

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 та засоби управління»**

ТЕМА: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконала:

студентка ІII курсу

групи КВ-02

Дахал К.

Перевірив:

доц. Павловський В.І.

Київ 2022

[Посилання](https://github.com/arrooww/bd_lab2) на gitbub.

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. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі. 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:

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.

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!

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.

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

**Логічна модель (схема) БД «Музичний гурт»**

Логічну модель (схему бази даних) наведено на рисунку 1.

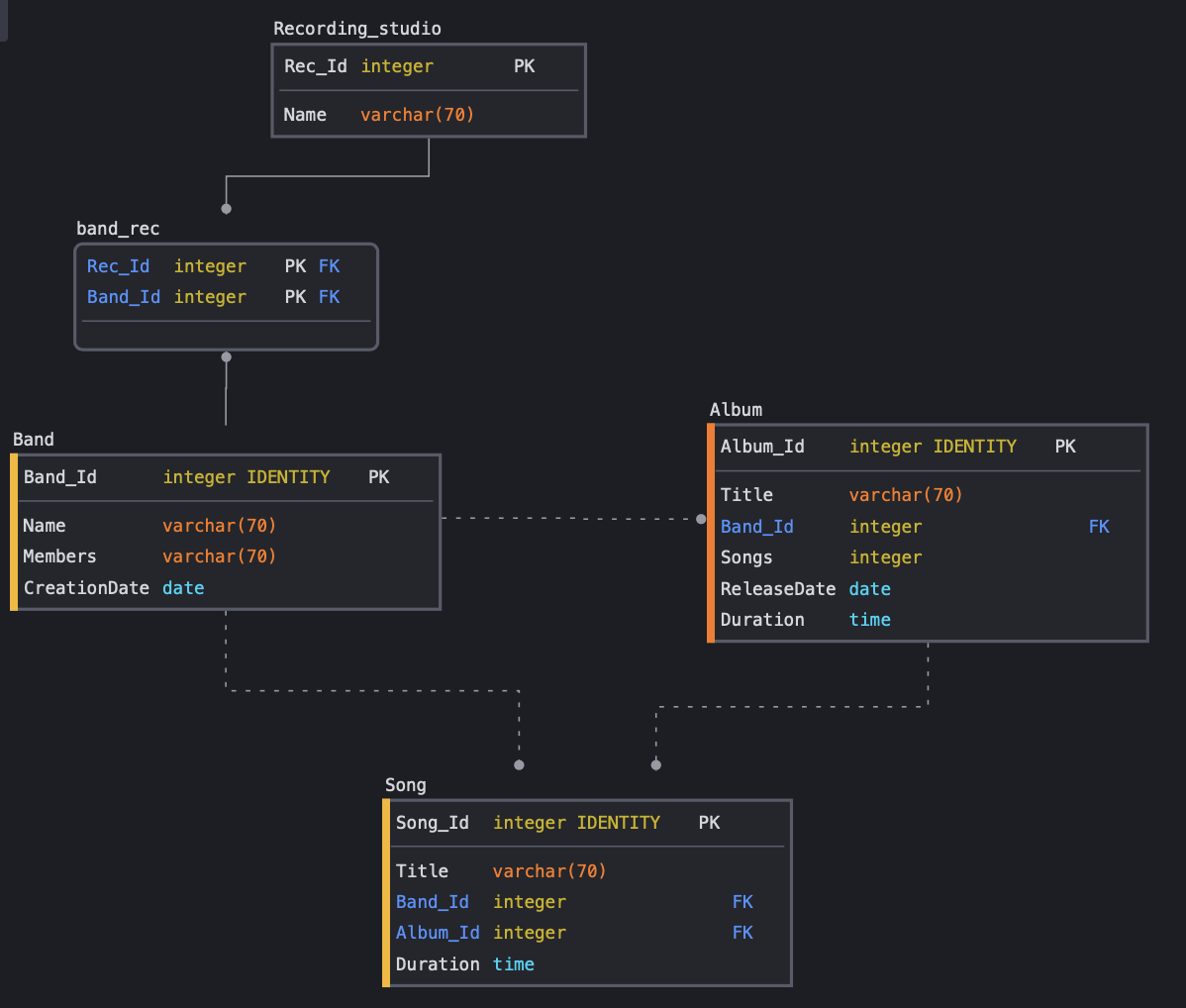


Рисунок 1 – Схема бази даних (інструмент: sqldbm.com)

