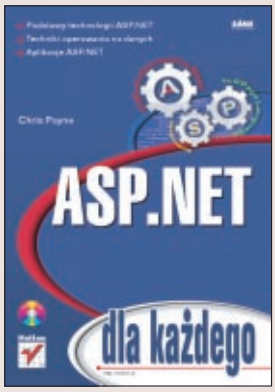
ASP.NET dla każdego



Autor: Chris Payne
Tłumaczenie: Andrzej Będkowski, Piotr Rajca
ISBN: 83-7197-607-0
Tytuł oryginału: Teach Yourself ASP.NET in 21 Days
Format: B5, stron: 710

Active Server Pages.NET, w skrócie ASP.NET, jest najnowszą wersją popularnej technologii ASP opracowanej przez firmę Microsoft i służącej do tworzenia dynamicznych aplikacji internetowych. ASP.NET jest jednak czymś więcej niż zwyczajnym uaktualnieniem klasycznej technologii ASP – zupełnie nowy model programistyczny oraz wiele nowych narzędzi to tylko dwie spośród wielu cech, którymi się wyróżnia. Pomiędzy klasyczną technologią ASP a jej nową wersją istnieje bardzo wiele różnic. Sprawiają one, iż nauczenie się ASP.NET nie jest łatwe. Niemniej jednak, dzięki niniejszej książce poznanie ASP.NET stanie się znacznie prostsze.
Bez wątpienia znajomość klasycznej technologii ASP, bądź innych rozwiązań umożliwiających tworzenie dynamicznych aplikacji internetowych, może pomóc w nauce ASP.NET, niemniej jednak aby korzystać z niniejszej książki, nie trzeba mieć żadnego doświadczenia programistycznego. Opis zagadnień omawianych w każdym z rozdziałów był opracowywany przy założeniu, że Czytelnik nie zajmował się dotąd technologią ASP. Jednak jeśli tworzyłeś już aplikacje internetowe przy użyciu ASP, to na pewno z chęcią przeczytasz punkty „To nie jest ASP!", które można znaleźć pod koniec każdego z rozdziałów książki. Zawierają one podstawowe informacje na temat różnic pomiędzy klasyczną technologią ASP a ASP.NET.
Nowe możliwości technologii ASP.NET sprawiają, że projektowanie i tworzenie dynamicznych stron WWW stało się wyjątkowo łatwe. Na przykład ASP.NET udostępnia wiele, niezwykle potężnych, elementów sterujących – znaczników, które przypominają znaczniki HTML i dają wiele różnych możliwości, jak na przykład: wyświetlanie kalendarza, losowo wybieranych reklam, czy też tabel HTML, których komórki zawierają informacje pobierane z baz danych. Te wyjątkowe elementy sterujące pozwalają programistom na generowanie złożonego kodu HTML zgodnego z obowiązującymi standardami, przy minimalnym nakładzie pracy. Podstawowe informacje na temat tych elementów sterujących oraz sposobów ich wykorzystania można znaleźć w rozdziale 5., „Podstawy tworzenia formularzy internetowych". Książka „ASP.NET dla każdego" składa się z 21 rozdziałów, które wyjaśniają wszelkie zawiłości związane z wykorzystaniem technologii ASP.NET. Każdy z rozdziałów zawiera przydatne informacje, które niezwłocznie będziesz mógł wykorzystać przy tworzeniu własnych aplikacji internetowych. Przyjęty w książce sposób przedstawiania wiedzy, polegający na tym, iż każdy kolejny rozdział omawia nowe zagadnienia, bazując przy tym na wiedzy wyniesionej z lektury poprzednich rozdziałów, jest najlepszy dla początkujących programistów, gdyż umożliwia im szybkie poznanie cech tej nowej, fascynującej technologii.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 10. Korzystanie z baz danych za pomocą obiektów ADO.NET

Po przeczytaniu dwóch poprzednich rozdziałów czytelnik znalazł się na najlepszej drodze, aby zostać ekspertem z dziedziny baz danych. Rozdział 8. „Podstawowe wiadomości na temat tworzenia baz danych" zawierał wprowadzenie do baz danych oraz krótkie omówienie zastosowania ich w środowisku ASP.NET. W rozdziale 9. opisano stosowanie obiektu sterującego `DataSet` i innych obiektów, dla których występuje wiązanie danych, w formularzach internetowych. Do tej pory jednak nie zamieszczono szczegółowych informacji na temat obiektów ADO.NET.

W niniejszym rozdziale zostanie opisane środowisko programowania ADO.NET oraz sposób współpracy tego środowiska ze środowiskiem ASP.NET. Znajdzie się tu sporo teorii wzbogaconej jednak wieloma przykładami. Na podstawie wiadomości z niniejszego rozdziału czytelnik powinien umieć odczytywać dane, zapisane w dowolnym magazynie danych (data store) z poziomu kodu strony ASP.NET, aby w wyniku otrzymać dynamiczne strony z obsługą danych (data-enabled pages).

W niniejszym rozdziale omówione zostaną następujące zagadnienia:

- modyfikowanie danych zapisanych w obiekcie `DataSet`,
- posługiwanie się obiektami `DataRow` oraz `DataTable`,
- automatyczne odzwierciedlanie w źródle danych zmian wprowadzanych w obiekcie `DataSet` za pomocą obiektu `OleDbCommandBuilder`,
- stosowanie obiektu `OleDbConnection`,
- stosowanie obiektu `OleDbCommand`,
- stosowanie obiektu `OleDbDataReader`,
- stosowanie obiektu `OleDbDataAdapter`.

Wprowadzenie do technologii ADO.NET

Obiekty ADO.NET jest to kolejny etap rozwoju obiektów ADO (ActiveX Data Object). Przy ich tworzeniu wykorzystano model dostępu do danych, u którego podstaw leżą takie cechy Internetu jak: skalowalność (scalability) i brak zachowywania danych stałych pomiędzy żądaniami użytkownika (statelesness) oraz język XML. Obiekty te stanowią interfejs do wszystkich źródeł danych zgodnych ze standardem OLE DB, umożliwiając łączenie się z takim źródłem, wykonywanie operacji na danych oraz aktualizację źródła. Można z nich korzystać z systemu zdalnego, za pomocą aplikacji rozproszonych lub za pomocą danych odłączonych (disconnected data).

