

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Степаненко Павло

Перевірив: Павловський В.І.

Київ – 2022

**Лабораторна робота № 2.**

**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**

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

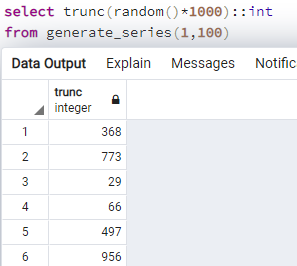
*Загальне завдання* роботи полягає у наступному:

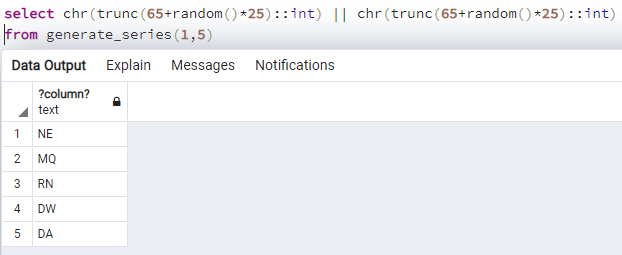
1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами 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:



Приклад генерації 5 псевдовипадкових рядків:

Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

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).

1. 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.
2. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/posts/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: http://initd.org/psycopg/docs/usage.html)

*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:*

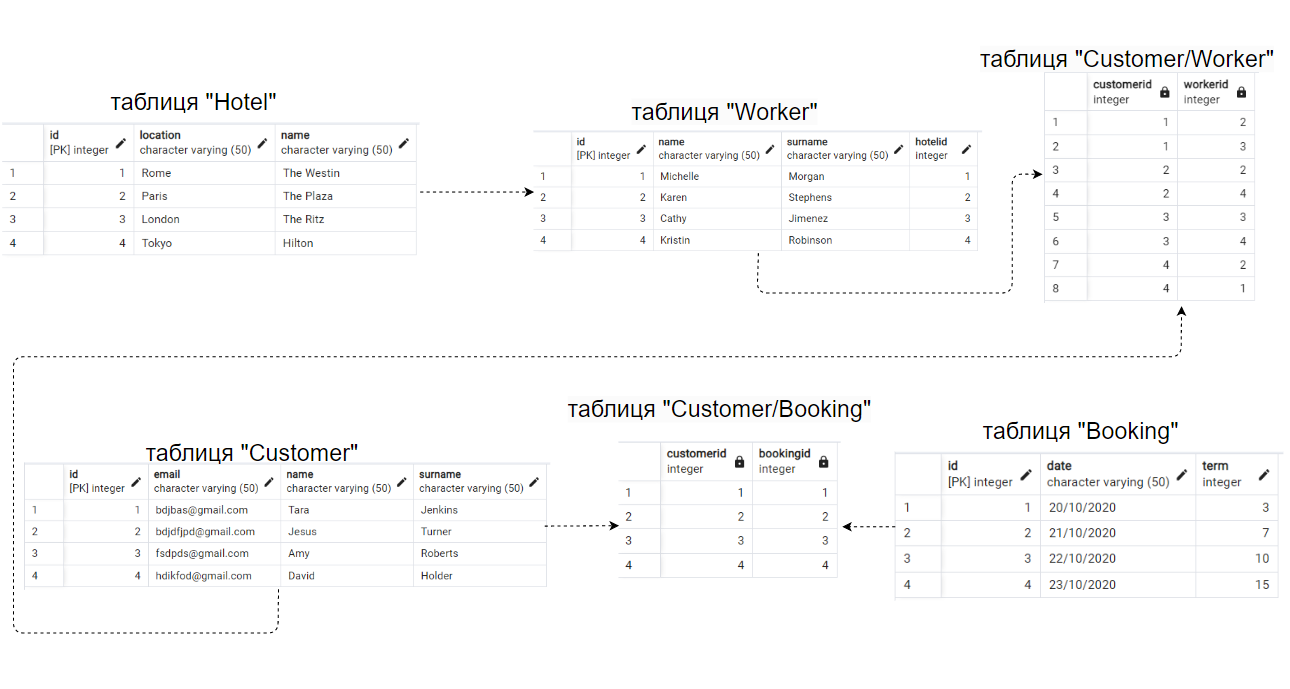
* ілюстрації уведення пошукового запиту та результатів виконання запитів;
* копії SQL-запитів, що ілюструють пошук із зазначеними початковими параметрами.

