NpgSQL

dr Szymon Murawski

February 26, 2020

1 ADO.NET

Database on its own is of little use. The possibility to interact with it from within programming language makes it truly useful.

In C# connecting to database is done by using ADO.NET classes or compatible drivers. ADO.NET can be used to connect not only to various relational database, but also to other kinds of persistent storage like XML files.

Each data provider has the following main objects:

- **Connection** - Establishes a connection to a specific data source
- **Command** - Executes a command against a data source
- **DataReader** - Reads a forward-only, read-only stream of data from a data source
- **DataAdapter** - Populates a `DataSet` and resolves updates with the data source
- **Transaction** - Enlists commands in transactions at the data source
- **Parameter** - Defines input, output, and return value parameters for commands and stored procedures
- **Exception** - Returned when an error is encountered at the data source

2 NpgSQL

2.1 Introduction

Connecting to PostgreSQL database is done via NpgSQL driver (`https://www.npgsql.org`). It is free, open source and 100% written in C#.

To start using NpgSQL first add NuGet package `NpgSQL` to your project and then type `using Npgsql` at the top of your project to import the library.

Following the convention of ADO.NET, all names of objects listed in previous section will start with `Npgsql` followed by object name. For example: `NpgsqlConnection`, `NpgsqlException`.

2.2 Sample application

```
1  using System;
2  using Npgsql;
3
4  class Program
5  {
6    static void Main(string[] args)
7    {
8      // Specify connection options and open an connection
9      NpgsqlConnection conn = new NpgsqlConnection("Server
       =127.0.0.1;User Id=postgres;Password=pwd;Database=postgres;");
10     conn.Open();
11
12     // Define a query
13     NpgsqlCommand cmd = new NpgsqlCommand("SELECT title FROM
       movies", conn);
14
15     // Execute a query
16     NpgsqlDataReader dataReader = cmd.ExecuteReader();
17
18     // Read all rows and output the first column in each row
19     while (dataReader.Read())
20       Console.Write($"title: {dataReader[0]}\n");
21
22     // Close connection
23     conn.Close();
24   }
25 }
```

3 Accessing the database

3.1 Connection and connection string

While connecting to database, first step is to always create a new connection object and then open this connection. Connection object `NpgsqlConnection` requires passing a **connection string** as its argument. Connection string is, as name implies, piece of text describing the details of requested connection.

Different databases/drivers use different forms of connection string, but in the case of NpgSQL it has the following form:

```
Server=127.0.0.1;Port=5432;Database=myDataBase;User Id=user;Password=pass;
```

This connection string can be then passed to NpgsqlConnection constructor and in return we will get **NpgsqlConnection** object, on which we can call `Open()` method. After we are done manipulating the database `Close()` method should be used to close connection to the database.

3.2 SQL command

To query and interact with database a command object must be created: `NpgsqlCommand`. Constructor for this object requires command string and `NpgsqlConnection` object:

`NpgsqlCommand cmd = new NpgsqlCommand("select title from movies", conn);`

Command string can be anything: SELECT statement, DML statement (INSERT, UPDATE, DELETE), DDL statement (CREATE, DROP, ALTER) or any other. Keep in mind, that we are still not executing the query, just a command object was created.

3.3 Executing command

Once we have a command object we can execute it. There are two main ways a command can be executed - either returning data (SELECT statements) or just manipulating data or schema of database (rest of the commands).

A non-select statement can be executed by executing `ExecuteNonQuery` method of command object:

```
1  using System;
2  using Npgsql;
3
4  class Program
5  {
6    static void Main(string[] args)
7    {
8      NpgsqlConnection conn = new NpgsqlConnection("Server
       =127.0.0.1;User Id=postgres;Password=pwd;Database=postgres;");
9      conn.Open();
10     NpgsqlCommand cmd = new NpgsqlCommand("DELETE FROM movies
       WHERE year < 1950", conn);
11     cmd.ExecuteNonQuery();
12     conn.Close();
13   }
14 }
```

3.4 Reading data

To read data from database we need to use data reader object: `NpgsqlDataReader`. This object is returned when a `ExecuteReader` method of command object is invoked.

Data reader returns data via a sequential stream. To read this data, you must pull data from a table row-by-row Once a row has been read, the previous row is no longer available. To read that row again, you would have to create a new instance of the `NpgsqlDataReader` and read through the data stream again.

```
1  using System;
2  using Npgsql;
3
4  class Program
5  {
6    static void Main(string[] args)
7    {
8      NpgsqlConnection conn = new NpgsqlConnection("Server
       =127.0.0.1;User Id=postgres;Password=pwd;Database=postgres;");
```

```
9       conn.Open();
10      NpgsqlCommand cmd = new NpgsqlCommand("SELECT title, year
        FROM movies", conn);
11      NpgsqlDataReader dataReader = cmd.ExecuteReader();
12      while (dataReader.Read())
13        Console.Write($"Movie {dataReader[0]} was produced in {
        dataReader[1]}\n");
14      conn.Close();
15    }
16  }
```

The return value of `Read` is type bool and returns true as long as there are more records to read. This allows us to use it in `while` loop. After the last record in the data stream has been read, Read returns false.

