APBD - Ćwiczenie 4

pgago@pja.edu.pl

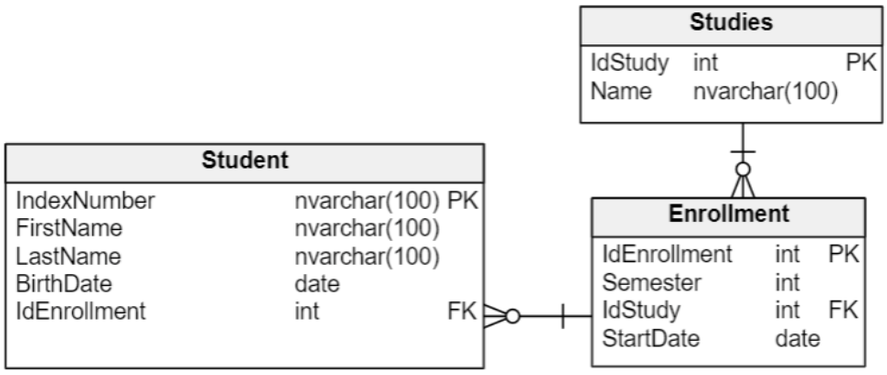
22 marca 2020

1 Wstęp

W niniejszych ćwiczeniach będziemy rozwijać aplikację napisaną w trakcie poprzednich ćwiczeń (ćwiczenia 3). Z tego powodu można użyć tego samego repozytorium. Proszę tylko pamiętać o odpowiednim nazywaniu commit'ów, tak aby sugerowały wykonanie ćwiczeń 3.

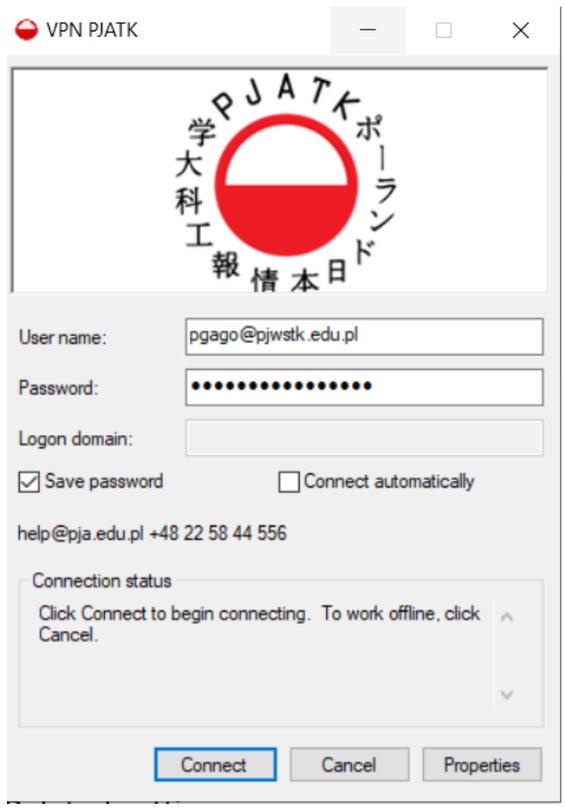
Zadanie 1 - przygotowanie bazy danych

W niniejszym zadaniu przygotujemy strukturę bazy danych, którą będziemy wykorzystywać w trakcie kolejnych zadań. Poniżej zaprezentowany jest jej diagram. W materiałach znajdziecie również skrypt o nazwie "db.sql", który zawiera kod DDL (w dialekcie Transact-SQL), który tworzy wspomnianą bazę.

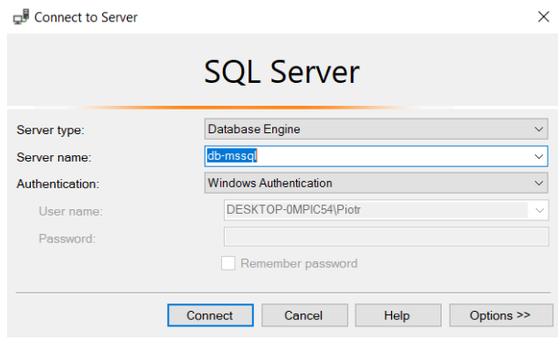


1. Prośba o połączenie się ze swoją szkolną bazę danych z pomocą SQL Management Studio do szkolnej bazy danych.

2. W tym celu należy nawiązać połączenie poprzez VPN. Uwaga. Podczas logowania przez VPN proszę podać pełen login razem z domeną w formie śxxxx@pjwstk.edu.pl".