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувалась мова програмування C# та середовище розробки Visual Studio Code. Для підключення до серверу бази даних PostgreSQL використано сторонню бібліотеку Npgsql.

**Шаблон проектування**

MVC - шаблон проектування, що був використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, в цій програмі відповідають всі компоненти які знаходять у папці Models.

View – консольний інтерфейс, з яким буде взаємодіяти наш користувач. Йому відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Отримує введені користувачем дані, обробляє їх та в залежності від результатів обробки відправляє користувачеві певний висновок.

**Структура програми та її опис**

Нижче наведено структуру програми:

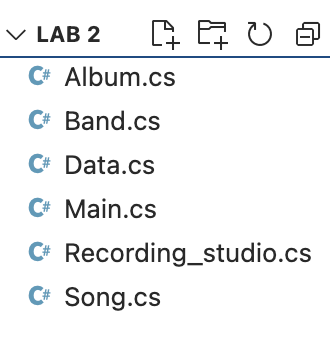


Рисунок 2 – Структура програми

Програма умовно поділена на 6 модулів:

* Main.cs – модуль, що викликає методи з контролерів та містить засоби для обробки виключень та повідомлення про помилки;
* Data.cs – модуль, що містить функції та засоби для підключення бази даних до програми;
* Band.cs – контролер для таблиці Band;
* Album.cs – контролер для таблиці Album;
* Song.cs – контролер для таблиці Song;
* Recording\_studio.cs – контролер для таблиці Recording\_studio;

**Структура меню програми**

Нижче наведено головне меню програми:

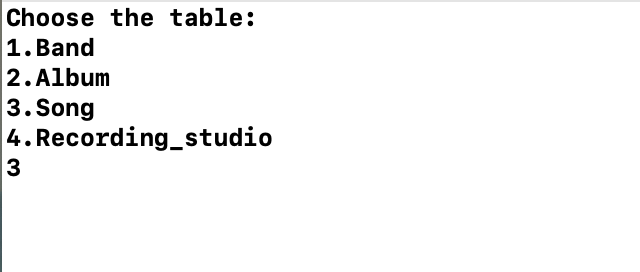


Рисунок 3 – Головне меню програми.

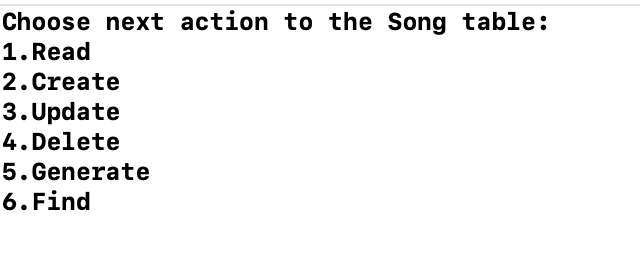
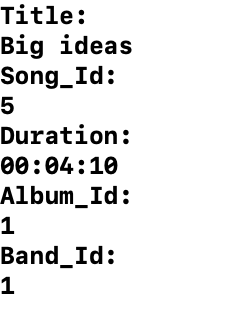


