|  |  |  |  |
| --- | --- | --- | --- |
| **Laboratorium 1**  **Sprawozdanie z realizacji laboratorium** | | | |
| **Temat:** Podstawy, nawiązywanie połączenia z bazą danych, zapisywanie rekordów do bazy danych | **Nr Albumu:** 028487 | **Grupa/zespół:** 1 / 1 | **Rok/semestr:** III / 6 |
| **Wykonał:**  Oleksii Hudzishevskyi | **Data wykonania:**  27/02/2023 | | **Data oddania:**  03/03/2023 |
| **Ocena:** | | **Podpis prowadzącego:** |

# Spis treści

[1. Spis treści 1](#_Toc128863691)

[2. Cel ćwiczenia 3](#_Toc128863692)

[3. Wymagania znajomości zagadnień 3](#_Toc128863693)

[4. Literatura, materiały dydaktyczne 3](#_Toc128863694)

[5. Wiadomości teoretyczne. 4](#_Toc128863695)

[6. Przebieg ćwiczenia 5](#_Toc128863696)

[7. Opracowanie wyników, sprawozdanie 6](#_Toc128863697)

[7.1 Opracowanie teoretyczne 6](#_Toc128863698)

[7.1.1 try-catch-finally 6](#_Toc128863699)

[7.1.2 SqlCommand 7](#_Toc128863700)

[7.1.3 connectionString 8](#_Toc128863701)

[7.1.4 SqlConnection 9](#_Toc128863702)

[7.1.5 SqlDataReader 10](#_Toc128863703)

[7.1.6 SqlException 11](#_Toc128863704)

[7.1.7 Using 11](#_Toc128863705)

[7.2 Opracowanie praktyczne 11](#_Toc128863706)

[8. Wnioski 11](#_Toc128863707)

[9. Bibliografia 11](#_Toc128863708)

[10. Spis ilustracji 12](#_Toc128863709)

[11. Spis snippetów 12](#_Toc128863710)

# 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.

# 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.

# 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>

# 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.

# 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)

# Opracowanie sprawozdania

## Opracowanie teoretyczne

### 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.

|  |
| --- |
| 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  } |

### SqlCommand

(Wymagany jest pakiet Nu-Get System.Data.SqlClient)

**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.

|  |
| --- |
| 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.

|  |
| --- |
| 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();  }  }  }  } |

### 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.

|  |
| --- |
| using System.Data.SqlClient;  namespace AJPPABLAB1  {  internal class Program  {  static async void Main(string[] args)  {  string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=MyLibrary;Trusted\_Connection=True";  SqlConnection connection = new SqlConnection(connectionString);  connection.Open();  SqlTransaction transaction = connection.BeginTransaction();    SqlCommand command = connection.CreateCommand();  command.Transaction = transaction;  connection.Close();  }  }  } |

### 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.

|  |
| --- |
| 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();  }  }  } |

### 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.

|  |
| --- |
| using System.Data.SqlClient;  namespace AJPPABLAB1  {  internal class Program  {  static async Task Main(string[] args)  {  string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=TestDatabase;Trusted\_Connection=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}");  }  }  }  }  }  } |

### SqlException

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

### 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().

|  |
| --- |
| using System.Data.SqlClient;  namespace AJPPABLAB1  {  internal class Program  {  static async Task Main(string[] args)  {  string connectionString = @"Server=LOCALHOST\LOCALDATABASE;Database=TestDatabase;Trusted\_Connection=True";  using (SqlConnection connection = new SqlConnection(connectionString))  {  }  }  }  } |