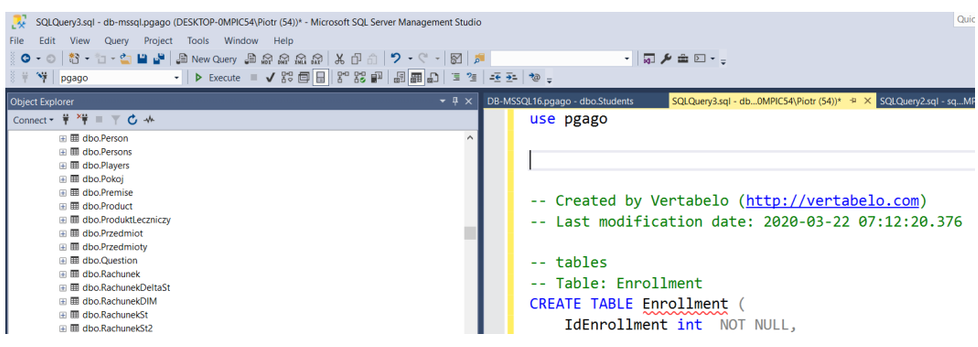
3. Na ekranie logowanie wystarczy podać d̈b-mssqlï pozostawić opcję "Windows Authentication".



4. Osoby, które nie korzystają z SSMS mogą wykorzystać aplikację DataGrip (JetBrains) do połączenia się z bazą danych.

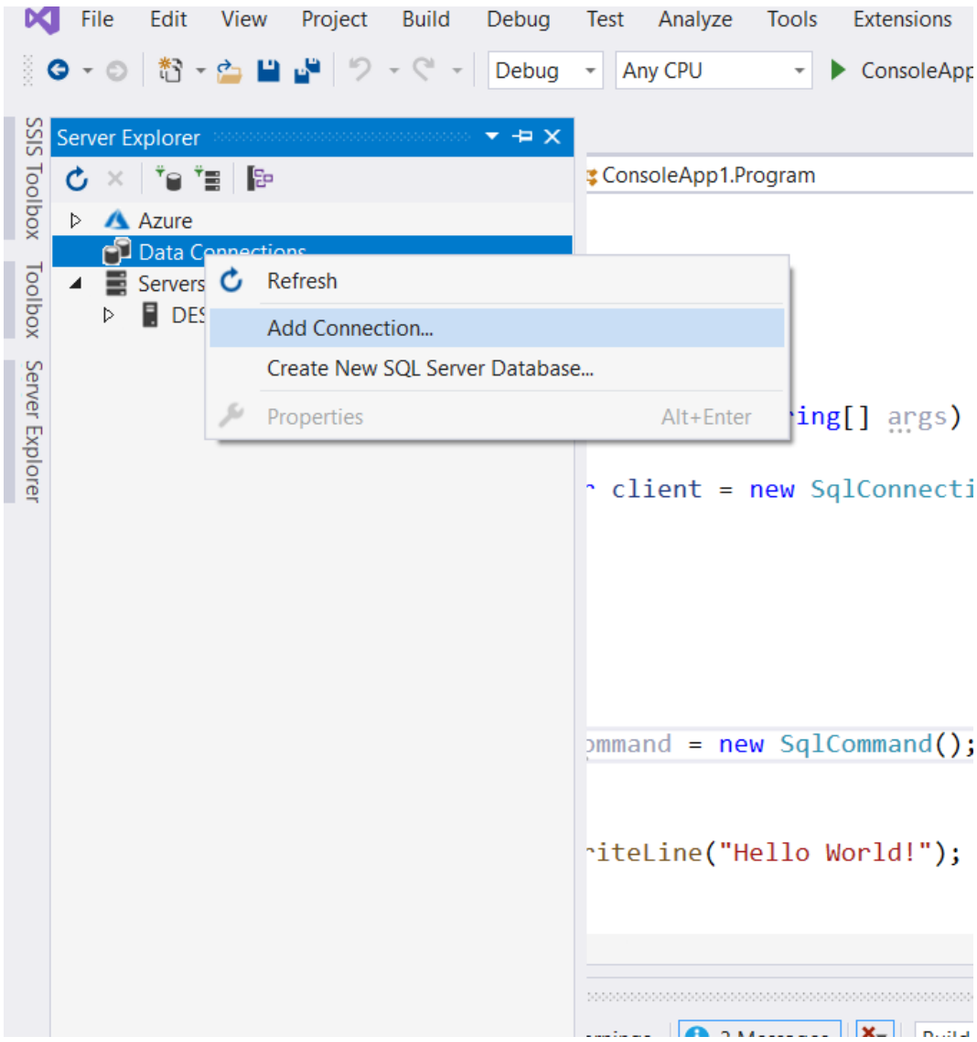
5. Można również wykorzystać lokalną bazę danych SQL Server.