Dzięki obiektom ADO.NET programista tworzący strony ASP.NET może używać w kodzie strony danych dowolnego typu. Obiekty te umożliwiają użytkownikom odczytywanie i zmianę danych zapisanych w dowolnym rodzaju składnicy danych, włączając w to bazy danych, pliki tekstowe oraz magazyny danych XML (XML data stores). Wskazane jest, aby dokładnie zapoznać się z obiektami ADO.NET, ponieważ spełniają one ważną rolę przy tworzeniu aplikacji dynamicznych. Poznanie wszystkich pułapek, jakie można napotkać przy ich stosowaniu, zaoszczędzi poźniejszych kłopotów.

ADO.NET kontra ADO

Chociaż firma Microsoft ogłosiła, że technologia ADO.NET jest tylko kolejnym etapem rozwoju obiektów ADO i zawiera częściowo takie same obiekty, technologia ta jest jednak zupełnie odmienna od swojego poprzednika. Podczas gdy w przypadku obiektów ADO konieczne było połączenie z magazynem danych, to obiekty ADO.NET komunikują się ze źródłem danych za pomocą języka XML. Dlatego obiekty ADO.NET są bardziej wydajne w przypadku aplikacji internetowych.

Podstawową zmianą, jaką wprowadzono w obiektach ADO.NET w stosunku do ADO, jest zastosowanie języka XML do wymiany danych. XML jest to *rozszerzalny język znaczników* (extensible markup language), zapisywany w formacie pliku tekstowego, podobny do HTML, który stanowi bardzo wydajny sposób przedstawiania danych (XML zostanie omówiony w następnym rozdziale, „Użycie XML w ASP.NET"). Technologia ADO.NET jest ściśle złączona z XML i korzysta z tego języka do wykonywania wszystkich operacji. Umożliwia to obiektom ADO.NET uzyskanie dostępu do magazynu danych, wymianę danych oraz zachowywanie magazynu danych łatwiej niż w przypadku obiektów ADO. Obiekty ADO.NET pracują również bardziej wydajnie, ponieważ można łatwo dokonywać konwersji typów danych wymienianych za pomocą języka XML, nie marnując czasu procesora na skomplikowane przekształcanie typów, jak miało to miejsce w przypadku obiektów ADO.

Inną poważną zmianą jest sposób współpracy obiektów ADO.NET z bazami danych. Obiekty ADO wymagały blokowania dostępu do zasobów bazy danych i nadmierne długich połączeń dla aplikacji napisanych w tej technologii, co nie ma miejsca w przypadku obiektów ADO.NET. Obiekty te korzystają z *odłączonych zbiorów danych* (disconnected data sets) (za pomocą obiektu DataSet), co pozwala uniknąć długotrwałych połączeń i blokowania baz danych. W ten sposób aplikacje ADO.NET stają się skalowalne, ponieważ użytkownicy nie rywalizują o dostęp do zasobów bazy danych.

Zmiany wprowadzone w technologii ADO.NET w stosunku do technologii ADO zebrano w tabeli 10.1.

Tabela 10.1. *Zmiany wprowadzone w technologii ADO.NET w stosunku do technologii ADO*

Technologia ADO	Technologia ADO.NET
Przedstawienie danych:	
Obiekt `Recordset`, przypominający pojedynczą tabelę lub wynik kwerendy.	Obiekt `DataSet`, który może zawierać wiele tabel z wielu źródeł danych.
Dostęp do danych:	
Sekwencyjny dostęp do wierszy zapisanych w obiekcie `Recordset`.	Umożliwia całkowicie niesekwencyjny dostęp do danych zapisanych w obiekcie `DataSet` za pomocą hierarchii kolekcji.
Relacje pomiędzy wieloma tabelami:	
Dane z wielu tabel można połączyć w jednym obiekcie `Recordset` za pomocą instrukcji SQL JOIN i UNION.	Do przechodzenia pomiędzy powiązanymi tabelami służą obiekty `DataRelation`.
Współdzielenie danych:	
Konieczne jest dokonanie konwersji typu danych na typ akceptowany przez system-odbiorcę, co obniża wydajność aplikacji.	Korzysta się z XML, więc konwersje typów nie są konieczne.
Możliwość programowania:	
Za pomocą obiektu `Connection` można przesłać dane do odpowiednich elementów składowych źródła danych.	Korzysta z mechanizmów silnej kontroli typów danych języka XML (strongly typed characteristics of XML); nie wymaga korzystania z części składowych magazynu danych (tabele, wiersze, kolumny); można odwoływać się do wszystkiego przez nazwę.
Skalowalność:	
Wynikiem walki o dostęp do źródła danych jest blokowanie dostępu do bazy danych oraz połączenia z bazą.	Nie występuje blokowanie dostępu do bazy danych ani długotrwałe aktywne połączenia, więc nie ma walki o dostęp do danych.
Zapory ogniowe:	
Stosowanie zapór ogniowych w tym przypadku jest problematyczne, ponieważ zapory blokują wiele rodzajów zapytań.	Można stosować zapory ogniowe, ponieważ umożliwia to zastosowanie języka XML.

Technologia ADO.NET a język XML

Język XML jest bardzo przydatnym narzędziem do dystrybucji danych. Jest całkowicie tekstowy, co oznacza, że łatwo jest w tym języku pisać i czytać aplikacje, które mogą być przesyłane w ramach środków bezpieczeństwa ustanowionych w Internecie.