*Вимоги до пункту №4 деталізованого завдання:*

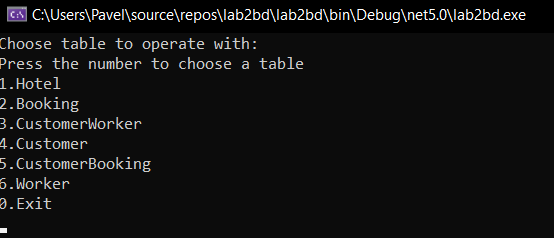
* ілюстрації програмного коду модуля “Model”, згідно із шаблоном MVC. Надати короткий опис функцій модуля.

**Нормалізована модель бази даних “Готель”**

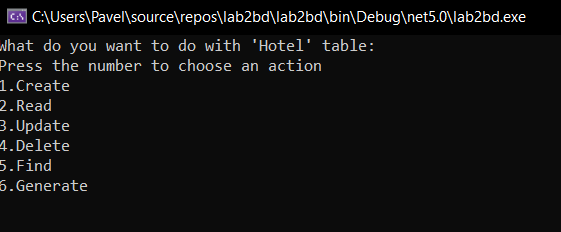
На рисунку зображена нормалізована модель даних БД “Готель”, розроблена на першій лабораторній роботі(трішки змінена).



Головне меню програми має вигляд:



Далі можна обрати операцію:



Основні модулі:

1) Program.cs – файл з меню;

2) BasedController.cs – файл з функціями для підключення та віключення БД від програми;

3) HotelController.cs – контроллер для таблиці Hotel;

4) BookingController.cs – контроллер для таблиці Booking;

5) CostumerController.cs – контроллер для таблиці Customer;

6) CustomerBookingController.cs – контроллер для таблиці customerbooking;

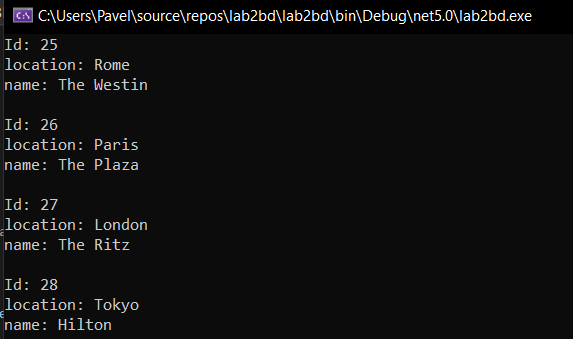
7) CustomerWorkerController.cs – контроллер для таблиці customerworker;

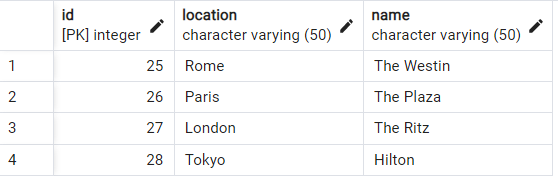
8) WorkerController.cs – контроллер для таблиці Worker;

Щоб підключити БД до програми використовуємо мову програмування c# та бібліотеку NPGSQL

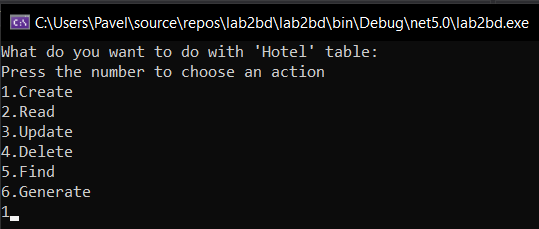
**1)**Функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних:

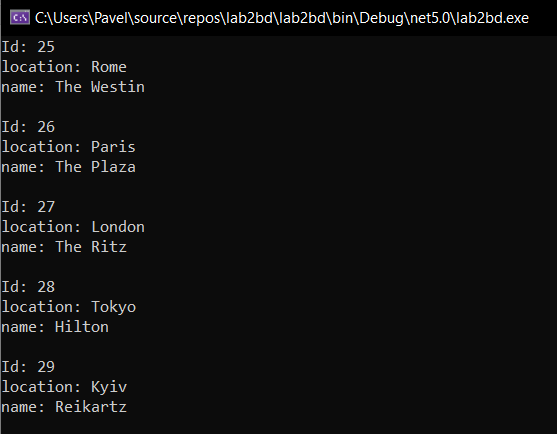
Перегляд таблиці “Hotel”:





Додамо данні до цієї таблиці:



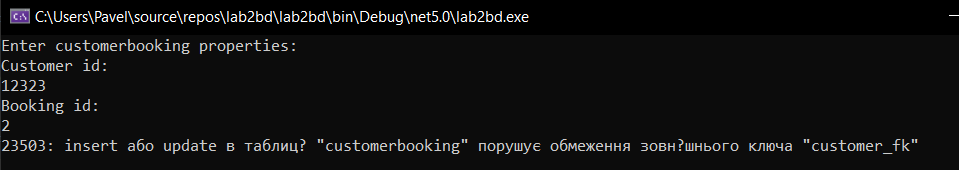


SQL-запит:



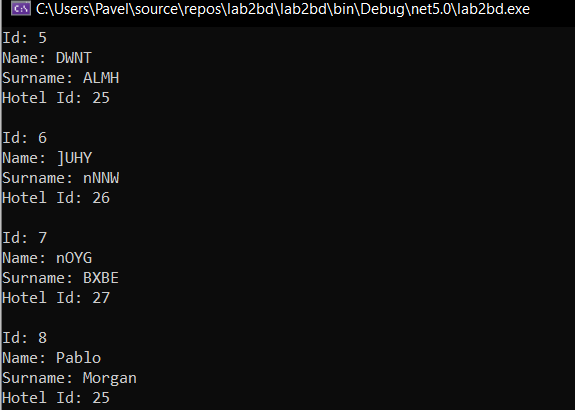
Контроль наявності відповідного рядка у батьківській таблиці при виконанні внесення до дочірної таблиці нових даних. Розглянемо на прикладі

CustomerBooking:



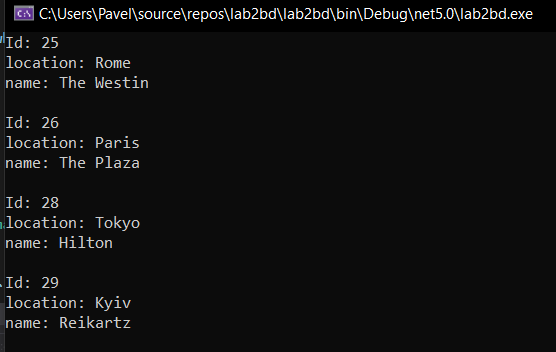
Редагування даних:



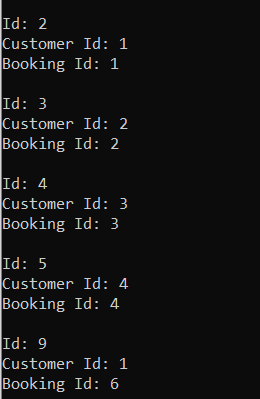


Видалення даних:

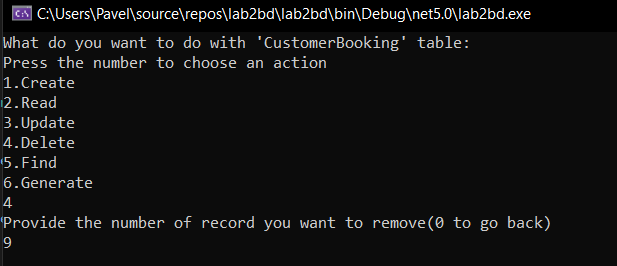


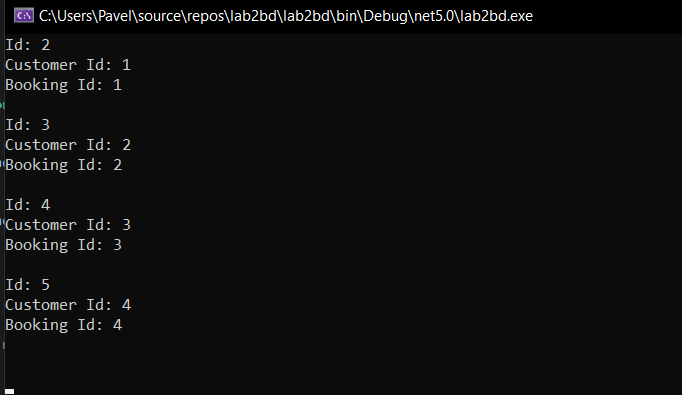


На прикладі таблиці CustomerBooking:

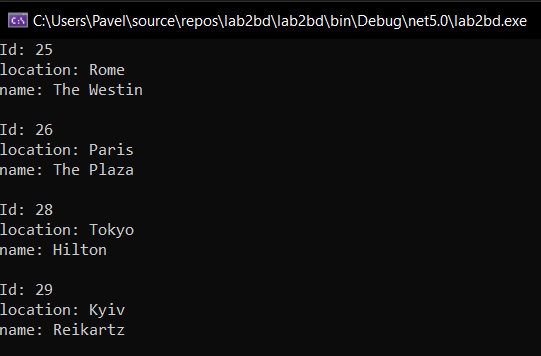


Видалимо запис з Id = 9

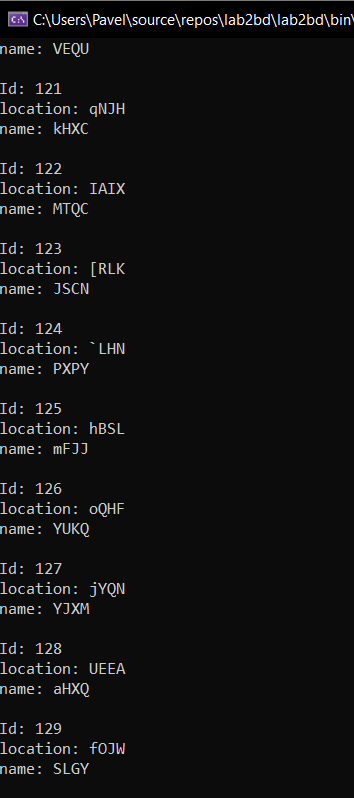




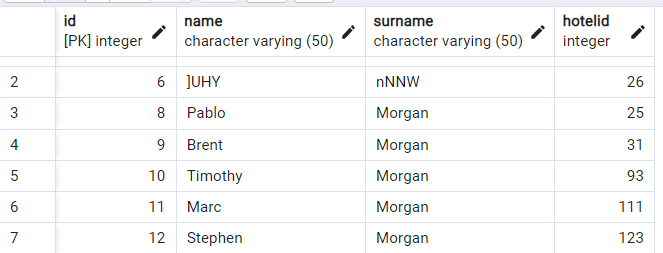
**2)** Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

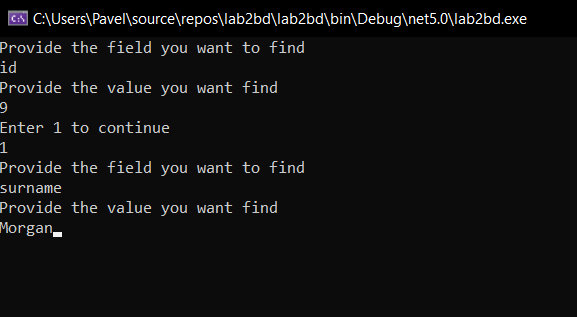
****

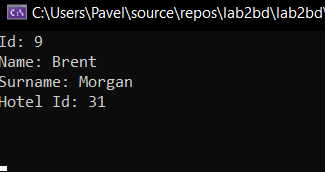
Згенеруємо 100 записів для таблиць “Hotel” та “Costumer”



**3)**Пошук за двома атрибутами одночасно:







Код програми:

**BasedController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **abstract** **class** BasedController

{

**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** 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** **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** BasedController(**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("Provide the field you want to find ");

fieldsToFind[i] = Console.ReadLine();

Console.WriteLine("Provide the 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("Provide the number of record you want to remove(0 to go 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("Provide the name of field you want to find:");

fieldToFind = Console.ReadLine();

Console.WriteLine("Provide the value in this field you want to find:");

valueToFind = Console.ReadLine();

Console.WriteLine("Provide the name of field you want to change:");

fieldToSet = Console.ReadLine();

Console.WriteLine("Provide the 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();

}

}

}

}

**Program.cs**

**using** BD2.Controllers;

**using** System;

**namespace** BD2

{

**class** Program

{

**static** **void** Main(**string**[] args)

{

**String** connectionString = "Host=localhost;Username=postgres;Password=Pavelstep117;Database=hoteln";

**int** table = 0;

**int** action = 0;

**do**

{

table = FirstMenu();

**if** (table == 0)

{

**return**;

}

BasedController controller = **null**;

**switch** (table)

{

**case** 1:

action = SecondMenu("Hotel");

controller = new HotelController(connectionString);

**break**;

**case** 2:

action = SecondMenu("Booking");

controller = new BookingController(connectionString);

**break**;

**case** 3:

action = SecondMenu("CustomerWorker");

controller = new CustomerWorkerController(connectionString);

**break**;

**case** 4:

action = SecondMenu("Customer");

controller = new CostumerController(connectionString);

**break**;

**case** 5:

action = SecondMenu("CustomerBooking");

controller = new CustomerBookingController(connectionString);

