Міністерство освіти та науки України

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

**ЗВІТ**

про виконання лабораторної роботи №5

на тему:

**«Технологія ADO.NET підключення/автономний рівень»**

**Виконав:** студент групи ІС-63

Портяний І.С.

**Перевірив:** ст.вик. Москаленко Н. В.

Київ 2018

**Варіант №14**

Для виконання даної лабораторної роботи я обрав власний варіант. **Постановка задачі має наступний вигляд:**

Кафедра забезпечує викладання певних дисциплін. Для кожної дисципліни передбачені певні години педагогічного навантаження по видах:

лекції, практичні заняття, лабораторні роботи, практика (це також вид дисципліни), дипломне проектування, екзамен. Має значення також посада викладача: певні види навантаження (як-от практичні та лабораторні заняття)

може виконувати асистент, інші – лише старший викладач, доцент або професор.

На кафедрі існують викладачі, як штатні, так і сумісники. Кожний викладач викладає кілька дисциплін (в середньому біля чотирьох).



Рис. 1 Структура БД

Створено та заповнено даними базу даних в MS SQL Server 2014.

У програмі створено клас **DB.cs** для роботи з базою даних. У класі реалізовані методи для додавання, видалення та оновлення даних. Виведення таблиць реалізовано за допомогою об’єктів класів DataAdapter та DataSet. Так як клас всього один, то діаграму класів не побудовано. Нижче наведено результати роботи програми. Показано таблиці як в самій програмі так і їх вигляд в SQL Server 2014 Management Studio. З скріншотів видно, що всі зміни, які відбуваються з таблицями під час роботи в програмі, можемо побачити й в Management Studio.

**Скріншоти роботи програми.**

Головне меню програми:

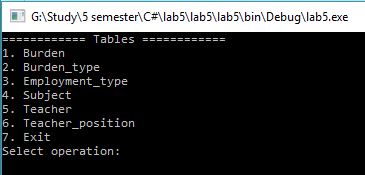
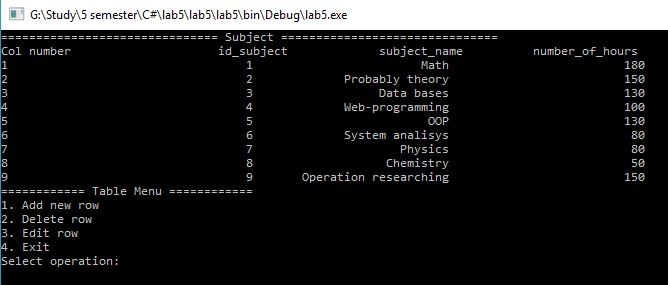


Рис. 2 Головне меню програми

Виведення кількох таблиць:



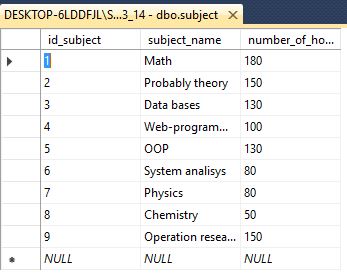


Рис. 3 Таблиця Subject в SQL Server та у програмі

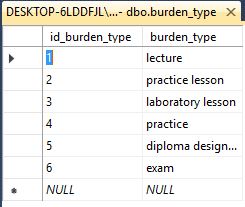
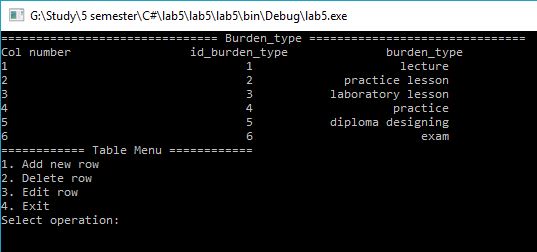
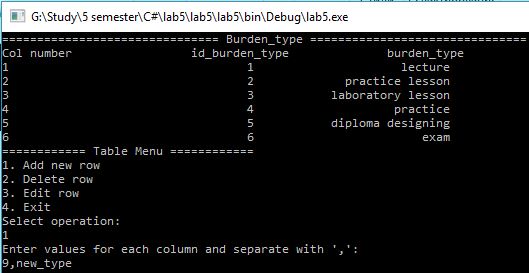
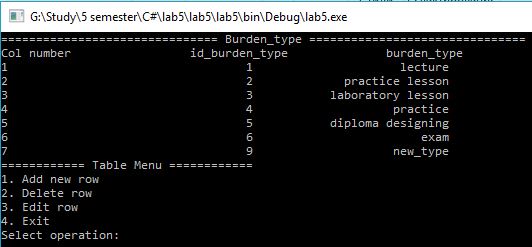


