

Опис (файл README.md) лабораторної роботи у **репозиторії GitHub** включає: назву лабораторної роботи, структуру бази даних з лабораторної роботи №1.

Репозиторій має містить файл звіту у форматі PDF та програмний код файлів мовою Python (або іншою).

Вимоги до звіту у форматі PDF (у електронній формі)

Загальні вимоги

- титульний аркуш, завдання, URL репозиторію з вихідним кодом та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання (див. нижче);
- діаграму сутність-зв’язок та структуру бази даних з лабораторної роботи №1, а також короткий опис бази даних;
- схему меню користувача з описом функціональності кожного пункту;
- назву мови програмування та бібліотек, що були використані;

Вимоги до пункту №1 деталізованого завдання:

- лістинги та скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних. Причини помилок мають бути пояснені;
- лістинги та скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці немає відповідного запису.

Вимоги до пункту №2 деталізованого завдання:

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць;
- копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах.

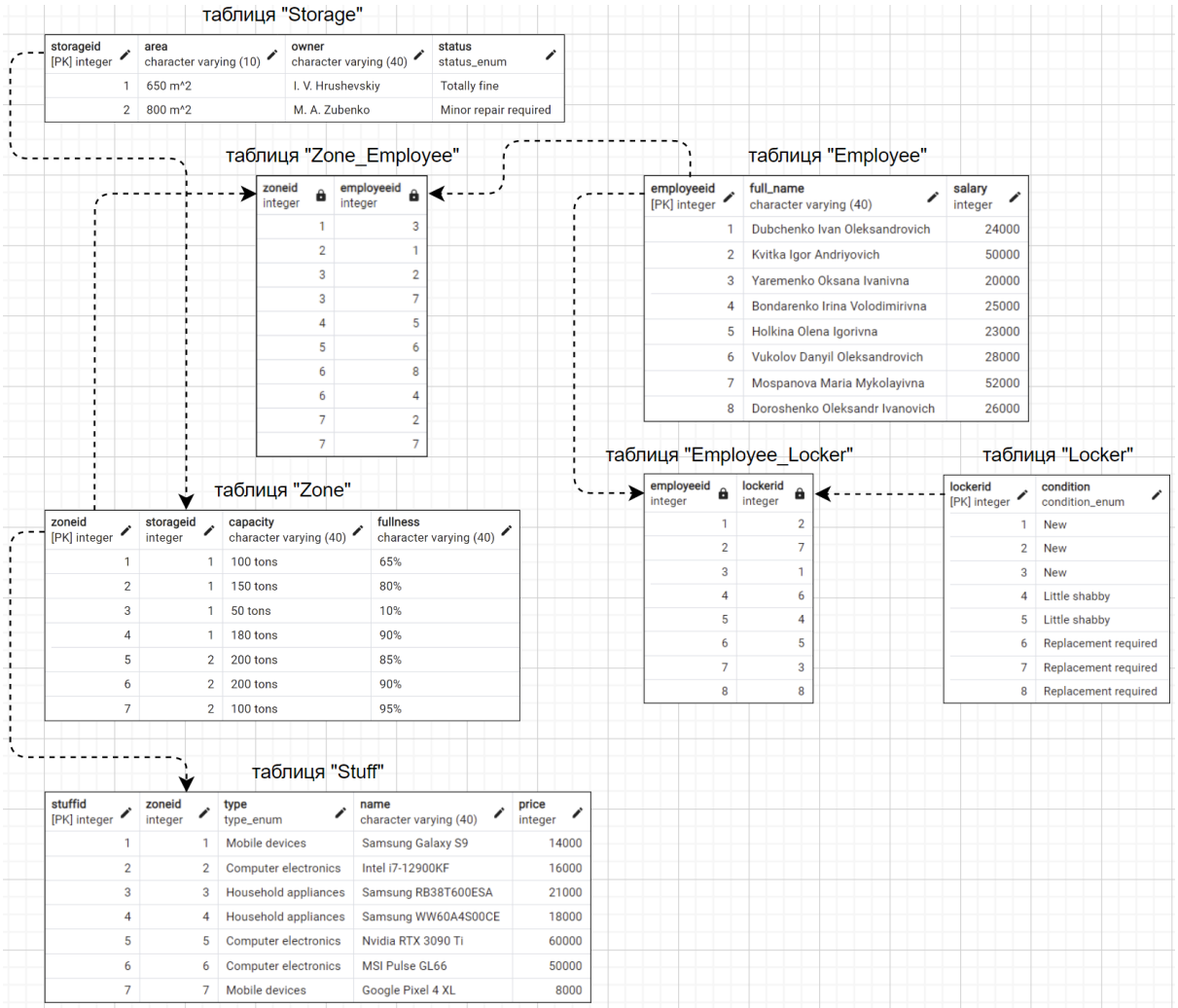
Вимоги до пункту №3 деталізованого завдання:

- ілюстрації уведення пошукового запиту та результатів виконання запитів;
- копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

Вимоги до пункту №4 деталізованого завдання:

- ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Надати короткий опис функцій модуля.

