Przykład pokazujący jak tworzyć aplikacje współpracującą z baza danych w trybie bezpołączeniowym tylko przy użyciu kodowania (bez kreatorów designu)

Klasy VB.NET do obsługi i komunikacji z różnymi bazami danych - przypomnienie teorii

VB.NET udostępnia wiele wbudowanych klas pozwalających na komunikację i obsługę baz danych. Niektóre tych z klas są specyficzne dla dostawcy (klasy dostawców), tzn. stanowią interfejs dla określonego typu systemu bazodanowego. Zadaniem ich jest połączenie z bazą danych, wykonywanie poleceń i odczytywanie danych. VB.NET wspomaga cztery odmiany dostawców baz danych: Microsoft SQL Server, OLE DB, ODBC oraz Oracle. W przykładzie omówiana jest współpracę z MS SQL Server.

W celach edukacyjnych nie podałem bezpośredniego kodu współpracy z baza typu Acess. Proszę wykorzystać informacje z quzi- wykładu oraz innych dostarczonych materiałów i przerobić kod do współpracy z bazą typu Access.

Jeszcze raz podkreślam, że praca z innymi dostawcami będzie bardzo podobna, np. dla programistów wykorzystujących silnik Microsoft JET (czyli dane są zapisane w pliku ACCESS-a) przeznaczone będą klasy wspierające OLE DB zorganizowane w ten sam sposób co klasy SQL. Należy pamiętać, że wszystkie klasy wspomagające pracę z bazami danych znajdują się w przestrzeni nazw **System.Data**.

Najważniejsze klasy dostawców w .NET to: **Connection**, **Command**, **DataReader** oraz **DataAdapter**. Ze względu na dostawców umieszczone są one w odpowiednich przestrzeniach nazw, np. **SQLClient** i **OLEDB** oraz zostały odpowiednio nazwane, np. **SQLCommand**.

Klasa dostawcy	Przestrzeń nazw SQLClient	Przestrzeń nazw OleDb
Connection	SQLConection	OleDbConnection
Command	SQLCommand	OleDbCommand
DataReader	SQLDataReader	OleDbDataReader
DataAdapter	SQLDataAdapter	OleDbDataAdapter

Klasa **Connection** powala nawiązać lub zakończyć połączenie z bazą danych. Do wykonania tej operacji jest potrzebny tzw. łańcuch połączeniowy (**ConnectionString**), czyli tekst zawierający najważniejsze informacje, pozwalające na połączenie z wybraną bazą danych udostępnioną przez wskazanego dostawcę. Łańcuch połączeniowy zawiera informacje w postaci par – klucz-wartość, specyficznych dla serwera baz danych, systemu zabezpieczeń. Przykładowe łańcuchy połączeń:

- **dla bazy obsługiwanej przez system SQL, gdzie system zabezpieczeń jest monitorowany przez serwer SQL (uwierzytelnianie SQL):** `Password=ala;Persist Security Info=True;User ID=sa;Initial Catalog=ksiazaki;Data Source=STACJA18`

- **dla bazy opartej na systemie Access (dostawca OleDb):**

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\AS.mdb;Persist Security Info=False
```

Połączenie z bazą polega na utworzeniu nowego obiektu klasy **System.Data.SqlClient.SqlConnection** zainicjowanego łańcuchem połączeniowym, a następnie wywołaniu metody **Open** dla tego obiektu, np.:

```
Dim strPolacz As String
```

```
strPolacz = "Password=ala;Persist Security Info=True;User ID=sa;Initial
Catalog=ksiazaki;Data Source=STACJA18"
objPolaczenie = New System.Data.SqlClient.SqlConnection(strPolacz)
objPolaczenie.Open()
```

Zamkniecie połączenia z bazą polega na wywołaniu metody **Close,** np.:

```
objPolaczenie.Close()
```

W trakcie działania aplikacji można sprawdzić, czy jest połączenie, wywołując właściwość **State,** np.:

```
If objPolaczenie.State = ConnectionState.Open Then
      objPolaczenie.Close()
End If
```

Fragment kodu powyżej sprawdza, czy połączenie z bazą jest otwarte; w przypadku pozytywnej odpowiedzi zamyka zbędne już połączenie (należy pamiętać, że próba zamknięcia połączenia, gdy zostało ono już wcześniej zakończone, spowoduje wystąpienie wyjątku).

Klasa **Command** pozwala wykonywać instrukcje SQL wraz z parametrami. Wykorzystujemy ją w celu, np. utworzenia bazy, tabeli, wpisu lub aktualizacji danych, wyszukania informacji z wykorzystaniem języka SQL.
Do wykonania zapytań wykorzystujemy 3 metody **ExecuteReader**, **ExecuteScalar** oraz **ExecuteNonQuery**. Działanie tych metod jest następujące:

- **ExecuteReader** –zwraca obiekt typu **DataReader** zawierający, np. zestaw rekordów będący wynikiem zapytania.
- **ExecuteScalar** – zwraca zawartość pierwszej kolumny z pierwszego wiersza zestawu wyników, otrzymanego w wyniku zapytania.
- **ExecuteNonQuery** wykonuje instrukcję SQL i zwraca ilość zmodyfikowanych wierszy.

Przykładowe wykorzystanie klasy **Command** do usunięcia rekordu o wskazanym identyfikatorze może wyglądać następująco:

```
      strSQL = "delete from ksiazki" & " where id_k=3"
```

Utworzenie zapytania SQL.