**break**;

**case** 6:

action = SecondMenu("Worker");

controller = new WorkerController(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 to operate with:");

Console.WriteLine("Press the number to choose a table");

Console.WriteLine("1.Hotel");

Console.WriteLine("2.Booking");

Console.WriteLine("3.CustomerWorker");

Console.WriteLine("4.Customer");

Console.WriteLine("5.CustomerBooking");

Console.WriteLine("6.Worker");

Console.WriteLine("0.Exit");

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

} **while** (choice < 0 || choice > 6 || correct == **false**);

**return** choice;

}

**public** **static** **int** SecondMenu(**string** tableToChange)

{

**var** choice = 0;

**var** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("What do you want to do with '" + tableToChange + "' table:");

Console.WriteLine("Press the number to choose an action");

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;

}

}

}

**HotelController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** HotelController : BasedController

{

**public** HotelController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, location, name from Hotel";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("location: {0}", rdr.GetValue(1));

Console.WriteLine("name: {0}", rdr.GetValue(2));

Console.WriteLine();

}

}

**catch** (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

**finally**

{

sqlConnection.Close();

}

Console.ReadLine();

}

**public** **override** **void** Create()

{

**string** sqlInsert = "Insert into hotel(location, name) VALUES(@location, @name)";

**string** location = **null**;

**string** name = **null**;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Provide the Hotel properties:");

Console.WriteLine("Locatoin:");

location = Console.ReadLine();

**if** (location.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of location shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

Console.WriteLine("name:");

name = Console.ReadLine();

**if** (name.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of name shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("location", location);

cmd.Parameters.AddWithValue("name", name);

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 hotel where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update hotel ");

}

**public** **override** **void** Find()

{

**base**.Find();

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** sqlGenerate = "insert into hotel(location, name) (select "

+ **base**.sqlRandomString

+ ", "

+ **base**.sqlRandomString

+ "from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

**base**.Generate(sqlGenerate);

}

}

}

**WorkerController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** WorkerController : BasedController

{

**public** WorkerController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, name, surname, hotelId from worker";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("Name: {0}", rdr.GetValue(1));

Console.WriteLine("Surname: {0}", rdr.GetValue(2));

Console.WriteLine("Hotel Id: {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 worker (name,surname, hotelid) VALUES(@name,@surname,@hotelid)";

**string** name = **null**;

**string** surname = **null**;

**int** hotel\_id = 0;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Enter worker properties:");

Console.WriteLine("Name:");

name = Console.ReadLine();

**if** (name.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of name shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Surname:");

surname = Console.ReadLine();

**if** (surname.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of surname shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Hotel id:");

correct = Int32.TryParse(Console.ReadLine(), **out** hotel\_id);

**if** (correct == **false**)

{

Console.WriteLine("Hotel id must be a number!");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("name", name);

cmd.Parameters.AddWithValue("surname", surname);

cmd.Parameters.AddWithValue("hotelid", hotel\_id);

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 worker where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update worker ");

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** sqlGenerate = "insert into worker(name, surname, hotelid) (select "

+ **base**.sqlRandomString

+ ", "

+ **base**.sqlRandomString

+ ",hotel.id from generate\_series(1, 1000000), hotel limit(" + recordsAmount + "))";

**base**.Generate(sqlGenerate);

}

}

}

**CostumerController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** CostumerController : BasedController

{

**public** CostumerController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, email, name, surname from customer";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("email: {0}", rdr.GetValue(1));

Console.WriteLine("Name: {0}", rdr.GetValue(2));

Console.WriteLine("Surname: {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 customer(email,name,surname) VALUES(@email, @name, @surname)";

**string** email = **null**;

**string** name = **null**;

**string** surname = **null**;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Enter Customer properties:");

Console.WriteLine("Email:");

email = Console.ReadLine();

**if** (email.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of email shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

name = Console.ReadLine();

**if** (name.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of name shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

surname = Console.ReadLine();

**if** (surname.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of surname shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("email", email);

cmd.Parameters.AddWithValue("name", name);

cmd.Parameters.AddWithValue("surname", surname);

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 customer where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update customer ");

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** sqlGenerate = "insert into customer(email,name,surname) (select "

+ **base**.sqlRandomString

+ ", "

+ **base**.sqlRandomString

+ ", "

+ **base**.sqlRandomString

+ "from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

**base**.Generate(sqlGenerate);

}

}

}

**CustomerWorkerController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** CustomerWorkerController : BasedController