Структура БД “Обслуговування складу”



storageid [PK] integer	area character varying (10)	owner character varying (40)	status status_enum
1	650 m^2	I. V. Hrushevskiy	Totally fine
2	800 m^2	M. A. Zubenko	Minor repair required

zoneid integer	employeeid integer
1	3
2	1
3	2
3	7
4	5
5	6
6	8
6	4
7	2
7	7

employeeid [PK] integer	full_name character varying (40)	salary integer
1	Dubchenko Ivan Oleksandrovich	24000
2	Kvitka Igor Andriyovich	50000
3	Yaremenko Oksana Ivanivna	20000
4	Bondarenko Irina Volodimirivna	25000
5	Holkina Olena Igorivna	23000
6	Vukolov Danyil Oleksandrovich	28000
7	Mospanova Maria Mykolayivna	52000
8	Doroshenko Oleksandr Ivanovich	26000

employeeid integer	lockerid integer
1	2
2	7
3	1
4	6
5	4
6	5
7	3
8	8

lockerid [PK] integer	condition condition_enum
1	New
2	New
3	New
4	Little shabby
5	Little shabby
6	Replacement required
7	Replacement required
8	Replacement required

zoneid [PK] integer	storageid integer	capacity character varying (40)	fullness character varying (40)
1	1	100 tons	65%
2	1	150 tons	80%
3	1	50 tons	10%
4	1	180 tons	90%
5	2	200 tons	85%
6	2	200 tons	90%
7	2	100 tons	95%

stuffid [PK] integer	zoneid integer	type type_enum	name character varying (40)	price integer
1	1	Mobile devices	Samsung Galaxy S9	14000
2	2	Computer electronics	Intel i7-12900KF	16000
3	3	Household appliances	Samsung RB38T600ESA	21000
4	4	Household appliances	Samsung WW60A4S00CE	18000
5	5	Computer electronics	Nvidia RTX 3090 Ti	60000
6	6	Computer electronics	MSI Pulse GL66	50000
7	7	Mobile devices	Google Pixel 4 XL	8000

Рис. 3 - Структурна модель предметної області “Обслуговування складу”

Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування C#, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC.

Меню програми

```
Choose table you want to work with or 0 to exit::
1.Storage
2.Zone
3.Stuff
4.Zone_Employee
5.Employee
6.Employee_Locker
7.Locker
```

Меню роботи з таблицею

```
Choose what you want to do with 'Storage' table or 0 to exit:
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
```

Використані бібліотеки:

Npgsql;

System;

System.Collections.Generic;

System.Text;

Завдання 1

Додавання даних до БД

```
Enter Zone properties:
StorageId:
2
Capacity:
34 tons
Fullness:
61%
```

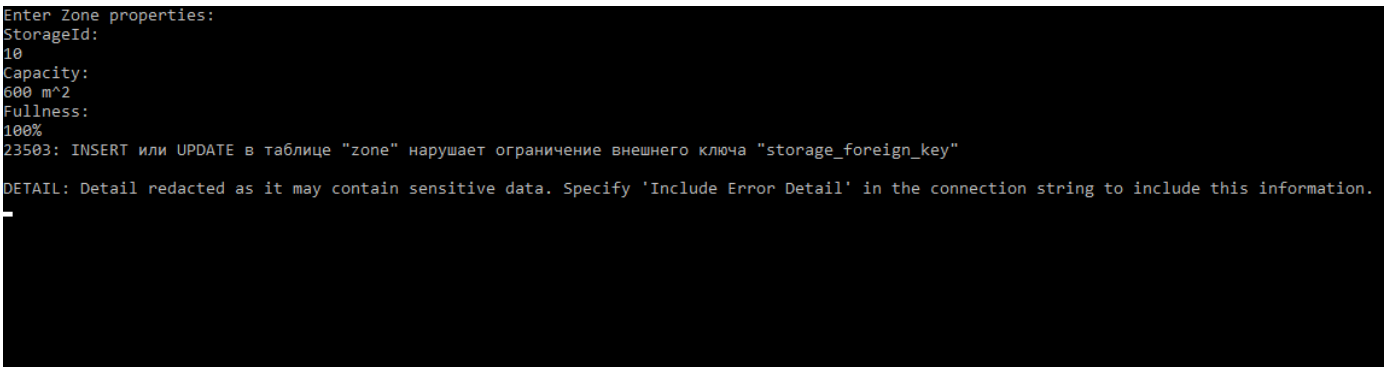
Результат:

```
ZoneId: 4
StorageId: 1
Capacity: 180 tons
Fullness: 90%

ZoneId: 5
StorageId: 2
Capacity: 200 tons
Fullness: 85%

ZoneId: 6
StorageId: 2
Capacity: 200 tons
Fullness: 90%

ZoneId: 7
StorageId: 2
Capacity: 100 tons
Fullness: 95%

ZoneId: 9
StorageId: 2
Capacity: 34 tons
Fullness: 61%
```