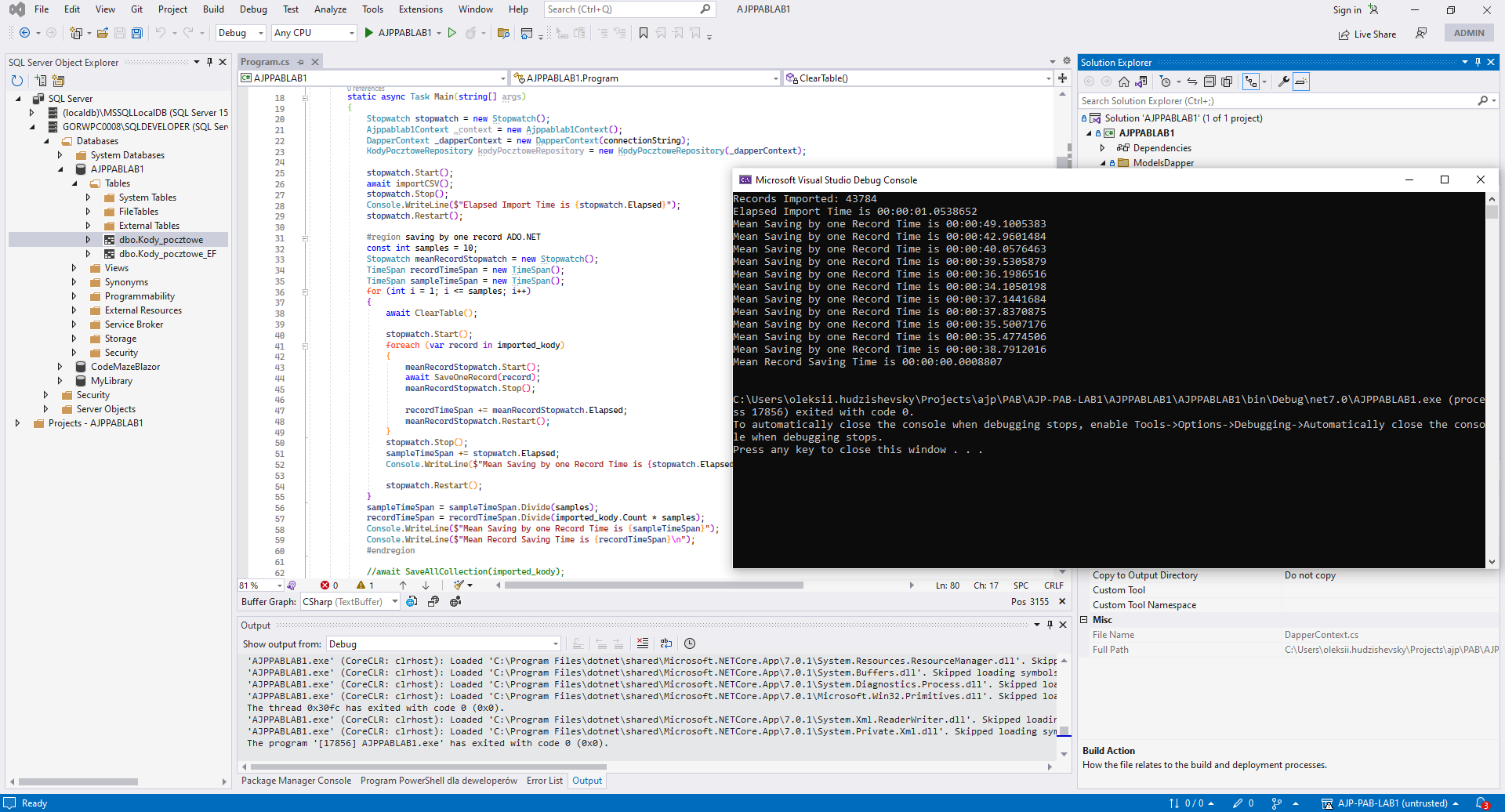
## Opracowanie praktyczne

### Importowanie danych z pliku CSV

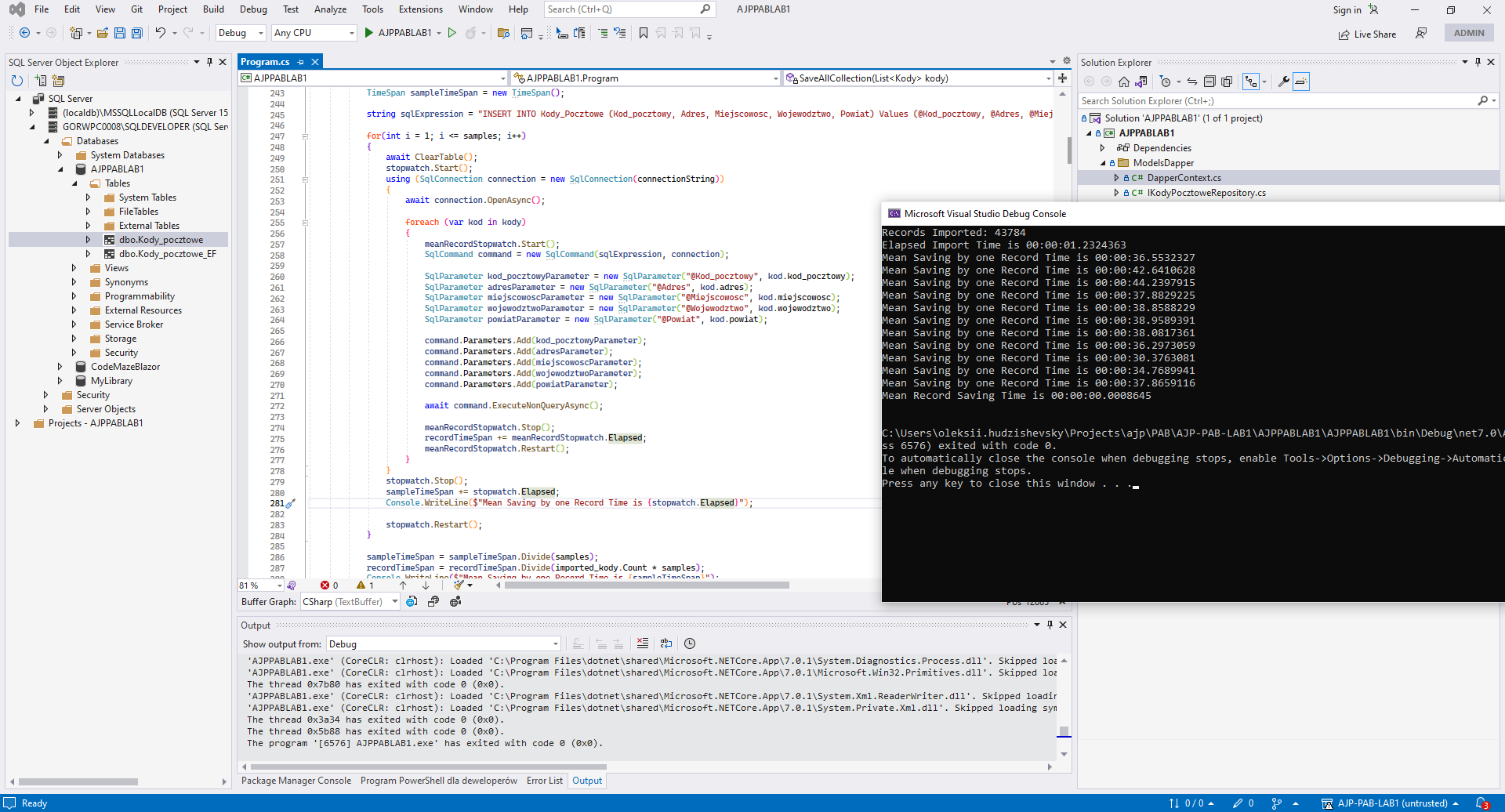
|  |
| --- |
| using CsvHelper;  using CsvHelper.Configuration;  using CsvHelper.Configuration.Attributes;  using System.Data.SqlClient;  using System.Globalization;  namespace AJPPABLAB1  {  internal class Program  {  static List<Kody> imported\_kody = new List<Kody>();  static async Task Main(string[] 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; } = "";  }  } |