XML zapisuje dane, stosując hierarchiczne przedstawienie pól i danych, które zawierają. Na przykład, baza danych `Użytkownicy`, zawierająca pola `Nazwisko`, `Identyfikator` oraz `DataUrodzenia`, mogłaby zostać przedstawiona w postaci tekstowej w następujący sposób:

```
<Użytkownicy>
<Użytkownik>
   <Nazwisko> </Nazwisko>
   <Identyfikator></Identyfikator>
   <DataUrodzenia> </DataUrodzenia>
</Użytkownik>
</Użytkownicy>
```

Jest to podstawowa struktura, która może być wykorzystana jako szablon dokumentu XML. (W rzeczywistości kod jest nieco bardziej skomplikowany niż powyższy przykład, ale informacje na ten temat wykraczają poza zakres niniejszej książki). Powyższy schemat można zastosować do przedstawiania wszystkich danych zapisanych w tabelach:

```
<Użytkownicy>
<Użytkownik>
   <Nazwisko>Jan Skrzetuski</Nazwisko>
   <Identyfikator>1</Identyfikator>
   <DataUrodzenia>22 Styczeń</DataUrodzenia>
</Użytkownik>
<Użytkownik>
   <Nazwisko>Helena Kurcewiczówna</Nazwisko>
   <Identyfikator>2</Identyfikator>
   <DataUrodzenia>6 Styczeń</DataUrodzenia>
</Użytkownik>
</Użytkownicy>
...
...
```

Powyższy kod można odczytywać za pomocą edytora tekstów (na przykład program Notatnik), podczas gdy odpowiadająca mu tabela bazy danych może być odczytywana tylko za pomocą narzędzi dostępnych w danej aplikacji bazodanowej lub poprzez dokonanie konwersji do innej bazy danych. Język XML jest niezależnym od implementacji, wydajnym narzędziem do zapisywania i przesyłania danych.

Dlatego też taka forma komunikacji została zaadaptowana dla potrzeb baz danych oraz ich interfejsów. Ułatwia to wszystkim życie. W technologii ADO.NET stosuje się język XML do wymiany danych i do wewnętrznego przedstawienia danych. Dane natychmiast po odczytaniu z bazy danych przekształcane są na postać XML i przesyłane wszędzie tam, gdzie trzeba. Ponieważ praktycznie każda aplikacja potrafi odczytać XML[1], zapewnia to zgodność postaci danych; dane mogą być przesyłane do dowolnego systemu i pewne jest, że odbiorca potrafi je odczytać.

Zastosowanie języka XML w technologii ADO.NET jest wielkim krokiem ku udostępnianiu aplikacji jako usług internetowych, co jest celem istnienia środowiska .NET. W tym miejscu przedstawione są tylko podstawowe wiadomości w skrócie, ale w kolejnych kilku rozdziałach przy tworzeniu programów rozproszonych ujawnią się zalety tego narzędzia.

[1] Autor ma na myśli łatwość odczytu danych zapisanych w formacie XML. Może to zrobić najprostszy edytor tekstowy. Nie spodziewajmy się jednak, że otworzymy taki dokument za pomocą programu do obróbki grafiki — *przyp. red.*

Model obiektowy ADO.NET

Środowisko ADO.NET składa się dwóch głównych części: obiektów `DataSet`, które omówiono szczegółowo w poprzednim rozdziale, oraz usługodawców zarządzanych (managed providers). Obiekt `DataSet` przedstawia dane przekazywane pomiędzy składnikami środowiska ADO.NET, na przykład z bazy danych do strony ASP.NET. Jest to mechanizm przedstawiania danych poza magazynem danych.

Nowy termin

Usługodawcy zarządzani (managed providers) służą jako warstwa komunikacyjna pomiędzy obiektami `DataSet` a magazynami danych. Umożliwiają łączenie się z magazynem danych zgodnym ze standardem OLE-DB (na przykład Microsoft Access), uzyskanie dostępu, wykonywanie różnych operacji oraz odczytywane danych z tego magazynu.

W firmie Microsoft opracowano dwóch usługodawców zarządzanych dla technologii ADO.NET: SQL Managed Provider oraz OLE DB Managed Provider. Pierwszy z nich służy wyłącznie do współpracy z SQL Serverem i zawiera wszystkie metody do komunikacji pomiędzy SQL Serverem i obiektem `DataSet`. Drugi z nich, OLE DB, pośredniczy w ustanowieniu komunikacji pomiędzy obiektem `DataSet` a dowolnym źródłem danych zgodnym ze standardem OLE DB. W obydwu przypadkach podstawowy zakres funkcji do współdziałania z magazynami danych jest jednakowy, więc na czym polega różnica?

SQL Managed Provider do wymiany danych z SQL Serverem korzysta z protokołu pod nazwą *tabelaryczny strumień danych* (tabular data stream). Jest to bardzo efektywny sposób komunikowania się z SQL Serverem, w którym nie korzysta się z OLE DB, ADO ani ODBC. Protokół ten jest całkowicie obsługiwany przez maszynę wirtualną CLR, więc ma wszystkie zalety opisane do tej pory. Dlatego właśnie Microsoft poleca stosowanie magazynów danych SQL Servera w technologii ADO.NET i ASP.NET.

Usługodawca SQL (SQL provider) w technologii ADO.NET współpracują tylko z SQL Serverem w wersji 7.0 lub wyższej. W przypadku korzystania z wcześniejszej wersji należy zastosować usługodawcę OLE DB.

Usługodawca OLE DB (OLE DB Provider) umożliwia efektywną komunikację z innymi magazynami danych — może być nawet w razie potrzeby stosowany do komunikacji z SQL Serverem.

Każdy z usługodawców zarządzanych składa się z trzech elementów:

- Interfejsów do łączenia się z magazynami danych, przetwarzania poleceń oraz współpracy z obiektem `DataSet`.
- Strumienia danych do uzyskania szybkiego i efektywnego dostępu do danych (przypomina obiekt `DataSet`, ale jest szybszy i ma mniej funkcji).
- Obiektów, które łączą się z bazą danych i wykonują polecenia bazy danych niskiego poziomu, zależnych od systemu.

