<table>
<tr><td colspan="4">Laboratorium 1
Sprawozdanie z realizacji laboratorium</td></tr>
<tr><td>Temat: Podstawy, nawiązywanie połączenia z bazą danych, zapisywanie rekordów do bazy danych</td><td>Nr Albumu:
028487</td><td>Grupa/zespół:
GL01</td><td>Rok/semestr: III / 6</td></tr>
<tr><td rowspan="2">Wykonał:
Oleksii Hudzishevskyi</td><td colspan="2">Data wykonania:
27/02/2023</td><td>Data oddania:
15/03/2023</td></tr>
<tr><td colspan="2">Ocena:</td><td>Podpis prowadzącego:</td></tr>
</table>

1. Spis treści

2. Cel ćwiczenia

Zapoznanie się z podstawowymi zagadnieniami dotyczącymi łączenia się z bazą danych (Microsoft SQL Server) z poziomu aplikacji napisanej w C# (WinForms – Net.Framework, WPF, WinFormsc- .Net Core lub też winforms .Net 6) – wedle indywidualnych preferencji studenta. Operacje Wykorzystanie dostawcy danych ADO.NET – SqlClient (System.Data.SqlClient) oraz ORM np. EF6.

Główne zagadnienia realizowane w części teoretycznej ćwiczeń to:

- Omówienie podstawowych klas i metod niezbędnych do prawidłowego zainicjalizowania połączenia z bazą danych
- Ustanowienie połączenia
- Przechwytywanie błędów (SqlException)
- Wykonywanie podstawowych operacji z grupy DQL – Data Query Language.
- Zamykanie połączenia
- Wykorzystywanie bloku using.

3. Wymagania znajomości zagadnień

- Pisanie prostych aplikacji w C# lub innym obiektowym języku wysokiego poziomu
- Podstawowa znajomość SQL, umiejętność pisania zapytań do bazy danych
- Wskazana podstawowa znajomość języka angielskiego lub też umiejętność korzystania z narzędzi tłumaczenia on-line. Wynika to z faktu, że większość użytecznej i najbardziej aktualnej dokumentacji jest publikowana właśnie w języku angielskim.

4. Literatura, materiały dydaktyczne

- https://docs.microsoft.com/pl-pl/dotnet/csharp/language-reference/keywords/try-catch-finally
- https://docs.microsoft.com/pl-pl/dotnet/csharp/
- https://www.sqlpedia.pl/
- https://www.mssqltips.com/sqlservertip/5771/querying-sql-server-tables-from-net/
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/

- https://docs.microsoft.com/en-us/dotnet/api/system.data.sqlclient?view=dotnet-plat-ext-5.0

5. Wiadomości teoretyczne.

W celu przygotowania się do części praktycznej ćwiczenia, należy odnaleźć w udostępnionej dokumentacji (linki podane w punkcie 3) oraz innych źródłach definicję wybranych pojęć. Wybrać te, które w ocenie studenta są najbardziej przystępne oraz najlepiej wyjaśniają wybrane zagadnienie. Zaleca się też, aby dla każdego zagadnienia student wstawił też fragment kodu z przykładem jego użycia tzw. Snippet.

- Blok try-catch-finally przyczyny stosowania, jakie są korzyści oraz przykłady użycia
- Klasa SqlCommand i jej główne składowe, szczególności zwrócić uwagę na składowe takie jak:
  - Transaction
  - Connection
  - CommandText
  - Parameters
  - ExecuteNonQuery()
  - ExecuteReader()
  - ExecuteScalar()
- Omówić connection string jakie składowe mogą się w nim zawierać.
- Główne polecenia dla SqlConnection:
  - Open()
  - BeginTransaction()
  - Close()
- Klasa SqlDataReader i jej główne składowe
- Klasa SqlException
- Blok using przyczyny stosowania, jakie są korzyści oraz przykłady użycia, nie mylić z dyrektywą using stosowaną w części deklaracji pliku. Chodzi tutaj o blok kodu (tak samo jak na przykład try-catch-finally.

6. Przebieg ćwiczenia

Wykorzystując powyższa bazę teoretyczną oraz wiedzę zdobytą w trakcie dotychczasowych studiów oraz oczywiście bazując na informacjach odnalezionych w Internecie, należy utworzyć nową bazę danych a w niej tabele „Kody_Pocztowe" zawierającą następujące kolumny: Kod_Pocztowy, Adres, Miejscowosc, Wojewodztwo, Powiat. Na zajęciach dostępne są lokalne instancje MS SQL Server, ale zaleca się korzystać z prywatnych komputerów. Wersja instalacyjna MS SQL Server 2019 Developer Edition dostępna jest w Internecie a obraz ISO u prowadzącego zajęcia. Wraz z instrukcją laboratoryjną studenci otrzymali również plik CSV (kody.csv) będący bazą kodów pocztowych w Polsce.

Zadaniem studentów jest napisanie prostego programu (może to być aplikacja konsolowa) który to wczyta zawartość pliku csv do pamięci a następnie zapisze go w bazie danych, równocześnie dokonując pomiaru całkowitego czasu zapisywania danych w bazie, przy czym przyjąć należy, że początek pomiaru czasu jest w momencie, gdy cały plik jest już wstępnie wczytany do pamięci i rozpoczyna się procedura zapisu (przed dokonaniem pierwszego wpisu oraz przed ustanowieniem połączenia z bazą danych), a za koniec pomiaru przyjąć moment, gdy zostanie zapisany ostatni rekord z pliku oraz połączenie do bazy zostanie zamknięte.

Ćwiczenie należy wykonać kilkukrotnie, każdorazowo zmieniając sposób zapisywania danych w bazie*. Za każdym razem do ćwiczenia wykorzystywany ma być dokładnie ten sam plik oraz ta sama tabela docelowa w bazie danych, z której też każdorazowo przed przystąpieniem do testów należy usunąć wszystkie dane. Opracować wyniki, w raz z omówieniem zależności czasu realizacji w zależności od użytej metody. Dla każdej z wykorzystanych metod przedstawić takie dane jak całkowity czas zapisu danych, wyliczony średni czas zapisu na pojedynczy rekord. Opcjonalnie można też przeprowadzić testy w trybie debugowania w Visual Studio i uwzględnić zużycie CPU oraz RAM. Wszystkie dane zestawić w jednej tabeli porównawczej a w oparciu o uzyskane wyniki wyprowadzić stosowne wnioski wraz z analizą odnotowanych różnic zmierzonych wartość starając się oczywiście wyjaśnić z czego różnice te wynikają.

*W ramach testowanych metod należy obowiązkowo uwzględnić 2 poniższe scenariusze:

- Metoda zapisu dotyczy pojedynczego rekordu (jako parametr metody przekazujemy jeden rekord) a wewnątrz metody otwierane jest połączenie do bazy danych,

wykonywany jest zapis rekordu a następnie połączenie to jest zamykane. Tak więc następuje tyle wywołań metody ile jest rekordów w bazie danych.

- Metoda zapisu dotyczy całej kolekcji, a więc przekazywane są do niej wszystkie rekordy, wewnątrz metody na samym początku ustanawiane jest połączenie z bazą danych, następuje przesłanie wszystkich rekordów, a następnie rozłączenie i wyjście z metody. Dodatkowo należy wykorzystać co najmniej 2 inne wybrane przez studenta sposoby zrealizowania zapisu danych do bazy, np.: SqlBulkCopy, lub też Wykorzystując ORM np. EF**. Można wykorzystać inny ORM lub też zaproponować inne rozwiązanie, które w ocenie studenta mogłoby być wydajniejsze czasowo.

**Przy czym w ramach w EF zadanie to też może zostać wykonane na kilka sposobów:

- DbContext.SaveChanges() wywoływane po każdej operacji dodania obiektu do kontekstu
- DbContext.SaveChanges() wywoływane na końcu (po dodaniu do kontekstu wszystkich obiektów)
- DbContext.SaveChanges() wywoływane co określoną liczbę rekordów (paczkowanie)

7. Opracowanie sprawozdania

7.1 Opracowanie teoretyczne

W tym rozdziale zostały opisane definicje wybranych pojęć na podstawie znalezionych w Internecie dokumentacji oraz innych źródeł pomocniczych. Także do niektórych z nich zostały dołączone snippet'y z kodem pokazujące ich działanie w praktyce.

7.1.1 try-catch-finally

Blok `try-catch-finally` jest stosowany w celu obsługiwania przewidzianych lub nieprzewidzianych błędów, które mogą wystąpić w kodzie, który zostanie umieszczony w bloku `try`. Działanie tego bloku kodu pokazane jest na *Snippet 1. try-catch-finally*.