```
Dim objZapytanie As New System.Data.SqlClient.SqlCommand(strSQL,
objPolaczenie)
```

Utworzenie nowego obiektu typu **System.Data.SqlClient.SqlCommand** (klasa **Command** dla dostawcy SQL) i zainicjowanie go zapytaniem SQL oraz obiektem klasy **Connection** (zawierającym informacje o połączeniu z bazą danych).

```
objZapytanie.ExecuteNonQuery()
```

Wykonanie zapytania SQL.

Klasa **DataReader** pozwala na tworzenie obiektów odczytujących strumień danych otrzymanych w wyniku zapytania. Obiekt typu **DataReader** posiada metody pozwalające na odczytanie kolejnych rekordów z bazy (tylko w przód) i nie ma możliwości modyfikacji danych w bazie. Przykładowe wykorzystanie klasy **DataReader** może wyglądać następująco:

```
strSQL = "delete from ksiazki where id_k=3"
Dim objZapytanie As New System.Data.SqlClient.SqlCommand(strSQL,
objPolaczenie)
```

Utworzenie zapytania SQL oraz nowego obiektu klasy SQLCommand.

```
Dim objOdczyt As System.Data.SqlClient.SqlDataReader
```

Utworzenie nowego obiektu **DataReader**.

```
objOdczyt= objZapytanie.ExecuteReader()
```

Wywołanie metody **ExecuteReader** (wykonanie zapytania SQL) i podstawienie wyniku wyszukiwań do obiektu typu **DataReader**.

```
Do While objOdczyt.Read
    objOdczyt.GetInt32(0)
    objOdczyt.GetString(1)
Loop
```

Pobranie w pętli kolejnych rekordów i odczytywanie pierwszych dwóch pól bazy (o podanych typach).

Klasa **DataAdapter** pozwala utworzyć obiekt pełniący kilka zadań:

- pobiera i przechowuje podzbiór danych uzyskany w wyniku zapytania i zapisuje go w obiekcie typu **DataSet**.
- umożliwia wykonywanie operacji na danych
- umożliwia przesłanie zaktualizowanych danych do bazy

Należy pamiętać, że obiekt **DataAdapter** pozwala na bezpołączeniowe operowanie na danych. Oznacza to, że dane pobrane z bazy są umieszczane w pamięci komputera i tam można wykonywać na nich różne operacje. Obiekt ten pozwala zwrócić informacje do bazy, dokonując jednocześnie ich aktualizacji. Do pobierania danych i umieszczania ich w obiekcie typu **DataSet** służy metoda **Fill**. Aktualizacja danych zapisanych w pamięci do bazy odbywa się za pomocą metody **Update**.

Klasa **DataSet** pozwala tworzyć obiekty przechowujące dane w pamięci. Mogą one zawierać obiekty typu **DataTable** będące odpowiednikami danych z tabelami oraz pozwalają na definiowanie relacji za pomocą metody **DataRelations**.

Klasa **DataTable** jest odpowiednikiem tabeli bazy danych, z tą różnicą, że dane przechowywane są w pamięci komputera. W klasie tej możemy wyróżnić kilka kolekcji pozwalających na uzyskanie informacji o danych, z których najważniejsze są **DataRow** i **DataColumn**. Pierwsza z nich pozwala na uzyskanie pojedynczego wiesza z obiektu **DataTable**, druga natomiast reprezentuje kolumnę w **DataTable**.

Klasa **DataView** pozwala na sortowanie i filtrowanie danych umieszczonych w obiekcie typu **dataTable**.

Przykład wykorzystania klasy **DataAdapter** może wyglądać następująco:

```
Dim objTabela As DataTable
Dim objWiersz As DataRow
```

Deklaracja nowych obiektów typu **DataTable** i **DataRow**.

```
Dim strSQL As String
strSQL = "select * from ksiazki"
```

Utworzenie ciągu znaków, stanowiącego zapytanie SQL.

```
Dim objZapytanie As New System.Data.SqlClient.SqlDataAdapter(strSQL,
objPolaczenie)
```

Utworzenie nowego obiektu klasy **DataAdapter** i zainicjowanie go zapytaniem SQL oraz obiektem typu **Connection**)

```
objDane = New DataSet("ksiazki")
```

Utworzenie nowego obiektu **DataSet** o nazwie **ksiazki**.

```
objZapytanie.Fill(objDane, "ksiazki")
```

Wypełnienie obiektu typu **DataSet** danymi, uzyskanymi w wyniku wykonania zapytania.

