APBD - Tutorial 4

pgago@pja.edu.pl

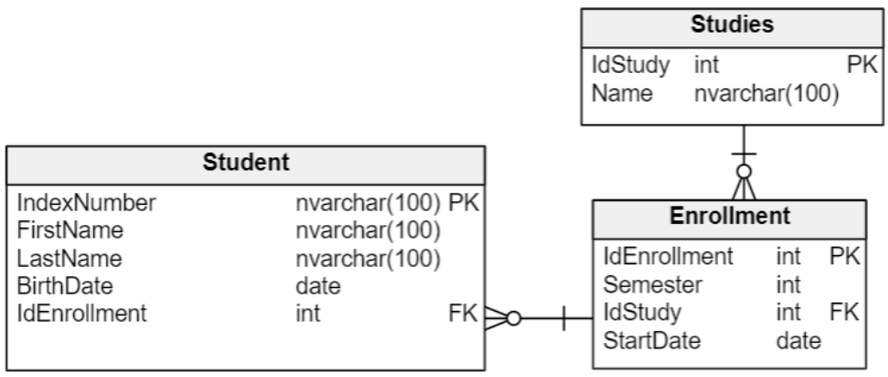
March 27, 2020

1 Introduction

During these exercises we will extend the application written in the tutorial 3. For this reason, you can use the same repository. Please remember to name commits appropriately to separate solutions from the tutorial 3.

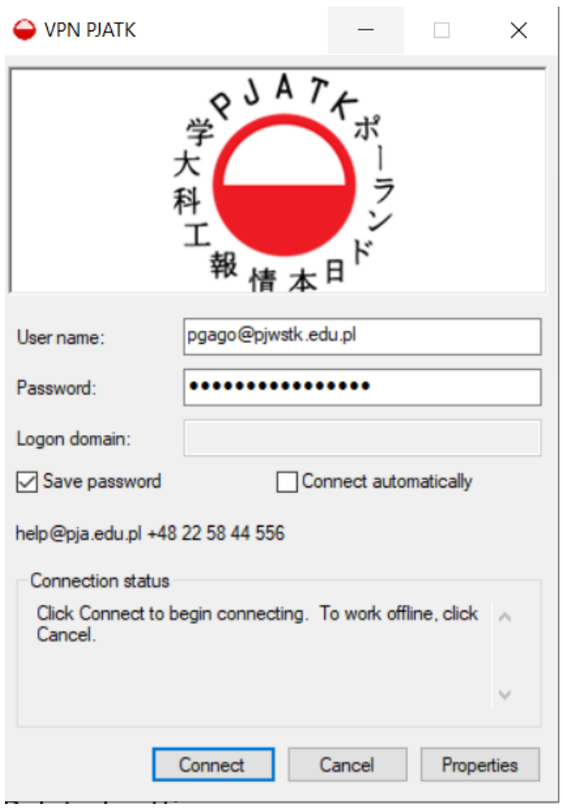
Task 1 – preparing the database

In this task we will prepare the structure of the database, which will be used during the next tasks. Below you can find the entity relational schema of the database. In the materials you can also find a script called "db.sql", which contains the DDL code (in the Transact-SQL dialect), which creates the mentioned database.

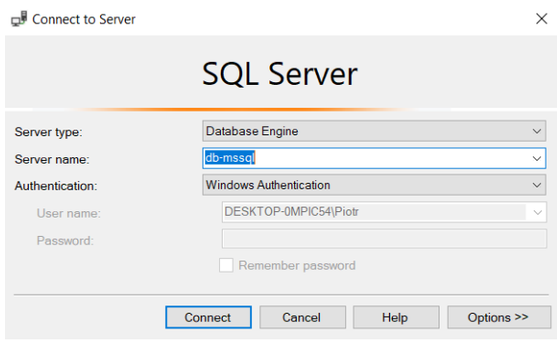


-

1. Please connect to your school database using SQL Management Studio.
2. For this purpose, a VPN connection is required. Attention. When logging in via VPN, please enter your full login together with domain as sxxxx@pjwstk.edu.pl".