Рис. 4 Таблиця Burden\_type в SQL Server та у програмі

Попрацюємо з таблицею **burden\_type**. Додамо, відредагуємо та видалимо рядок.

Додамо новий рядок:





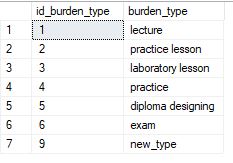
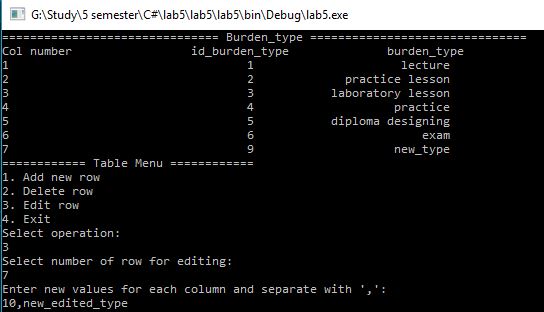
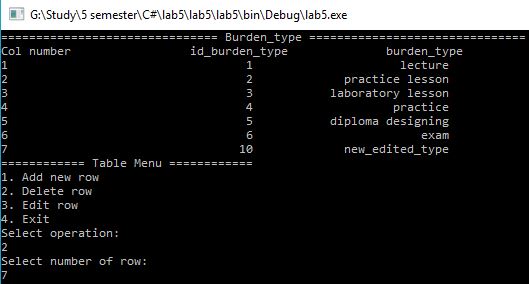


Рис. 5 Таблиця Burden\_type з новим рядком в SQL Server та у програмі

Змінимо цей рядок:





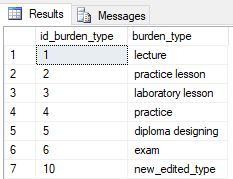
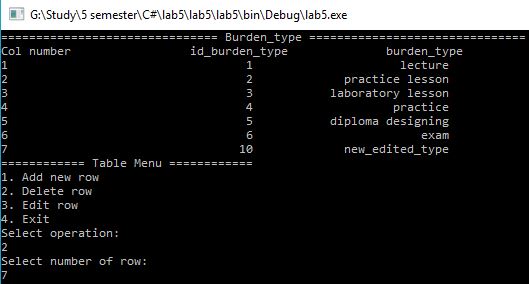


Рис. 6 Таблиця Burden\_type з редагованим рядком в SQL Server та у програмі

Тепер видалимо цей запис:



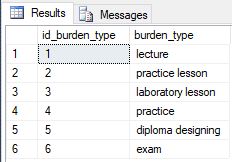
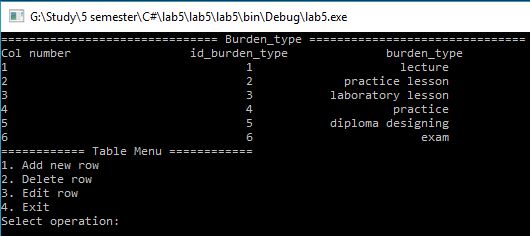


Рис. 7 Таблиця Burden\_type з видаленим останнім записом в SQL Server та у програмі

**Код проекту**

**Клас для роботи з базою даних**

**DB.cs**

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab5

{

class DB

{

DataSet ds = new DataSet();