W dalszej części niniejszej książki stosowany będzie usługodawca OLE DB, ponieważ umożliwia on dostęp do typów danych, które będą wykorzystywane. Składnia w przypadku obydwu usługodawców jest podobna, więc zmiana usługodawcy na usługodawcę SQL nie powinna być trudna.

W rzeczywistości wszystkie obiekty usługodawców ADO mają przedrostek `OleDb`. W większości przypadków wystarczy zastąpić go przedrostkiem SQL, importować przestrzeń nazw `System.Data.SQL` i w ten sposób umożliwić stosowanie usługodawcy SQL zamiast usługodawcy OLE DB.

Obiekt DataSet — ciąg dalszy

W poprzednim rozdziale pominięto kilka pojęć i atrybutów związanych z obiektem `DataSet`.

Nowy termin

Znając już obiekt `DataSet` ważne jest, aby pamiętać, że jest on jednostką całkowicie niezależną od źródła danych. Z tego powodu obiekt `DataSet` określa się jako *odłączony* (disconnected). Dzięki temu każdy z użytkowników otrzymuje swoją kopię danych, z którymi może zrobić, co tylko zechce. Zmiany dokonane w obiekcie `DataSet` nie mają automatycznego odzwierciedlenia w źródle danych. Każda zmiana musi być w sposób jawny wprowadzona do źródła danych za pomocą metod, które zostaną podane w dalszej części niniejszego rozdziału.

W tabeli 10.2 zamieszczono atrybuty obiektu `DataSet`.

Tabela 10.2. *Atrybuty obiektu DataSet*

Atrybut	Opis
`CaseSensitive`	Wskazuje, czy przy porównywaniu łańcuchów znaków w obiektach `DataTable` będą uwzględniane duże i małe litery.
`DataSetName`	Odczytuje lub nadaje nazwę bieżącego obiektu `DataSet`.
`DefaultView`	Wyświetla dane z obiektu `DataSet` w postaci dostosowanej do konkretnych potrzeb. Umożliwia to wyszukiwanie danych, filtrowanie, przechodzenie pomiędzy tabelami itd.
`EnforceConstraints`	Wskazuje, czy istniejące ograniczenia w bazie danych powinny być widoczne przy aktualizacji.
`ExtendedProperties`	Pobiera zbiór danych użytkownika.
`HasErrors`	Wskazuje, czy wystąpiły błędy w wierszach tablic obiektu `DataSet`.
`Relations`	Pobiera zbiór relacji pomiędzy tabelami obiektu `DataSet`.
`Tables`	Pobiera zbiór tabel obiektu `DataSet`.
`GetXML`	Pobiera lub ustawia dane XML albo schemat XML (XML schema) obiektu `DataSet`.
`XMLData`	Pobiera lub ustawia tylko dane XML obiektu `DataSet`.
`GetXMLSchema`	Pobiera lub ustawia tylko schemat XML obiektu `DataSet`.

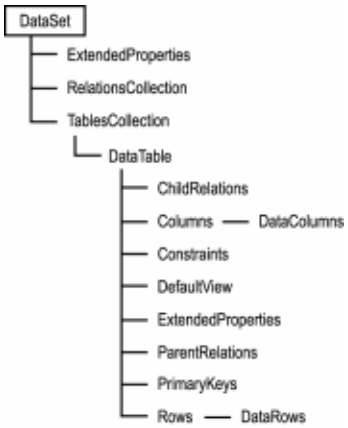
W tabeli 10.3 zamieszczono metody obiektu DataSet.

Tabela 10.3. *Metody obiektu DataSet*

Metoda	Opis
AcceptChanges	Przekazuje wszystkie zmiany dokonane w obiekcie DataSet, od jego pobrania lub od ostatniego wywołania metody AcceptChanges.
Clear	Usuwa wszystkie wiersze we wszystkich tabelach obiektu DataSet. Nie usuwa rzeczywistej zawartości bazy danych.
Clone	Kopiuje strukturę obiektu DataSet, łącznie z tabelami danych (obiekty DataTable), relacjami i ograniczeniami.
Copy	Kopiuje strukturę i dane zapisane w danym obiekcie DataSet.
GetChanges	Zwraca kopię obiektu DataSet, zawierającą wszystkie zmiany dotyczące danych, wprowadzone od ostatniego pobrania.
GetChildRelations	Pobiera zbiór relacji potomnych (podrzędnych) określonej tablicy.
GetParentRelations	Pobiera zbiór relacji macierzystych (nadrzędnych) określonej tablicy.
HasChanges	Wskazuje, czy obiekt DataSet zawiera jakiekolwiek zmiany.
Merge	Scala dany obiekt DataSet z innym.
ReadXML	Wczytuje schemat XML (XML schema) i dane XML (XML data) do obiektu DataSet.
ReadXMLData	Wczytuje dane XML do obiektu DataSet.
ReadXMLSchema	Wczytuje schemat XML do obiektu DataSet.
RejectChanges	Odrzuca wszystkie zmiany dokonane w danym obiekcie DataSet.
ResetRelations	Zmienia wartość atrybutu relations na wartość domyślną.
ResetTables	Zmienia wartość atrybutu tables na wartość domyślną.
ShouldPersistRelations	Wskazuje, czy atrybut relations powinien zostać zachowany.
ShouldPersistTables	Wskazuje, czy atrybut tables powinien być zachowany.
WriteXML	Zapisuje kod XML przedstawiający obiekt DataSet, łącznie z danymi i schematem, do pliku XML.
WriteXMLData	Zapisuje kod XML przedstawiający obiekt DataSet (tylko dane) do pliku XML.
WriteXMLSchema	Zapisuje kod XML przedstawiający obiekt DataSet (tylko schemat) do pliku XML.

W powyższych tabelach nie zamieszczono wszystkich atrybutów i metod obiektu DataSet. Pominięto niektóre dziedziczone atrybuty i metody. Więcej informacji na ten temat można znaleźć w dokumentacji firmowej .NET Framework SDK lub w dodatku C. „Obiekty ADO.NET — właściwości i metody".