SQL-запит:

```
string sqlInsert = "Insert into Zone (StorageId, Capacity, Fullness) VALUES(@StorageId, @Capacity, @Fullness)";
```

Перевірка наявності відповідного рядка у батьківській таблиці при внесенні змін до дочірньої:



Код для перехоплення помилки:

```
try
{
    using NpgsqlDataReader rdr = cmd.ExecuteReader();

    while (rdr.Read())
    {
        Console.WriteLine("ZoneId: {0}", rdr.GetValue(0));
        Console.WriteLine("StorageId: {0}", rdr.GetValue(1));
        Console.WriteLine("Capacity: {0}", rdr.GetValue(2));
        Console.WriteLine("Fullness: {0}", rdr.GetValue(3));
        Console.WriteLine();
    }
    Console.WriteLine();
}
catch (Exception ex)
{
```

```
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}
finally
{
    sqlConnection.Close();
}
```

Видалення даних

```
Choose what you want to do with 'Zone' table or 0 to exit:
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
4
Enter number of record you want to delete (or 0 to step back):
9
```

Результат:

```
ZoneId: 4
StorageId: 1
Capacity: 180 tons
Fullness: 90%

ZoneId: 5
StorageId: 2
Capacity: 200 tons
Fullness: 85%

ZoneId: 6
StorageId: 2
Capacity: 200 tons
Fullness: 90%

ZoneId: 7
StorageId: 2
Capacity: 100 tons
Fullness: 95%
```

SQL-запит:

```
public override void Delete()
{
    base.Delete("delete from Zone where ZoneId = ");
}
```

```
using var cmd = new NpgsqlCommand(sqlDelete + id, sqlConnection);
```

Завдання 2

Генерування даних в таблиці Zone

```
Choose what you want to do with 'Zone' table or 0 to exit:
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
6
How many records do you want?
100000
```

Результат:

```
ZoneId: 99906
StorageId: 2
Capacity: UDIU
Fullness: TUAJ

ZoneId: 99907
StorageId: 1
Capacity: aDCT
Fullness: `DLB

ZoneId: 99908
StorageId: 2
Capacity: jEMH
Fullness: [BTY

ZoneId: 99909
StorageId: 1
Capacity: GIOL
Fullness: MKLW

ZoneId: 99910
StorageId: 2
Capacity: hNEG
Fullness: iCOE
```

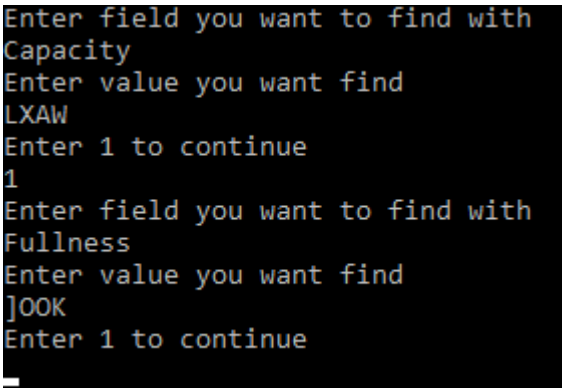
SQL-запит:

```
string sqlGenerate = "insert into Zone(StorageId, Capacity, Fullness) (select storage.StorageId"
    + ", "
    + base.sqlRandomString
    + ", "
    + base.sqlRandomString
    + " from generate_series(1, 1000000), storage limit(" + recordsAmount + "))";
base.Generate(sqlGenerate);
```

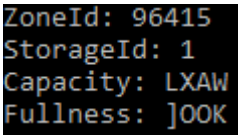
public readonly string sqlRandomString = "chr(trunc(65 + random() * 50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)";

Завдання 3

Пошукові запити



Результат:



SQL-запит:

"select ZoneId, StorageId, Capacity, Fullness from Zone where Capacity = 'LXAW' and Fullness = ']OOK'"