SqlDataAdapter adapter = new SqlDataAdapter();

string connectionString = ConfigurationManager.ConnectionStrings["Lab5Connection"].ConnectionString;

public string[] Tables { get; private set; } = new string[] { "Burden", "Burden\_type", "Employment\_type", "Subject", "Teacher", "Teacher\_position" };

public void ShowTable(int numberOfTable)

{

const int sizeTitle = 25;

const int sizeContent = 28;

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(GetSQLCode(numberOfTable), connection);

SqlDataReader reader = command.ExecuteReader();

Console.WriteLine($"=============================== {Tables[numberOfTable]} ===============================");

if (reader.HasRows)

{

Console.Write("Col number\t");

for (int i = 0; i < reader.FieldCount; i++)

{

if (numberOfTable == 4)

{

Console.Write($"{reader.GetName(i), 20}");

}

else

Console.Write($"{reader.GetName(i),sizeTitle}");

}

Console.WriteLine();

int index = 0;

while (reader.Read())

{

index++;

Console.Write(index + "\t");

for (int i = 0; i < reader.FieldCount; i++)

{

if (numberOfTable == 4)

{

Console.Write($"{reader.GetValue(i), 21}");

}

else

Console.Write($"{reader.GetValue(i),sizeContent}");

}

Console.WriteLine();

}

}

else

{

Console.WriteLine("Empty table");

}

}

}

private string GetSQLCode(int numberOfTable)

{

if (numberOfTable < 0 || numberOfTable >= Tables.Length)

{

throw new Exception("Incorrect number of table!");

}

return "Select \* From " + Tables[numberOfTable];

}

public bool Add(int numberOfTable, string row)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

adapter.SelectCommand = new SqlCommand(GetSQLCode(numberOfTable), connection);

adapter.Fill(ds);

DataRow newRow;

switch (numberOfTable)

{

case 0:

{

newRow = EditBurden(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

case 1:

{

newRow = EditBurdenType(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

case 2:

{

newRow = EditEmploymentType(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

case 3:

{

newRow = EditSubject(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

case 4:

{

newRow = EditTeacher(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

case 5:

{

newRow = EditTeacherPosition(row, ds.Tables[0].NewRow());

if (newRow == null)

{

connection.Close();

return false;

}

break;

}

default:

{

connection.Close();

return false;

}

}

ds.Tables[0].Rows.Add(newRow);

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);

adapter.Update(ds);

}

return true;

}

public bool Delete(int numberOfTable, int rowIndex)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

adapter.SelectCommand = new SqlCommand(GetSQLCode(numberOfTable), connection);

adapter.Fill(ds);

DataTable dt = ds.Tables[0];

Console.WriteLine($"Number of rows: {dt.Rows.Count}");

if (dt.Rows.Count < rowIndex)

{

connection.Close();

if (dt.Rows.Count == 0)

{

return true;

}

return false;

}

dt.Rows[rowIndex - 1].Delete();

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);

adapter.Update(ds);

}

return true;

}

public bool Edit(int numberOfTable, int rowIndex, string row)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

adapter.SelectCommand = new SqlCommand(GetSQLCode(numberOfTable), connection);

adapter.Fill(ds);

if (rowIndex > ds.Tables[0].Rows.Count)

{

connection.Close();

if (ds.Tables[0].Rows.Count == 0)

{

return true;

}

return false;

}

DataRow editRow = ds.Tables[0].Rows[rowIndex - 1];

switch (numberOfTable)

{

case 0:

{

if (EditBurden(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

case 1:

{

if (EditBurdenType(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

case 2:

{

if (EditEmploymentType(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

case 3:

{

if (EditSubject(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

case 4:

{

if (EditTeacher(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

case 5:

{

if (EditTeacherPosition(row, editRow) == null)

{

connection.Close();

return false;

}

break;

}

default:

{

connection.Close();

return false;

}

}

SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);

adapter.Update(ds);

}

return true;