Teraz widać, że obiekt DataSet ma wiele funkcji, które wcześniej pominięto, na przykład odczytywanie i zapisywanie danych za pomocą języka XML. Nie należy bać się eksperymentowania z ustawieniami podanych powyżej atrybutów. Obiekty DataSet i DataRow mają również większość takich samych atrybutów jak obiekt DataSet, więc nie zostały opisane. Rysunek 10.1 przedstawia model obiektu DataSet.



Rysunek 10.1. *Model obiektu DataSet*

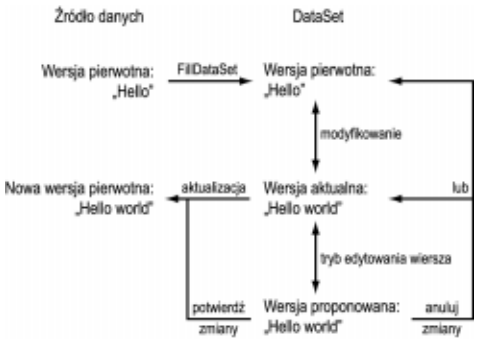
Modyfikowanie danych w wierszu (obiekt DataRow)

Ponieważ działanie obiektu `DataSet` jest już znane, opisane zostanie teraz modyfikowanie danych zapisanych w danym obiekcie. Dane zapisane są w obiekcie `DataSet` w postaci przypominającej bazę danych; obiekt taki zawiera tabele, kolumny i wiersze. Często modyfikuje się dane zapisane w obiekcie `DataSet` za pomocą wyrażeń SQL, które zmieniają dane w wielu rekordach jednocześnie, ale czasem konieczny jest ściślejszy nadzór nad każdym z wierszy. Obiekt `DataRow` przedstawia wiersz danych zapisanych w obiekcie `DataTable`. Jak podano w poprzednim rozdziale, można edytować bezpośrednio zawartość każdego z wierszy (obiektów `DataRow`).

Należy też znać kilka innych właściwości obiektów `DataRow` i `DataTable`. Po pierwsze, jest to atrybut `RowState`, który wskazuje stan bieżącego wiersza. Atrybut ten może mieć wartość `Detached`, `Unchanged`, `New`, `Deleted` oraz `Modified`. `Detached` oznacza, że wiersz został utworzony, ale nie jest częścią żadnego zbioru wierszy (obiekt `RowsCollection`) obiektu `DataSet`. Znaczenie kolejnych czterech nie wymaga objaśnień.

Jako część atrybutu `RowState` obiekt `DataTable` zawiera trzy wersje każdego z wierszy: pierwotną, aktualną oraz proponowaną. Wersje te służą do określenia stanu wiersza (atrybut `RowState`). Wersja pierwotna jest to stan wiersza po dodaniu po raz pierwszy do obiektu `DataTable`. Zazwyczaj są to takie same wartości jak w bazie danych. Wersja aktualna jest to wersja po wprowadzeniu zmian. Wersja proponowana występuje tylko w jednym przypadku — kiedy dla danego wiersza wywołano metodę `BeginEdit`.

Metoda `BeginEdit` jest stosowana do dokonywania zmian w wierszach bez konieczności stosowania się do reguł poprawności. Na przykład, jeśli mamy kilka wierszy, które muszą zostać dodane do określonej wartości, można przejść do trybu edytowania i wykonywać dowolne operacje. Wywołanie metody `EndEdit` lub `AcceptChanged` kończy tryb edytowania i przywraca stosowanie reguł poprawności. Tryb edytowania można również wykorzystać do anulowania proponowanych zmian. Proces modyfikowania wiersza przedstawiono na rysunku 10.2.



Rysunek 10.2. *Modyfikowanie wiersza*

Wartość pierwotna (ze źródła danych) jest przenoszona do obiektu `DataSet`, kiedy wywoływana jest metoda `Fill`. Po dokonaniu zmian wartość ta staje się wartością aktualną. Na tym etapie można przywrócić wartość pierwotną, dokonać aktualizacji źródła danych, wpisując do niego wartość aktualną, lub też powrócić do trybu edytowania. W trybie edytowania można potwierdzić wprowadzone zmiany i dokonać aktualizacji magazynu danych lub anulować zmiany i przywrócić wersję pierwotną lub aktualną. W rzeczywistości można przywrócić dowolną wersję, a następnie dokonać aktualizacji magazynu danych w trybie edytowania. Każda z tych wersji jest dostępna za pomocą atrybutów `DataRowVersion.Original`, `DataRowVersion.Current` oraz `DataRowVersion.Proposed`.

Przy modyfikowaniu danych w wierszu (`DataRow`) mogą z różnych przyczyn wystąpić błędy. Każdy z błędów zapisany jest w atrybucie `RowError` obiektu `DataRow` w postaci tekstowej. Można również ręcznie wpisywać błędy do tego atrybutu. Wszystkie błędy można odczytać jednocześnie za pomocą metody `GetErrors`, która zwraca tablicę obiektów `DataRow` (wierszy). Jeśli wystąpi jakikolwiek błąd, nie zostanie wykonane łączenie ani aktualizacja źródła danych. Najpierw należy usunąć przyczyny błędów. Nie należy przejmować się, jeśli brzmi to niezrozumiale. Wszystko będzie jaśniejsze po zapoznaniu się z przykładami.

Obiekt `DataRow` zawiera dwie metody, `Delete` oraz `Remove`, które wydają się bardzo podobne, ale jest między nimi bardzo istotna różnica. Metoda `Delete` całkowicie usuwa wiersz razem z danymi, które zawiera. Potem nie ma już dostępu do tych danych. Za pomocą metody `Remove` usuwa się wiersz z tabeli (obiekt `DataTable`) i w ten sposób nie ma już do niego dostępu z poziomu programu. Jednak rzeczywiste źródło danych nie zostało naruszone, więc dane nadal są na swoim miejscu, ale nie widać ich. Jest to przydatne, jeśli nie wszystkie wiersze tabeli (obiektu `DataTable`) będą wykorzystywane.