```
objTabela = objDane.Tables("ksiazki")
```

Podstawienie pod obiekt typu **DataTable** tabeli **ksiazki,** znajdującej się w obiekcie typu **DataSet**.

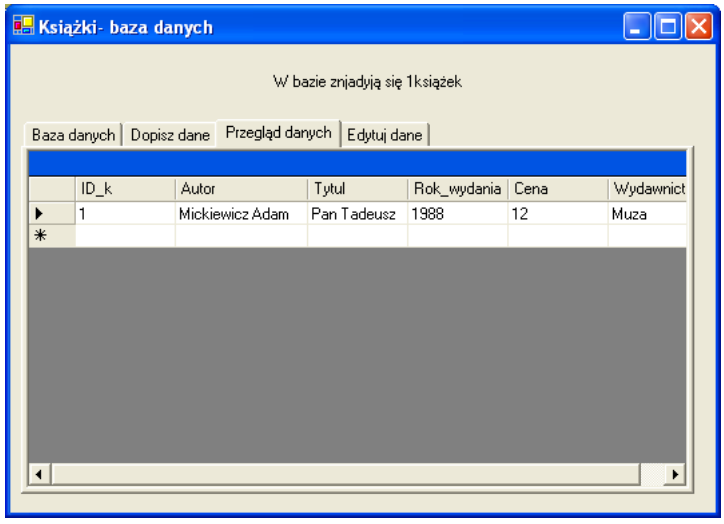
```
objWiersz = objTabela.Rows(wiersz)
```

Pobranie jednego wiersza tabeli danych z obiektu typu **DataTable** i podstawienie go do obiektu typu **DataRow** (parametr wiersza określa numer rekordu z zestawu danych, zapisanych w pamięci).

```
txtEID_k.Text = objWiersz.Item(0)
txtEAutor.Text = objWiersz.Item(1)
txtETytul.Text = objWiersz.Item(2)
```

Odczyt kolejnych pól rekordu i umieszczenie pobranych danych w kontrolach typu **TextBox**.

Kontrolka **DataGrid** pozwala wyświetlać dane w postaci tabeli umieszczonej na formularzu. Jest to najprostszy sposób wyświetlenia wyników zapytania w postaci tabelarycznej. Np. tak jak poniżej na rysunku.



Do wypełnienia kontrolki **DatGrid** danymi wykorzystuje się wcześniej poznane klasy. Fragment kodu realizujący to zadanie może wyglądać następująco:

```
Dim strSql As String
strSql = "select * from ksiazki"
Dim objZapytanie As New System.Data.SqlClient.SqlDataAdapter(strSql,
objPolaczenie)
Dim objDane As New DataSet("ksiazki")
Dim objDaneWidok As DataView
Dim objTabela As DataTable
```

Utworzenie obiektów potrzebnych do pobrania i przeglądania danych.

```
objZapytanie.Fill(objDane, "ksiazki")
```

```
objTabela = objDane.Tables("ksiazki")
```

Wypełnienie obiektu typu **DataSet** danymi, uzyskanymi w wyniku wykonania zapytania.

```
objDaneWidok = objTabela.DefaultView
```

Utworzenie nowego obiektu typu **DataView** określającego sposób wyświetlania danych.

```
DataGrid1.DataSource = objDane
DataGrid1.DataMember = "ksiazki"
```

Przypisanie do kontrolki **DataGrid** zawartość obiektu typu **DataSet** (metoda **DataSource**). Ponieważ obiekt klasy **DataSet** może zawierać wiele tabel wskazanie poprzez właściwość **DataMember**, która z tabel będzie wyświetlana w kontrolce.

Prosta aplikacja bazodanowa Książki oparta na ADO.NET - przykład użycia

Stwórzmy prosta aplikacja bazodanowa pozwalająca na przeglądanie, edycje i wprowadzanie danych do bazy. Baza danych może zostać oparta np. na SQL Server ale również na Access (wymagana jest wtedy zmiana nazw niektórych klas). Interfejs aplikacji będą stanowiły cztery karty kontrolki **TabControl**.

Kodowanie programu rozpoczniemy od utworzenia zmiennych globalnych oraz procedur umożliwiających połączenie z bazą i sprawdzenie ilości rekordów w bazie. W celu wykorzystania klas służących do obsługi baz danych, należy zaimportować przestrzeń nazw **System.Data**.

```
Imports System.Data
```

Deklaracja zmiennych globalnych wygląda następująco:

```
Dim objPolaczenie As System.Data.SqlClient.SqlConnection
```

Deklaracja obiektu typu **SQLConnection**, służącego do połączenia z bazą danych.

```
Dim objDane As New DataSet("ksiazki")
```

Deklaracja obiektu typu **DataSet**, służącego do przechowywania danych będących wynikiem zapytań SQL.

```
Dim nr_rekordu As Integer = 0
```

Deklaracja zmiennej, w której będzie zapamiętany numer aktualnie przeglądanego rekordu.

```
Dim objTabela As DataTable
```

Deklaracja obiektu typu **DataTable**, służącego zapamiętania tabeli przechowywanej w obiekcie typu **DataSet**.

```
Dim objWiersz As DataRow
```

Deklaracja obiektu typu **DataRow**, pobierającego wiersz danych z tabeli przechowywanej w obiekcie typu **DataTable**.

Połączenie z bazą danych zostanie realizowane poprzez procedurę, która będzie wielokrotnie wywoływania w aplikacji.