}

private DataRow EditBurden(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idBurden;

int numberOfHours;

int fkIDTeacher;

int fkIDSubject;

int fkIDBurdenType;

if (rowData.Length < 5)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idBurden))

{

return null;

}

if (!int.TryParse(rowData[1].Replace(" ", string.Empty), out numberOfHours))

{

return null;

}

if (!int.TryParse(rowData[2].Replace(" ", string.Empty), out fkIDTeacher))

{

return null;

}

if (!int.TryParse(rowData[3].Replace(" ", string.Empty), out fkIDSubject))

{

return null;

}

if (!int.TryParse(rowData[4].Replace(" ", string.Empty), out fkIDBurdenType))

{

return null;

}

newRow["id\_burden"] = idBurden;

newRow["number\_of\_hours"] = numberOfHours;

newRow["fk\_id\_teacher"] = fkIDTeacher;

newRow["fk\_id\_subject"] = fkIDSubject;

newRow["fk\_id\_burden\_type"] = fkIDBurdenType;

return newRow;

}

private DataRow EditBurdenType(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idBurdenType;

string burdenTypeName;

if (rowData.Length < 2)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idBurdenType))

{

return null;

}

burdenTypeName = rowData[1];

newRow["id\_burden\_type"] = idBurdenType;

newRow["burden\_type"] = burdenTypeName;

return newRow;

}

private DataRow EditEmploymentType(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idEmploymentType;

string employmentTypeName;

if (rowData.Length < 2)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idEmploymentType))

{

return null;

}

employmentTypeName = rowData[1];

newRow["id\_employment\_type"] = idEmploymentType;

newRow["employment\_type\_name"] = employmentTypeName;

return newRow;

}

private DataRow EditSubject(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idSubject;

string subjectName;

int numberOfHours;

if (rowData.Length < 3)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idSubject))

{

return null;

}

subjectName = rowData[1];

if (!int.TryParse(rowData[2].Replace(" ", string.Empty), out numberOfHours))

{

return null;

}

newRow["id\_subject"] = idSubject;

newRow["subject\_name"] = subjectName;

newRow["number\_of\_hours"] = numberOfHours;

return newRow;

}

private DataRow EditTeacher(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idTeacher;

string firstName;

string lastName;

string adress;

string phoneNumber;

int numberOfHours;

int fkIDPosition;

int fkIDEmploymentType;

if (rowData.Length < 8)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idTeacher))

{

return null;

}

firstName = rowData[1];

lastName = rowData[2];

adress = rowData[3];

phoneNumber = rowData[4];

if (!int.TryParse(rowData[5].Replace(" ", string.Empty), out numberOfHours))

{

return null;

}

if (!int.TryParse(rowData[6].Replace(" ", string.Empty), out fkIDPosition))

{

return null;

}

if (!int.TryParse(rowData[7].Replace(" ", string.Empty), out fkIDEmploymentType))

{

return null;

}

newRow["id\_teacher"] = idTeacher;

newRow["firstname"] = firstName;

newRow["lastname"] = lastName;

newRow["adress"] = adress;

newRow["phone\_number"] = phoneNumber;

newRow["number\_of\_hours"] = numberOfHours;

newRow["fk\_id\_position"] = fkIDPosition;

newRow["fk\_id\_employment\_type"] = fkIDEmploymentType;

return newRow;

}

private DataRow EditTeacherPosition(string row, DataRow newRow)

{

string[] rowData = row.Split(',');

int idPosition;

string positionName;

if (rowData.Length < 2)

{

return null;

}

if (!int.TryParse(rowData[0].Replace(" ", string.Empty), out idPosition))

{

return null;

}

positionName = rowData[1];

newRow["id\_position"] = idPosition;

newRow["position\_name"] = positionName;

return newRow;

}

}

}

**Main**

**Program.cs**

//Особисто запропонований варіант

//Кафедра забезпечує викладання певних дисциплін.