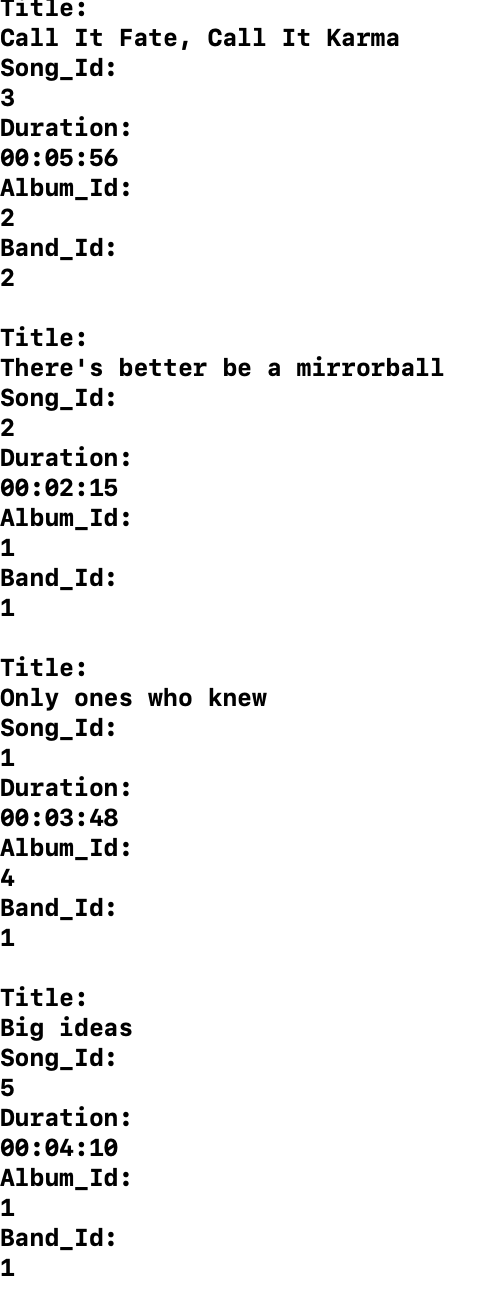
Рисунок 4 – Другорядне меню програми.

**Результати роботи програми**

1. Додавання даних до бази даних.



Результат операції:



SQL-запит:

string sqlInsert = "Insert into Song(Title, Band\_Id, Album\_Id, Duration) VALUES(@Title, @Band\_Id, @Album\_Id, @Duration)";

Для контролю наявності відповідного рядка у батьківській таблиці під час внесення даних до дочірньої таблиці в коді передбачена спеціальна процедура:

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

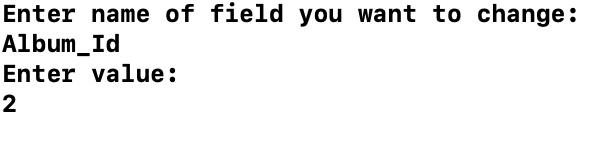
finally

{

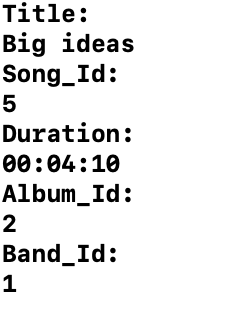
sqlConnection.Close();

}

1. Редагування даних.



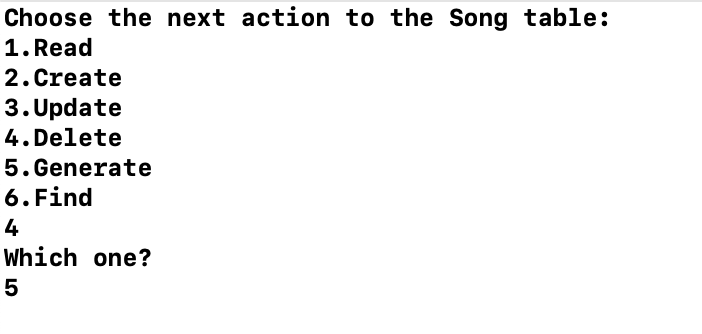
Результат операції:



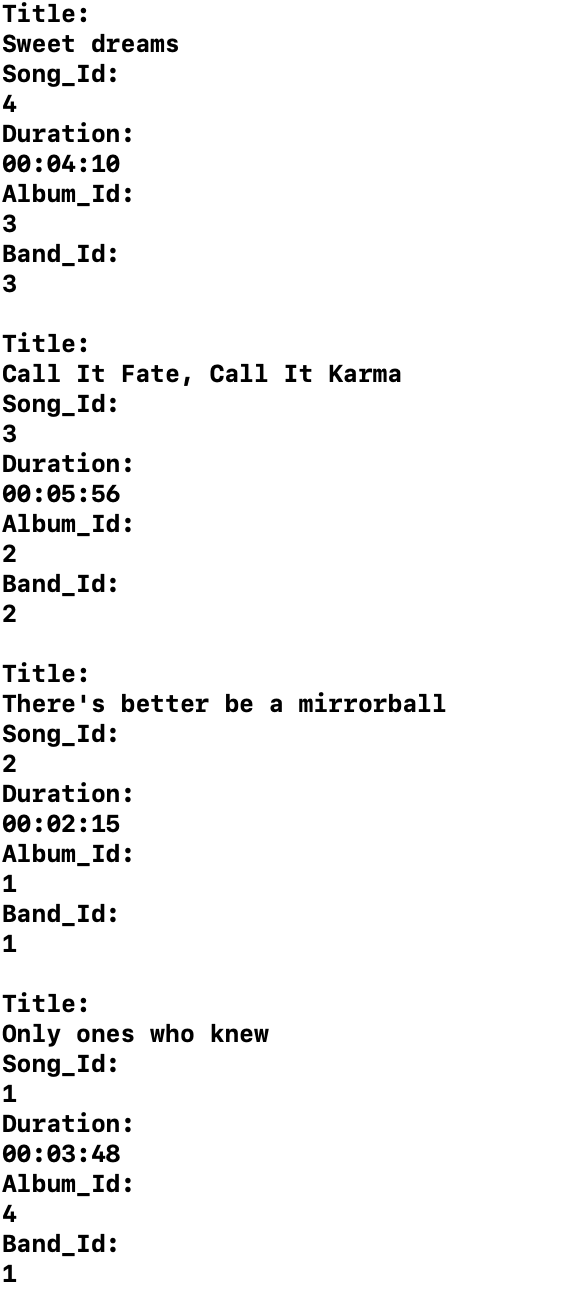
SQL-запит:

string sqlUpdate = "Update @table set @field\_to\_update = @new\_value where @field\_to\_find = @old\_value";

1. Видалення даних.

****

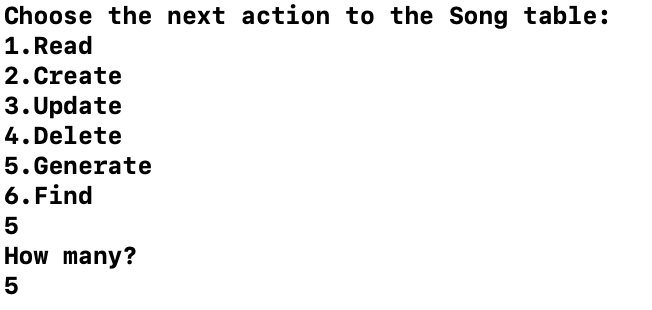
Результат операції:

****

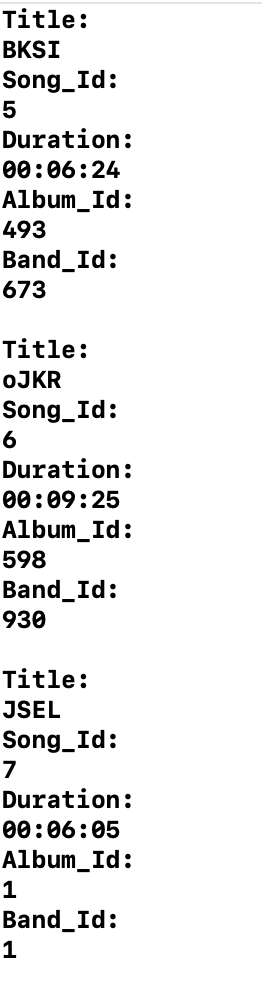
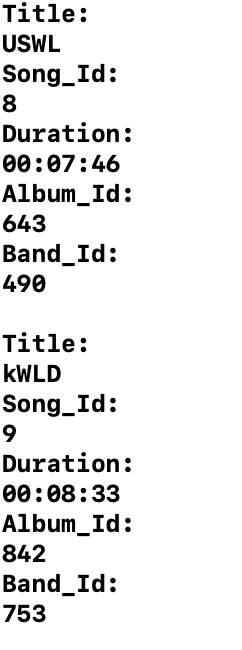
SQL-запит:

base.Delete("Delete from Song where Song\_Id = ");

1. Додавання одразу декількох в таблицю за допомогою операції Generate.



Результат додавання:

SQL-запит:

string sqlGenerate = "Insert into Song(Title, Band\_Id, Album\_Id, Duration) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

**public** **readonly** string sqlUpdate = "Update @table set @field\_to\_update = @new\_value where @field\_to\_find = @old\_value";

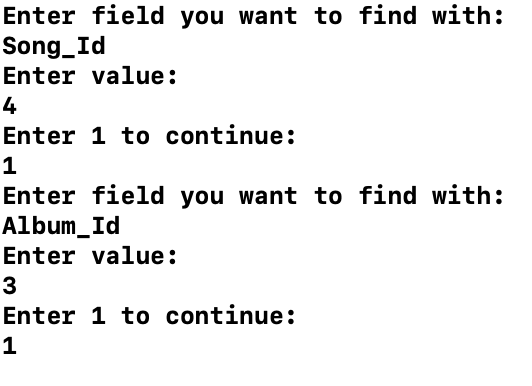
**public** **readonly** string sqlRandomString = "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)";

**public** **readonly** string sqlRandomInteger = "trunc(random()\*1000)::int";

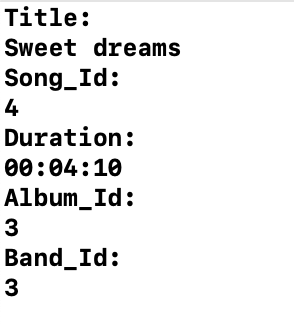
**public** **readonly** string sqlRandomDate = "timestamp '00:10:59' + random() \* (timestamp '00:10:59' - timestamp '00:05:00')";

**public** **readonly** string sqlRandomBoolean = "trunc(random()\*2)::int::boolean";

1. Пошук за декількома атрибутами одночасно.



Результат пошуку:



**Лістинг програми**

**Main.cs**

**using** bd\_lab2.Controllers;

**using** System;

**namespace** bd\_lab2

{

**class** Program

{

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

{

string connectionString = "Host=localhost; Username=postgres; Password=4545; Database=BD";

**int** table = 0;

**int** action = 0;

**do**

{

table = FirstMenu();

**if** (table == 0)

{

**return**;

}

BaseController controller = null;

**switch** (table)

{

**case** 1:

action = SecondMenu("Band");

controller = **new** Band(connectionString);

**break**;

**case** 2:

action = SecondMenu("Album");

controller = **new** Album(connectionString);

**break**;

**case** 3:

action = SecondMenu("Song");

controller = **new** Song(connectionString);

**break**;

**case** 4:

action = SecondMenu("Recording studio");

controller = **new** Recording\_studio(connectionString);

**break**;

}

**switch** (action)

{

**case** 1:

controller.Read();

**break**;

**case** 2:

controller.Create();

**break**;

**case** 3:

controller.Update();

**break**;

**case** 4:

controller.Delete();

**break**;

**case** 5:

controller.Generate();

**break**;

**case** 6:

controller.Find();

**break**;

}

} **while** (**true**);

}

**public** **static** **int** FirstMenu()

{

var choice = 0;

var correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Choose the table:");