Snippet 1. try-catch-finally

```
try{
    // Kod programu
}
```

```
catch(Exception ex){
    // Kod programu wykonywany jeśli w bloku wyżej zostanie wychwycony Exception
    Console.WriteLine(ex.Message);
}
finally {
    // Blok kodu wykonywany zawsze, nawet jeśli zostanie wychwycony Exception albo nie
}
```

7.1.2 SqlCommand

Connection – obiekt klasy `SqlConnection`, który przedstawia używane połączenie.

CommandText – obiekt klasy `SqlCommand`, który przechowuje wyrażenie SQL, które będzie wykonywane. **Parameters** – obiekt klasy `SqlParameters`, który jest wykorzystywany do podstawiania parametrów do wykonywanego wyrażenia SQL.

Żeby móc skorzystać z tych klas wymagany jest pakiet Nu-Get System.Data.SqlClient Obiekty te zostały pokazane w praktyce na *Snippet 2. SqlCommand, Connection, CommandText, Parameters*.

Snippet 2. SqlCommand, Connection, CommandText, Parameters

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async void Main(string[] args)
        {
            string connectionString = "connectionString";

            int age = 23;
            string name = "Olek";
            string sqlExpression = "INSERT INTO Users (Name, Age) VALUES (@name, @age)";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                await connection.OpenAsync();

                SqlCommand command = new SqlCommand(sqlExpression, connection);

                SqlParameter nameParameter = new SqlParameter("@name", name);
                SqlParameter ageParameter = new SqlParameter("@age", age);
                command.Parameters.Add(nameParameter);
                command.Parameters.Add(ageParameter);

                int count = await command.ExecuteNonQueryAsync();
                Console.WriteLine(count);
            }
        }
    }
}
```

ExecuteNonQuery() – metoda, która wykonuje wyrażenie SQL i zwraca ilość zmodyfikowanych wierszy. Wykorzystywane jest ze słowami kluczowymi INSERT, UPDATE, DELETE.

ExecuteReader() – metoda, która wykonuje wyrażenie SQL i zwraca wiersze z tabeli. Wykorzystywane jest ze słowem kluczowym SELECT.

ExecuteScalar() – wykonuje wyrażenie SQL i zwraca jedną wartość skalarną. Wykorzystywane jest w połączeniu SELECT z funkcjami min, max, sum i count.

Transaction – obiekt klasy `SqlTransaction`, który umożliwia wykonanie zbioru operacji w postaci jednego pakietu, a w przypadku niepowodzenia możliwości cofnięcia zmian dokonanych tymi operacjami.

Część tych metod została zaprezentowana na *Snippet 3. ExecuteNonQueryAsync, Transaction*Snippet 3.

Snippet 3. ExecuteNonQueryAsync, Transaction

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async void Main(string[] args)
        {
            string connectionString = "connectionString";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                await connection.OpenAsync();
                SqlTransaction transaction = connection.BeginTransaction();

                SqlCommand command = connection.CreateCommand();
                command.Transaction = transaction;

                try
                {
                    command.CommandText = "INSERT INTO Users (Name, Age) Values ('Olek', '23')";
                    await command.ExecuteNonQueryAsync();
                    command.CommandText = "INSERT INTO Users (Name, Age) Values ('Stary Olek', '73')";
                    await command.ExecuteNonQueryAsync();

                    await transaction.CommitAsync();
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                    await transaction.RollbackAsync();
                }
            }
        }
}
```

7.1.3 connectionString

connectionString jest to zbiór parametrów przechowywanych jako tekst, za pomocą którego wykonywane jest połączenie z serwerem bazy danych. connectionString może/musi zawierać takie parametry:

Application Name – nazwa aplikacji

AttachDBFileName – cała ścieżka do dołączanej bazy danych

Connect Timeout – określenie w jakim czasie będzie dokonane wykonane połączenie

Server – nazwa serwera

Encrypt – flaga ustawiająca szyfrowanie SSL, może być true, false lub yes, no.

Database – nazwa bazy danych

Trusted_Connection – ustawia tryb autentykacji, może przyjąć wartości true, false lub yes, no. Jeśli ustawimy true, zostaną wykorzystane poświadczenia konta Windows.

Packet Size – rozmiar przesyłanych pakietów sieciowych.

Workstation ID – wskazuje nazwę komputera, na którym uruchomiana jest instancja serwera.

Password – hasło użytkownika.

User ID – login użytkownika.

Na *Snippet 4. connectionString* zostało zaprezentowane połączenie z bazą danych za pomocą connectionString, który zawiera dane o serwerze, dane o bazie danych, i sposób połączenia za pomocą konta AD.

Snippet 4. connectionString

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async void Main(string[] args)
        {
            string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=MyLibrary;Trusted_Connec-
tion=True";

            SqlConnection connection = new SqlConnection(connectionString);
            connection.Open();

            SqlTransaction transaction = connection.BeginTransaction();

            SqlCommand command = connection.CreateCommand();
            command.Transaction = transaction;

            connection.Close();
        }
    }
}
```

7.1.4 SqlConnection

`SqlConnection` jest to klasa, za pomocą której jest dokonywane połączenie z bazą danych. Metoda `Open()` służy do otwierania połączenia. Metoda `Close()` służy do zamykania połączenia. Metoda `BeginTransaction()` służy do rozpoczęcia transakcji. Działanie obiektu tej klasy zostało pokazane w praktyce na *Snippet 5. SqlConnection*.

Snippet 5. SqlConnection

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async void Main(string[] args)
        {
            string connectionString = "connectionString";

            SqlConnection connection = new SqlConnection(connectionString);
            connection.Open();

            SqlTransaction transaction = connection.BeginTransaction();

            SqlCommand command = connection.CreateCommand();
            command.Transaction = transaction;

            connection.Close();
        }
    }
}
```

7.1.5 SqlDataReader

Klasa `SqlDataReader` pozwala sczytywać i operować danymi, które otrzymujemy za pomocą wykonania polecenia SQL, klasa zawiera następne właściwości i metody:

FieldCount – ilość kolumn w danym wierszu.

HasRows – wskazuje, czy obiekt klasy zawiera co najmniej jeden wierszu.

IsClosed – zwraca bool, który wskazuje czy dane egzemplarz `SqlDataReader` jest zamknięty.

Item[liczba], Item[string] – zwraca wartość z wiersza wg. wskazanego w nawiasach indeksa.

Close() – metoda, która zamyka obiekt `SqlDataReader`.

GetValue(liczba) – metoda, która zwraca wartość z wiersza wg. wskazanego wiersza.

Read() – sczytywanie następnego wiersza.

Część możliwości tej klasy zostało zaprezentowane na *Snippet 6. SqlDataReader*.

Snippet 6. SqlDataReader

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async Task Main(string[] args)
        {
            string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=TestDatabase;Trusted_Con-
nection=True";
            string sqlExpression = "SELECT * FROM Users";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                await connection.OpenAsync();

                SqlCommand command = new SqlCommand(sqlExpression, connection);
                SqlDataReader reader = await command.ExecuteReaderAsync();

                if(reader.HasRows)
                {
                    string columnName1 = reader.GetName(0);
                    string columnName2 = reader.GetName(1);
                    string columnName3 = reader.GetName(2);

                    Console.WriteLine($"{columnName1}\t{columnName2}\t{columnName3}");

                    while(await reader.ReadAsync())
                    {
                        var id = reader.GetValue(0);
                        var name = reader.GetValue(1);
                        var age = reader.GetValue(2);

                        Console.WriteLine($"{id}\t{name}\t{age}");
                    }
                }
            }
        }
    }
}
```

7.1.6 SqlException

Wyjątek klasy `SqlException` jest wyrzucany, gdy SQL Server zwraca ostrzeżenie lub błąd, ta klasa nie może być dziedziczona

7.1.7 Using

Blok `using` jest stosowany do utworzenia obiektu, wykorzystaniu tego obiektu w bloku `using` i ostatecznie po zakończeniu tego bloku bezpiecznego i wygodnego usunięcia tego obiektu z pamięci. Obiekt, który jest tworzony w tym bloku musi dziedziczyć interfejs `IDisposable` w celu możliwości wykonania metody `Dispose()`. Blok `using` został pokazany w praktyce na *Snippet 7. Blok using*.

Snippet 7. Blok using

```
using System.Data.SqlClient;

namespace AJPPABLAB1
{
    internal class Program
    {
        static async Task Main(string[] args)
        {
```

```
            string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=TestDatabase;Trusted_Con-
nection=True";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {

            }
        }
    }
}
```

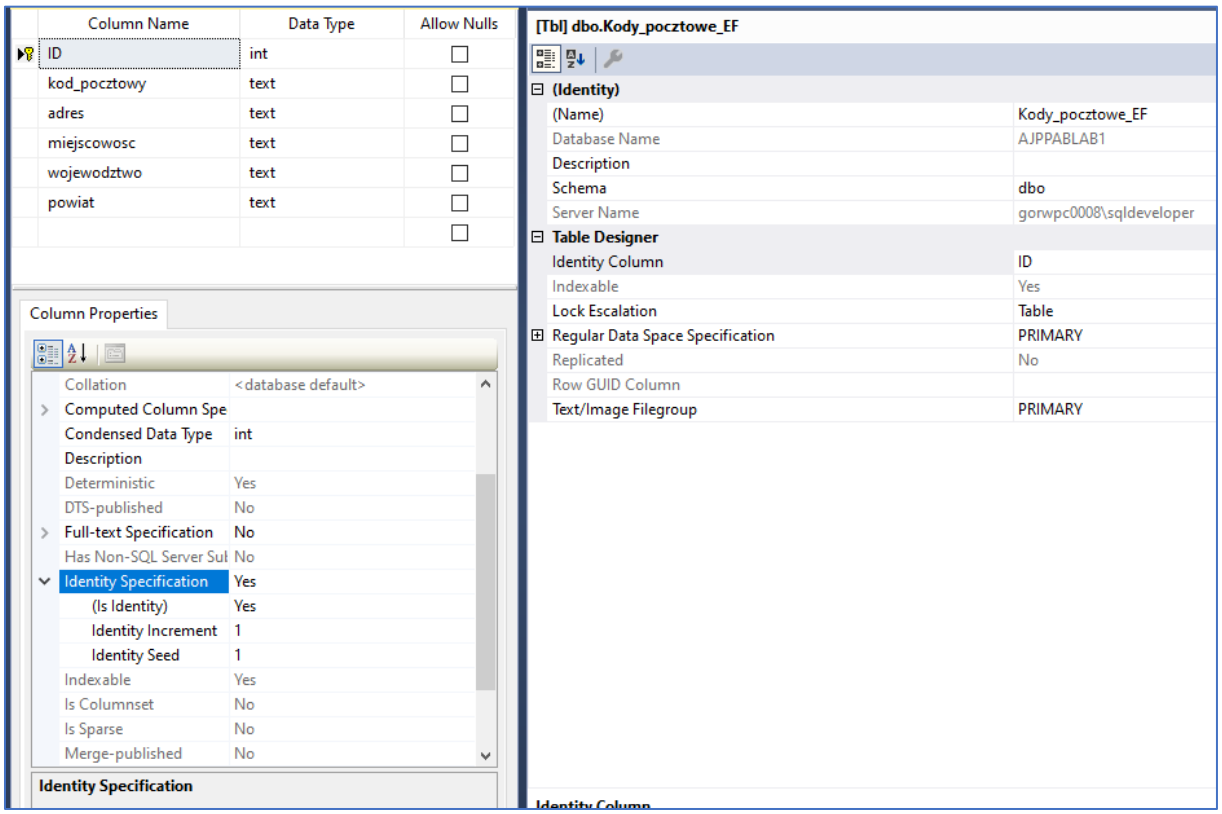
7.2 Opracowanie praktyczne

W tym rozdziale na podstawie zdobytej wiedzy w trakcie dotychczasowych studiów oraz bazując na informacjach odnalezionych w Internecie została utworzona baza danych zgodna z wymaganiami laboratoryjnymi, a także została utworzona aplikacja konsolowa, w której wykonane zostały testy pomiaru czasu dodawania danych wierszy do bazy danych umieszczonej na SQL Server 2019.

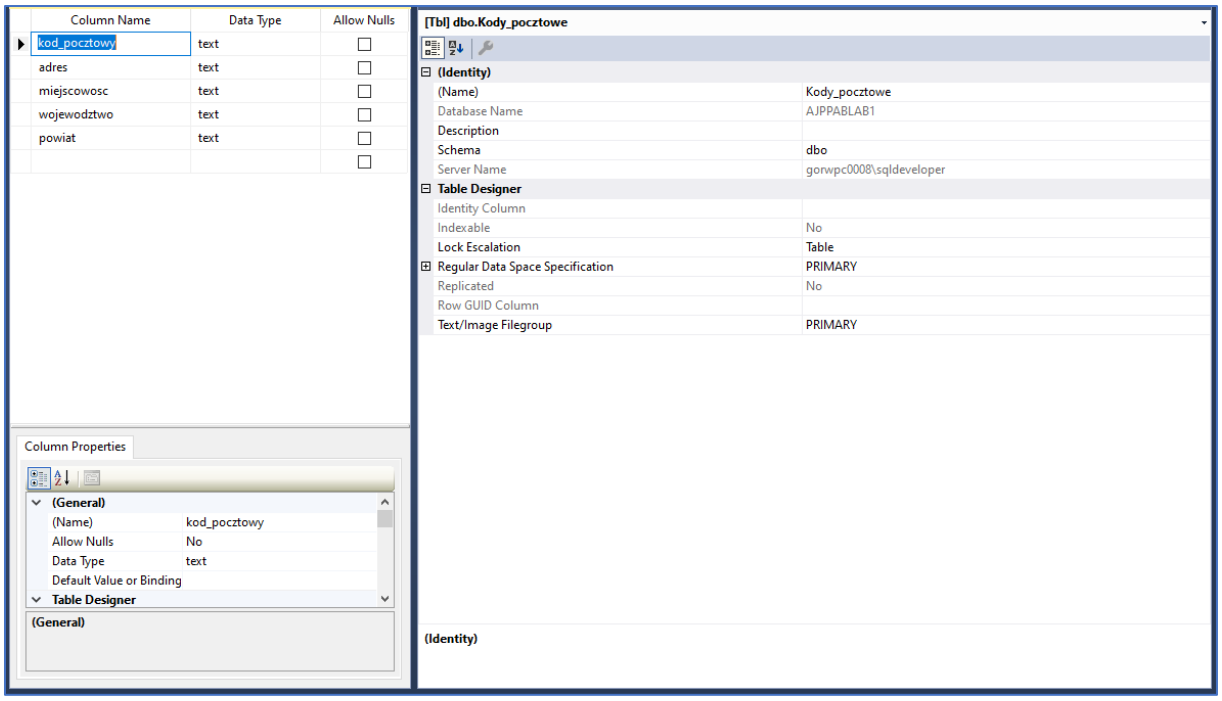
Do wykonaniu pomiarowych testów zostały wykorzystane klasy systemowe `Stopwatch` i `Timespan`, które umożliwiają odliczanie czasu w wykonywanym programie.

7.2.1 Baza danych

W celu wykonania testów różnych sposobów zapisywania danych do bazy danych zostały utworzone dwie tabele: tabela **dbo.Kody_pocztowe** i tabela **dbo.Kody_pocztowe_EF**. Tabela **dbo.Kody_pocztowe** jest utworzona do przetestowania metod z ADO.NET, Dapper i SqlBulkCopy. Tabela **dbo.Kody_pocztowe_EF** została utworzona do przetestowania metod z Entity-Framerok i różni się od tabeli **dbo.Kody_pocztowe** tylko dodatkową kolumną z kluczem głównym (ID). Na rysunkach (*Rysunek 1. Tabela dbo.Kody_pocztowe_EF*, *Rysunek 2. Tabela dbo.Kody_pocztowe*) zostały zaprezentowane struktury tych tabel.



Rysunek 1. Tabela dbo.Kody_pocztowe_EF



Rysunek 2. Tabela dbo.Kody_pocztowe

7.2.2 Importowanie danych z pliku CSV

Snippet 8. Importowanie danych z pliku CSV przedstawia kod, za pomocą którego jest wykonywane importowanie danych z pliku kody.csv do listy składającej się z obiektów klasy Kody. Importowanie jest wykonane za pomocą metod i klas z pakietu Nu-Get CsvHelper. Klasa kody została utworzona na podstawie kolumn, które zawiera plik CSV a także dodatkowe atrybuty z pakietu CsvHelper.

Snippet 8. Importowanie danych z pliku CSV