6. Po podłączeniu należy uruchomić skrypt i stworzyć wymagane tabele w bazie danych. Proszę pamiętać o zmianę bazy master na własną.



Zadanie 3 (opcjonalnie) - zapoznanie Server Explorer

Dla osób korzystających z Microsoft Visual Studio warto zapoznać się z oknem Śerver Explorer". Jest to okno, które pozwala nam nawiązać bezpośrednie połączenie z bazą danych z poziomu Visual Studio. Proces logowania do bazy jest identyczny jak w przypadku SSMS.



Zadanie 4.1 - pobieranie danych - dodawanie połączenie

1. W tym zadaniu zmodyfikujemy końcówkę służącą do pobierania danych na temat studentów.

2. W tym celu dodaj do bazy danych dane testowe. Z pomocą SSMS wstawiamy do trzech tabel po kilka rekordów danych.

3. Następnie wracamy do metody odpowiadającej za żądanie Get i zwracającą zamockowaną wcześniej listę studentów. Chcemy rozpocząć komunikację z bazą danych.

4. Skorzystamy z przestrzeni nazw System.Data. W momencie pracy z bazą relacyjną musimy podjąć decyzję czy:

   (a) Czy chcemy użyć sterownika konkretnie zaprojektowanego pod daną bazę danych co pozwoli nam uzyskać lepszą wydajność, jak również skorzystać z mechanizmów, które implementowane są wyłącznie w tej bazie danych? Minusem tego rozwiązania jest oczywiście silniejsze związanie naszego kodu z konkretnym rozwiązaniem bazodanowym.

   (b) Czy chcemy użyć sterownika, który wykorzystuje ogólne standardy (ODBC - Open Database Connectivity), który może łączyć się z różnymi bazami relacyjnymi. Niestety nie będziemy jednak mogli wykorzystać cech specyficznych dla danej bazy danych.

5. Na ten moment skorzystamy ze sterownika specyficznego.

6. Klasa SqlConnection pozwala nam nawiązać połączenie z bazą SQL Server. Ta klasa służy do zarządzania połączeniem z bazą danych.

7. Klasa SqlCommand reprezentuje komendę wysyłaną do bazy danych (zazwyczaj w postaci kodu SQL).

8. Warto zwrócić uwagę na ten podział, który jest widocznym odzwierciedleniem zasady SRP (ang. Single Responsibility Principle).

9. Klasa SqlConnection powinna być w odpowiedni sposób zamknięta. Innymi słowy posiada ona metodę Dispose(), którą powinniśmy wywołać. My do tego celu użyjemy bloku using. Poniżej zaprezentowany jest przykład kodu w metodzie Get.

```
using (var client = new SqlConnection("[CONNECTION STRING]"))
{


}
```

10. Jak widać klasa SqlConnection wymaga podania parametru reprezentującego Ćonnection String". Jest to wartość string reprezentująca połączenie do bazy danych - czyli

wszystkie informacje, które pozwalają nam takie połączenie nawiązać (adres serwera, numer portu, nazwa bazy danych, login, hasło). Przykłady znajdziecie pod poniższym adresem.
https://www.connectionstrings.com/sql-server/

11. W naszym wypadku Connection string powinien mieć następującą postać:

```
Data Source=db-mssql;Initial Catalog=pgago;Integrated Security=True
```

Oczywiście zamiast pgago należy podać swój login (sxxxx).

12. "Data sourceóznacza serwer. "Initial Catalogóznacza bazę danych do której domyślnie uzyskujemy dostęp. "Integrated Securityóznacza wykorzystanie "Windows Authentication"podczas logowania do serwera.

Zadanie 4.2 - pobieranie danych - dodanie obiektu SqlCommand

1. Klasa SqlCommand reprezentuje wyłącznie komendę wysyłaną do bazy danych. Do pracy wymagane są min. dwa parametry: obiekt SqlConnection i komenda SQL.

2. Obiekt ten pozwala na parametryzację wysyłanego żądania co zobaczymy w dalszych lekcjach.

3. Proszę spojrzeć poniżej na wykorzystanie klasy SqlCommand. W miejsce zapytania SQL należy wstawić kod SQL, które połączy dane z tabel Student, WpisNaSemestr i Studia. Chcemy zwrócić dane w postaci imie, nazwisko, dataUrodzenia, nazwa studiow i numer semestru.

4. Domyślnie obiekt SqlCommand pozwala nam na wysłanie żądania SQL na kilka sposobów - zależnie od tego czy oczekujemy, że serwer wyśle nam jakieś dane zwrotne - m.in. ExecuteReader() vs ExecuteNonQuery(). Pierwsza metoda służy zazwyczaj do pracy z zapytaniem SELECT. Metoda ExecuteNonQuery() służy do wysyłania zapytań na które nie oczekujemy odpowiedzi.