### ADO.NET

Metoda pojedynczego zapisu



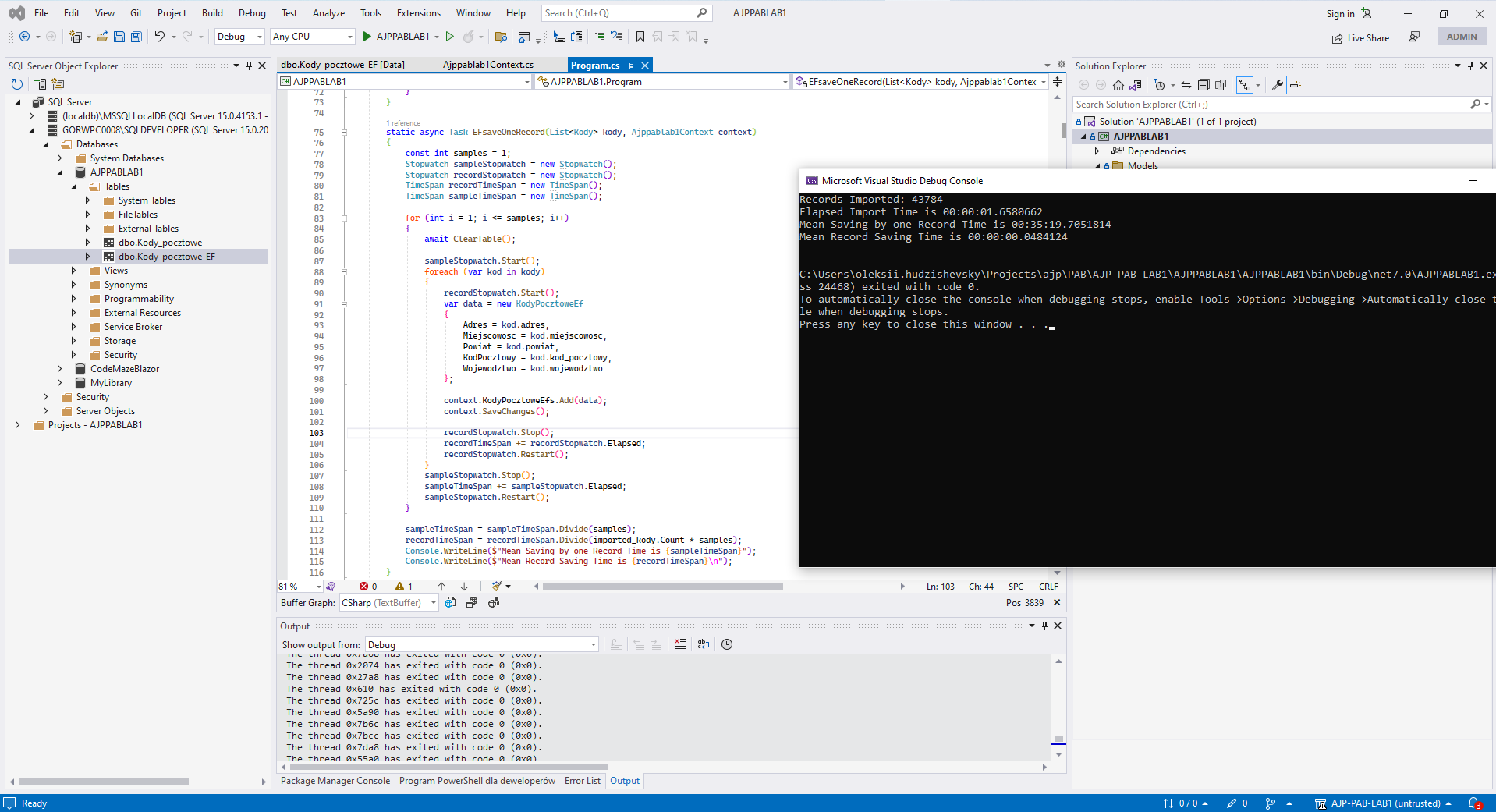
Metodą zapisu całej kolekcji



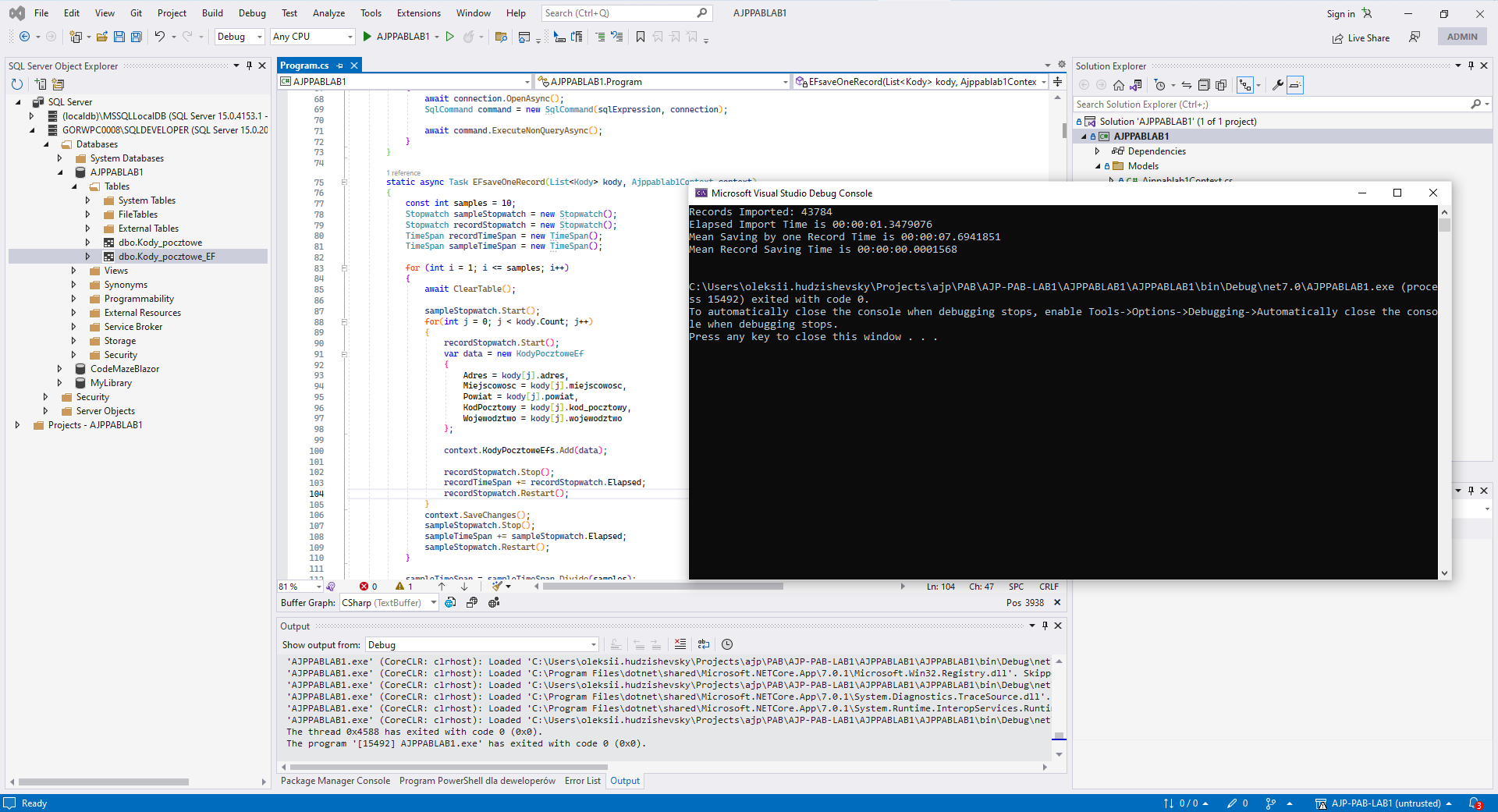
### Zapisywanie za pomocą Entity Framework