```
Private Sub polacz_z_baza()
Dim strPolacz As String
```

Deklaracja zmiennej typu **String**, w której zostanie zapamiętany łańcuch połączeniowy.

```
strPolacz = "Password=janusz;Persist Security Info=True;User ID=sa;Initial
Catalog=ksiazki;Data Source=(local)"
```
Ustalenie wartości łańcucha połączeniowego (zostało to już opisane wcześniej).

```
Try
```
Rozpoczęcie bloku kodu chronionego.

```
   objPolaczenie = New System.Data.SqlClient.SqlConnection(strPolacz)
```
Utworzenie nowego obiektu klasy **SQLConnection**.

```
   objPolaczenie.Open()
```
Otwarcie połączenia z bazą.

```
Catch ex As Exception
   MessageBox.Show("Blad w połączeniu z bazą danych", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
End Sub
```
Obsługa ewentualnych wyjątków, mogących wystąpić przy połączeniu z bazą.

Aplikacja będzie informowała użytkownika o ilości rekordów w bazie. Informacja ta będzie także wykorzystywana podczas obsługi karty **Edytuj dane**. Funkcja **ile_rekordow** wykorzystuje poznane klasy do obliczenia ilości rekordów w tabeli **ksiazki**.

```
Private Function ile_rekordow() As Integer
Dim strSQL As String
objDane.Clear()
```
Deklaracja zmiennej typy **String** przechowującej zapytanie SQL oraz wyczyszczenie obiektu typu **DataSet**.

```
strSQL = "select * from ksiazki"
```
Zapamiętanie zapytania SQL, które będzie wykonane w funkcji.

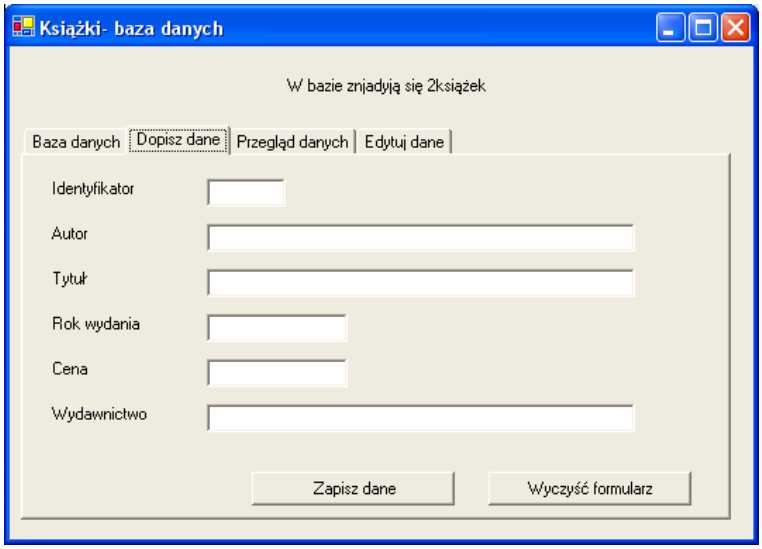
```
Dim objZapytanie As New System.Data.SqlClient.SqlDataAdapter(strSQL,
objPolaczenie)
```
Utworzenie nowego obiektu typu **DataAdapter**.

```
If objPolaczenie.State = ConnectionState.Open Then
  Try
```
Sprawdzenie, czy jest połączenie z bazą i rozpoczęcie bloku kodu chronionego.

```
    objZapytanie.Fill(objDane, "ksiazki")
```
Wypełnienie obiektu typu **DataSet** (**objDane**) wynikiem zapytania SQL, wykonanego na tabeli **ksiazki**.

```
    Return objDane.Tables("ksiazki").Rows.Count
```
Zwrócenie ilości rekordów w bazie (dokładnie ilości wierszy zapamiętanych w obiekcie typu **DataSet** w tabeli **ksiazki**.

```
  Catch ex As DataException
    MessageBox.Show("Odczyt z bazy nie  jest możliwy", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  End Try
End If
End Function
```
Obsługa ewentualnych wyjątków, mogących wystąpić przy połączeniu z bazą i zakończenie kodu funkcji.



Zakładka **Dopisz dane** umożliwia wprowadzanie nowych danych do tabeli **ksiazki**.
W programie dane są wprowadzane do pól tekstowych, następnie sprawdzana jest ich poprawność. Wpis danych jest realizowany poprzez wykonanie zapytania SQL, służącego do wprowadzania danych. Procedura realizująca to zadanie może wyglądać następująco:

```
Private Sub btnWpis_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnWpis.Click
Dim strWpis As String
If txtID_k.Text.Length = 0 OrElse Not IsNumeric(txtID_k.Text) Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  txtID_k.Focus()
  Exit Sub
End If
```

Sprawdzenie poprawności wprowadzonych danych do pola tekstowego. W przypadku błędnych danych (pole puste lub niewłaściwy typ danych, np. tekst) działanie procedury jest przerywane a kursor jest ustawiany w polu z błędem.