```
using CsvHelper;
using CsvHelper.Configuration;
using CsvHelper.Configuration.Attributes;
using System.Data.SqlClient;
using System.Globalization;

namespace AJPPABLAB1
{
    internal class Program
    {
        static List<Kody> imported_kody = new List<Kody>();

        static async Task Main(string[] args)
        {
            await importCSV();
        }

        static async Task importCSV()
        {
            var csvConfig = new CsvConfiguration(CultureInfo.InvariantCulture)
            {
                HasHeaderRecord= true,
                Delimiter = ";",
                MemberTypes = MemberTypes.Properties,
                HeaderValidated = null,
                MissingFieldFound = null,
            };

            using(var reader = new StreamReader(@"C:\Users\oleks\Projects\ajp\AJP-PAB-LAB1\kody.csv"))
            using(var csv = new CsvReader(reader, csvConfig))
            {
                imported_kody = csv.GetRecords<Kody>().ToList();
                Console.WriteLine($"Records Imported: {imported_kody.Count}");
            }
        }
    }

    public class Kody {
        [Name("KOD POCZTOWY")]
        [Index(0)]
        public string kod_pocztowy { get; set; } = "";
        [Name("ADRES")]
        [Index(1)]
        public string adres { get; set; } = "";
        [Name("MIEJSCOWOŚĆ")]
        [Index(2)]
        public string miejscowosc { get; set; } = "";
        [Name("WOJEWÓDZTWO")]
        [Index(3)]
        public string wojewodztwo { get; set; } = "";
        [Name("POWIAT")]
        [Index(4)]
        public string powiat { get; set; } = "";
    }
}
```

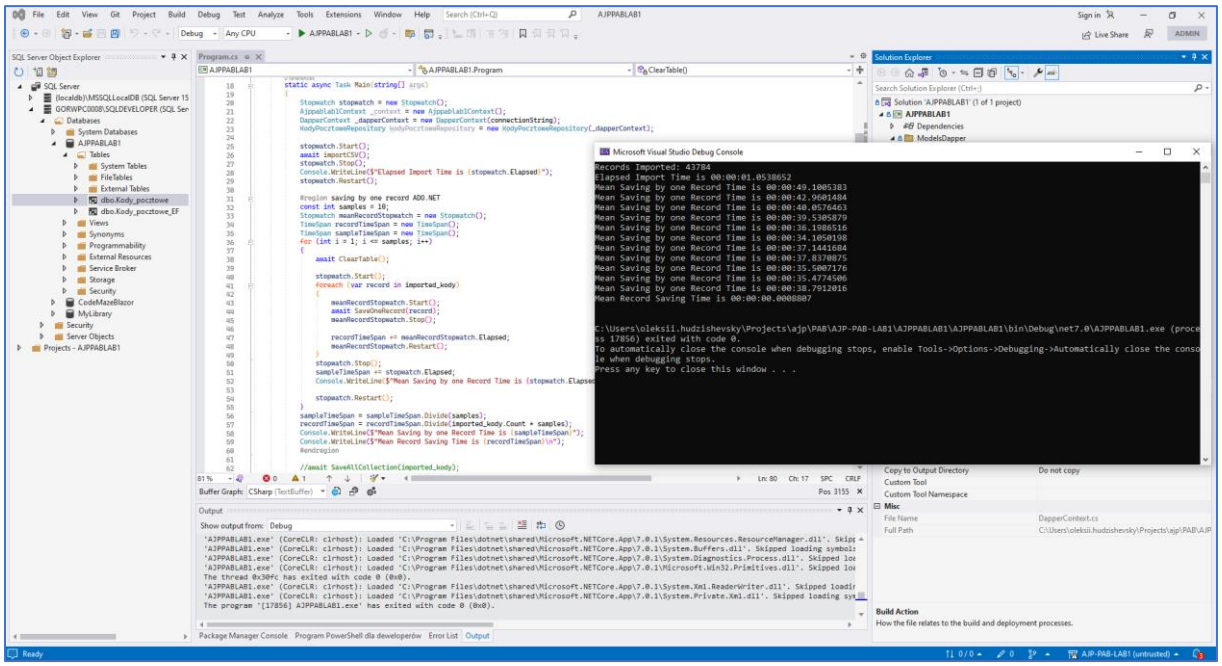
7.2.3 ADO.NET

Snippet 9. ADO.NET – pojedynczy zapis przedstawia kod, za pomocą którego jest wykonywane pojedyncze otwieranie połączenia, dodawanie wiersza i następne zamknięcie połączenia, taki zapis jest wykonywany dla każdego z zaimportowanych rekordów z pliku CSV. Wynik tej metody jest przedstawiony na *Rysunek 3. ADO.NET – pojedynczy zapis*.

Snippet 9. ADO.NET – pojedynczy zapis

```
using AJPPABLAB1;
using System.Diagnostics;

const int samples = 10;
Stopwatch meanRecordStopwatch = new Stopwatch();
TimeSpan recordTimeSpan = new TimeSpan();
TimeSpan sampleTimeSpan = new TimeSpan();
for (int i = 1; i <= samples; i++)
{
    await ClearTable();

    stopwatch.Start();
    foreach (var record in imported_kody)
    {
        meanRecordStopwatch.Start();
        await SaveOneRecord(record);
        meanRecordStopwatch.Stop();

        recordTimeSpan += meanRecordStopwatch.Elapsed;
        meanRecordStopwatch.Restart();
    }
    stopwatch.Stop();
    sampleTimeSpan += stopwatch.Elapsed;
    Console.WriteLine($"Mean Saving by one Record Time is {stopwatch.Elapsed}");

    stopwatch.Restart();
}
sampleTimeSpan = sampleTimeSpan.Divide(samples);
recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");

static async Task SaveOneRecord(Kody kody)
{
    string sqlExpression = "INSERT INTO Kody_Pocztowe (Kod_pocztowy, Adres, Miejscowosc, Woje-
wodztwo, Powiat) Values (@Kod_pocztowy, @Adres, @Miejscowosc, @Wojewodztwo, @Powiat)";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        await connection.OpenAsync();
        SqlCommand command = new SqlCommand(sqlExpression, connection);

        SqlParameter kod_pocztowyParameter = new SqlParameter("@Kod_pocztowy", kody.kod_pocz-
towy);
        SqlParameter adresParameter = new SqlParameter("@Adres", kody.adres);
        SqlParameter miejscowoscParameter = new SqlParameter("@Miejscowosc", kody.miejscow-
osc);
        SqlParameter wojewodztwoParameter = new SqlParameter("@Wojewodztwo", kody.wojewodz-
two);
        SqlParameter powiatParameter = new SqlParameter("@Powiat", kody.powiat);

        command.Parameters.Add(kod_pocztowyParameter);
        command.Parameters.Add(adresParameter);
        command.Parameters.Add(miejscowoscParameter);
        command.Parameters.Add(wojewodztwoParameter);
        command.Parameters.Add(powiatParameter);

        await command.ExecuteNonQueryAsync();
    }
```

```
}
```



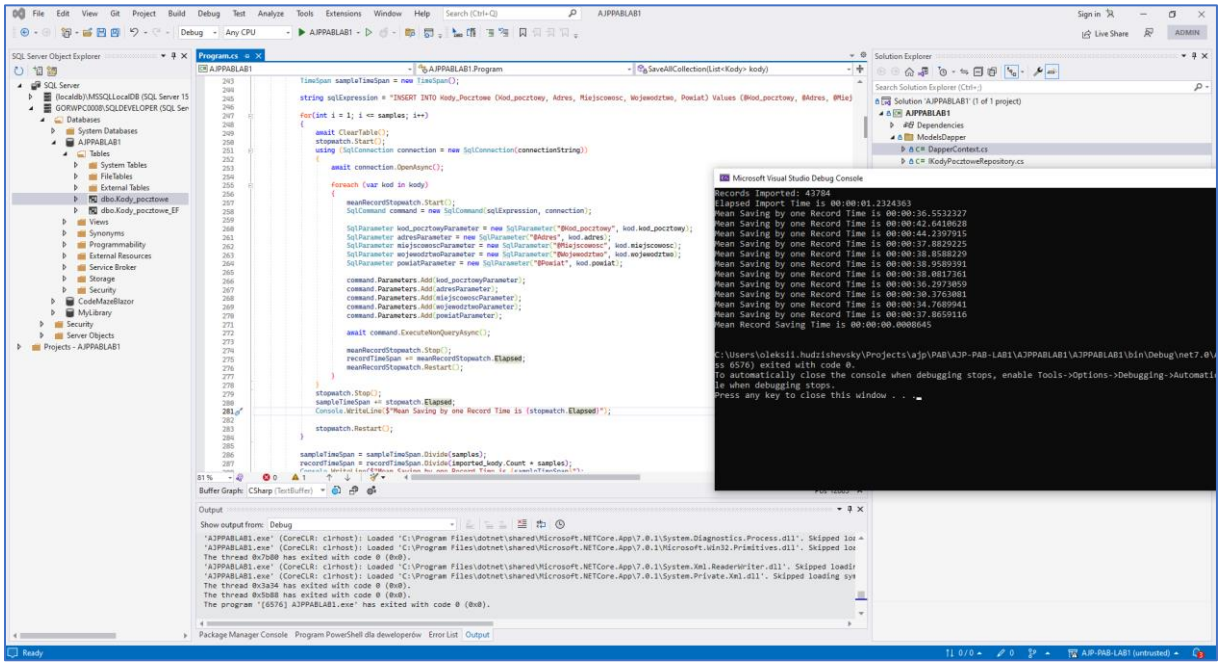
Rysunek 3. ADO.NET – pojedynczy zapis

Snippet 10. ADO.NET – zapis całej kolekcji przedstawia kod, za pomocą którego jest wykonywane pojedyncze otwieranie połączenia, dodanie wszystkich wierszy i następne zamknięcie połączenia. Wynik tej metody jest przedstawiony na *Rysunek 4. ADO.NET – Zapis całej kolekcji*.

Snippet 10. ADO.NET – zapis całej kolekcji