Ostatnia z omawianych metod, `RejectChanges`, unieważnia wszystkie zmiany wprowadzone od pobrania danego wiersza lub od ostatniego wywołania metody `AcceptChanges`. Na przykład, poniższy fragment kodu wczytuje dane do obiektu `DataSet`, modyfikuje wartości zapisane w pierwszym wierszu, a następnie odrzuca wprowadzone zmiany:

```
dim objConn as new OleDbConnection _
   ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
   "Data Source="C:\ASPNET\dane\banking\mdb")
dim objCmd as new OleDbDataAdapter _
   ("select * from tblUsers", objConn)
```

```
dim ds as DataSet = new DataSet()
objCmd.Fill(ds, "tblUsers)

ds.Tables("tblUsers").Rows(0)("Imię") = "Tadeusz"

'rób cokolwiek

ds.Tables("tblUsers").Rows(0).RejectChanges
```

Przeglądanie danych zapisanych w tabeli (obiekt DataTable)

Obiekt `DataTable` zawiera metodę `Select` umożliwiającą filtrowanie i sortowanie danych zapisanych w danej tabeli. Metoda ta zwraca tablicę wierszy (obiektów `DataRow`). Wywołuje się ją w następujący sposób:

```
NazwaTabeli.Select(wyrażenie filtru, porządek sortowania, _
   DataRowViewState)
```

Oto przykład zastosowania tej metody:

```
dim ds as new DataSet("MójDataSet")
dim dTable as new DataTable("MojaTabela")
'wpisz dane do obiektu DataSet i DataTable
dim MojeWiersze() as DataRow = ds.Tables("MojaTabela").Select _
   (Nothing, "NazwaUżytkownika", DataViewRowState.CurrentRows)
```

Powyższy kod zwraca tablicę wszystkich zmodyfikowanych wierszy (obiektów `DataRow`) posortowanych według pola „NazwaUżytkownika”. Dla każdego parametru, który ma być pominięty, należy wpisać `Nothing`. Dlatego zwracana może być jedna z wersji wiersza lub wszystkie wersje wiersza, które można sortować i (lub) filtrować. Poniżej zamieszczono kolejny przykład.

Wydruk 10.1. *Odczytywanie wierszy za pomocą metody Select*

```
1:  <%@ Page Language="VB" %>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
6:     sub Page_Load(obj as object, e as eventargs)
7:        dim objConn as new OleDbConnection _
8:           ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
9:           "Data Source=C:\ASPNET\dane\banking.mdb")
10:
11:       dim objCmd as new OleDbDataAdapter _
12:          ("select * from tblUsers", objConn)
13:
14:       dim ds as DataSet = new DataSet()
15:       objCmd.Fill(ds, "tblUsers")
16:
17:       dim dTable as DataTable = ds.Tables("tblUsers")
18:       dim AktualneWiersze() as DataRow = dTable.Select(Nothing, _
19:          Nothing, DataViewRowState.CurrentRows)
```

```
20:       dim I, J as integer
21:       dim strOutput as string
22:
23:       For I = 0 to AktualneWiersze.Length-1
24:          For J = 0 to dTable.Columns.Count-1
25:             strOutput = strOutput & dTable.Columns(J). _
26:                ColumnName & " = " & _
27:                AktualneWiersze(I)(J).ToString & "<br>"
28:          next
29:       next
30:
31:       Response.Write(strOutput)
32:    end sub
33: </script>
34:
35: <html><body>
36:
37: </body></html>
```

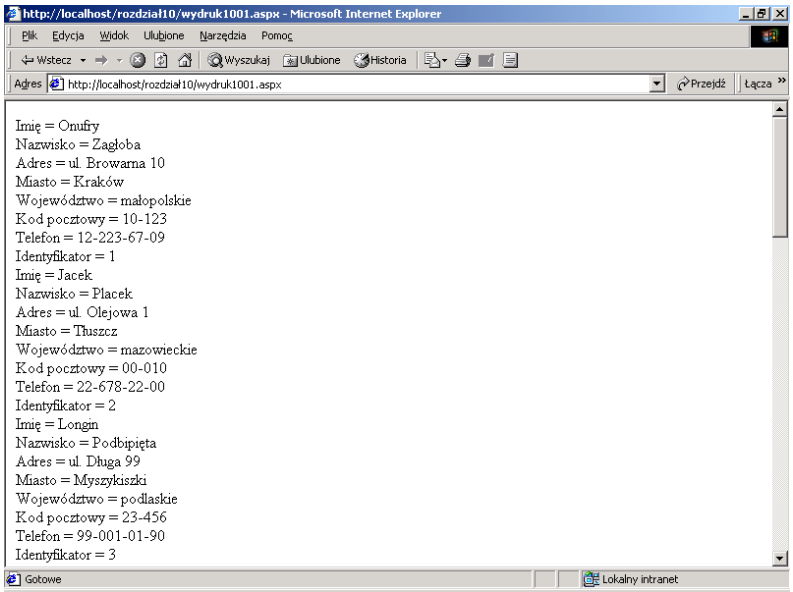
Analiza Powyższy program przykładowy odczytuje z obiektu `DataSet` wszystkie aktualne wiersze i wyświetla poszczególne pola i ich zawartość w oknie przeglądarki. W kodzie metody `Page_Load` deklaruje się obiekty `OleDbConnection` oraz `OleDbDataAdapter` (wiersze 7. – 12.). Po zapoznaniu się z poprzednim rozdziałem powinno być to zrozumiałe. Następnie, tworzony jest obiekt `DataSet`, który za pomocą metody `Fill` jest wypełniany danymi (wiersze 14. i 15.). Dalej odczytywana jest jedyna tabela obiektu `DataSet`, którą zapisuje się do zmiennej `dTable`, aby ułatwić dostęp do danych w dalszej części kodu.