W przypadku dokonania testów z Entity Framework’iem musiała być utworzona tabela o innej strukturze zawierającej dodatkowo klucz główny, ponieważ Entity Framework pracuje tylko z tablicami zawierającymi klucz główny.

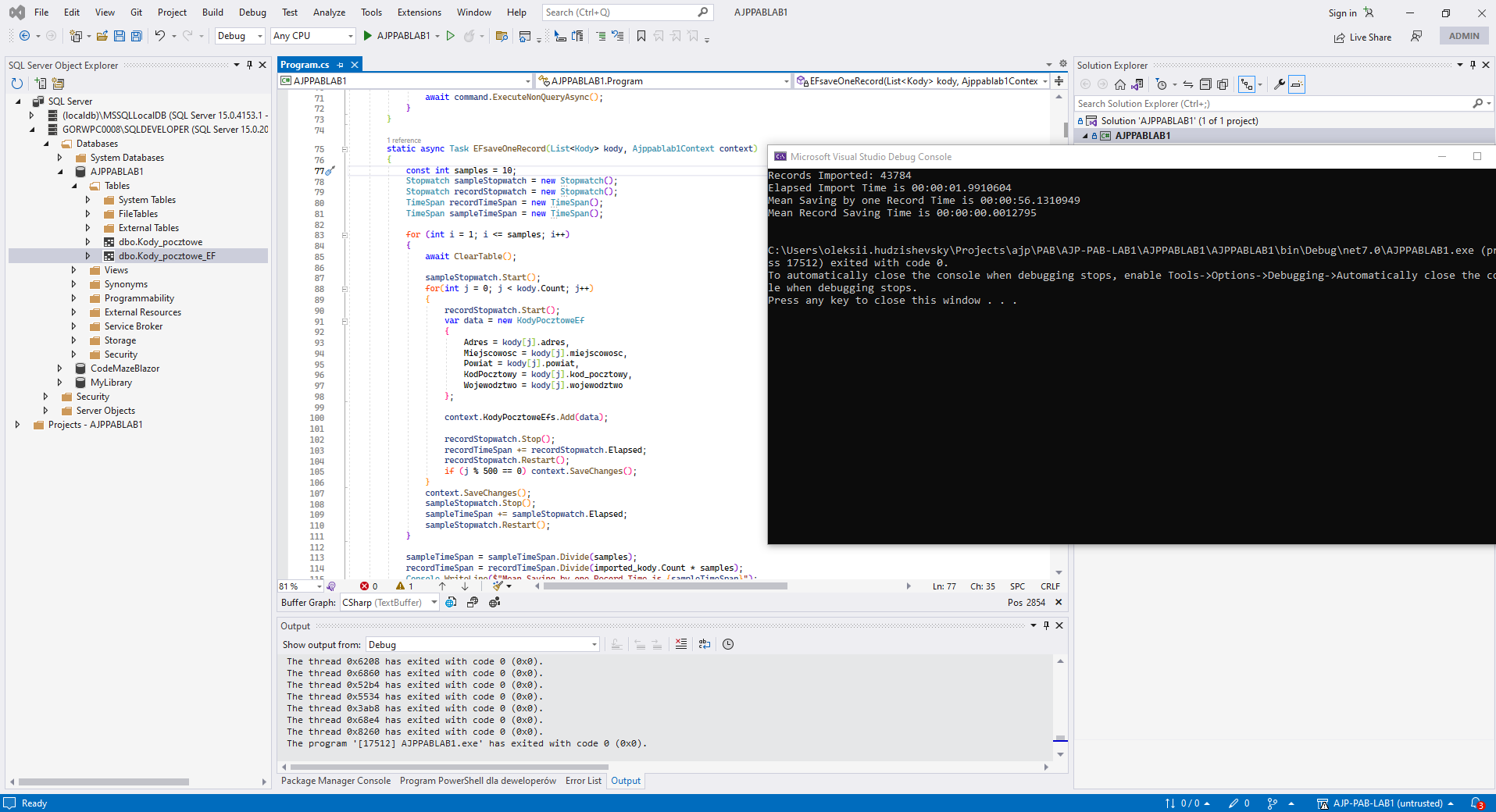
Metodą zapisu zmian za każdym dodaniem wierszu