5. W tym wypadku oczekujemy danych zwrotnych, więc wykorzystamy metodę ExecuteReader(). Wywołanie tej metody otwiera nam aktywne połączenie do bazy danych. Jako obiekt zwrotny otrzymujemy kolekcję SqlDataReader.

6. Ten element jest bardzo ważny. Jest to kolekcja typu "forward-only", gdzie możemy przeglądać zwracane dane rekord-po-rekordzie. Jest tak dlatego, że nigdy nie wiemy jak dużo danych baza nam prześle. Z tego powodu w momencie wykonywania zapytania SQL otwierane jest aktywne połączenie, gdzie fragment-po-fragmencie przesyłane są poprzez aktywne połączenie. W razie kiedy aplikacja serwerowa nie jest w stanie obsługiwać ich w odpowiednim czasie - wtedy są buforowane po stronie serwera.

7. W naszym wypadku wykorzystamy pętlę while, gdzie sukcesywnie będziemy wywoływać metodę Read(). Metoda pozwala nam na wczytanie kolejnego rekordu, którego obsługą możemy się zająć. W naszym wypadku będziemy je zamieniać na kolekcję obiektów Student, które zostaną zwrócone z końcówki.

8. Poniżej znajduje się przykład parsowania danych. Jak widać wykorzystujemy do tego nazwy kolumn. Można również wykorzystać indeks numeryczny, ale jest to kiepska praktyka ze względu na to, że kolejność kolumn w zapytanie częściej ulega zmianie niż ich nazwa.

9. Zmodyfikuj odpowiednio klasę student, tak aby pozwalała na zapisanie i przekazanie wszystkich niezbędnych informacji.

10. Następnie zwróć je z kontrolera i sprawdź rezultat z pomocą Postmana

```
using (var con = new SqlConnection("Data Source=db-mssql;Initial Catalog=pgago;Integrated Security=True"))
using(var com=new SqlCommand())
{
    com.Connection = con;
    com.CommandText = "select * from Students .....";

    con.Open();
    var dr = com.ExecuteReader();
    while (dr.Read())
    {
        var st = new Student();
        st.FirstName = dr["FirstName"].ToString();
        //...
    }

}
```

Zadanie 4.3 - pobieranie danych - końcówka zwracające wpisy na semestr

1. W tym zadaniu proszę samodzielnie dodać końcówkę służącą do zwracania wpisów na semestr danego studenta.
2. W tym celu należy przekazać parametry do metody Get w postaci numer id studenta
3. Następnie musicie przygotować odpowiednie zapytanie SQL, które pozwoli na zwrócenie tylko wpisów na semestr danego studenta. Można zastosować tutaj konkatenację tak, aby uzyskać odpowiednie zapytanie SQL z wykorzystaniem warunku WHERE.
4. Z pomocą Postmana sprawdźcie czy możecie uzyskać z końcówki niezbędne dane.

Zadanie 4.4 - pobieranie danych - SQLInjection

1. W tym zadaniu spróbujemy wykorzystać podatność naszej aplikacji na wstrzyknięcia kodu SQL.
2. Identyfikatorem studenta jest numer indeksu, czyli wartość typu string. Zastanów się czy jesteś w stanie przekazać taką wartość parametru, aby aplikacja webowa zachowała się w sposób niepoprawny. Np. wykorzystując swoją znajomość kodu T-SQL spróbuj usunąć tabelę Students.
3. Tego typu podatności nazywane są ŚQL Injection", czyli atak poprzez wstrzyknięcie kodu SQL. Jest to jeden z najczęściej stosowanych ataków względem aplikacji webowych wykorzystujących relacyjne bazy danych.

Zadanie 4.5 - poprawka naszej metody Get

1. W tym zadaniu spróbujemy się zabezpieczyć przed atakiem typu SQL Injection.
2. Wprowadź do swojego kodu parametr. Parametry wprowadzane są do kodu SQL poprzez użycie znaku @.
3. Następnie z pomocą obiektu SqlCommand możemy uzupełnić parametry określonymi wartościami. W przykładzie poniżej uzupełniamy parametr o nazwie id.
4. Spróbuj wprowadzić przedstawioną modyfikację w kodzie i ponownie przeprowadzić atak polegający na wstrzyknięciu kodu SQL.

```
string id = "s1234";

using (var con = new SqlConnection("Data Source=db-mssql;Initial Catalog=pgago;Integrated Security=True"))
using(var com=new SqlCommand())
{
    com.Connection = con;
    com.CommandText = "select * from Students where FirstName=@id";
    com.Parameters.AddWithValue("id", id);
```