# Laboratorium 6 – MS SQL Server

**Temat:** Tworzenie aplikacji baz danych za pomocą ADO.NET

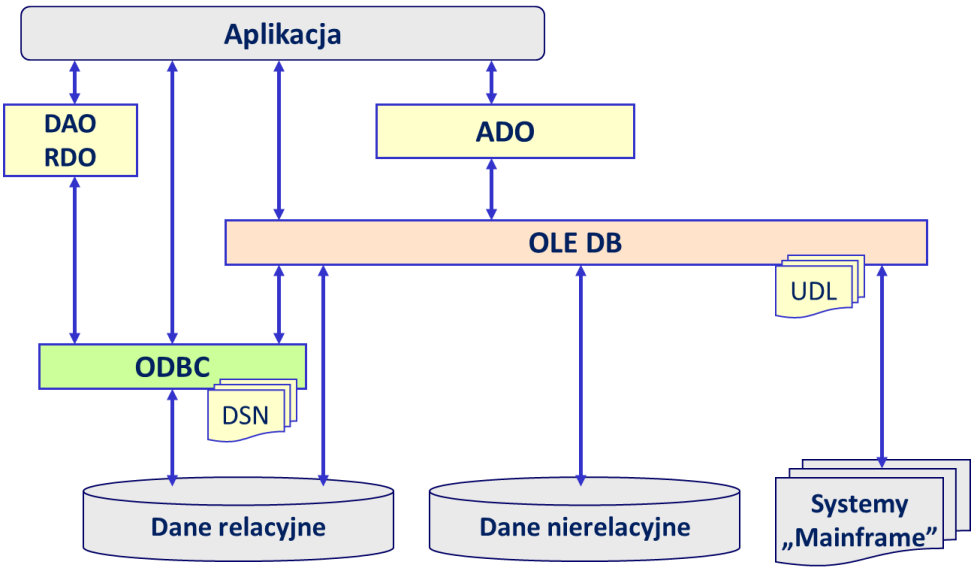
Opracowanie: A.Dydejczyk

Tematyka prezentowana w trakcie laboratorium:

1. Architektura ADO.NET
2. Przestrzenie nazw ADO.NET
3. Podstawowe klasy w przestrzeni nazw ADO.NET System.Data
4. Obiekt klasy SqlConnection
5. Obiekt klasy SqlCommand
6. Obiekt klasy SqlDataReader
7. Obiekt klasy SqlDataAdapter
8. Ćwiczenia prezentujące omówione zagadnienia interfejsu ADO.NET
9. Zadanie

## Wprowadzenie

Dostęp do baz danych w ramach rozwiązań firmy Microsoft został przedstawiony na rys.1. Początkowo aplikacje uzyskiwały dostęp do baz danych poprzez interfejs DAO (*Data Access Object* ) lub RDO (*Remote DataBase Objects)*. Kolejną formą połączenia z bazą danych jest interfejs ODBC. Pod koniec lat 80-tych opracowano interfejs OLE DB (*Object Linking and Embeding).* Obecnie do połączenia z bazą danych wykorzystujemy interfejs ADO.