Metodą zapisu zmian za dodaniem wszystkich wierszy

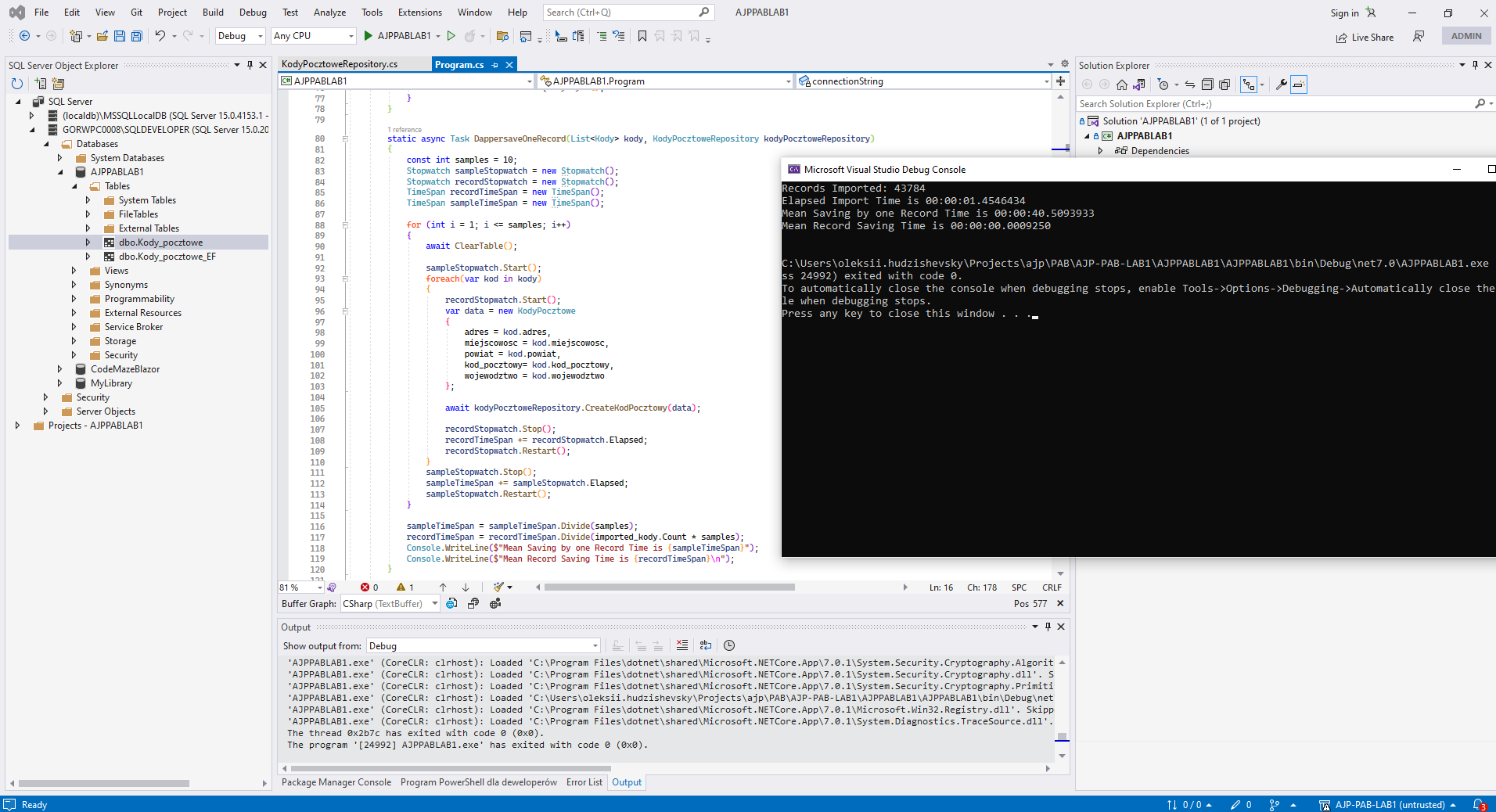


Zapisywanie metodą „paczkowania (500)”