Metoda `Select`, umieszczona w wierszu 18., odczytuje wszystkie wiersze obiektu `DataTable`, które uległy zmianie (wiersze aktualne), i umieszcza je w tablicy. Do przechodzenia do kolejnych wierszy tablicy służy pętla `for` (wiersz 23.); do przechodzenia do kolejnych kolumn danego wiersza służy kolejna pętla `for` (wiersz 24.). Odczytywane są nazwy pól oraz ich zawartość, które po konwersji na typ `string` wyświetlane są za pomocą metody `Response.Write`. Wynik działania powyższego programu zamieszczono na rysunku 10.3.

Innym sposobem sortowania i filtrowania danych jest zastosowanie obiektu `DataView`. Obiekt `DataView` przedstawia obiekt `DataTable`, ale w przeciwieństwie do tego obiektu można go powiązać z internetowymi obiektami sterującymi. Dla jednego obiektu `DataTable` można utworzyć wiele obiektów `DataView`.

W przypadku stron ASP.NET umożliwia to stosowanie dwóch różnych obiektów sterujących powiązanych z tym samym obiektem `DataTable`, ale wyświetlających różne dane. Poniższy fragment kodu pokazuje, w jaki sposób tworzyć atrybuty obiektu `DataView` i nadawać im wartości:

```
dim MójWidok as new DataView(dTable)
MójWidok.RowStateFilter = DataViewRowState.ModifiedOriginal
MójWidok.Sort = "Identyfikator ASC"
MójWidok.RowFilter = "Miasto = Gliwice"
```



Rysunek 10.3. *Zawartość obiektu DataSet odczytana za pomocą metody Select oraz pętli for*

Najpierw ze zmiennej `dTable` tworzony jest nowy obiekt `DataView` (wiersz 1.). (Zmienna `dTable` jest to wcześniej utworzony obiekt `DataTable`, wypełniony danymi). W wierszu 2. powyższego przykładu określa się filtr — odrzucane będą wszystkie wersje wierszy oprócz wersji pierwotnej. W trzecim wierszu określono porządek sortowania. Ostatni wiersz podaje kryterium zwracania wierszy. Powyższy przykład pokazuje, że obiekt `DataView` zawiera wiele tych samych atrybutów, które metoda `Select` używa do odczytywania wierszy danych. Znajomość tych atrybutów będzie bardzo pomocna podczas rzeczywistego modyfikowania źródeł danych.

Współbieżność

Nowy termin

Ponieważ każdy z użytkowników ma swój własny podgląd obiektu `DataSet`, może wykonywać dowolne operacje na danych i aktualizować źródło danych po ich zakończeniu w dowolnym momencie. Co się jednak stanie, jeśli kilku użytkowników będzie chciało równocześnie dokonać aktualizacji tych samych danych? *Współbieżność* (concurrency) jest sposobem zapobiegania problemom, które mogłyby wystąpić w takiej sytuacji. Są dwa rodzaje współbieżności: *pesymistyczny* (pessimistic) i *optymistyczny* (optimistic).

W przypadku współbieżności pesymistycznej, za każdym razem, kiedy któryś z użytkowników uzyskuje dostęp do danych lub próbuje dokonać zmian w danych, ustanawiana jest blokada i żaden z pozostałych użytkowników nie ma dostępu do tych danych. Po zakończeniu pracy przez użytkownika, który był pierwszy, pozostali użytkownicy mogą znów próbować uzyskać dostęp do danych.

W przypadku współbieżności optymistycznej nie występuje blokowanie dostępu do danych. Zamiast tego wiersze są monitorowane w celu określenia, czy dane nie zostały zmienione, a następnie zapisuje te zmiany. Przyjmijmy, że dwóch użytkowników

odczytało dane dotyczące statku o nazwie *M/S Stefan Batory*. Obydwaj mają ten sam zestaw danych. Pierwszy użytkownik zmienił nazwę na *M/S Stefan Batory I*. Jeśli drugi użytkownik spróbuje później zmienić nazwę po raz kolejny, zmiany te nie zostaną uwzględnione, ponieważ jego zestaw danych nie będzie już aktualny. Użytkownik ten musi pobrać wersję aktualną danych i spróbować jeszcze raz. Używając określeń z dziedziny baz danych, można napisać, że jeśli użytkownicy pobiorą ten sam zestaw danych, uwzględnione zostaną tylko te zmiany, które zostaną wprowadzone jako pierwsze. Dane drugiego użytkownika są już nieważne, ponieważ zmianie uległa zawartość źródła danych, więc jego próby wprowadzenia zmian zakończą się niepowodzeniem.

W technologii ADO.NET można stosować obydwa rodzaje współbieżności. Środowisko ADO.NET zawiera procedury obsługi obydwu rodzajów współbieżności, które działają w sposób niezauważalny dla użytkownika. Jednak wiedza o tym, jak działają takie procedury, może być przydatna w sytuacji, kiedy wystąpią jakieś problemy.

Korzystanie z baz danych za pomocą technologii ADO.NET

Jak już wspomniano w rozdziale 8. „Podstawowe wiadomości o tworzeniu baz danych", wyróżniamy pięć etapów uzyskiwania dostępu do danych za pomocą strony internetowej ASP.NET:

1. Utworzenie obiektu łączącego z bazą danych.
2. Otwarcie połączenia z bazą danych.
3. Wypełnienie obiektu `DataSet` odpowiednimi danymi.
4. Skonfigurowanie obiektu `DataView` w celu wyświetlania danych.
5. Powiązanie serwerowego obiektu sterującego z obiektem `DataView`.

Punkt piąty został omówiony szczegółowo w poprzednim rozdziale, zatem w dalszej części niniejszego rozdziału zamieszczono informacje dotyczące pierwszych czterech. Pewne działania można wykonywać na kilka sposobów, stosując różne obiekty, więc zostaną one opisane po kolei. Najpierw jednak należy zapoznać się z danymi, które są konieczne do połączenia się z bazą danych.