```
If txtAutor.Text.Length = 0 Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  txtAutor.Focus()
  Exit Sub
End If
If txtTytul.Text.Length = 0 Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  txtTytul.Focus()
  Exit Sub
End If
If txtRok_wydania.Text.Length = 0 OrElse Not IsNumeric(txtRok_wydania.Text)
Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  txtRok_wydania.Focus()
  Exit Sub
End If
If txtCena.Text.Length = 0 OrElse Not IsNumeric(txtCena.Text) Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
```

```
  txtCena.Focus()
  Exit Sub
End If
If txtWydawnictwo.Text.Length = 0 Then
  MessageBox.Show("Proszę uzupełnić brakujace dane", "Brak danych",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  txtWydawnictwo.Focus()
  Exit Sub
End If
```

Sprawdzenie pozostałych pól formularza.

```
strWpis = "insert into ksiazki(id_k,autor,tytul,rok_wydania," & _
"cena,wydawnictwo) values (" & txtID_k.Text & "," & "'" & txtAutor.Text &
"'" & "," &  "'" & txtTytul.Text & "'" & "," & txtRok_wydania.Text & "," &
txtCena.Text & "," & "'" & txtWydawnictwo.Text & "'" & ")"
```

Utworzenie zapytania SQL służącego do wprowadzania danych.

```
polacz_z_baza()
```

Połączenie z bazą.

```
Dim objZapytanie As New System.Data.SqlClient.SqlCommand(strWpis,
objPolaczenie)
```

Utworzenie nowego obiektu typu **SQLCommand**.

```
If objPolaczenie.State = ConnectionState.Open Then
  Try
  objZapytanie.ExecuteNonQuery()
```

Sprawdzenie, czy jest połączenie z bazą i wykonanie w bloku kodu chronionego zapytania poprzez wywołanie metody **ExecuteNonQuery**.

```
lblIleKsiazek.Text = "W bazie znajduje się " & ile_rekordow() & "książek"
```

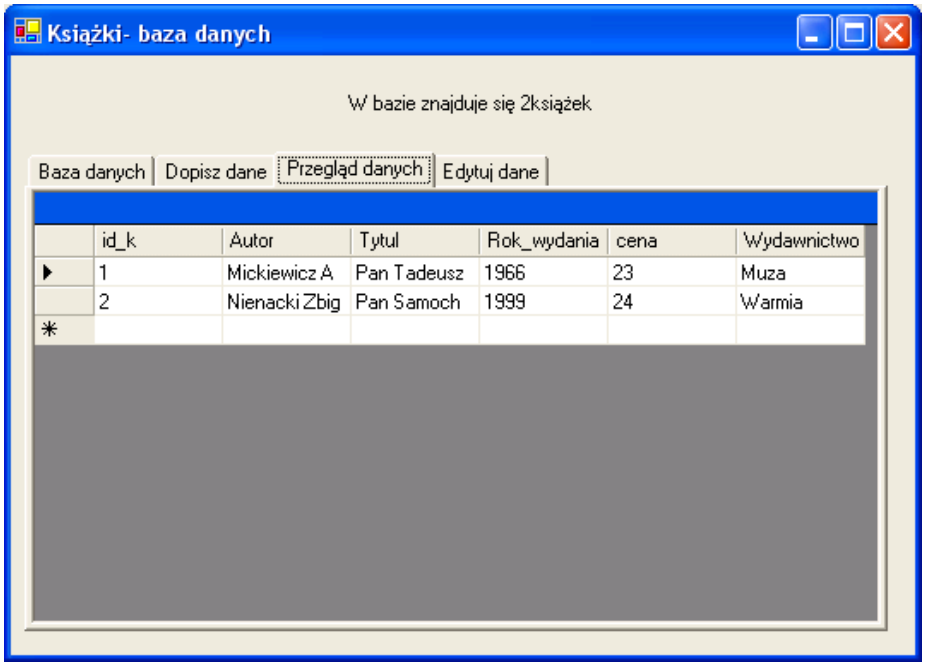
Aktualizacja informacji o ilości rekordów w bazie.

```
Catch ex As Exception
  MessageBox.Show("Bład podczas wpisywania danych", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
Finally
  objPolaczenie.Close()
End Try
```

Obsługa ewentualnych wyjątków mogących wystąpić przy połączeniu z bazą i zakończenie połączenia z bazą w bloku **Finally** (w przypadku wystąpienia wyjątku połączenie na pewno zostanie zamknięte).

```
czysc_wpis()
MessageBox.Show("Dane zostały wpisane do bazy", "Wpis udany",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk)
End If
End Sub
```

Wyczyszczenie pól tekstowym po wprowadzeniu danych oraz poinformowanie użytkownika o przeprowadzonej operacji.



Karta **przegląd danych** wyświetla zwartość bazy w kontrolce typu **DataGrid**. Zawartość kontrolki jest zawsze aktualizowana przy zmianie karty w kontrolce **TabControl**. Wywołanie aktualizacji następuje podczas przejścia na odpowiednią kartę.

```
    Private Sub tclBaza_SelectedIndexChanged(ByVal sender As Object, ByVal
e As System.EventArgs) Handles tclBaza.SelectedIndexChanged
```

Procedura obsługi zdarzenia zmiany aktywnej karty w kontrolce typu **TabControl**..