Console.WriteLine("1.Band");

Console.WriteLine("2.Album");

Console.WriteLine("3.Song");

Console.WriteLine("4.Recording studio");

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

} **while** (choice < 0 || choice > 4 || correct == **false**);

**return** choice;

}

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

{

var choice = 0;

var correct = **false**;

**do**

{

Console.Clear();

Console.WriteLine("Choose next action to the'" + tableToChange + "' table:");

Console.WriteLine("1.Read");

Console.WriteLine("2.Create");

Console.WriteLine("3.Update");

Console.WriteLine("4.Delete");

Console.WriteLine("5.Generate");

Console.WriteLine("6.Find");

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

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

**return** choice;

}

}

}

**Data.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** bd\_lab2.Controllers

{

**public** abstract **class** Data

{

**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 '00:10:59' + random() \* (timestamp '00:10:59' - timestamp '00:05:00')";

**public** readonly string sqlRandomBoolean = "trunc(random()\*2)::int::boolean";

**public** Data(string connectionString)

{

**this**.connectionString = connectionString;

**this**.sqlConnection = **new** NpgsqlConnection(connectionString);

}

**public** **void** Read()

{

Read("");

}

**public** **virtual** **void** Create()

{

**throw** **new** NotImplementedException();

}

**public** **virtual** **void** Update()

{

**throw** **new** NotImplementedException();

}

**public** **virtual** **void** Delete()

{

**throw** **new** NotImplementedException();

}

**virtual** **public** **void** Generate()

{

**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:");

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** Read(string whereCondition)

{

}

**protected** **void** Delete(string sqlDelete)

{

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

**int** id = 0;

**do**

{

Console.WriteLine("Which one?");

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

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

{

Console.WriteLine("Something went wrong, try again");

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:");

valueToFind = Console.ReadLine();

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

fieldToSet = Console.ReadLine();

Console.WriteLine("Enter value:");

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();

}

}

}

}

**Band.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** bd\_lab2.Controllers

{

**public** **class** Band : Data

{

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

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

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select Band\_Id, Name, Members, CreationDate from Band";

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

**try**

{

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

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

{

Console.WriteLine("Band\_Id: {0}", rdr.GetValue(0));

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

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

Console.WriteLine("CreationDate: {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 Band(Name, Members, CreationDate) VALUES(@Name, @Members, @CreationDate)";

string Name = null;

string Members = null;

DateTime CreationDate = 0;

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

**do**

{

Console.Clear();

Console.WriteLine("Name:");

Name = Console.ReadLine();

**if** (name.Length > 70)

{

correct = **false**;

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Members:");

Members = Console.ReadLine();

**if** (members.Length > 70)

{

correct = **false**;

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

**continue**;

}

Console.WriteLine("CreationDate:");

correct = DateTime.TryParse(Console.ReadLine(), out CreationDate);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

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

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

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

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

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 Band where Band\_Id = ");

}

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

{

base.Update("Update Band ");

}

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

{

Console.WriteLine("How many?");

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

**int** recordsAmount;

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

string sqlGenerate = "Insert into categories(Name, Members, CreationDate) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

}

}

}

**Album.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** bd\_lab2.Controllers

{

**public** **class** Album : Data

{

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

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

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select Album\_Id, Title, Band\_Id, Songs, ReleaseDate, Duration from Album";

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

**try**

{

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

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

{

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

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

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

Console.WriteLine("Songs: {0}", rdr.GetValue(3));

Console.WriteLine("ReleaseDate: {0}", rdr.GetValue(4));

Console.WriteLine("Duration: {0}", rdr.GetValue(5));

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 Album(Title, Band\_Id, Song, ReleaseDate, Duration) VALUES(@Title, @Band\_Id, @Song, @ReleaseDate, @Duration)";

string Title = null;

**int** Band\_Id = 0;

**int** Song = **int**;

DataTime ReleaseDate = 0;

timestamp Duration = 0;