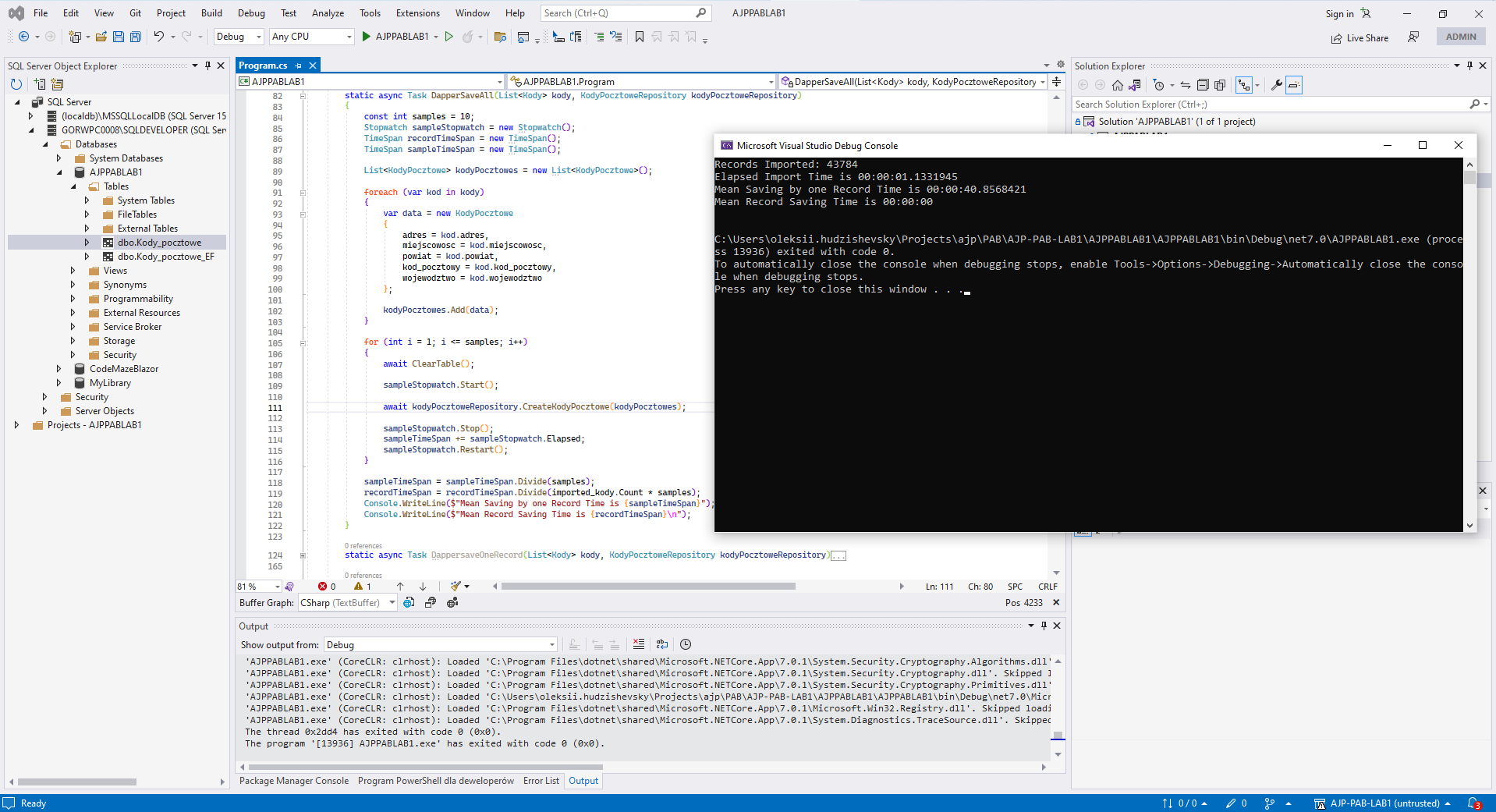


### Zapisywanie za pomocą Dapper

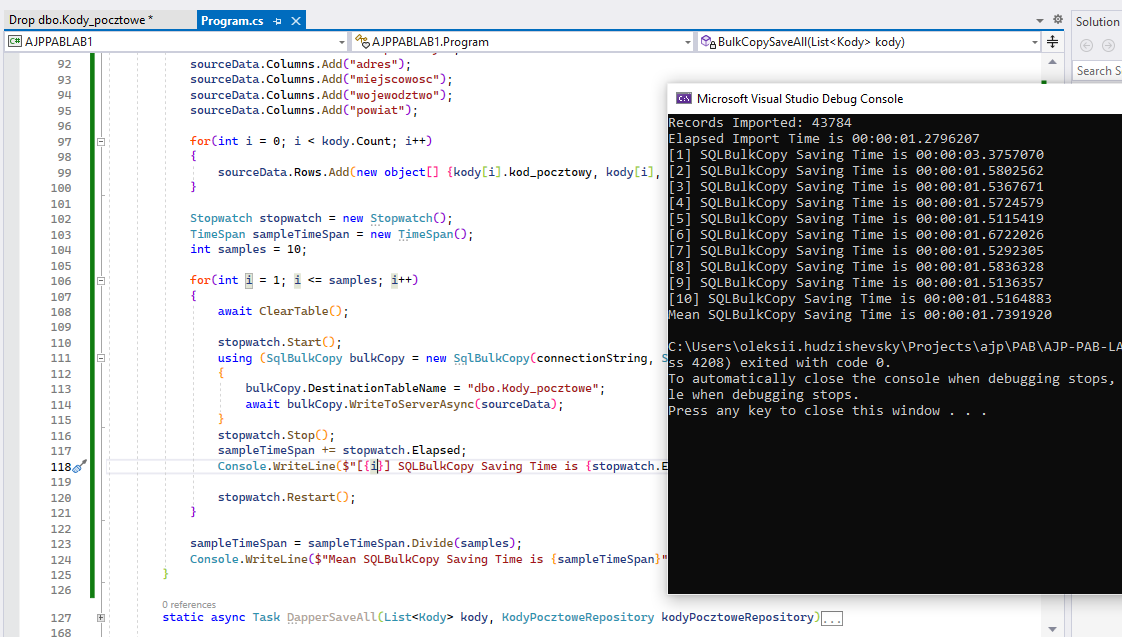
Zapisywanie każdego rekordu osobno



Zapisywanie całej kolekcji na raz



### Zapisywanie za pomocą SQLBulkCopy



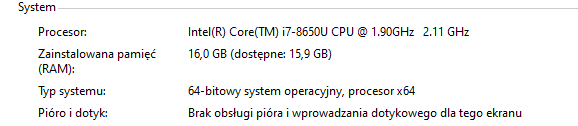
### Podsumowanie wyników

Specyfikacja komputera, na którym były przeprowadzane testy:

**RAM**: 16 GB

**CPU**: Intel Core i7-8650U 1.90 GHz 2.11GHz

**IDE**: Visual Studio 2022 Community



Przeprowadzono zostało 7 testów z wyliczeniem średniego czasu z dziesięciu prób zapisu informacji z pliku csv do bazy danych. Najlepszy wynik pokazała metoda zapisu zmian na końcu za pomocą Entity Framework

|  |  |  |
| --- | --- | --- |
| 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., 0.79120 ms. | 0.00088 ms. |
| ADO.NET – zapis całej kolekcji | 37 sek., 0. 86591 ms. | 0.00086 ms. |
| EntityFramework – pojedynczy zapis | 35 min., 19 sek., 0.70518 ms. | 0.04841 ms. |
| EntityFramework – zapis całej kolekcji | 7 sek., 0.69418 ms. | 0.00015 ms. |
| EntityFramework – zapis metodą „paczkowania (500)” | 56 sek., 0.13109 ms. | 0.00127 ms. |
| Dapper – pojedynczy zapis | 40 sek., 0.50939 ms. | 0.00092 ms. |