Код програми

Лістинг Prog.cs

```
using BD.Controllers;
using System;


namespace BD
{
    class Prog
    {
        static void Main(string[] args)
        {
            String connectionString =
"Host=localhost;Username=postgres;Password=super_KEKL;Database=Mainte
nance of the warehouse";


            int table = 0;
            int action = 0;
            do
            {
                table = FirstMenu();
                if (table == 0)
                {
                    return;
                }

                Base controller = null;
```

```
switch (table)
{
    case 1:
        action = SecondMenu("Storage");
        controller = new StorageC(connectionString);
        break;
    case 2:
        action = SecondMenu("Zone");
        controller = new ZoneC(connectionString);
        break;
    case 3:
        action = SecondMenu("Stuff");
        controller = new StuffC(connectionString);
        break;
    case 4:
        action = SecondMenu("Zone_Employee");
        controller = new Zone_EmployeeC(connectionString);
        break;
    case 5:
        action = SecondMenu("Employee");
        controller = new EmployeeC(connectionString);
        break;
    case 6:
        action = SecondMenu("Employee_Locker");
        controller = new Employee_LockerC(connectionString);
        break;
```

```
    case 7:
        action = SecondMenu("Locker");
        controller = new LockerC(connectionString);
        break;
}


switch (action)
{
    case 1:
        controller.Create();
        break;
    case 2:
        controller.Read();
        break;
    case 3:
        controller.Update();
        break;
    case 4:
        controller.Delete();
        break;
    case 5:
        controller.Find();
        break;
    case 6:
        controller.Generate();
        break;
```

```
            }



        } while (true);
    }

    public static int FirstMenu()
    {
        var choice = 0;
        var correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Choose table you want to work with or 0 to
exit::");
            Console.WriteLine("1.Storage");
            Console.WriteLine("2.Zone");
            Console.WriteLine("3.Stuff");
            Console.WriteLine("4.Zone_Employee");
            Console.WriteLine("5.Employee");
            Console.WriteLine("6.Employee_Locker");
            Console.WriteLine("7.Locker");
            correct = Int32.TryParse(Console.ReadLine(), out choice);
        } while (choice < 0 || choice > 7 || correct == false);
```

```
            return choice;
        }

        public static int SecondMenu(string tableToChange)
        {
            var choice = 0;
            var correct = false;
            do
            {
                Console.Clear();
                Console.WriteLine("Choose what you want to do with '" +
tableToChange + "' table or 0 to exit:");
                Console.WriteLine("1.Create");
                Console.WriteLine("2.Read");
                Console.WriteLine("3.Update");
                Console.WriteLine("4.Delete");
                Console.WriteLine("5.Find");
                Console.WriteLine("6.Generate");
                correct = Int32.TryParse(Console.ReadLine(), out choice);
            } while (choice < 0 || choice > 6 || correct == false);



            return choice;
        }
    }
}
```

Лістинг Base.cs

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

namespace BD.Controllers
{
    public abstract class Base
    {
        public string connectionString;
        protected NpgsqlConnection sqlConnection;

        string fieldToFind = null;
        string valueToFind = null;
        string fieldToSet = null;
        string valueToSet = null;

        string[] fieldsToFind = new string[10];
        string[] valuesToFind = new string[10];



        public readonly string sqlUpdate = "Update @table set @field_to_update = @new_value where @field_to_find = @old_value";
        public readonly string sqlRandomString = "chr(trunc(65 + random() * 50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)";
        public readonly string sqlRandomInteger = "trunc(random()*1000)::int";
```

```
        public readonly string sqlRandomDate = "timestamp '2014-01-10 20:00:00' + random() * (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')";
        public readonly string sqlRandomBoolean = "trunc(random()*2)::int::boolean";



        public Base(string connectionString)
        {
            this.connectionString = connectionString;
            this.sqlConnection = new NpgsqlConnection(connectionString);
        }



        public virtual void Create()
        {
            throw new NotImplementedException();
        }
        public void Read()
        {
            Read("");
        }
        public virtual void Update()
        {
            throw new NotImplementedException();
        }
        public virtual void Delete()
        {
```

```
    throw new NotImplementedException();
}
public virtual void Find()
{
    Console.Clear();
    int actualSize = 0;
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine("Enter field you want to find with");
        fieldsToFind[i] = Console.ReadLine();
        Console.WriteLine("Enter value you want find");
        valuesToFind[i] = Console.ReadLine();
        Console.WriteLine("Enter 1 to continue");
        actualSize++;

        int choose = 0;
        bool correct = Int32.TryParse(Console.ReadLine(), out choose);
        if (correct = false || choose != 1)
        {
            break;
        }
    }

    string whereCondition = " where ";

    int parseInt;
    if (Int32.TryParse(valuesToFind[0], out parseInt) == false)
```