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

**do**

{

Console.Clear();

Console.WriteLine("Title:");

name = Console.ReadLine();

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

{

correct = **false**;

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Band Id:");

correct = Int32.TryParse(Console.ReadLine(), out Band\_Id);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Song:");

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

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Release Date:");

correct = Date.TryParse(Console.ReadLine(), out ReleaseDate);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Duration:");

correct = timestamp.TryParse(Console.ReadLine(), out Duration);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("Band\_Id", Band\_Id);

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

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

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

cmd.Prepare();

**try**

{

cmd.ExecuteNonQuery();

}

**catch** (Exception ex)

{

Console.WriteLine($"Can't be created: {ex.Message}");

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

**public** **override** **void** Delete()

{

base.Delete("Delete from Album where Album\_Id = ");

}

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

{

base.Update("Update Album: ");

}

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

{

Console.WriteLine("How many?");

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

**int** recordsAmount;

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

string sqlGenerate = "Insert into customer(Title, Band\_Id, Song, ReleaseDate, Duration) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

}

}

}

**Song.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** bd\_lab2.Controllers

{

**public** **class** Song : Data

{

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

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

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select Song\_Id, Title, Band\_Id, Album\_Id, Duration availability from Song";

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

**try**

{

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

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

{

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

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

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

Console.WriteLine("Album Id: {0}", rdr.GetValue(3));

Console.WriteLine("Duration: {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 Song(Title, Band\_Id, Album\_Id, Duration) VALUES(@Title, @Band\_Id, @Album\_Id, @Duration)";

string Title = null;

**int** Band\_Id = 0;

**int** Album\_Id = 0;

timestamp Duration = 0;

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

**do**

{

Console.Clear();

Console.WriteLine("Title:");

name = Console.ReadLine();

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

{

correct = **false**;

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

**continue**;

}

Console.WriteLine("Band Id:");

correct = Int32.TryParse(Console.ReadLine(), out Band\_Id);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Album Id:");

correct = Int32.TryParse(Console.ReadLine(), out Album\_Id);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Duration:");

correct = timestamp.TryParse(Console.ReadLine(), out Duration);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

correct = **true**;

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

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("Band\_Id", Band\_Id);

cmd.Parameters.AddWithValue("Album\_Id", Album\_Id);

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

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 Song where Song\_Id = ");

}

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

{

base.Update("Update Song ");

}

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

{

base.Find();

}

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

{

Console.WriteLine("How many?");

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

**int** recordsAmount;

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

string sqlGenerate = "Insert into Song(Title, Band\_Id, Album\_Id, Duration) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

}

}

}

**Recording\_studio.cs**

**using** Npgsql;

**using** System;

**using** System.Collections.Generic;

**using** System.Text;

**namespace** bd\_lab2.Controllers

{

**public** **class** Recording\_studio : Data

{

**public** Recording\_studio(string connectionString) : base(connectionString) { }

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

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select Rec\_Id, Title from Recording\_studio";

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

**try**

{

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

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

{

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

Console.WriteLine("Name: {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 Recording\_studio (Rec\_Id, Name) VALUES(@Rec\_Id, @Name)";

**int** Rec\_Id= 0;

string Name = null;

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

**do**

{

Console.Clear();

Console.WriteLine("Rec\_Id:");

correct = Int32.TryParse(Console.ReadLine(), out Rec\_Id);

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

{

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

}

Console.WriteLine("Name:");

name = Console.ReadLine();

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

{

correct = **false**;

Console.WriteLine("Something went wrong, try again");

Console.ReadLine();

**continue**;

}

correct = **true**;

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

sqlConnection.Open();

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

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

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 Recording\_studio where Rec\_Id = ");

}

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

{

base.Update("Update Recording\_studio ");

}

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

{

Console.WriteLine("How many?");

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

**int** recordsAmount;

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

string sqlGenerate = "Insert into Recording\_studio(Name) (select "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomDate

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

}

}

}