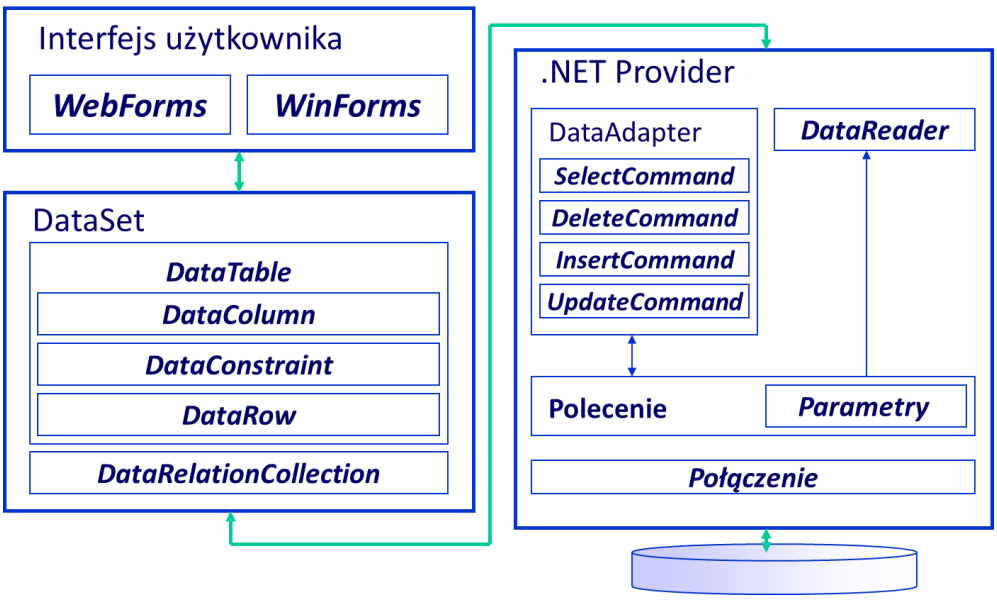

Rys. 1. Dostęp do baz danych w ramach technologii udostępnianych przez Microsoft

W ramach technologii ADO.NET wprowadzonej wraz platformą .NET Framework użytkownicy otrzymują następujące funkcjonalności:

**Integracja z platformą .NET Framework - Dane załadowane do pamięci operacyjnej mogą zostać udostępnione jako zwykłe struktury środowiska .NET Framework (na przykład tablice i kolekcje), co zapewnia spójne metody dostępu do danych relacyjnych.**

- ✓ **Lepsza integracja z XML** - Walidacja danych, wykonywanie zapytań hierarchicznych i transformacji danych relacyjnych
- ✓ **Poprawiona obsługa danych w trybie off-line** - Obiekt o nazwie DataSet, działa jak standardowa, znajdująca się w pamięci relacyjna reprezentacja danych. Ponieważ obiekt DataSet z założenia nie posiada stałego połączenia z bazą danych, doskonale nadaje się do pakowania, wymiany, buforowania, przechowywania i ładowania danych.
- ✓ **Jawna kontrola sposobu dostępu do danych** - ADO charakteryzuje się niejawnym definiowaniem zachowań, co nie zawsze jest potrzebne w aplikacjach, dlatego może to ograniczać ich wydajność. ADO.NET zapewnia dobrze zdefiniowane komponenty o dobrze wyodrębnionej funkcjonalności.

Architektura ADO.NET została przedstawiona na rys.2.




Rys. 2. Architektura ADO.NET

W ramach architektury ADO.NET w strukturze .NET Framework otrzymujemy przestrzenie nazw System.Data.*, które zawierają klasy i wyliczenia, które są głównym narzędziem dającym dostęp do bazy danych. Klasy z przestrzeni nazw System.Data.* umożliwiają dostęp do baz danych w czasie rzeczywistym lub dla klas DataSet, DataTable i DataAdapter dostęp offline.

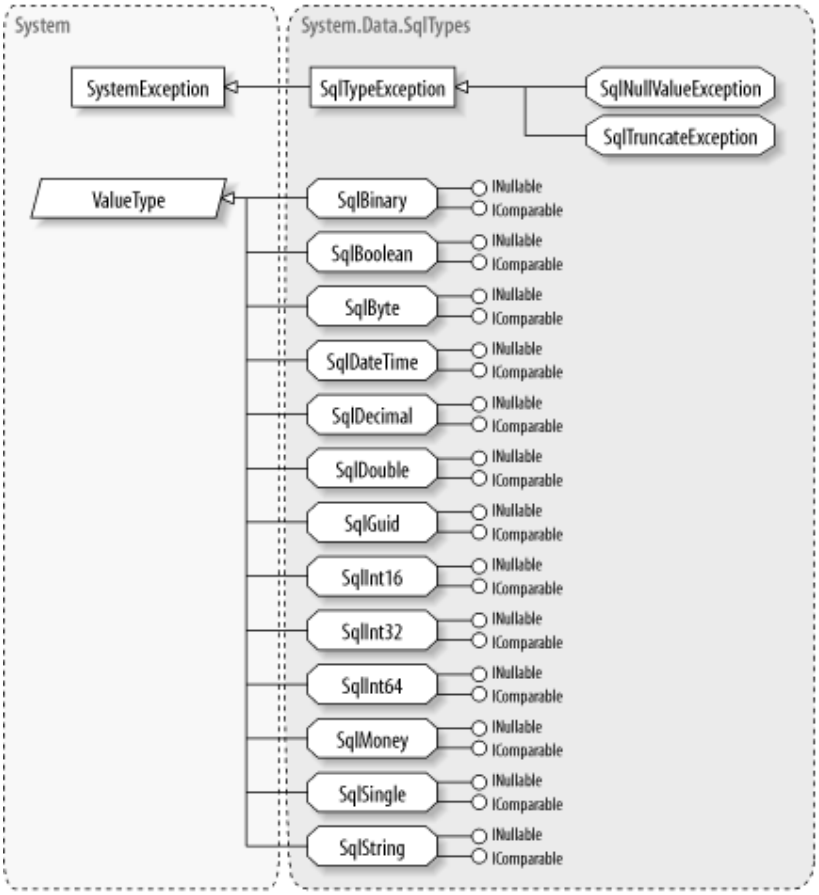
Najczęściej używane przestrzenie nazw dające dostęp do SQL Server'a:

- **System.Data** - daje dostęp do klas implementujących architekturę ADO.NET, takich jak DataSet i DataTable.
- **System.Data.Common** - umożliwia dostęp do klas współdzielonych przez sterowniki dostępu do danych .NET Framework, takie jak klasa DbProviderFactory.

- **System.Data.SqlClient** - podstawowa przestrzeń nazw dla natywnych mechanizmów łączenia z SQL Server. Przestrzeń nazw zawiera klasy oferujące zoptymalizowany dostęp do SQL Server (w wersji 7.0 i nowszych) za pomocą SQL Server Native Client. Klasy z tej przestrzeni nazw są zaprojektowane specjalnie w taki sposób, by wykorzystywać specyficzne dla SQL Server mechanizmy i nie współpracują z innymi źródłami danych.
- **System.Data.Odbc** - udostępnia zarządzany dostęp do sterowników ODBC starego typu. Mechanizm ODBC został opracowany jako uniwersalny standard do łączenia wielu różnych źródeł danych. Ponieważ jego misją była standaryzacja dostępu do danych z różnych źródeł, ODBC udostępnia typowy interfejs, który czasami nie wykorzystuje specyficznych dla SQL Server lub innych baz danych (DBML) mechanizmów. Oznacza to, że ODBC nie jest tak wydajny jak klient SQL, ale jest użyteczną opcją przy łączeniu z różnorodnymi źródłami danych, takimi jak arkusze Excel i inne DBMS-y.
- **System.Data.OleDb** – umożliwia łączenie się z różnymi źródłami danych, w tym z SQL Server. Jest to opcja dla aplikacji, które muszą mieć dostęp do danych na wielu platformach, takich jak SQL Server i Microsoft Access. Mechanizm OLEDB został oznaczony przez Microsoft jako przestarzały na korzyść bardziej rozpowszechnionego ODBC, mimo że mechanizm OLEDB powstał później niż ODBC.
- **System.Data.SqlTypes** - (rys.3) udostępnia klasy .NET reprezentujące natywne, zerowane typy danych SQL Server. Te specyficzne dla SQL Server typy danych .NET korzystają w dużej mierze z tej samej reprezentacji, co odpowiadające im natywne typy danych SQL Server, co pozwala zredukować problemy z utratą precyzji. Korzystanie z tych typów przyspiesza łączenie z SQL Server, ponieważ pozwala wyeliminować wewnętrzne konwersje. Wszystkie typy danych inaczej niż standardowe typy wartości .NET mają wbudowaną możliwość obsługi wartości NULL.

| Klasa System.Data.SqlTypes T-SQL | Natywny typ danych |
|---|---|
| SqlBinary | binary, image, timestamp, varbinary |
| SqlBoolean | bit |
| SqlByte | tinyint |
| SqlDateTime | datetime, smalldatetime |
| SqlDecimal | decimal, numeric |
| SqlDouble | float |
| SqlGuid | uniqueidentifier |
| SqlInt16 | smallint |
| SqlInt32 | int |
| SqlInt64 | bigint |
| SqlMoney | money, smallmoney |
| SqlSingle | real |
| SqlString | char, nchar, ntext, nvarchar, text, varchar |
| SqlXml | xml |