```
            {
                valuesToFind[0] = "'" + valuesToFind[0] + "'";
            }
            whereCondition += fieldsToFind[0] + " = " + valuesToFind[0];

            for (int i = 1; i < actualSize; i++)
            {
                if (Int32.TryParse(valuesToFind[i], out parseInt) == false)
                {
                    valuesToFind[i] = "'" + valuesToFind[i] + "'";
                }
                whereCondition += " and " + fieldsToFind[i] + " = " +
valuesToFind[i];
            }

            Read(whereCondition);
        }
        virtual public void Generate()
        {
            throw new NotImplementedException();
        }

        virtual public void Read(string whereCondition)
        {

        }
```

```
protected void Delete(string sqlDelete)
{
    bool correct = false;
    int id = 0;
    do
    {
        Console.WriteLine("Enter number of record you want to delete (or 0 to step back):");
        correct = Int32.TryParse(Console.ReadLine(), out id);
        if (correct == false)
        {
            Console.WriteLine("Id must be a number...");
            Console.ReadLine();
            continue;
        }
    } while (correct == false || id < 0);

    sqlConnection.Open();

    using var cmd = new NpgsqlCommand(sqlDelete + id, sqlConnection);

    try
    {
        cmd.Prepare();
        cmd.ExecuteNonQuery();
    }
```

```
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }

    }

    private void Update(string table, string field_to_update, string new_value,
string field_to_find, string old_value)
    {
        sqlConnection.Open();

        StringBuilder updateString = new StringBuilder("Update", 200);

        int new_int;
        if (!Int32.TryParse(new_value, out new_int))
        {
            new_value = "'" + new_value + "'";
        }
        if (!Int32.TryParse(old_value, out new_int))
        {
```

```
            old_value = "'" + old_value + "'";
        }

        updateString.AppendFormat(" {0} set {1} = {2} where {3} = {4}",
table, field_to_update, new_value, field_to_find, old_value);

        using var cmd = new NpgsqlCommand(updateString.ToString(),
sqlConnection);

        try
        {
            cmd.Prepare();
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }

    protected void Update(string sqlUpdate)
    {
```

```
        Console.Clear();
        Console.WriteLine("Enter name of field you want to find:");
        fieldToFind = Console.ReadLine();
        Console.WriteLine("Enter value in this field you want to find:");
        valueToFind = Console.ReadLine();


        Console.WriteLine("Enter name of field you want to change:");
        fieldToSet = Console.ReadLine();
        Console.WriteLine("Enter new value in this field");
        valueToSet = Console.ReadLine();


        int ParseInt = 0;
        if (Int32.TryParse(valueToFind, out ParseInt) == false)
        {
            valueToFind = "'" + valueToFind + "'";
        }
        if (Int32.TryParse(valueToSet, out ParseInt) == false)
        {
            valueToSet = "'" + valueToSet + "'";
        }

        string sqlQuery = sqlUpdate + "set " + fieldToSet + " = " + valueToSet +
" where " + fieldToFind + " = " + valueToFind;
```

```
    sqlConnection.Open();

    using var cmd = new NpgsqlCommand(sqlQuery, sqlConnection);

    try
    {
        cmd.Prepare();
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }

}

protected void Generate(string sqlGenerate)
{

    sqlConnection.Open();

    using var cmd = new NpgsqlCommand(sqlGenerate, sqlConnection);
```

```
            try
            {
                cmd.Prepare();
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.ReadLine();
            }
            finally
            {
                sqlConnection.Close();
            }
        }
    }
}
```

Лістинг StorageC.cs

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

namespace BD.Controllers
{
    public class StorageC : Base
```

```
{
    public StorageC(string connectionString) : base(connectionString) { }
    public override void Read(string whereCondition)
    {
        Console.Clear();

        sqlConnection.Open();

        string sqlSelect = "select StorageId, Area, Owner, Status from Storage";

        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine("StorageId: {0}", rdr.GetValue(0));
                Console.WriteLine("Area: {0}", rdr.GetValue(1));
                Console.WriteLine("Owner: {0}", rdr.GetValue(2));
                Console.WriteLine("Status: {0}", rdr.GetValue(3));
                Console.WriteLine();
            }
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }



        Console.ReadLine();
    }


    public override void Create()
    {
        string sqlInsert = "Insert into Storage(Area, Owner) VALUES(@Area,
@Owner)";


        string Area = null;
        string Owner = null;


        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Enter Storage properties:");
            Console.WriteLine("Area:");
            Area = Console.ReadLine();
```

```
    if (Area.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of Area > 10. It is wrong.");
        Console.ReadLine();
        continue;
    }

    Console.WriteLine("Owner:");
    Owner = Console.ReadLine();
    if (Owner.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of Owner > 40. It is wrong.");
        Console.ReadLine();
        continue;
    }


    correct = true;
} while (correct == false);


sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("Area", Area);
```

```
    cmd.Parameters.AddWithValue("Owner", Owner);
    cmd.Prepare();

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }
}

public override void Delete()
{
    base.Delete("delete from Storage where StorageId = ");
}
public override void Update()
{
    base.Update("Update Storage ");
}
public override void Find()
```

```
        {
            base.Find();
        }

        public override void Generate()
        {
            Console.WriteLine("How many records do you want?");
            bool correct = false;
            int recordsAmount;

            correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

            string sqlGenerate = "insert into Storage(Area, Owner) (select "
                + base.sqlRandomString
                + ", "
                + base.sqlRandomString
                + " from generate_series(1, 1000000)  limit(" + recordsAmount + "))";
            base.Generate(sqlGenerate);
        }


    }
}
```