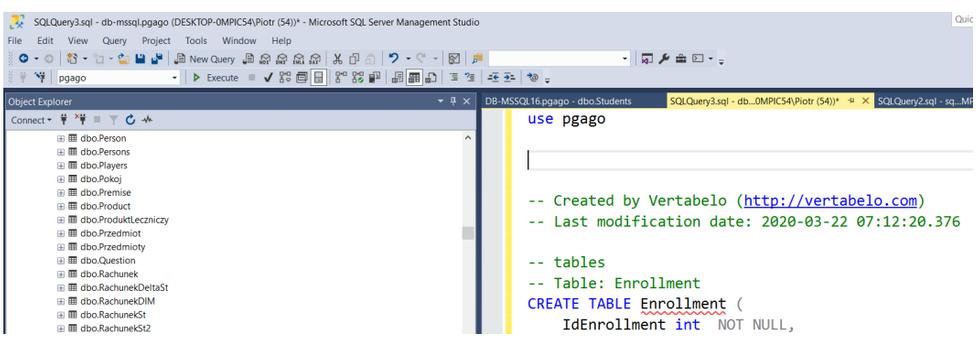
3. On the login screen, just enter the “db-mssql” leave the option "Windows Authentication".



4. People who do not use SSMS may use DataGrip (JetBrains) application to connect to the database.

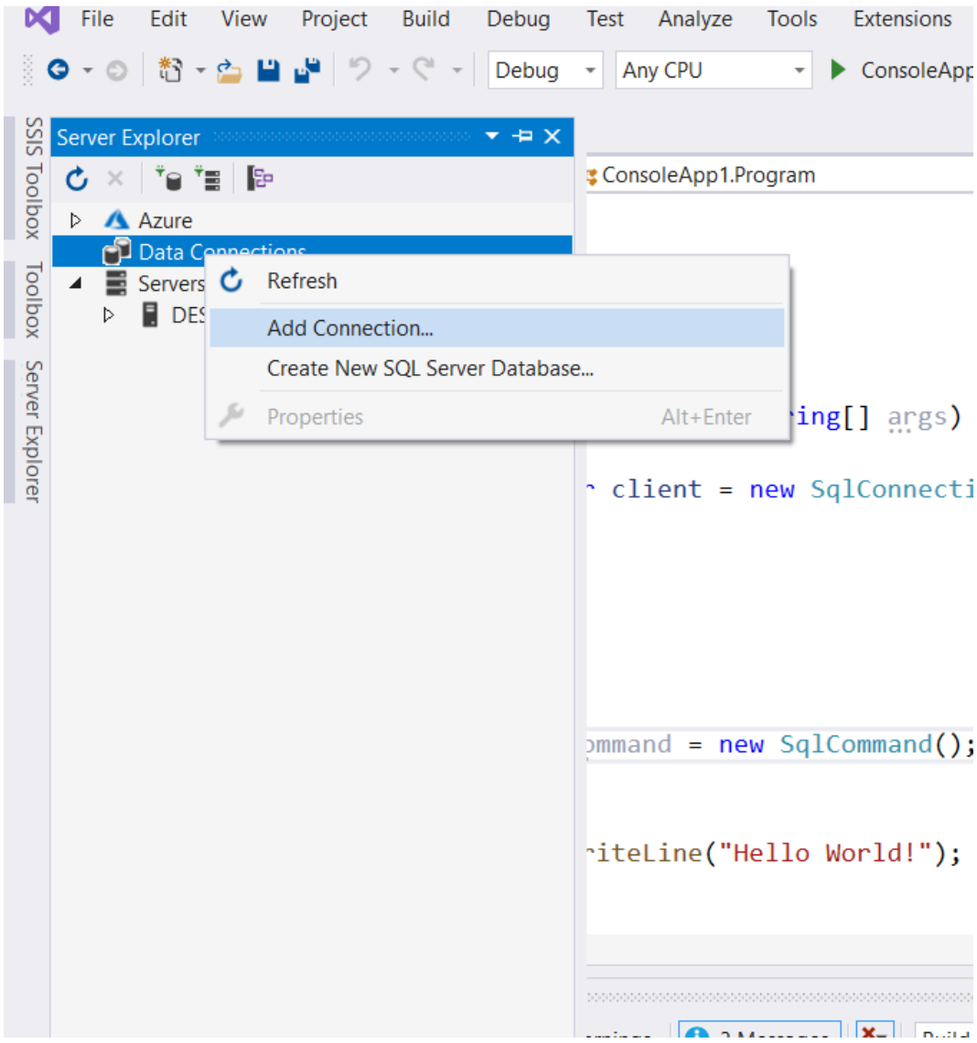
5. You may also use the local SQL Server database.

6. Once connected, run the script and create the required tables in the database. Remember to change the master database to your own.



Task 2 (optional) – introduction to Server Explorer

For students who use Microsoft Visual Studio, it is worthwhile to take a look at the Server Explorer window. This is a window that allows us to make direct connections to the database from the Visual Studio level. The process of logging into the database is identical to one in SSMS.