Tabela1. Konwersja typów System.Data.SqlTypes
https://msdn.microsoft.com/pl-pl/library/system.data.sqltypes%28v=vs.110%29.aspx




Rys.3. Dostępne typy w ramach przestrzeni System.Data.SqlTypes

Do połączenia z bazą SQL Server wykorzystujemy natywnego klienta SQL dla .NET (SQLNCLI). Najczęściej wykorzystywane klasy w ramach tego klienta przedstawiono w tabeli 2.

| **Klasa System.Data.SqlClient** | **Opis** |
|---|---|
| SqlCommand | *Reprezentuje wyrażenie SQL lub procedurę składowaną do wykonania.* |
| SqlCommandBuilder | *Automatycznie generuje polecenia dla pojedynczej tabeli uzgadniające zmiany wprowadzone do DataSet z ADO.NET.* |
| SqlConnection | *Ustanawia połączenie z SQL Server.* |
| SqlConnectionStringBuilder | *Tworzy ciąg znaków opisujący połączenie do wykorzystania w obiektach SqlConnection.* |
| SqlDataAdapter | *Opakowuje zestaw obiektów SqlCommand i SqlConnection, które mogą być wykorzystywane do wypełnienia DataSet z ADO.NET i aktualizacji bazy danych SQL Server.* |
| SqlDataReader | *Udostępnia metody do odczytu pobieranego strumienia wierszy z bazy danych SQL Server.* |
| SqlException | *Daje dostęp do specyficznych dla SQL Server wyjątków. Klasa ta może być wykorzystana do przechwycenia błędu lub ostrzeżenia z SQL Server.* |
| SqlParameter | *Reprezentuje parametr do SqlCommand.* |
| SqlParameterCollection | *Kolekcja obiektów SqlParameter związanych z SqlCommand.* |
| SqlTransaction | *Umożliwia inicjalizowanie transakcji SQL Server od strony klienta i zarządzanie nimi.* |

Tabela 2. Klasy natywnego klienta SQL

## Ćwiczenie A. Połączenie ze źródłem danych, SqlDataReader

Tworzymy aplikację konsolową w języku C#, która pobiera dane on-line w jedną stronę i tylko w trybie do odczytu. Kolejność wykonanych operacji w interfejsie Visual Studio przedstawiono w poniższych punktach.

a) Otwieramy program Visual Studio.
b) Tworzymy nowy projekt w języku Visual C# typu Console Application i nadajemy nazwę np. Lab6_01.
c) Po wprowadzeniu zawartości skryptu kompilujemy F7 ( Build, Build Solution) i wykonujemy Ctrl-F5 ( Debug, Start Without Debugging )

```
// Skrypt Lab06.01
using System;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_1
   {
      static void Main(string[] args)
      {
         string sqlconnection = @"DATA SOURCE=MSSQLServer;" +
         "INITIAL CATALOG=AdventureWorks; INTEGRATED SECURITY=SSPI;";
         string sqlcommand = "SELECT  DepartmentId, Name,  GroupName " +
         " FROM HumanResources.Department  ORDER BY DepartmentId";
         SqlConnection connection = new SqlConnection(sqlconnection);
         try
         {
            //connection.ConnectionString = sqlconnection ;
            connection.Open();
          SqlCommand command = new SqlCommand(sqlcommand, connection);
            SqlDataReader datareader = command.ExecuteReader();
            while (datareader.Read())
            {
               Console.WriteLine  (  "{0}\t{1}\t{2}",
               datareader["DepartmentId"].ToString(),
               datareader["Name"].ToString(),
               datareader["GroupName"].ToString()   );
            }
         }
         catch (SqlException ex)
         {    Console.WriteLine(ex.Message);        }
         finally
         {    connection.Close();   }
         Console.Write("Press a Key to Continue...");
         Console.ReadKey();
      }
   }
}
```

Uwagi:

1. Połączenie z bazą danych realizowane jest przez klasę SQLConnection w oparciu o szeregu par klucz-wartość oddzielonych średnikiem:
   DATA SOURCE=MSSQLServer; INITIAL CATALOG=AdventureWorks; INTEGRATED SECURITY=SSPI;
2. Konstrukcja **try … catch** zapewnia poprawne wykonanie kodu w przypadku błędów w czasie przetwarzania danych na serwerze baz danych. Możliwe zakończenie połączenia **connection.Close()** lub **connection.Dispose().**
3. Metoda ExecuteReader() zwraca instancję SqlDataReader, która pozwala pobrać w wiersze zawierające wyniki tylko do odczytu.
4. Połączenie z bazą danych „Connection String”
   https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-string-syntax

| Dostawca | Struktura zapytania |
|---|---|
| SqlCLient | *Integrated Security=true;*<br>*Integrated Security=SSPI;* |
| OleDB | *Integrated Security=SSPI;* |
| ODBC | *Trusted_Connection=yes;* |
| OracleClient | *Integrated Security=yes;* |

   Dodatkowe materiały omawiające zagadnienie połączenia z bazą danych:
   - The Connection Strings Reference
     https://www.connectionstrings.com/
   - SQL Server Connection Pooling (ADO.NET)
     https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql-server-connection-pooling

5. Połączenie z bazą danych „AdvantureWorks” na lokalnym serwerze wykorzystując metodę „windows authentication” z dostawcą SqlClient.

```
"Persist Security Info=False;Integrated Security=true;
   Initial Catalog=AdventureWorks;Server=MSSQL1"
"Persist Security Info=False;Integrated Security=SSPI;
   database=AdventureWorks;server=(local)"
"Persist Security Info=False;Trusted_Connection=True;
   database=AdventureWorks;server=(local)"
```

6. Połączenie z bazą danych „AdvantureWorks” wykorzystując uwierzytelnienie na serwerze SQL Server z dostawcą SqlClient.

```
“Persist Security Info=False;User ID=*****;Password=*****;
   Initial Catalog=AdventureWorks;Server=MySqlServer"
```

## Ćwiczenie B. Połączenie ze źródłem danych, SqlDataAdapter

Klasa SqlDataAdapter tworzy odłączony zbiór danych, który jest zapisany w pamięci podręcznej. Daje on elastyczność, ponieważ nie musimy utrzymywać stałego połączenia z bazą danych, by odpytywać i modyfikować dane.
W ramach Visual Studio tworzymy projekt zawierający aplikację konsolową w języku C# wykorzystującą SqlDataAdapter. Wykorzystując SqlDataAdapter wypełniamy DataSet a następnie wyświetlimy wyniki.

```
// Skrypt Lab06.02
using System;
using System.Data;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_2
   {
      static void Main(string[] args)
      {
         string sqlconnection = @"DATA SOURCE=MSSQLServer;" +
         "INITIAL CATALOG=AdventureWorks; INTEGRATED SECURITY=SSPI;";
         string sqlcommand = "SELECT  DepartmentId, Name, GroupName " +
         " FROM HumanResources.Department  ORDER BY DepartmentId";
         SqlDataAdapter adapter = null;
         DataSet dataset = null;
         try
         {
            adapter = new SqlDataAdapter(sqlcommand, sqlconnection);
            dataset = new DataSet();
            adapter.Fill(dataset);
            foreach (DataRow row in dataset.Tables[0].Rows)
            {
               Console.WriteLine  (  "{0}\t{1}\t{2}",
               row["DepartmentId"].ToString(),
               row["Name"].ToString(),
               row["GroupName"].ToString()    );
            }
         }
         catch (SqlException ex)
         {   Console.WriteLine(ex.Message);   }
         finally
         {
            if (dataset != null)
               dataset.Dispose();
            if (adapter != null)
               adapter.Dispose();
         }
         Console.Write("Press a Key to Continue...");
         Console.ReadKey();
      }
   }
}
```

Uwagi:

1. W ramach tego zadania wykorzystujemy deklarację SqlDataAdapter i DataSet zamiast SqlConnection, SqlCommand i SqlDataReader.
2. Kod pobierający dane tworzy nowe instancje SqlDataAdapter i DataSet, a następnie wypełnia DataSet za pomocą metody Fill() z SqlDataAdapter.
3. Dane odczytujemy z tabeli (**dataset.Tables[0]**) zwróconej przez DataSet w pętli wykorzystując obiekt DataRow.