Лістинг ZoneC.cs

```
using Npgsql;
using System;
```

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

namespace BD.Controllers
{
    public class ZoneC : Base
    {
        public ZoneC(string connectionString) : base(connectionString) { }

        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select ZoneId, StorageId, Capacity, Fullness from
Zone";

            using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
            try
            {
                using NpgsqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
```

```
            Console.WriteLine("ZoneId: {0}", rdr.GetValue(0));
            Console.WriteLine("StorageId: {0}", rdr.GetValue(1));
            Console.WriteLine("Capacity: {0}", rdr.GetValue(2));
            Console.WriteLine("Fullness: {0}", rdr.GetValue(3));
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }


    Console.ReadLine();
}

public override void Create()
{
    string sqlInsert = "Insert into Zone (StorageId, Capacity, Fullness)
VALUES(@StorageId, @Capacity, @Fullness)";
```

```
int Storage_id = 0;
string Capacity = null;
string Fullness = null;

bool correct = false;
do
{
    Console.Clear();
    Console.WriteLine("Enter Zone properties:");

    Console.WriteLine("StorageId:");
    correct = Int32.TryParse(Console.ReadLine(), out Storage_id);
    if (correct == false)
    {
        Console.WriteLine("StorageId must be a number!");
        Console.ReadLine();
    }


    Console.WriteLine("Capacity:");
    Capacity = Console.ReadLine();
    if (Capacity.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of Capacity > 40. It is wrong.");
        Console.ReadLine();
        continue;
```

```
    }

    Console.WriteLine("Fullness:");
    Fullness = Console.ReadLine();
    if (Fullness.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of Fullness > 40. It is wrong.");
        Console.ReadLine();
        continue;
    }

    correct = true;
} while (correct == false);


sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("StorageId", Storage_id);
cmd.Parameters.AddWithValue("Capacity", Capacity);
cmd.Parameters.AddWithValue("Fullness", Fullness);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
```

```
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }
}
public override void Delete()
{
    base.Delete("delete from Zone where ZoneId = ");
}
public override void Update()
{
    base.Update("Update Zone ");
}

public override void Generate()
{
    Console.WriteLine("How many records do you want?");
    bool correct = false;
    int recordsAmount;

    correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);
```

```
            string sqlGenerate = "insert into Zone(StorageId, Capacity, Fullness)
(select storage.StorageId"
                + ", "
                + base.sqlRandomString
                + ", "
                + base.sqlRandomString
                + " from generate_series(1, 1000000), storage limit(" +
recordsAmount + "))";
            base.Generate(sqlGenerate);
        }
    }
}
```

Лістинг StuffC.cs

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

namespace BD.Controllers
{
    public class StuffC : Base
    {
        public StuffC(string connectionString) : base(connectionString) { }

        public override void Read(string whereCondition)
```

```
    {
        Console.Clear();


        sqlConnection.Open();


        string sqlSelect = "select StuffId, ZoneId, Type, Name, Price from
Stuff";



        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();


            while (rdr.Read())
            {
                Console.WriteLine("StuffId: {0}", rdr.GetValue(0));
                Console.WriteLine("ZoneId: {0}", rdr.GetValue(1));
                Console.WriteLine("Type: {0}", rdr.GetValue(2));
                Console.WriteLine("Name: {0}", rdr.GetValue(3));
                Console.WriteLine("Price: {0}", rdr.GetValue(4));
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }



        Console.ReadLine();
    }


    public override void Create()
    {
        string sqlInsert = "Insert into Stuff (ZoneId, Type, Name, Price)
VALUES(@ZoneId, @Type, @Name, @Price)";


        int Zone_id = 0;
        string Type = null;
        string Name = null;
        int Price = 0;


        bool correct = false;
        do
        {
            Console.Clear();
```

```
Console.WriteLine("Enter Stuff properties:");

Console.WriteLine("ZoneId:");
correct = Int32.TryParse(Console.ReadLine(), out Zone_id);
if (correct == false)
{
   Console.WriteLine("ZoneId must be a number!");
   Console.ReadLine();
}


Console.WriteLine("Type:");
Type = Console.ReadLine();
if (Type.Length > 40)
{
   correct = false;
   Console.WriteLine("Length of Type > 40. It is wrong.");
   Console.ReadLine();
   continue;
}

Console.WriteLine("Name:");
Name = Console.ReadLine();
if (Name.Length > 40)
{
   correct = false;
   Console.WriteLine("Length of Name > 40. It is wrong.");
```

```
        Console.ReadLine();
        continue;
    }

    Console.WriteLine("Price:");
    correct = Int32.TryParse(Console.ReadLine(), out Price);
    if (correct == false)
    {
        Console.WriteLine("Price must be a number!");
        Console.ReadLine();
    }

    correct = true;
} while (correct == false);



sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("ZoneId", Zone_id);
cmd.Parameters.AddWithValue("Type", Type);
cmd.Parameters.AddWithValue("Name", Name);
cmd.Parameters.AddWithValue("Price", Price);
cmd.Prepare();

try
{
```

```
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
    public override void Delete()
    {
        base.Delete("delete from Stuff where StuffId = ");
    }
    public override void Update()
    {
        base.Update("Update Stuff ");
    }

    public override void Generate()
    {
        Console.WriteLine("How many records do you want?");
        bool correct = false;
        int recordsAmount;
```

```
            correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);


            string sqlGenerate = "insert into Stuff(ZoneId, Type, Name, Price)
(select zone.ZoneId"
                + ", "
                + base.sqlRandomString
                + ", "
                + base.sqlRandomString
                + ", "
                + base.sqlRandomInteger
                + " from generate_series(1, 1000000), zone limit(" + recordsAmount
+ "))";
            base.Generate(sqlGenerate);
        }
    }
}
```