```
using AJPPABLAB1;
using System.Diagnostics;

static async Task SaveAllCollection(List<Kody> kody)
{
    const int samples = 10;
    Stopwatch stopwatch = new Stopwatch();

    Stopwatch meanRecordStopwatch = new Stopwatch();
    TimeSpan recordTimeSpan = new TimeSpan();
    TimeSpan sampleTimeSpan = new TimeSpan();

    string sqlExpression = "INSERT INTO Kody_Pocztowe (Kod_pocztowy, Adres, Miejscowosc, Woje-
wodztwo, Powiat) Values (@Kod_pocztowy, @Adres, @Miejscowosc, @Wojewodztwo, @Powiat)";

    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();
        stopwatch.Start();
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            await connection.OpenAsync();

            foreach (var kod in kody)
            {
                meanRecordStopwatch.Start();
                SqlCommand command = new SqlCommand(sqlExpression, connection);
```

```
                SqlParameter kod_pocztowyParameter = new SqlParameter("@Kod_pocztowy",
kod.kod_pocztowy);
                SqlParameter adresParameter = new SqlParameter("@Adres", kod.adres);
                SqlParameter miejscowoscParameter = new SqlParameter("@Miejscowosc", kod.miej-
scowosc);
                SqlParameter wojewodztwoParameter = new SqlParameter("@Wojewodztwo", kod.woje-
wodztwo);
                SqlParameter powiatParameter = new SqlParameter("@Powiat", kod.powiat);

                command.Parameters.Add(kod_pocztowyParameter);
                command.Parameters.Add(adresParameter);
                command.Parameters.Add(miejscowoscParameter);
                command.Parameters.Add(wojewodztwoParameter);
                command.Parameters.Add(powiatParameter);

                await command.ExecuteNonQueryAsync();

                meanRecordStopwatch.Stop();
                recordTimeSpan += meanRecordStopwatch.Elapsed;
                meanRecordStopwatch.Restart();
            }
        }
        stopwatch.Stop();
        sampleTimeSpan += stopwatch.Elapsed;
        Console.WriteLine($"Mean Saving by one Record Time is {stopwatch.Elapsed}");

        stopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
    Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
    Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");
}
```



Rysunek 4. ADO.NET – Zapis całej kolekcji

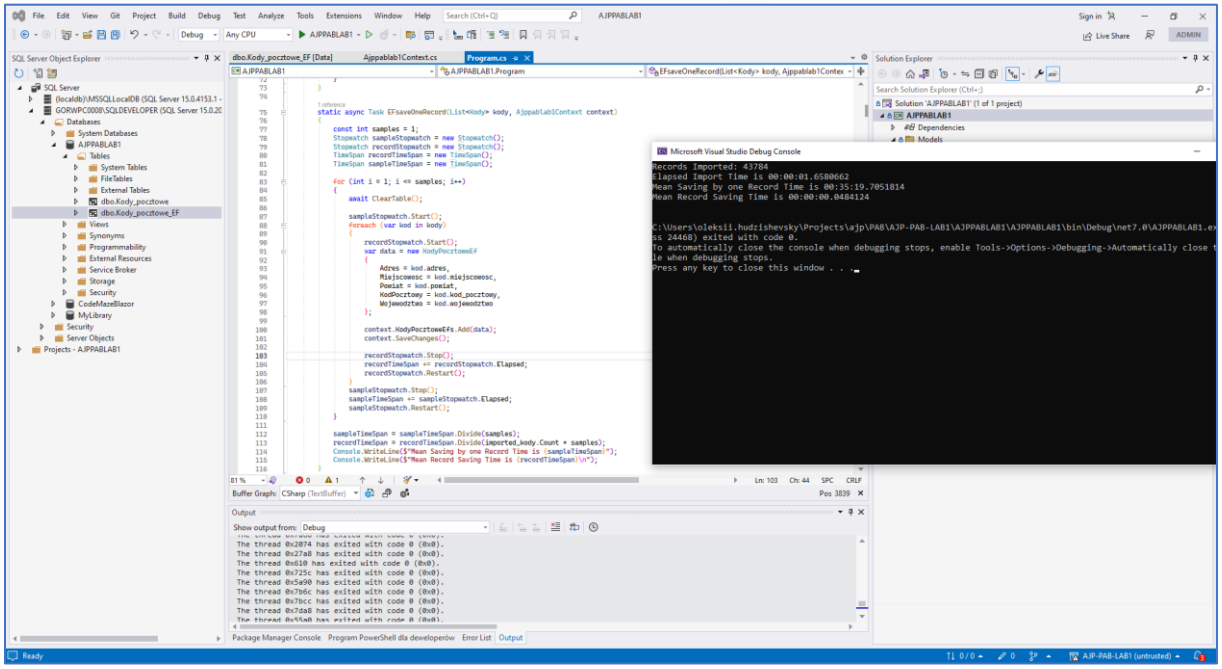
7.2.4 Zapisywanie za pomocą Entity Framework

W przypadku dokonywania testów z Entity Framework'iem musiała być wykorzystana tabela o innej strukturze, zawierająca dodatkowo klucz główny, ponieważ Entity Framework pracuje tylko z tablicami zawierającymi klucz główny.

Snippet 11. Entity Framework – pojedynczy zapis przedstawia kod, za pomocą którego jest wykonywane dodawanie wiersza i następne zapisywanie zmian metodą `SaveChanges()` dla każdego z zaimportowanych rekordów z pliku CSV. Wynik tej metody jest przedstawiony na *Rysunek 5. Entity Framework – pojedynczy zapis*.

Snippet 11. Entity Framework – pojedynczy zapis

```
using AJPPABLAB1.ModelsEF;
using AJPPABLAB1;
using System.Diagnostics;

static async Task EFsaveOneRecord(List<Kody> kody, Ajppablab1Context context)
{
    const int samples = 10;
    Stopwatch sampleStopwatch = new Stopwatch();
    Stopwatch recordStopwatch = new Stopwatch();
    TimeSpan recordTimeSpan = new TimeSpan();
    TimeSpan sampleTimeSpan = new TimeSpan();

    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();

        sampleStopwatch.Start();
        for (int j = 0; j < kody.Count; j++)
        {
            recordStopwatch.Start();
            var data = new KodyPocztoweEf
            {
                Adres = kody[j].adres,
                Miejscowosc = kody[j].miejscowosc,
                Powiat = kody[j].powiat,
                KodPocztowy = kody[j].kod_pocztowy,
                Wojewodztwo = kody[j].wojewodztwo
            };

            context.KodyPocztoweEfs.Add(data);
            context.SaveChanges();

            recordStopwatch.Stop();
            recordTimeSpan += recordStopwatch.Elapsed;
            recordStopwatch.Restart();
        }
        sampleStopwatch.Stop();
        sampleTimeSpan += sampleStopwatch.Elapsed;
        sampleStopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
    Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
    Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");
}
```



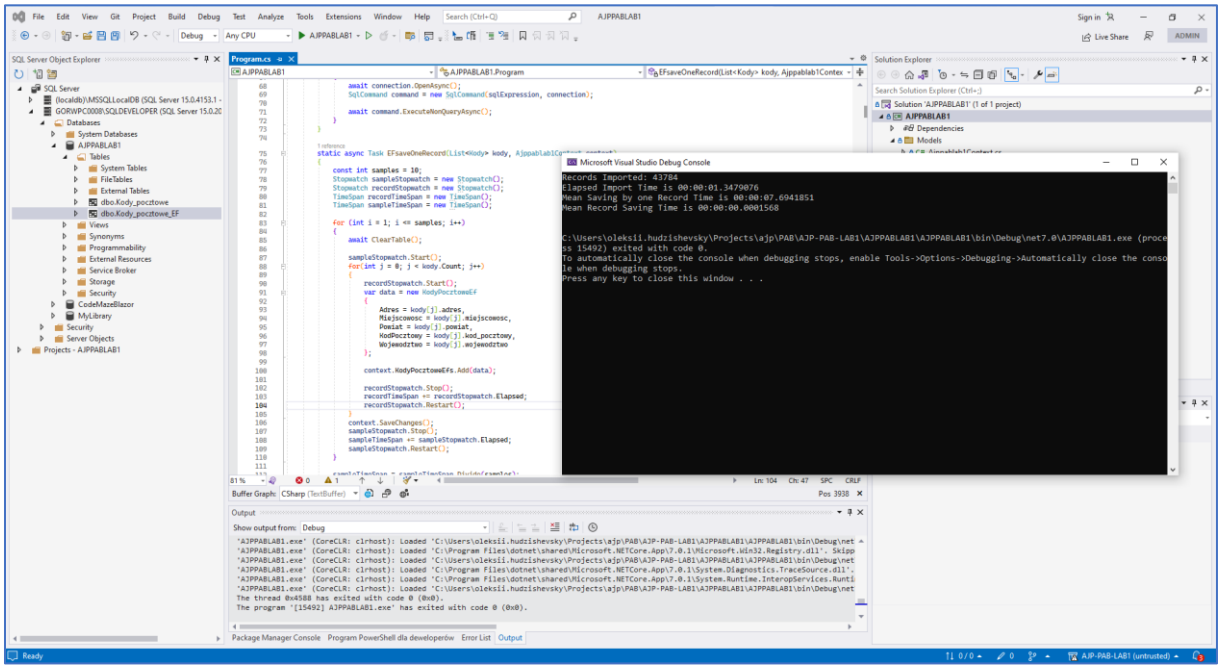
Rysunek 5. Entity Framework – pojedynczy zapis

Snippet 12. Entity Framework – zapis całej kolekcji przedstawia kod, za pomocą którego jest wykonywane dodanie wszystkich wierszy i na końcu zapisywanie zmian metodą `SaveChanges()`. Wynik tej metody jest przedstawiony na *Rysunek 6. Entity Framework – zapis całej kolekcji*.

Snippet 12. Entity Framework – zapis całej kolekcji