## Ćwiczenie C. Zapytanie parametryzowane

W ramach tego ćwiczenia opracujemy aplikację konsolową wykorzystującą bezpieczną metodę przekazywania parametrów do procedur składowanych
lub wyrażeń SQL nazywaną parametryzacją dostępną w ADO.NET. Każdy obiekt SqlCommand udostępnia właściwość SqlParameterCollection o nazwie Parameters. Metoda Add kolekcji Parameters pozwala dodać parametry do polecenia SqlCommand. Parametry z kolekcji Parameters są przekazywane do SQL Server z wyrażeniem SQL podczas wywoływania metod ExecuteReader(), ExecuteScalar(), ExecuteNonQuery() czy ExecuteXmlReader() z SqlCommand.

W przypadku tworzenia zapytań do bazy danych wykorzystujących SqlParametr należy korzystać z enumeracji dostępnej w przestrzeni nazw **System.Data.SqlDbTypes**. Lista typów zdefiniowanych w ramach tej przestrzeni dostępna jest pod poniższym adresem:

https://msdn.microsoft.com/pl-pl/library/system.data.sqldbtype%28v=vs.110%29.aspx

Dodanie obiektu Parameter do SqlCommand jest krytyczne; jest to ten fragment kodu, który zabezpiecza przed atakami ze wstrzykiwaniem SQL.

```
// Skrypt Lab06.03
using System;
using System.Data;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_3
   {
      static void Main(string[] args)
      {
         string name = "SMITH";
         string sqlconnection = @"SERVER=MSSQLServer; " +
         "INITIAL CATALOG=AdventureWorks2008; INTEGRATED SECURITY=SSPI;";
         string sqlcommand = "SELECT " +
         " BusinessEntityID, FirstName,  MiddleName, LastName " +
         "FROM Person.Person WHERE LastName = @name";
         SqlConnection connection = null;
         SqlCommand command = null;
         SqlDataReader datareader = null;
         try
         {
            connection = new SqlConnection(sqlconnection);
            connection.Open();
            command = new SqlCommand(sqlcommand, connection);
            command.Parameters.Add("@name", SqlDbType.NVarChar, 50).Value = name;
```

```
            datareader = command.ExecuteReader();
            while (datareader.Read())
            {
               Console.WriteLine   (  "{0}\t{1}\t{2}\t{3}",
               datareader["BusinessEntityID"].ToString(),
               datareader["LastName"].ToString(),
               datareader["FirstName"].ToString(),
               datareader["MiddleName"].ToString()   );
            }
         }
         catch (Exception ex)
         {   Console.WriteLine(ex.Message);    }
         finally
         {   connection.Close();   }
         Console.WriteLine("Press any key...");
         Console.ReadKey();
      }
   }
}
```

Uwagi:

1. Tworzymy obiekt korzystając z klasy SqlCommand na podstawie wcześniej zdefiniowanego polecenia SQL.
2. Wykorzystując metodę Add dodajemy do kolekcji Parameters parametr @name typu nvarchar o długości 50 dostępny w ramach przestrzeni nazw **System.Data.SqlDbTypes**.