| Dapper – zapis całej kolekcji | 40 sek., 0.85684 ms. | 0.00093 ms. |
| SQLBulkCopy – zapis całej kolekcji | 1 sek., 0.73919 ms. | - |

# Wnioski

# Bibliografia

(Linki aktualne na dzień 03/03/2023)

* <https://code-maze.com/>
* <https://learn.microsoft.com/>
* <https://metanit.com/sharp>
* <https://github.com/DapperLib/Dapper>
* <https://code-maze.com/csharp-read-data-from-csv-file/>
* <https://www.techiedelight.com/measure-execution-time-csharp/>
* <https://code-maze.com/using-dapper-with-asp-net-core-web-api/>
* <https://peterdaugaardrasmussen.com/2022/01/05/csharp-program-does-not-contain-a-static-main-method-suitable-for-an-entry-point-when-building-solution/>
* <https://stackoverflow.com/questions/47058423/csvhelper-no-members-are-mapped-for-type/47059214#47059214>
* <https://stackoverflow.com/questions/74155920/no-members-mapped-for-type-csv-helper>
* <https://stackoverflow.com/questions/74467642/entity-framework-core-7-connection-certificate-trust-exception>
* <https://stackoverflow.com/questions/8602395/timeout-expired-the-timeout-period-elapsed-prior-to-completion-of-the-operation>

# Spis ilustracji

# Spis snippetów

**Nie można odnaleźć pozycji dla spisu ilustracji.**