```
using AJPPABLAB1.ModelsEF;
using AJPPABLAB1;
using System.Diagnostics;

static async Task EFSaveAll(List<Kody> kody, Ajppablab1Context context)
{
    const int samples = 10;
    Stopwatch sampleStopwatch = new Stopwatch();
    Stopwatch recordStopwatch = new Stopwatch();
    TimeSpan recordTimeSpan = new TimeSpan();
    TimeSpan sampleTimeSpan = new TimeSpan();

    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();

        sampleStopwatch.Start();
        for (int j = 0; j < kody.Count; j++)
        {
            recordStopwatch.Start();
            var data = new KodyPocztoweEf
            {
                Adres = kody[j].adres,
                Miejscowosc = kody[j].miejscowosc,
                Powiat = kody[j].powiat,
                KodPocztowy = kody[j].kod_pocztowy,
                Wojewodztwo = kody[j].wojewodztwo
            };

            context.KodyPocztoweEfs.Add(data);
```

```
            recordStopwatch.Stop();
            recordTimeSpan += recordStopwatch.Elapsed;
            recordStopwatch.Restart();
        }
        context.SaveChanges();
        sampleStopwatch.Stop();
        sampleTimeSpan += sampleStopwatch.Elapsed;
        sampleStopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
    Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
    Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");
}
```



Rysunek 6. Entity Framework – zapis całej kolekcji

7.2.5 Zapisywanie za pomocą Dapper

Snippet 13. Dapper – pojedynczy zapis przedstawia kod, za pomocą którego jest wykonywane dodawanie wiersza i zapisywanie za pomocą metody `CreateKodPocztowy()` dla każdego z zaimportowanych rekordów z pliku CSV. Metoda `CreateKodPocztowy()` znajduje się w klasie `KodyPocztoweRepository`, klasa ta została utworzona dla komunikacji z kontekstem Dapper'a. *Snippet 14. Dapper - metoda CreateKodPocztowy()* przedstawia kod tej metody. Wynik tej metody jest przedstawiony na *Rysunek 7. Dapper – pojedynczy zapis*.

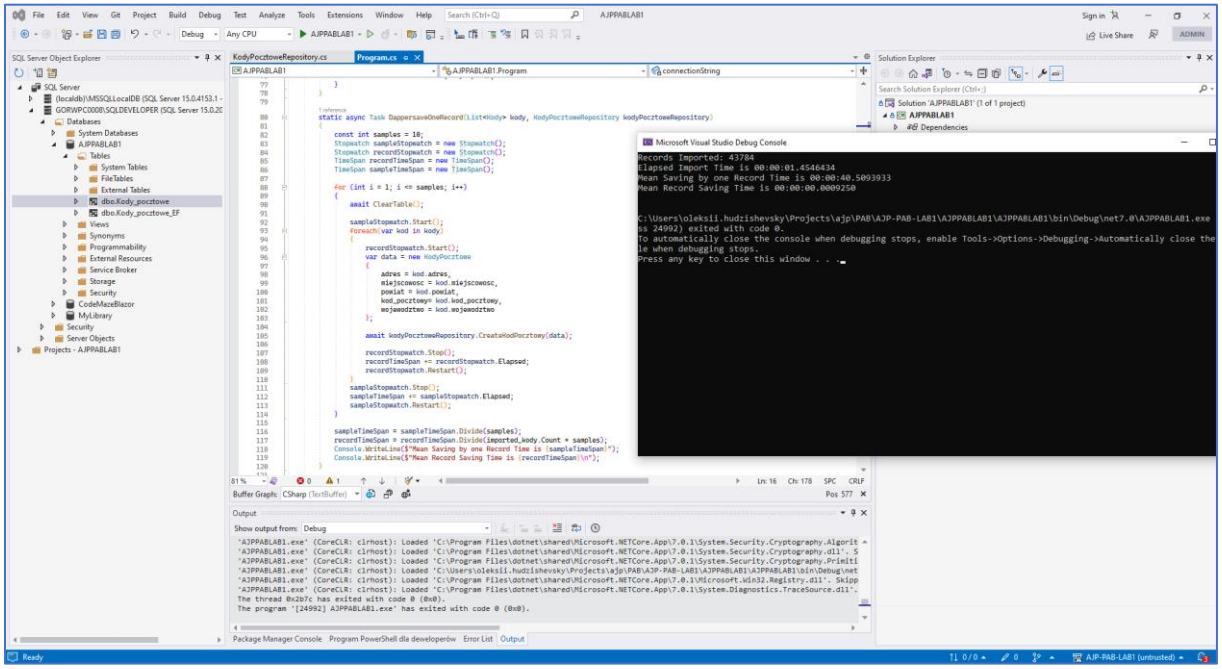
Snippet 13. Dapper – pojedynczy zapis

```
using AJPPABLAB1.ModelsDapper;
using AJPPABLAB1;
using System.Diagnostics;
```

```
static async Task DappersaveOneRecord(List<Kody> kody, KodyPocztoweRepository kodyPocztoweRe-
pository)
{
    const int samples = 10;
    Stopwatch sampleStopwatch = new Stopwatch();
    Stopwatch recordStopwatch = new Stopwatch();
    TimeSpan recordTimeSpan = new TimeSpan();
    TimeSpan sampleTimeSpan = new TimeSpan();

    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();

        sampleStopwatch.Start();
        foreach (var kod in kody)
        {
            recordStopwatch.Start();
            var data = new KodyPocztowe
            {
                adres = kod.adres,
                miejscowosc = kod.miejscowosc,
                powiat = kod.powiat,
                kod_pocztowy = kod.kod_pocztowy,
                wojewodztwo = kod.wojewodztwo
            };

            await kodyPocztoweRepository.CreateKodPocztowy(data);

            recordStopwatch.Stop();
            recordTimeSpan += recordStopwatch.Elapsed;
            recordStopwatch.Restart();
        }
        sampleStopwatch.Stop();
        sampleTimeSpan += sampleStopwatch.Elapsed;
        sampleStopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
    Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
    Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");
}
```

Snippet 14. Dapper - metoda CreateKodPocztowy()

