|  |  |  |  |
| --- | --- | --- | --- |
| **Laboratorium 2**  **Sprawozdanie z realizacji laboratorium** | | | |
| **Temat:** Praca z bazą danych SqlClient | **Nr Albumu:** 028487 | **Grupa/zespół:** GL01 | **Rok/semestr:** III / 6 |
| **Wykonał:**  Oleksii Hudzishevskyi | **Data wykonania:**  13/03/2023 | | **Data oddania:**  15/03/2023 |
| **Ocena:** | | **Podpis prowadzącego:** |

# Spis treści

[1. Spis treści 1](#_Toc129787881)

[2. Cel ćwiczenia 2](#_Toc129787882)

[3. Wymagania znajomości zagadnień 2](#_Toc129787883)

[4. Literatura, materiały dydaktyczne 2](#_Toc129787884)

[5. Wiadomości teoretyczne. 2](#_Toc129787885)

[6. Przebieg ćwiczenia 7](#_Toc129787886)

[7. Opracowanie sprawozdania 8](#_Toc129787887)

[8. Wnioski 8](#_Toc129787888)

[9. Bibliografia 8](#_Toc129787889)

[10. Spis ilustracji 8](#_Toc129787890)

[11. Spis snippetów 8](#_Toc129787891)

# Cel ćwiczenia

Praktyczne wprowadzanie do Przetwarzania transakcyjnego. Wszystkie zadania w trakcie tego laboratorium wykonywane będą z wykorzystaniem dostawcy danych ADO.NET – SqlClient ( System.Data.SqlClient ). Na tym etapie studenci nie powinni wykorzystywać do tego celu żadnego ORM-a

# 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/>
* <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>
* <https://docs.microsoft.com/pl-pl/dotnet/api/system.data.sqlclient.sqltransaction?view=dotnet-plat-ext-5.0>
* <https://www.webtrainingroom.com/adonet/transaction>
* <https://www.plukasiewicz.net/Artykuly/SQL_Transactions>

# Wiadomości teoretyczne.

Przydatne wskazówki:

* **Pobranie ID ostatnio dodanego do bazy danych rekordu**

W przypadku gdy mamy w bazie danych tabele dla której klucz główny (id) nadawane są przez bazę automatycznie i chcemy uzyskać dostęp do przydzielonego id czyli mamy włączone IDENTITY z opcją auto-increment, to możemy to zrealizować na przykład poprzez:

Na końcu CommandText w SqlCommand dopisujemy:

cmd.CommandText += " SELECT SCOPE\_IDENTITY();";

teraz możemy wysłać polecenie do bazy ale wykorzystując do tego ExecuteScalar():

**object** value = cmd.ExecuteScalar();

if(value != DBNull.Value)

{

**string** ResultId = value.ToString();

}

Oczywiście jeżeli chcemy pobrać wartość jako na przykład **long** to musimy zastosować odpowiednie rzutowanie.

Wywołując **ExecuteScalar** najbezpieczniej jest zastosowanie przypisania do **object**, wynika to z faktu, że zwrócone może być DBNull przy czym ważna uwaga to nie jest to samo co **null** w C#. Nie możemy bezpośrednio rzutować się pomiędzy tymi typami.

* **Zapisywanie do bazy danych rekordu zawierającego w danych pola null**

W sytuacji gdy chcemy zapisać do bazy danych jakikolwiek obiekt, który zawiera atrybuty nullable (dopuszczające null) lub też typu string musimy uwzględnić to przy budowaniu polecenia Insert (lub Update). Ponieważ w przypadku SqlClient zaleca się wykorzystanie parametryzowanych kwerend, to nie możemy przypisywać do parametru wartości null (jak już wspomniano nie jest to tym samym **null** z bazy danych). Przykład w jaki sposób można wykonać właściwe rzutowanie typów.

**SqlCommand** myCommand = new SqlCommand();

myCommand.CommandText = "INSERT INTO [InitialCatalog].[dbo].[Table] ( "

+ "[ColumnName1], "

+ "[ColumnName2], "

+ "[ColumnName3] "

+ " ) "

+ "VALUES ( @val1, @val2, @val3 ) ; ";

myCommand.Parameters.AddWithValue("@val1", var1);

if (var2 == null) myCommand.Parameters.AddWithValue("@val2", DBNull.Value);

else myCommand.Parameters.AddWithValue("@val2", var2);

myCommand.Parameters.AddWithValue("@val3", var3);

Jak widzimy w przytoczonym przykładzie, zmienne var1 oraz var3 są zmiennymi niedopuszczającymi wartości **null** (wiemy że takowa nie wystąpi) więc przy bindowaniu parametrów można swobodnie wykorzystać Parameters.AddWithValue przy czym co istotne nie musimy wnikać jakiego dokładnie typu to będą dane (int, string, DateTime, decimal …. ), nie ma znaczenia – środowisko sobie poradzi o ile nie będziemy mieli do czynienia z null. Dla tej sytuacji (zmienna var2) z przykładu widzimy uwarunkowanie bindowanej wartości: w przypadku null przekazujemy DBNull.Value, w pozostałych zaś przypadkach postępujemy tak samo jak w przypadku innych zmiennych z przykładu.