```
        If tclBaza.SelectedIndex = 1 Then
            czysc_wpis()
        End If
```

W przypadku, gdy jest aktywna karta **Dopisz dane** (indeks 1), zostaje wywoływana procedura czyszczenia pół tekstowych na tej karcie.

```
        If tclBaza.SelectedIndex = 2 Then
            polacz_z_baza()
            przeglad_bazy()
        End If
```

W przypadku, gdy jest aktywna karta **Przegląd danych** (indeks 2), zostaje nawiązywane połączenie z bazą i wywoływana procedura wypełniająca kontrolkę typu **DataGrid** danymi.

```
        If tclBaza.SelectedIndex = 3 Then
            nr_rekordu = 0
            polacz_z_baza()
            odczyt_do_edycji(0)
        End If
    End Sub
```

W przypadku, gdy jest aktywna karta **Edytuj dane** (indeks 3) zostaje nawiązywane połączenie z bazą i wywoływana procedura wyświetlająca pierwszy rekord bazy.

Wyświetlanie danych w kontrolce typu **DataGrid** jest realizowane w następującej procedurze.

```
Private Sub przeglad_bazy()
Dim strSql As String
```

```
strSql = "select * from ksiazki"
```
Utworzenie zapytania SQL.

```
Dim objZapytanie As New System.Data.SqlClient.SqlDataAdapter(strSql,
objPolaczenie)
```
Utworzenie nowego obiektu typu **SQLDataAdapter**.

```
Dim objDane As New DataSet("ksiazki")
```
Utworzenie nowego obiektu typu **DataSet**.

```
Dim objDaneWidok As DataView
```
Utworzenie nowego obiektu typu **DataView**.

```
Dim objTabela As DataTable
```
Utworzenie nowego obiektu typu **DataTable**.

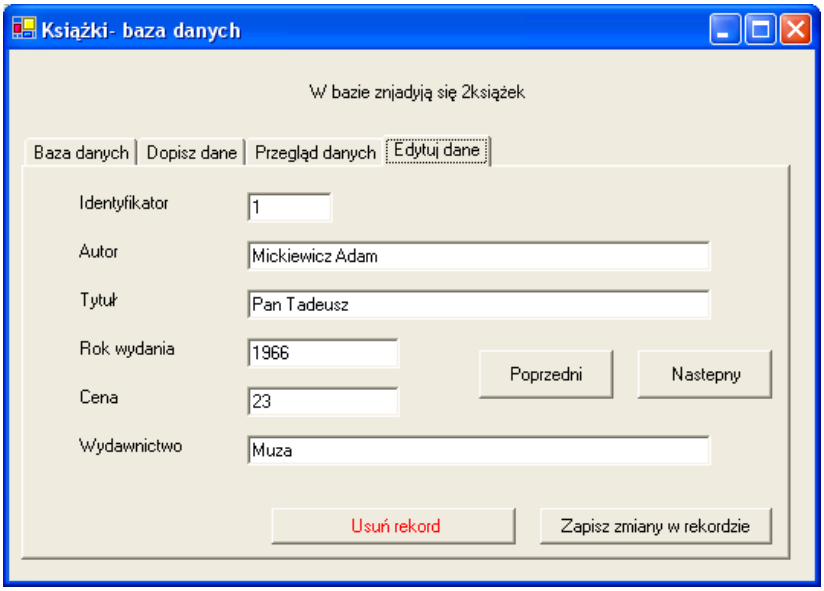
```
If objPolaczenie.State = ConnectionState.Open Then
   Try
```
Sprawdzenie, czy jest połączenie z bazą danych i rozpoczęcie bloku kodu chronionego.

```
        objZapytanie.Fill(objDane, "ksiazki")
```
Wypełnienie obiektu typu **DataSet** danymi będącymi wynikiem zapytania.

```
        objTabela = objDane.Tables("ksiazki")
        objDaneWidok = objTabela.DefaultView
```
Zapamiętanie tabeli **ksiazki** w obiekcie typu **DataTable** i ustawienie sposobu sortowania danych.

```
        DataGrid1.DataSource = objDane
        DataGrid1.DataMember = "ksiazki"
```
Wypełnienie kontrolki typu **DataGrid** danymi.

```
   Catch ex As DataException
       MessageBox.Show("Odczyt z bazy nie  jest możliwy", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
Finally
  objPolaczenie.Close()
  End Try
End If
End Sub
```
Obsługa ewentualnych wyjątków mogących wystąpić przy połączeniu z bazą i zakończenie połączenia z bazą w bloku **Finally** (w przypadku wystąpienia wyjątku połączenie na pewno zostanie zamknięte).



Karta **Edytuj dane** pozwala na przeglądanie pojedynczych rekordów. Użytkownik może zmodyfikować zwartość rekordu i zapisać zmiany oraz usunąć rekord. Funkcje te są realizowane poprzez odpowiednie zapytania SQL. Odczyt danych jest realizowany przez następującą procedurę:

```
Private Sub odczyt_do_edycji(ByVal nr_rekordu As Integer)
Dim strSQL As String
strSQL = "select * from ksiazki"
Dim objZapytanie As New System.Data.SqlClient.SqlDataAdapter(strSQL,
objPolaczenie)
objDane = New DataSet("ksiazki")
If objPolaczenie.State = ConnectionState.Open Then
  Try
   objDane.Clear()
   objZapytanie.Fill(objDane, "ksiazki")
   Catch ex As System.Exception
   MessageBox.Show("Odczyt z bazy nie  jest możliwy", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  Finally
   objPolaczenie.Close()
  End Try
End If
```

Zapamiętanie wszystkich rekordów bazy w obiekcie typu **DataTable**.