```
public async Task CreateKodPocztowy(KodyPocztowe kodyPocztowe)
{
    var query = "INSERT INTO Kody_pocztowe (kod_pocztowy, adres, miejscowosc, wojewodztwo, po-
wiat) VALUES (@kod_pocztowy, @adres, @miejscowosc, @wojewodztwo, @powiat)";

    var parameters = new DynamicParameters();
    parameters.Add("kod_pocztowy", kodyPocztowe.kod_pocztowy);
    parameters.Add("adres", kodyPocztowe.adres);
    parameters.Add("miejscowosc", kodyPocztowe.miejscowosc);
    parameters.Add("wojewodztwo", kodyPocztowe.wojewodztwo);
    parameters.Add("powiat", kodyPocztowe.powiat);

    using (var connection = _context.CreateConnection())
    {
        await connection.ExecuteAsync(query, parameters);
    }
}
```



Rysunek 7. Dapper – pojedynczy zapis

Snippet 15. Dapper – zapis całej kolekcji przedstawia kod, za pomocą którego jest wykonywane dodawanie wszystkich wierszy i zapisywanie za pomocą metody `CreateKodyPocztowe()`. Metoda `CreateKodyPocztowe()` znajduje się w klasie `KodyPocztoweRepository`, klasa ta została utworzona dla komunikacji z kontekstem Dapper'a. *Snippet 16. Dapper - CreateKodyPocztowe()* przedstawia kod tej metody. Wynik tej metody jest przedstawiony na *Rysunek 8. Dapper - zapis całej kolekcji*.

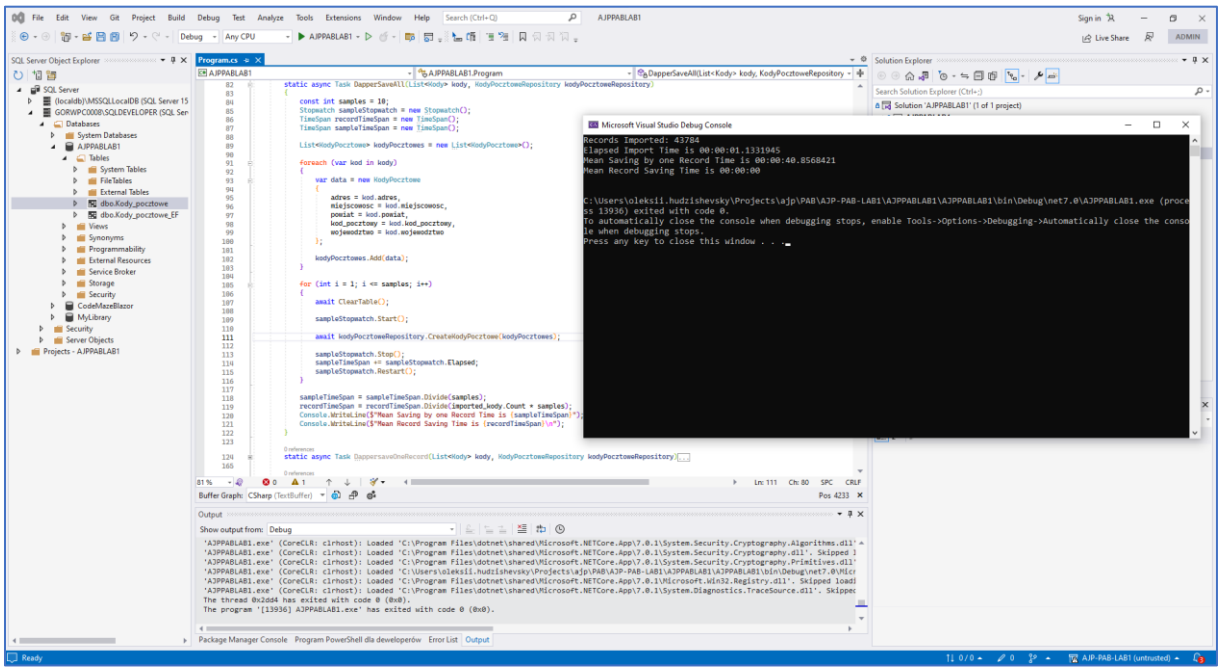
Snippet 15. Dapper – zapis całej kolekcji

```
using AJPPABLAB1.ModelsDapper;
using AJPPABLAB1;
using System.Diagnostics;

static async Task DapperSaveAll(List<Kody> kody, KodyPocztoweRepository kodyPocztoweReposi-
tory)
{
    const int samples = 10;
    Stopwatch sampleStopwatch = new Stopwatch();
    TimeSpan recordTimeSpan = new TimeSpan();
    TimeSpan sampleTimeSpan = new TimeSpan();

    List<KodyPocztowe> kodyPocztowes = new List<KodyPocztowe>();

    foreach (var kod in kody)
    {
        var data = new KodyPocztowe
        {
            adres = kod.adres,
            miejscowosc = kod.miejscowosc,
            powiat = kod.powiat,
            kod_pocztowy = kod.kod_pocztowy,
            wojewodztwo = kod.wojewodztwo
        };

        kodyPocztowes.Add(data);
    }
```

```
    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();

        sampleStopwatch.Start();

        await kodyPocztoweRepository.CreateKodyPocztowe(kodyPocztowes);

        sampleStopwatch.Stop();
        sampleTimeSpan += sampleStopwatch.Elapsed;
        sampleStopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    recordTimeSpan = recordTimeSpan.Divide(imported_kody.Count * samples);
    Console.WriteLine($"Mean Saving by one Record Time is {sampleTimeSpan}");
    Console.WriteLine($"Mean Record Saving Time is {recordTimeSpan}\n");
}
```

Snippet 16. Dapper - CreateKodyPocztowe()

```
public async Task CreateKodyPocztowe(List<KodyPocztowe> kodyPocztowe)
{
    Stopwatch recordStopwatch = new Stopwatch();

    using (var connection = _context.CreateConnection())
    {
        foreach (var kod in kodyPocztowe)
        {
            recordStopwatch.Start();
            var query = "INSERT INTO Kody_pocztowe (kod_pocztowy, adres, miejscowosc, woje-
wodztwo, powiat) VALUES (@kod_pocztowy, @adres, @miejscowosc, @wojewodztwo, @powiat)";

            var parameters = new DynamicParameters();

            parameters.Add("kod_pocztowy", kod.kod_pocztowy);
            parameters.Add("adres", kod.adres);
            parameters.Add("miejscowosc", kod.miejscowosc);
            parameters.Add("wojewodztwo", kod.wojewodztwo);
            parameters.Add("powiat", kod.powiat);

            await connection.ExecuteAsync(query, parameters);
            recordStopwatch.Stop();

            recordStopwatch.Restart();
        }
    }
}
```



Rysunek 8. Dapper - zapis całej kolekcji

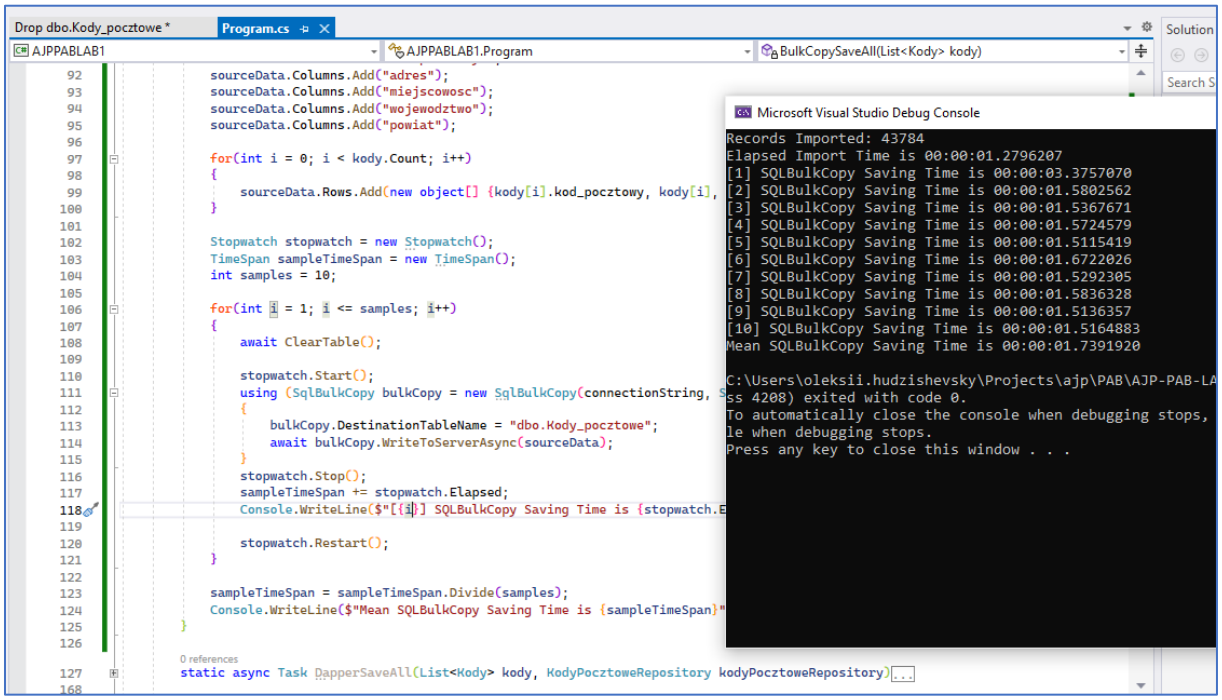
7.2.6 Zapisywanie za pomocą SQLBulkCopy

Snippet 17. SQLBulkCopy – zapis całej kolekcji przedstawia kod, w którym wszystkie zaimportowane rekordy z pliku CSV zostały dodany do obiektu klasy `DataTable` i następnie po otwarciu połączenia za pomocą metody `WriteToServer()` cały obiekt klasy `DataTable` został dodany do bazy danych. Wynik tej metody jest przedstawiony na *Rysunek 9. SQLBulkCopy - zapis całej kolekcji*.

Snippet 17. SQLBulkCopy – zapis całej kolekcji