## Ćwiczenie D. Wiele aktywnych zbiorów wyników

W ramach ćwiczenia wykorzystamy możliwość korzystania z wielu aktywnych zbiorów wyników w ramach serwera SQL Server – technologię MARS (*Multiple Active Result Sets*). MARS umożliwia przetwarzanie wielu otwartych zbiorów wyników w jednym połączeniu. W ramach zadania przedstawiono sposób wykorzystania MARS do wykonywania różnych zadań w jednym połączeniu.

```
 // Skrypt Lab06.04
using System;
using System.Data;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_4
   {
      static string sqlconnection = @"SERVER=MSSQLServer; " +
      "INITIAL CATALOG=AdventureWorks; INTEGRATED SECURITY=SSPI; " +
      "MULTIPLEACTIVERESULTSETS=true; ";
      static string sqlcommand1 = "SELECT " +
      " DepartmentID, Name, GroupName " +
      "FROM HumanResources.Department; ";
      static string sqlcommand2 = "SELECT " +
      " ShiftID, Name, StartTime, EndTime " +
      "FROM HumanResources.Shift; ";
```

```
static SqlConnection connection = null;
static SqlCommand command1 = null;
static SqlCommand command2 = null;
static SqlDataReader datareader1 = null;
static SqlDataReader datareader2 = null;
static void Main(string[] args)
{
   try
   {
      connection = new SqlConnection(sqlconnection);
      connection.Open();
      command1 = new SqlCommand(sqlcommand1, connection);
      command2 = new SqlCommand(sqlcommand2, connection);
      datareader1 = command1.ExecuteReader();
      datareader2 = command2.ExecuteReader();
      int i = 0;
      Console.WriteLine("===========");
      Console.WriteLine("Departments");
      Console.WriteLine("===========");
      while (datareader1.Read() && i++ < 3)
      {
         Console.WriteLine (  "{0}\t{1}\t{2}",
         datareader1["DepartmentID"].ToString(),
         datareader1["Name"].ToString(),
         datareader1["GroupName"].ToString()  );
      }
      Console.WriteLine("======");
      Console.WriteLine("Shifts");
      Console.WriteLine("======");
      while (datareader2.Read())
      {
         Console.WriteLine (  "{0}\t{1}\t{2}\t{3}",
         datareader2["ShiftID"].ToString(),
         datareader2["Name"].ToString(),
         datareader2["StartTime"].ToString(),
         datareader2["EndTime"].ToString()   );
      }
      Console.WriteLine("======================");
      Console.WriteLine("Departments, Continued");
      Console.WriteLine("======================");
      while (datareader1.Read())
      {
         Console.WriteLine  (  "{0}\t{1}\t{2}",
         datareader1["DepartmentID"].ToString(),
         datareader1["Name"].ToString(),
         datareader1["GroupName"].ToString()  );
      }
   }
   catch (SqlException ex)
   {  Console.WriteLine(ex.Message);  }
   finally
   {
      if (datareader1 != null)
         datareader1.Dispose();
      if (datareader2 != null)
         datareader2.Dispose();
      if (command1 != null)
         command1.Dispose();
      if (command2 != null)
         command2.Dispose();
```

```
            if (connection != null)
               connection.Dispose();
         }
         Console.WriteLine("Press a key to end...");
         Console.ReadKey();
      }
   }
}
```

Uwagi:

1. Kluczowa dla uruchomienia MARS jest para klucz-wartość MULTIPLEACTIVERESULTSETS=true w ciągu opisującym połączenie.
2. W ramach aplikacji deklarujemy jedno połączenie wykorzystując SqlConnection, tworzymy dwa obiekty SqlCommand i dwa obiekty SqlDataReader. Po otwarciu połączenia, tworzone są dwa obiekty SqlCommand pozwalające w jednym połączeniu pobrać dwa zbiory wyników.

## Ćwiczenie E. Połączenie ze źródłem danych – polecenie „using”

Realizacja zadania z ćwiczenia A (połączenie z bazą danych) z wykorzystaniem polecenia „using”.

```
// Skrypt Lab06.05
using System;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_5
   {
      static void Main(string[] args)
      {
         string sqlconnection = @"DATA SOURCE=MSSQLServer;" +
         "INITIAL CATALOG=AdventureWorks; INTEGRATED SECURITY=SSPI;";
         string sqlcommand = "SELECT  DepartmentId, Name, GroupName " +
         " FROM HumanResources.Department  ORDER BY DepartmentId";
         using ( SqlConnection connection = new SqlConnection() )
         {
            connection.ConnectionString = sqlconnection ;
            connection.Open();
            SqlCommand command = new SqlCommand(sqlcommand, connection);
            using ( SqlDataReader datareader = command.ExecuteReader() )
            {
              while (datareader.Read())
              {
                Console.WriteLine
                (
                "{0}\t{1}\t{2}",
                datareader["DepartmentId"].ToString(),
                datareader["Name"].ToString(),
                datareader["GroupName"].ToString()
                );
              }
            }
```

```
            }
            Console.Write("Press a Key to Continue...");
            Console.ReadKey();
        }
    }
}
```

## Ćwiczenie F. Metody wykonania wyrażeń SQL klasy SqlCommand

W ramach tego ćwiczenia przedstawione zostaną metody wykonania poleceń SQL dostępne dla klasy SqlCommand (również dla OleDbCommand). W ramach poprzednich zadań przedstawiona została metoda ExecuteReader zwracająca wynik wykonania zapytania SQL Select na źródłach danych. Pozostałe metody wykorzystywane do realizacji poleceń SQL to ExecuteNonQuery, ExecuteScalar i ExecuteXMLReader.