```
If objDane.Tables("ksiazki").Rows.Count = 0 Then

  MessageBox.Show("Baza pusta- brak rekordów", "Baza ",
MessageBoxButtons.OK, MessageBoxIcon.Warning)
btnPoprzedni.Enabled = False
btnNastepny.Enabled = False
btnUsunRekord.Enabled = False
btnEZapiszDane.Enabled = False
wyswietl_rekord(-1)
Exit Sub
```

Sprawdzenie ilości rekordów w bazie. W przypadku, gdy baza jest pusta, poinformowanie użytkownika o tym fakcie i wyłączenie przycisków służących do operacji na danych. Wywołanie procedury **wyswietl_rekord** z wartością ujemną powoduje wyczyszczenie pól tekstowych.

```
Else
```

```
btnNastepny.Enabled = True
btnPoprzedni.Enabled = True
btnUsunRekord.Enabled = True
btnEZapiszDane.Enabled = True
wyswietl_rekord(nr_rekordu)
End If
End Sub
```

W przypadku gdy w bazie są dane włącznie przycisków służących do operacji na danych i wywołanie procedury **wyswietl_rekord** z numerem rekordu, który ma być aktualnie wyświetlony.

Procedura **wyswietl_rekord** wyświetla jeden wiersz danych z obiektu typu **DataTable**.

```
Private Sub wyswietl_rekord(ByVal wiersz As Integer)
If wiersz >= 0 Then
    objTabela = objDane.Tables("ksiazki")
    objWiersz = objTabela.Rows(wiersz)
    txtEID_k.Text = objWiersz.Item(0)
    txtEAutor.Text = objWiersz.Item(1)
    txtETytul.Text = objWiersz.Item(2)
    txtERok_wydania.Text = objWiersz.Item(3)
    txtECena.Text = objWiersz.Item(4)
    txtEWydawnictwo.Text = objWiersz.Item(5)
```

W przypadku, gdy w bazie są dane, do wyświetlenia zawartości pól wybranego rekordu wykorzystany jest obiekt typu **DataRows**.

```
Else
    czysc_edycja()
End If
End Sub
```

W przypadku, gdy baza jest pusta, wywoływana zostaje procedura czyszcząca pola tekstowe.

Przyciski **Następny** i **Poprzedni** pozwalają na poruszanie się pomiędzy rekordami. Procedury zdarzenia z nimi związane zwiększają (lub zmniejszają) numer rekordu, jaki ma być wyświetlony i wywołują procedurę **wyswietl_rekord**. Dodatkowo informują użytkownika o osiągnięciu pierwszego lub ostatniego rekordu.

```
Private Sub btnNastepny_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnNastepny.Click
If nr_rekordu < objDane.Tables("ksiazki").Rows.Count - 1 Then
   nr_rekordu += 1
   wyswietl_rekord(nr_rekordu)
Else
   MessageBox.Show("Ostatni rekord - przejście dalej nie jest możliwe",
"Uwaga", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
End Sub

Private Sub btnPoprzedni_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnPoprzedni.Click
If nr_rekordu > 0 Then
   nr_rekordu -= 1
   wyswietl_rekord(nr_rekordu)
Else
   MessageBox.Show("Pierwszy rekord - przejście dalej nie jest możliwe",
„Uwaga", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
End Sub
```

Zapis zmian w danych następuje po kliknięciu przycisku **Zapisz zmiany w rekordzie**. Do modyfikacji rekordu jest wykorzystywane polecenie SQL w połączeniu z klasami VB.NET służącymi do obsługi baz danych. Procedura wypełniająca to zadanie może wyglądać następująco:

```
Private Sub btnEZapiszDane_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnEZapiszDane.Click
Dim strWpis As String
Dim strSQL As String
If txtEID_k.Text.Length = 0 OrElse Not IsNumeric(txtEID_k.Text) Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtEID_k.Focus()
  Exit Sub
End If
```

Sprawdzenie poprawności wprowadzonych danych do pola tekstowego. W przypadku błędnych danych (pole puste lub niewłaściwy typ danych, np. tekst) działanie procedury jest przerywane a kursor zostaje ustawiony w polu z błędem.