//Для кожної дисципліни передбачені певні години

//педагогічного навантаження по видах: лекції, практичні

//заняття, лабораторні роботи, практика(це також вид дисципліни),

//дипломне проектування, екзамен. Має значення також посада викладача:

//певні види навантаження(як-от практичні та лабораторні заняття) може

//виконувати асистент, інші – лише старший викладач, доцент або професор.

//На кафедрі існують викладачі, як штатні, так і сумісники.

//Кожний викладач викладає кілька дисциплін(в середньому біля чотирьох).

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab5

{

class Program

{

static DB db = new DB();

static void Main(string[] args)

{

MainMenu();

Console.WriteLine("Press any key...");

Console.Read();

}

private static void MainMenu()

{

Console.Clear();

bool flag = true;

while (flag)

{

Console.WriteLine("============ Tables ============");

int iter = 0;

for (int i = 0; i < db.Tables.Length; i++)

{

Console.WriteLine($"{i + 1}. {db.Tables[i]}");

iter = i;

}

Console.WriteLine($"{iter + 2}. Exit");

int operation;

Console.WriteLine("Select operation:");

string input = Console.ReadLine();

if (!int.TryParse(input, out operation))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect operation!");

Console.WriteLine();

}

else if (operation < 1 || operation > db.Tables.Length + 1)

{

flag = true;

Console.WriteLine();

Console.WriteLine($"Number of operation must be between 1 and {db.Tables.Length + 1}");

Console.WriteLine();

}

else if (operation == iter + 2)

{

System.Environment.Exit(1);

}

else

{

TableMenu(operation - 1);

flag = true;

}

}

}

private static void TableMenu(int numberOfTable)

{

Console.Clear();

bool flag = true;

db.ShowTable(numberOfTable);

while (flag)

{

Console.WriteLine("============ Table Menu ============");

Console.WriteLine("1. Add new row");

Console.WriteLine("2. Delete row");

Console.WriteLine("3. Edit row");

Console.WriteLine("4. Exit");

Console.WriteLine("Select operation:");

int operation;

string input = Console.ReadLine();

if (!int.TryParse(input, out operation))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect operation!");

Console.WriteLine();

}

else

{

switch (operation)

{

case 1:

{

flag = false;

Add(numberOfTable);

break;

}

case 2:

{

flag = false;

Delete(numberOfTable);

break;

}

case 3:

{

flag = false;

Edit(numberOfTable);

break;

}

case 4:

{

flag = false;

MainMenu();

break;

}

default:

{

flag = true;

Console.WriteLine();

Console.WriteLine("Number of operation must be between 1 and 4");

Console.WriteLine();

break;

}

}

}

}

}

private static void Add(int numberOfTable)

{

bool flag = true;

while (flag)

{

Console.WriteLine("Enter values for each column and separete with ',':");

string input = Console.ReadLine();

if (!db.Add(numberOfTable, input))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect data!");

Console.WriteLine();

}

else

flag = false;

}

}

private static void Delete(int numberOfTable)

{

bool flag = true;

while (flag)

{

Console.WriteLine("Select number of row:");

int rowNumber;

string input = Console.ReadLine();

if (!int.TryParse(input, out rowNumber))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect number!");

Console.WriteLine();

}

else if (!db.Delete(numberOfTable, rowNumber))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Input number must be between 1 and number of latest row");

Console.WriteLine();

}

else

{

flag = false;

}

}

}

private static void Edit(int numberOfTable)

{

bool flag = true;

while (flag)

{

Console.WriteLine("Select number of row for editing:");

int rowNum;

string input1 = Console.ReadLine();

Console.WriteLine("Enter new values for each column and separete with ',':");

string input2 = Console.ReadLine();

if (!int.TryParse(input1, out rowNum))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect number!");

Console.WriteLine();

}

else if (!db.Edit(numberOfTable, rowNum, input2))

{

flag = true;

Console.WriteLine();

Console.WriteLine("Incorrect data!");

Console.WriteLine();

}

else

{

flag = false;

}

}

}

}

}