Лістинг Zone_EmployeeC.cs

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

namespace BD.Controllers
{
    public class Zone_EmployeeC : Base
    {
```

```
    public Zone_EmployeeC(string connectionString) : base(connectionString)
{ }
    public override void Read(string whereCondition)
    {
        Console.Clear();


        sqlConnection.Open();


        string sqlSelect = "select ZoneId, EmployeeId from Zone_Employee";



        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();


            while (rdr.Read())
            {
                Console.WriteLine("ZoneId: {0}", rdr.GetValue(0));
                Console.WriteLine("EmployeeId: {0}", rdr.GetValue(1));
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }



        Console.ReadLine();
    }
    public override void Create()
    {
        string sqlInsert = "Insert into Zone_Employee (ZoneId, EmployeeId)
VALUES(@ZoneId, @EmployeeId)";


        int ZoneId = 0;
        int EmployeeId = 0;


        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Enter Zone_Employee properties:");
            Console.WriteLine("ZoneId:");
            correct = Int32.TryParse(Console.ReadLine(), out ZoneId);
            if (correct == false)
```

```
    {
        Console.WriteLine("ZoneId must be a number!");
        Console.ReadLine();
    }

    Console.WriteLine("EmployeeId:");
    correct = Int32.TryParse(Console.ReadLine(), out EmployeeId);
    if (correct == false)
    {
        Console.WriteLine("EmployeeId must be a number!");
        Console.ReadLine();
    }


    correct = true;
} while (correct == false);


sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("ZoneId", ZoneId);
cmd.Parameters.AddWithValue("EmployeeId", EmployeeId);
cmd.Prepare();

try
{
```

```
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
    public override void Delete()
    {
        base.Delete("delete from Zone_Employee where EmployeeId = ");
    }
    public override void Update()
    {
        base.Update("Update Zone_Employee ");
    }

    public override void Generate()
    {
        Console.WriteLine("How many records do you want?");
        bool correct = false;
        int recordsAmount;
```

```
            correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);