There are two ways of accessing column values. First is the index notation: `dataReader[index]`, where index is an integer. This is not very readable, so there is another way, string index notation `dataReader[stringIndex]`, where `stringIndex` is column name from SQL query. Above example using string index would then look as follow:

```
1 NpgsqlCommand cmd = new NpgsqlCommand("SELECT title, year FROM
      movies", conn);
2 NpgsqlDataReader dataReader = cmd.ExecuteReader();
3 while (dataReader.Read())
4   Console.Write($"Movie {dataReader["title"]} was produced in {
      dataReader["year"]}\n");}
```

3.5 Parametrization

Suppose we want to execute query with some external parameter, like display only movies from year of users choice. A VERY BAD way of doing it would be something like this:

```
1 string year = Console.ReadLine();
2 NpgsqlCommand cmd = new NpgsqlCommand($"SELECT title, year FROM
      movies WHERE year = {year}", conn);
3 NpgsqlDataReader dataReader = cmd.ExecuteReader();
4 Console.WriteLine($"Movies produced in {year}");
5 while (dataReader.Read())
6   Console.Write($"Title: {dataReader["title"]}\n");}
```

Above example leads to potentially catastrophic results, as anything that the user inputs will be pasted into the query. A clever user might prepare a harmful string, that reads the content of whole database or even destroys it!

A proper way of passing parameters into the query is to use SQL command parameters functionality:

```
1 string year = Console.ReadLine();
2 NpgsqlCommand cmd = new NpgsqlCommand($"SELECT title, year FROM
      movies WHERE year = @year", conn);
3 cmd.Parameters.AddWithValue("@year", year);
4 NpgsqlDataReader dataReader = cmd.ExecuteReader();
5 Console.WriteLine($"Movies produced in {year}");
6 while (dataReader.Read())
7   Console.Write($"Title: {dataReader["title"]}\n");}
```

In the command string we put *placeholders* in places where a value should be entered, i.e. `@year`. Later on we add parameters values into the command using `AddWithValue` method. This has the following advantages over embedding the value directly in SQL:

- Avoid SQL injection for user-provided inputs: the parameter data is sent separately from the SQL, and is never interpreted as SQL.
- Required to make use of prepared statements, which dramatically improve performance if you execute the same SQL many times.
- Parameter data is sent in an efficient, binary format, rather than being represented as a string in SQL.

Note that PostgreSQL does not support parameters in arbitrary locations - you can only parameterize data values. For example, trying to parameterize a table or column name will fail - parameters aren't a simple way to stick an arbitrary string in your SQL.

3.6 Transactions

Atomicity of SQL is guaranteed by transactions. Multiple commands can be a part of a single transaction and either all of them successfully finish, or transaction will be rolled back.

In NpgSQL transactions are created using `BeginTransaction` method of `NpgsqlConnection` object. After that transaction must be assigned to `NpgsqlCommand` object. From now executing commands won't have effect on the database, until `transaction.Commit()` is invoked. We can rollback the transaction using `transaction.Rollback()` method.

```
1  using (NpgsqlConnection connection = new NpgsqlConnection(
       connectionString))
2  {
3          connection.Open();
4          NpgsqlCommand command = connection.CreateCommand();
5          NpgsqlTransaction transaction = connection.
       BeginTransaction();
6          command.Connection = connection;
7          command.Transaction = transaction;
8          try
9          {
10             command.CommandText =
11                 "INSERT INTO movies(movie_id, title, year) VALUES
        (102, 'Ace Ventura 2', 1995)";
12             command.ExecuteNonQuery();
13             command.CommandText =
14                 "INSERT INTO movies(movie_id, title, year) VALUES
        (101, 'Ace Ventura', 1994)";
15             command.ExecuteNonQuery();
16
17             // Attempt to commit the transaction.
18             transaction.Commit();
```

```
19              Console.WriteLine("Both records are written to
    database.");
20          }
21          catch (Exception ex)
22          {
23            Console.WriteLine("Error occured, rolling back
    transaction");
24              transaction.Rollback();
25          }
26  }
```

3.7 Exercise

- Create a program, that takes `movie_id` from the user and displays details about the movie
- Also display all actors that are starring in the movie
- Also display all the copies of the movie
- Also display status of those copies - are they rented and if so to who?
- Add an option for the user to create a new movie