Task 3.1 – fetching data – adding the connection

1. In this task, we will modify the endpoint for fetching the data about students.
2. To do so, add test data to the database. With the help of SSMS, insert several records into each of three tables.
3. Next, we go back to the method responsible for the Get task which returns the previously mocked list of students. We want to start the communication with the database.
4. We'll use the System.Data namespace. When working with the database, we have to make a decision:
   (a) Do we want to use a driver specifically designed for a given database, which allows to achieve better performance, as well as to use mechanisms that are implemented only in this database? The downside of this solution is of course a tighter link between our code and a specific database solution.
   (b) Do you want to use a driver that uses general standards (ODBC - Open Database Connectivity), which can connect to different relational databases? Unfortunately, we will not be able to use database-specific features.
5. For now, we will use a specific driver (first approach).
6. The SqlConnection class allows us to connect to the SQL Server database. This class is used to manage the connection to the database.
7. The SqlCommand class represents the command which is sent to the database (usually in the form of SQL).
8. It is worth noting this separation, which is a visible reflection of the Single Responsibility Principle (SRP).
9. The SqlConnection class should be properly closed. In other words, it has a Dispose() method should be called. We will use the using block for this. Below there is an example of code in the method Get.

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


}
```

10. As you can see the SqlConnection class requires a parameter representing "Connection String". This is the value of the string representing the connection to the database – that is all the information that allows us to make such a connection (server address, port number, database name, login, password). Examples can be found at the address below.
    https://www.connectionstrings.com/sql-server/

11. In our case, the connection string should have the following form:

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

    Of course, instead of the pgago, please enter your login (sxxxx).

12. "Data Source" means the server. "Initial Catalog" means the database we're going connect to. "Integrated Security" means the use of "Windows Authentication" when logging into the server.

Task 3.2 - fetching data - adding SqlCommand object

1. The SqlCommand class represents only the command sent to the database. At least two parameters are required: SqlConnection object and SQL command.

2. This object allows you to parameterize the sent query as we will see in the next lessons.

3. Take a look at the use of the SqlCommand class below. In the place of SQL query, insert the SQL code that connects data from tables Student, WpisNaSemester and Studia. We want to return the data in the form of name, surname, date of birth, name of studies and semester number.

4. By default, the SqlCommand object allows us to send an SQL query in several ways - depending on whether we expect the server to send us any feedback - ExecuteReader() vs ExecuteNonQuery(). The first method is usually used to work with the SELECT query. The ExecuteNonQuery() method is used to send queries when we do not expect a response.

5. In this case we expect feedback, so we will use the ExecuteReader() method. Calling this method creates an active connection to the database. As a return object we get SqlDataReader collection.

6. This element is very important. This is a forward-only collection, where we can review the returned data record-by-record. This is because we never know how much data the database will send us. For this reason, when executing SQL query, an active connection is opened, where fragments are sent via an active connection one-by-one. If the server application is not able to handle them in time - then they are cached on the server side.

7. In our case we will use while loop, where we will successively call up the Read() method. The method allows us to fetch next record, which we can handle. In our case, we will convert them into the collection of Student objects, which our endpoint will return.

8. Below there is an example of data parsing. As you can see, we use column names for this. You can also use a numerical index, but this is a bad practice because the order of columns in a query changes more often than their name.

9. Modify the Student class accordingly so that all necessary information can be saved and sent.

10. Next return them in the controller and check the result using the Postman.

```
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();
        //...
    }

}
```

Task 3.3 - fetching data – endpoint returns semester entries (WpisNaSemestr)

1. In this task, please add an endpoint which returns semester entries for particular student.
2. For this purpose, the parameters should be transferred to the Get method in the form of the student's id number
3. Next you have to write an appropriate SQL query that will allow you to return only entries for the semester of a given student. You can apply the concatenation here in order to obtain the proper SQL query using the WHERE condition.
4. With the help of Postman, check if you get the necessary data from the endpoint.

Task 3.4 – fetching data – SQLInjection

1. With the help of Postman, check if you get the necessary data from the endpoint.
2. The student's identifier is the index number, which has a value of the type string. Think whether you are able to pass such a value of the parameter that the web application behaves incorrectly. For example, using your knowledge of T-SQL, try to delete the table Student.
3. This type of vulnerability is called SQL Injection, which is an attack by injecting SQL. This is one of the most frequently used attacks on web applications that use relational databases.

Task 3.5 - improvement of our method Get

1. In this task we will try to protect ourselves against SQL Injection attack.
2. Enter a parameter in your code. Parameters are entered into your SQL code using @ symbol.
3. Next, we can use the SqlCommand object to add parameters to your code with specified values. In the example below, we set the parameter named id.
4. Try to make the modifications in the code above mentioned and launch SQL injection attack again.

```
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);
```