            string sqlGenerate = "insert into Zone_Employee(ZoneId, EmployeeId)
(select Zone.ZoneId, Employee.EmployeeId"
                + " from Zone, Employee limit(" + recordsAmount + "))";
            base.Generate(sqlGenerate);
        }
    }
}
```

Лістинг EmployeeC.cs

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

namespace BD.Controllers
{
    public class EmployeeC : Base
    {
        public EmployeeC(string connectionString) : base(connectionString) { }
        public override void Read(string whereCondition)
        {
            Console.Clear();


            sqlConnection.Open();
```

```
        string sqlSelect = "select EmployeeId, Full_name, Salary from
Employee";



        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();


            while (rdr.Read())
            {
                Console.WriteLine("EmployeeId: {0}", rdr.GetValue(0));
                Console.WriteLine("Full_name: {0}", rdr.GetValue(1));
                Console.WriteLine("Salary: {0}", rdr.GetValue(2));
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
```

```
        }


        Console.ReadLine();
    }


    public override void Create()
    {
        string sqlInsert = "Insert into Employee(Full_name, Salary)
VALUES(@Full_name, @Salary)";


        string Full_name = null;
        int Salary = 0;


        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Enter patient properties:");
            Console.WriteLine("Full_name:");
            Full_name = Console.ReadLine();
            if (Full_name.Length > 40)
            {
                correct = false;
                Console.WriteLine("Length of Full_name > 40. It is wrong.");
                Console.ReadLine();
```

```
        continue;
    }

    Console.WriteLine("Salary:");
    correct = Int32.TryParse(Console.ReadLine(), out Salary);
    if (correct == false)
    {
        Console.WriteLine("Salary must be a number!");
        Console.ReadLine();
    }
    correct = true;
} while (correct == false);



sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("Full_name", Full_name);
cmd.Parameters.AddWithValue("Salary", Salary);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
}
catch (Exception ex)
{
```

```
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }

    public override void Delete()
    {
        base.Delete("delete from Employee where EmployeeID = ");
    }
    public override void Update()
    {
        base.Update("Update Employee ");
    }

    public override void Generate()
    {
        Console.WriteLine("How many records do you want?");
        bool correct = false;
        int recordsAmount;

        correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

        string sqlGenerate = "insert into Employee(Full_name, Salary) (select "
```

```
            + base.sqlRandomString
            + ", "
            + base.sqlRandomString
            + " from generate_series(1, 1000000)  limit(" + recordsAmount + "))";
        base.Generate(sqlGenerate);
    }
  }
}
```

Лістинг Employee_LockerC.cs

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

namespace BD.Controllers
{
  public class Employee_LockerC : Base
  {



    public Employee_LockerC(string connectionString) :
base(connectionString) { }
    public override void Read(string whereCondition)
    {
      Console.Clear();
```

```
        sqlConnection.Open();


        string sqlSelect = "select EmployeeID, LockerId from
Employee_Locker";




        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();


            while (rdr.Read())
            {
                Console.WriteLine("EmployeeID: {0}", rdr.GetValue(0));
                Console.WriteLine("LockerId: {0}", rdr.GetValue(1));
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
```

```
            sqlConnection.Close();
        }



        Console.ReadLine();
    }
    public override void Create()
    {
        string sqlInsert = "Insert into Employee_Locker (EmployeeID,
LockerId) VALUES(@EmployeeID, @LockerId)";


        int EmployeeID = 0;
        int LockerId = 0;


        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Enter Employee_Locker properties:");
            Console.WriteLine("EmployeeID:");
            correct = Int32.TryParse(Console.ReadLine(), out EmployeeID);
            if (correct == false)
            {
                Console.WriteLine("EmployeeID must be a number!");
                Console.ReadLine();
            }
```

```
    Console.WriteLine("LockerId:");
    correct = Int32.TryParse(Console.ReadLine(), out LockerId);
    if (correct == false)
    {
        Console.WriteLine("LockerId must be a number!");
        Console.ReadLine();
    }


    correct = true;
} while (correct == false);


sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("EmployeeID", EmployeeID);
cmd.Parameters.AddWithValue("LockerId", LockerId);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
```

```
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
    public override void Delete()
    {
        base.Delete("delete from Employee_Locker where EmployeeID = ");
    }
    public override void Update()
    {
        base.Update("Update Employee_Locker ");
    }

    public override void Generate()
    {
        Console.WriteLine("How many records do you want?");
        bool correct = false;
        int recordsAmount;

        correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

        string sqlGenerate = "insert into Employee_Locker(EmployeeId,
LockerId) (select Employee.EmployeeId, Locker.LockerId"
            + " from Employee, Locker limit(" + recordsAmount + "))";
```

```
            base.Generate(sqlGenerate);
        }
    }
}
```

Лістинг LockerC.cs

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

namespace BD.Controllers
{
    public class LockerC : Base
    {
        public LockerC(string connectionString) : base(connectionString) { }
        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select LockerId, Condition from Locker";

            using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
            try
            {
```

```
        using NpgsqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Console.WriteLine("LockerId: {0}", rdr.GetValue(0));
            Console.WriteLine("Condition: {0}", rdr.GetValue(1));
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }


    Console.ReadLine();
}


public override void Create()
{
```

```
            Console.Clear();
            Console.WriteLine("This function is not available in this table");
            System.Threading.Thread.Sleep(3000);
        }

        public override void Delete()
        {
            base.Delete("delete from Locker where LockerId = ");
        }
        public override void Update()
        {
            base.Update("Update Locker ");
        }

        public override void Generate()
        {
            Console.Clear();
            Console.WriteLine("This function is not available in this table");
            System.Threading.Thread.Sleep(3000);
        }
    }
}
```