* **Odczytanie z bazy danych rekordu zawierającego w danych pola null**

W przypadku pobierania danych z obiektu Reader musimy postępować analogicznie, tzn. przed mapowaniem należy się upewnić czy zwracana dana ma wartość czy też jest to DBNull i w zależności od tego postępować różnie w trakcie odczytu. Przykład dla tego samego obiektu co powyżej:

List<NaszTyp> results = new List<NaszTyp>();

myCommand.CommandText = "SELECT "

+ "[ColumnName1], "

+ "[ColumnName2], "

+ "[ColumnName3] "

+ "FROM [InitialCatalog].[dbo].[Table] "

+ "WHERE …………… ";

try

{

myCommand.Connection.Open();

SqlDataReader reader = myCommand.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

int var1 = (int)reader["ColumnName1"];

DateTime? Var2 = null;

if (!reader.IsDBNull(reader.GetOrdinal("ColumnName2")))

{

var2 = (DateTime)reader["ColumnName2"];

}

string var3 = ((string)reader["ColumnName3"]).Trim();

result.Results.Add(new NaszTyp(var1, var2, var3));

}

}

reader.Close();

}

catch (SqlException e)

{

… (Jakaś obsługa wyjątku, logowanie etc.)

}

finally

{

if (myCommand.Connection != null) myCommand.Connection.Close();

}

* **Bardzo uproszczony przykład użycia transakcji w kodzie**

using (SqlConnection connection = new SqlConnection("Nasz ConnectionString"))

{

try

{

connection.Open();

using (SqlTransaction trans = connection.BeginTransaction())

{

Tutaj nasze operacje na bazie danych (Insert, Update, Delete);

…

…

trans.Commit();

}

}

catch (SqlException ex)

{

(Jakaś obsługa wyjątku, logowanie etc.)

}

}

# Przebieg ćwiczenia

Stworzyć w bazie danych co najmniej 2 tabele (można sobie tę bazę przygotować z wykorzystaniem dowolnych narzędzi). Tabele mają zawierać jakieś ograniczenia, np. „minimum Value”. Następnie przygotować prosty program (aplikacja WinForms) pobierający dane od użytkownika (z formularza na ekranie) następnie wykonujący operacje zapisu tych danych do wskazanych tabel po naciśnięciu przycisku na ekranie. W pierwszych 2 częściach zadania wykonać to z wykorzystaniem ADO.NET – przykładowe snipety kodu były załączone do instrukcji laboratoryjnej. Uwaga, to są jedynie wycinki kodu (niekoniecznie kompletne) więc w razie problemów, wszelkie brakujące fragmenty omówimy na zajęciach.

W pierwszej części ćwiczenia zadanie wykonać, jako dwie kolejne niepowiązane ze sobą operacje. W trakcie testów wprowadzić kolejno, zestaw danych zawierający prawidłowe wartości dla obu tabel, sprawdzić poprawność wykonania programu. Następnie podjąć próbę zapisania danych z zestawu zawierającego błędne dane, przy czym nieprawidłowość ta ma dotyczyć jednej z wybranych tabel. Sprawdzić zachowanie się programu oraz zmiany zachodzące w bazie danych po wywołaniu polecenia dodania nowych rekordów do tabel (w obu scenariuszach), spisać swoje uwagi oraz spostrzeżenia.

Po ukończeniu pierwszej części, należy zmodyfikować kod programu tak, aby operacje dodania wpisów do tych tabel zrealizowane zostały w obrębie jednej transakcji. Powtórzyć próby dodania danych analogicznie jak w poprzednim przypadku: zestaw prawidłowych danych, oraz zestaw zawierający błędy. Ponownie zweryfikować rezultat wykonywanych operacji w bazie danych, wyprowadzić i zanotować wnioski. Uzupełnić powyższe kody tak, by możliwym było zapisywanie rekordów zawierających null, zamiast wartości (oczywiście z uprzednią modyfikacją bazy danych, jeżeli będzie to konieczne). Mogą to być na przykład wartości liczbowe (integer) z dopuszczeniem null, czyli po stronie aplikacji są to typu nullable. Zaprezentować wyniki działania kodu oraz omówić działanie. Jaka jest różnica pomiędzy NULL w aplikacji oraz NULL w bazie danych, co to w praktyce oznacza w trakcie pisania kodu dodającego takowe „wartości” do bazy danych.

W następnej części ćwiczenia postarać się odpowiedzieć na pytanie: co jest ekwiwalentem transakcji w przypadku ORM (EF 6). Przebudować kod z laboratorium tak by zrealizować oba powyższe punktu (niezależne oraz transakcyjne dodawanie danych do tabel) tak aby został on zrealizowany za pomocą EF 6. W razie wątpliwości bądź pytań, wszystkie niejasności mogą zostać wyjaśnione przez prowadzącego NA ZAJĘCIACH, osoby nieobecne będą musiały zmierzyć się z problemem bez dodatkowych wskazówek ze strony prowadzącego. Ale biorąc pod uwagę frekwencję na zajęciach, zakładam, że nikt z nieobecnych raczej problemów mieć nie będzie, bo wygląda na to, że dla wszystkich tematyka zajęć jest w pełni zrozumiała i nie jest wymagane dodatkowe omawianie tych problemów.

Uwaga, dla wszystkich części laboratorium dodać widok w aplikacji, wyświetlający wszystkie dane z tabel w stosownej kontrolce, DataGridView, przykład użycia kontrolki postaramy omówić się w trakcie zajęć.

# Opracowanie sprawozdania

# Wnioski

# Bibliografia

1. Źródła pomocnicze
   1. Tytuł [link], dostęp: dd.mm.rrrr
2. Napotkane problem i ich rozwiązania
   1. Tytuł [link], dostęp: dd.mm.rrrr

# Spis ilustracji

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

# Spis snippetów

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