```
If txtEAutor.Text.Length = 0 Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtEAutor.Focus()
  Exit Sub
End If
If txtETytul.Text.Length = 0 Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtETytul.Focus()
  Exit Sub
End If
If txtERok_wydania.Text.Length = 0 OrElse Not
IsNumeric(txtERok_wydania.Text) Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtERok_wydania.Focus()
  Exit Sub
End If
If txtECena.Text.Length = 0 OrElse Not IsNumeric(txtECena.Text) Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtECena.Focus()
  Exit Sub
End If
If txtEWydawnictwo.Text.Length = 0 Then
  MessageBox.Show("Błędny wpis", "Błąd", MessageBoxButtons.OK,
MessageBoxIcon.Error)
  txtEWydawnictwo.Focus()
  Exit Sub
End If
```

Sprawdzenie pozostałych pól formularza

```
polacz_z_baza()
```

Połączenie z bazą danych

```
strWpis = "id_k=" & txtEID_k.Text & ", autor='" & txtEAutor.Text &
"',Tytul='" & _
txtETytul.Text & "', rok_wydania=" & txtERok_wydania.Text & ", cena=" &
txtECena.Text & ", Wydawnictwo='" & _
txtEWydawnictwo.Text & "'"
```

```
strSQL = "update ksiazki set " & strWpis & " where id_k=" & txtEID_k.Text
```
Utworzenie zapytania SQL aktualizującego dane.

```
If objPolaczenie.State = ConnectionState.Open Then
  Try
```
Sprawdzenie, czy jest połączenie z bazą i otwarcie bloku kodu chronionego.

```
   Dim objZapytanie As New System.Data.SqlClient.SqlCommand(strSQL,
objPolaczenie)
   objZapytanie.ExecuteNonQuery()
```
Utworzenie nowego obiektu typu **SQLCommand** i wykonanie aktualizacji danych za pomocą zapytania SQL.

```
   odczyt_do_edycji(nr_rekordu)
```
Aktualizacja danych na formularzu.

```
   MessageBox.Show("Rekord został zaktualizowany", "Informacja",
MessageBoxButtons.OK, MessageBoxIcon.Information)
```
Poinformowanie użytkownika o wykonaniu aktualizacji danych.

```
  Catch ex As DataException
   MessageBox.Show("Błąd podczas aktualizacji danych", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
  Finally
   objPolaczenie.Close()
  End Try
End If
End Sub
```
Obsługa ewentualnych wyjątków mogących wystąpić przy połączeniu z bazą i zakończenie połączenia z bazą w bloku **Finally** (w przypadku wystąpienia wyjątku połączenie na pewno zostanie zamknięte).

Ostatnim przyciskiem, który należy oprogramować jest **Usuń rekord**. Ponownie zostanie wykorzystane polecenie SQL służące do usuwania danych oraz klasy VB.NET.

```
Private Sub btnUsunRekord_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnUsunRekord.Click
polacz_z_baza()
```
Połączenie z bazą danych.

```
If objPolaczenie.State = ConnectionState.Open Then
If ile_rekordow() > 0 Then
```
Sprawdzenie, czy jest połączenie z bazą, i czy są rekordy w bazie.

```
  Dim Pytanie As DialogResult
  Pytanie = MessageBox.Show("Czy chcesz slasować bieżacy rekord?",
"Usuwanie danych", MessageBoxButtons.YesNo, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button2)
  If Pytanie = DialogResult.Yes Then
```
Potwierdzenie usunięcia danych przez użytkownika.

```
   Dim strSQL As String
   strSQL = "delete from ksiazki" & " where id_k=" & txtEID_k.Text
```
Utworzenie zapytania SQL..

```
   Try
    Dim objZapytanie As New System.Data.SqlClient.SqlCommand(strSQL,
objPolaczenie)
```

```
    objZapytanie.ExecuteNonQuery()
```

Utworzenie nowego obiektu typu **SQLCommand** i usunięcie rekordu za pomocą zapytania SQL.

```
    MessageBox.Show("Rekord został usunięty", "Informacja",
MessageBoxButtons.OK, MessageBoxIcon.Information)
    lblIleKsiazek.Text = "W bazie znajdują się " & ile_rekordow() &
"książek"
    odczyt_do_edycji(0)
```

Potwierdzenie usunięcia rekordu oraz aktualizacji informacji o ilości rekordów w bazie. W formularzu zostanie wyświetlony pierwszy rekord.

```
   Catch ex As DataException
    MessageBox.Show("Bład podczas aktualizacji danych", "Błąd",
MessageBoxButtons.OK, MessageBoxIcon.Error)
   Finally
    objPolaczenie.Close()
   End Try
  End If
End If
Else
  MessageBox.Show("Brak połaczenia z bazą", "Bład", MessageBoxButtons.OK,
MessageBoxIcon.Error)
End If
End Sub
```

Obsługa ewentualnych wyjątków mogących wystąpić przy połączeniu z bazą i zakończenie połączenia z bazą w bloku **Finally** (w przypadku wystąpienia wyjątku połączenie na pewno zostanie zamknięte).

Przedstawiony powyżej przykład pokazuje podstawowe mechanizmy obsługi baz danych wykorzystywane w VB.NET. Wiele rzeczy można zrobić także inaczej, np. aktualizację danych wykonywać za pomocą obiektów klasy **DataAdapter**. Powyższy przykład miał pokazać, jak można połączyć efektywnie język SQL z klasami .NET Framework.