{

**public** CustomerWorkerController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, customerid, workerid from customerworker";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("Customer Id: {0}", rdr.GetValue(1));

Console.WriteLine("Worker Id: {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 customerworker (customer, worker) VALUES(@customer, @worker)";

**int** customer\_id = 0;

**int** worker\_id = 0;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Enter customerworker properties:");

Console.WriteLine("customer id:");

correct = Int32.TryParse(Console.ReadLine(), **out** customer\_id);

**if** (correct == **false**)

{

Console.WriteLine("Customer id must be a number!");

Console.ReadLine();

}

Console.WriteLine("Worker id:");

correct = Int32.TryParse(Console.ReadLine(), **out** worker\_id);

**if** (correct == **false**)

{

Console.WriteLine("Worker id must be a number!");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("customerid", customer\_id);

cmd.Parameters.AddWithValue("workerid", worker\_id);

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 customerworker where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update customerworker ");

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** sqlGenerate = "insert into customerworker(customerid, workerid) (select customer.id, worker.id"

+ " from customer, worker limit(" + recordsAmount + "))";

**base**.Generate(sqlGenerate);

}

}

}

**BookingController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** BookingController : BasedController

{

**public** BookingController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, date, term from booking";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("Date: {0}", rdr.GetValue(1));

Console.WriteLine("Term: {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 booking(date, term) VALUES(@date, @term)";

**string** date = **null**;

**int** term = 0;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Enter Booking properties:");

Console.WriteLine("date:");

**if** (date.Length > 40)

{

correct = **false**;

Console.WriteLine("Length of date shouldn't be bigger than 40.");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Term:");

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

**if** (correct == **false**)

{

Console.WriteLine("Term must be a number!");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("date", date);

cmd.Parameters.AddWithValue("term", term);

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 booking where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update booking ");

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** sqlGenerate = "insert into booking(date, term) (select "

+ **base**.sqlRandomString

+ ", "

+ **base**.sqlRandomInteger

+ " bookingid from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

**base**.Generate(sqlGenerate);

}

}

}

**CustomerBookingController.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** BD2.Controllers

{

**public** **class** CustomerBookingController : BasedController

{

**public** CustomerBookingController(**string** connectionString) : **base**(connectionString) { }

**public** **override** **void** Read(**string** whereCondition)

{

Console.Clear();

sqlConnection.Open();

**string** sqlSelect = "select id, customerid, bookingid from customerbooking";

**using** **var** cmd = new NpgsqlCommand(sqlSelect + whereCondition, sqlConnection);

**try**

{

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

**while** (rdr.Read())

{

Console.WriteLine("Id: {0}", rdr.GetValue(0));

Console.WriteLine("Customer Id: {0}", rdr.GetValue(1));

Console.WriteLine("Booking Id: {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 customerbooking (customerid, bookingid) VALUES(@customerid, @bookingid)";

**int** customer\_id = 0;

**int** booking\_id = 0;

**bool** correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Enter customerbooking properties:");

Console.WriteLine("Customer id:");

correct = Int32.TryParse(Console.ReadLine(), **out** customer\_id);

**if** (correct == **false**)

{

Console.WriteLine("Customer id must be a number!");

Console.ReadLine();

}

Console.WriteLine("Booking id:");

correct = Int32.TryParse(Console.ReadLine(), **out** booking\_id);

**if** (correct == **false**)

{

Console.WriteLine("Booking id must be a number!");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

**using** **var** cmd = new NpgsqlCommand(sqlInsert, sqlConnection);

cmd.Parameters.AddWithValue("customerid", customer\_id);

cmd.Parameters.AddWithValue("bookingid", booking\_id);

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 customerbooking where id = ");

}

**public** **override** **void** Update()

{

**base**.Update("Update customerbooking");

}

**public** **override** **void** Generate()

{

Console.WriteLine("What amount of records do you want?");

**bool** correct = **false**;

**int** recordsAmount;

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

**string** subquery = "with pa as ( select customer.id from customer where customer.id not in ( select customer.id from customerbooking join customer on customerbooking.customerid = customer.id ) limit(1)), "

+ "va as ( select booking.id from booking where booking.id not in ( select booking.id from customerbooking join booking on customerbooking.bookingid = bookingid ) limit(1))";

**string** sqlGenerate = subquery + " insert into customerbooking(customerid, bookingid) (select pa.id, va.id from pa, va limit(1))";

**for** (**int** i = 0; i < recordsAmount; i++)

{

**base**.Generate(sqlGenerate);

}

}

}

}