```
using AJPPABLAB1;
using System.Data;
using System.Diagnostics;

static async Task BulkCopySaveAll(List<Kody> kody)
{
    DataTable sourceData = new DataTable();
    sourceData.Columns.Add("kod_pocztowy");
    sourceData.Columns.Add("adres");
    sourceData.Columns.Add("miejscowosc");
    sourceData.Columns.Add("wojewodztwo");
    sourceData.Columns.Add("powiat");

    for (int i = 0; i < kody.Count; i++)
    {
        sourceData.Rows.Add(new object[] { kody[i].kod_pocztowy, kody[i], kody[i].miejscowosc,
kody[i].wojewodztwo, kody[i].powiat });
    }

    Stopwatch stopwatch = new Stopwatch();
    TimeSpan sampleTimeSpan = new TimeSpan();
    int samples = 10;

    for (int i = 1; i <= samples; i++)
    {
        await ClearTable();
```

```
        stopwatch.Start();
        using (SqlBulkCopy bulkCopy = new SqlBulkCopy(connectionString, SqlBulkCopyOptions.Ke-
epIdentity))
        {
            bulkCopy.DestinationTableName = "dbo.Kody_pocztowe";
            await bulkCopy.WriteToServerAsync(sourceData);
        }
        stopwatch.Stop();
        sampleTimeSpan += stopwatch.Elapsed;
        Console.WriteLine($"[{i}] SQLBulkCopy Saving Time is {stopwatch.Elapsed}");

        stopwatch.Restart();
    }

    sampleTimeSpan = sampleTimeSpan.Divide(samples);
    Console.WriteLine($"Mean SQLBulkCopy Saving Time is {sampleTimeSpan}");
}
```



Rysunek 9. SQLBulkCopy - zapis całej kolekcji

7.2.7 Wyniki testów

Specyfikacja komputera, na którym były przeprowadzane testy:

- **Pamięć RAM** –16 GB
- **CPU** – Intel Core i7-8650U 1.90 GHz 2.11GHz
- **Software** – Visual Studio 2022 Community, Microsoft SQL Server Management Studio 18, SQL Server 2019

Przeprowadzono zostało 7 testów z wyliczeniem średniego czasu z dziesięciu prób zapisu informacji z pliku „kody.csv” do bazy danych. Wyniki z tych testów zostały zaprezentowane

na *Tabela 1. Wyniki pomiarowe z testów*. W tabeli został podkreślony wiersz z najlepszym wynikiem.

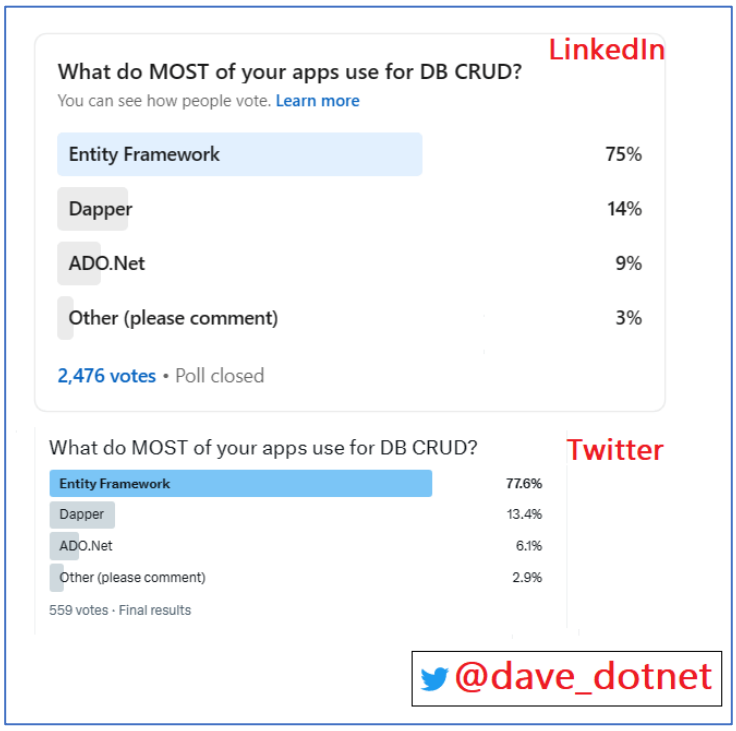
Tabela 1. Wyniki pomiarowe z testów

Metoda / Pomiar	Średni całkowity czas na 10 prób [mm.ss.ms]	Średni czas zapisu pojedynczego rekordu [mm.ss.ms]
ADO.NET – pojedynczy zapis	38 sek., 791.20 ms.	0.88 ms.
ADO.NET – zapis całej kolekcji	37 sek., 865.91 ms.	0.86 ms.
Entity Framework – pojedynczy zapis	35 min., 19 sek., 705.18 ms.	0.04841 ms.
Entity Framework – zapis całej kolekcji	7 sek., 694.18 ms.	0.15 ms.
Dapper – pojedynczy zapis	40 sek., 509.39 ms.	0.00092 ms.
Dapper – zapis całej kolekcji	40 sek., 856.84 ms.	0.93 ms.
SQLBulkCopy – zapis całej kolekcji	1 sek., 739.19 ms.	-

8. Wnioski

W trakcie wykonania tego laboratorium udało się osiągnąć główny cel, czyli zapoznać się z podstawowymi zagadnieniami dotyczącymi łączenia się z bazą danych (Microsoft SQL Server) z poziomu aplikacji konsolowej napisanej w C#. Udało się także zapoznać się z różnymi metodami dodawania nowych danych do bazy danych, m.in.: dodawanie za pomocą ADO.NET który jest częścią bibliotek .NET, dodawanie za pomocą takiego ORM jak Entity Framework a także microORM'a Dapper i na końcu dodawanie za pomocą metody `SQLBulkCopy`.

Przeglądając Internet można zauważyć, że największą popularnością zyskuje Entity Framework (*Rysunek 10. Popularność framework'ów do pracy z bazą danych*), chociaż w testach wykonanych w laboratorium posiada najgorsze wyniki. Taka popularność wynika z wygodności w użyciu, którą sprawia ten framework, ponieważ pozwala on wejść na poziom abstrakcyjny i operować bazą danych i jej tabelami nie korzystając z SQL kwerend.



Rysunek 10. Popularność framework'ów do pracy z bazą danych

9. Bibliografia

1. Źródła pomocnicze
    a. Guide to ADO.NET and working with databases in .NET 6 (oryg. Руководство по ADO.NET и работе с базами данных в .NET 6) [https://metanit.com/sharp/adonetcore/], dostęp: 03.03.2023
    b. How to Read Data From a CSV File in C# [https://code-maze.com/csharp-read-data-from-csv-file/], dostęp: 03.03.2023
    c. Measure execution time in C# [https://www.techiedelight.com/measure-execution-time-csharp/], dostęp: 03.03.2023
    d. Using Dapper with ASP.NET Core Web API [https://code-maze.com/using-dapper-with-asp-net-core-web-api/], dostęp: 06.03.2023
    e. Extremely easy way to bulk insert data into SQL Server using SqlBulkCopy class [https://www.youtube.com/watch?v=WBxuwJUazGM], dostęp: 08.03.2023
    f. Dapper vs Entity Framework vs ADO.NET Performance Benchmarking [https://www.exceptionnotfound.net/dapper-vs-entity-framework-vs-ado-net-performance-benchmarking/], dostęp: 13.03.2023

g. Speed Comprasion: Daper vs Entity Framework [https://dontpaniclabs.com/blog/post/2014/05/01/speed-comparison-dapper-vs-entity-framework/], dostęp: 15.03.2023

2. Napotkane problem i ich rozwiązania
   a. C# - Error: "Program does not contain a static 'main' method suitable for an entry point" when building solution [https://peterdaugaardrasmussen.com/2022/01/05/csharp-program-does-not-contain-a-static-main-method-suitable-for-an-entry-point-when-building-solution/], dostęp: 03.03.2023
   b. Entity Framework Core 7 connection certificate trust exception [https://stackoverflow.com/questions/74467642/entity-framework-core-7-connection-certificate-trust-exception], dostęp: 03.03.2023
   c. Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding. The statement has been terminated [https://stackoverflow.com/questions/8602395/timeout-expired-the-timeout-period-elapsed-prior-to-completion-of-the-operation], dostęp: 03.03.2023
   d. No members mapped for type - CSV Helper [https://stackoverflow.com/questions/74155920/no-members-mapped-for-type-csv-helper], dostęp: 03.03.2023

10. Spis ilustracji

11. Spis snippetów