| Metoda | Opis |
|---|---|
| ExecuteNonQuery | Metodę używa się do wykonania wyrażenia na połączonym źródle danych. Stosuje się do wyrażeń DDL, zapytań operacji tj. INSERT, UPADTE i DELETE. Zwracana jest ilość wierszy, ale nie zwraca parametrów wyników czy zestawu wyników. |
| ExecuteReader | Metoda służy do realizacji zapytań SQL Select. |
| ExecuteScalar | Metoda wykorzystywana do realizacji procedury składowanej lub wyrażenia SQL, gdy zwracana jest pojedyncza wartość skalarna. |
| ExecuteXMLReader | Metoda wykorzystywana do realizacji wyrażeń FOR XML Select. |

Tabela 3. Funkcjonalność metod realizowanych w ramach klasy SqlCommand

```
// Skrypt Lab06.06
using System;
using System.Data;
using System.Data.SqlClient;
namespace Lab06.Examples
{
   class Example_Lab06_6
   {
      static void Main(string[] args)
      {
         string sqlconnection = @"SERVER=MSSQLServer; " +
         "INITIAL CATALOG=tempdb; INTEGRATED SECURITY=SSPI;";
         string sqlcommand1 = "CREATE TABLE t1 ( a1 char(5), a2 char(5) )";
         string sqlcommand2 = "INSERT INTO  t1 VALUES ('test1','test2')";
         string sqlcommand3 = "SELECT * FROM t1";
         string sqlcommand4 = "SELECT COUNT(*) FROM t1";
         SqlConnection connection = null;
         SqlCommand command = null;
         SqlDataReader datareader = null;
         try
         {
            connection = new SqlConnection(sqlconnection);
            connection.Open();
            Console.WriteLine("====== CREATE TABLE ======");
```

```
            command = new SqlCommand(sqlcommand1, connection);
            command.ExecuteNonQuery();
            Console.WriteLine("====== INSERT DATA  ======");
            command = new SqlCommand(sqlcommand2, connection);
            command.ExecuteNonQuery();
            Console.WriteLine("====== SELECT  ======");
            command = new SqlCommand(sqlcommand3, connection);
            datareader = command.ExecuteReader();
            while (datareader.Read())
            {
               Console.WriteLine("{0}\t{1}",
               datareader["a1"].ToString(), datareader["a2"].ToString());
            }
            datareader.Close();
            Console.WriteLine("====== COUNT()  ======");
            command = new SqlCommand(sqlcommand4, connection);
            Object count = command.ExecuteScalar();
            Console.WriteLine(count.ToString());
         }
         catch (Exception ex)
         { Console.WriteLine(ex.Message); }
         finally
         { connection.Close(); }
         Console.WriteLine("Press any key...");
         Console.ReadKey();
      }
   }
}
```

## Zadanie

1. Opracować skrypt T-SQL tworzący użytkownika Lab6user w bazie SQL Server ( jeżeli istnieje usunąć ), bazę danych Lab6db ( jeżeli istnieje usunąć ) oraz na koniec przypisać użytkownika Lab6user jako **db_owner** do bazy danych Lab6db.
2. Opracować procedurę T-SQL tworzącą następujące tabele w utworzonej bazie danych Lab6db oraz odpowiednie relacje pomiędzy tabelami:
   student ( id int PK,  fname varchar(30), lname varchar(30) )
   wykladowca ( id int PK , fname varchar(30), lname varchar(30) )
   przedmiot ( id int PK , name varchar(50) )
   grupa ( id_wykl int, id_stud int, id_przed int )
   (id_wykl FK(wykladowca id), id_stud FK(student id), id_przed FK(przedmiot id)) PK
3. Opracować program w języku C# realizujący następującą funkcjonalność:
   a. Realizujący procedurę opracowaną w punkcie 2 - połączenie z bazą danych wykonane dla  użytkownika Lab6user;
   b. Wprowadzenie danych do tabel z plików CSV - połączenie z bazą danych wykonane dla  użytkownika Lab6user;
   c. Wykonanie zapytania zwracającego liczbę studentów dla każdego wykładowcy oraz liczbę studentów zapisanych na określony przedmiot - połączenie z bazą danych wykonane dla  użytkownika Lab6user.