Dane dotyczące połączenia z bazą danych

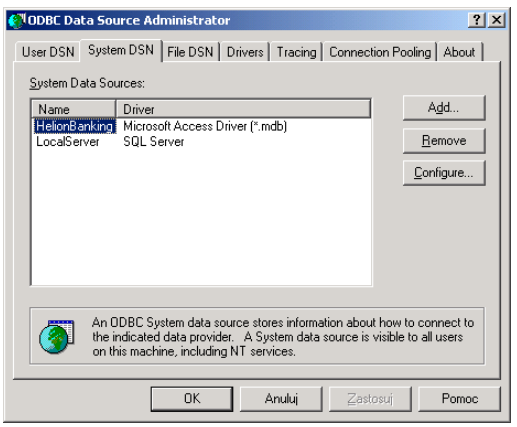
Nowy termin

Zanim w kodzie stron ASP.NET do korzystania z baz danych będzie można zastosować obiekty ADO.NET, konieczne jest uzyskanie danych dotyczących konkretnej bazy danych, do której trzeba uzyskać dostęp. Dane te to lokalizacja bazy danych, rodzaj bazy danych (na przykład MS Access, SQL Server lub Oracle), wersja bazy danych itd. Dane te są przekazywane do obiektów ADO.NET za pomocą *łańcucha połączenia* (connection string), który tworzy się ręcznie.

Najprostszym sposobem przekazania wymienionych wyżej danych jest utworzenie pliku *System DSN* (System Data Source Name). Plik taki, zawierający dane domyślne dla niektórych magazynów danych zainstalowanych przez system operacyjny, powinien już istnieć. Wystarczy tylko dodać dane do tego pliku. Na szczęście, jest to operacja bardzo prosta. Jako przykład przedstawione zostanie dodawanie danych dotyczących bazy danych użytkowników, której tworzenie opisano w rozdziale 8.

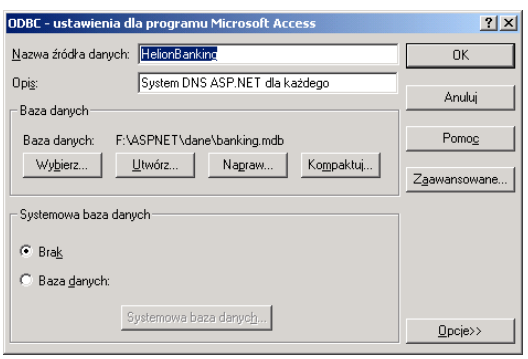
1. Jeśli baza danych użytkowników jest otwarta, zamknij ją.
2. W systemie Windows 2000 przejdź do menu *Start/Ustawienia/Panel sterowania, Narzędzia administracyjne/Źródła danych (ODBC)*.
3. Wybierz zakładkę System DSN. Powinno pojawić się okno podobne do przedstawionego na rysunku 10.4. Za pomocą tej zakładki można tworzyć, edytować lub usuwać istniejące źródła danych ODBC.

Rysunek 10.4. *Dane zakładki System DSN*



4. Naciśnij przycisk *Dodaj*. Z listy wybierz pozycję *Microsoft Access Driver (*.mdb)* i naciśnij przycisk *Zakończ*.
5. Podaj nazwę DSN `HelionBanking`. Można także wprowadzić opis.
6. Naciśnij przycisk *Wybierz* i przejdź do bazy danych Accessa, która została utworzona na podstawie rozdziału 8. Wybierz tę bazę, naciskając przycisk *OK*. Powinno pojawić się okno podobne do przedstawionego na rysunku 10.5. Naciśnij przycisk *OK*, potem jeszcze raz *OK* i gotowe!

Rysunek 10.5. *System DSN dla bazy danych banking.mdb*



System DSN bazy danych zawiera wszystkie dane, które są konieczne, aby obiekty ADO.NET mogły znaleźć daną bazę. Teraz, korzystając z obiektów ADO.NET w kodzie strony ASP.NET, można podać następujący łańcuch połączenia:

```
"DSN=HelionBanking"
```

Aby uniknąć kłopotów związanych z tworzeniem systemowego połączenia DSN (system Data Source Name), można zastosować połączenia nie korzystające z DSN (DSN-less connection). Wtedy konieczne jest podanie wszystkich koniecznych danych w łańcuchu połączenia. Dla przykładowej bazy danych łańcuch połączenia może wyglądać następująco:

```
"Provider=Microsoft.Jet.OLEDB.4.0;DataSource=
➥C:\ASPNET\dane\banking.mdb"
```

W przypadku bazy danych SQL łańcuch połączenia mógłby wyglądać następująco:

```
"Provider=SQLOLEDB.1;Initial Catalog=Northwind;
➥Data Source=MyServer;User ID=sa;"
```

Łańcuch połączenia podaje obiektom ADO.NET usługodawcę (provider), z którego mają korzystać, oraz lokalizację bazy danych. Jest wiele innych parametrów, które można podać, na przykład `UID` — nazwę użytkownika oraz `PWD` — hasło, konieczne do uzyskania dostępu do bazy danych. Jednak najczęściej stosuje się parametry podane powyżej, więc opis pozostałych został pominięty.

Jest wiele argumentów za stosowaniem systemowego połączenia DSN, oprócz estetycznego (kod wygląda tym lepiej, im łańcuchy połączeń są krótsze). Najważniejszą zaletą jest to, że nie trzeba sprawdzać danych zawartych w łańcuchu połączenia, przy każdym łączeniu się z bazą danych, co jest konieczne w przypadku połączeń bez połączenia DSN. Wystarczy sprawdzić dane tylko raz, kiedy tworzy się DSN. Wynikiem jest poprawa wydajności systemu.

Obiekt OleDbConnection

Teraz, kiedy już wiadomo, jak ustanowić połączenie z bazą danych, podany zostanie sposób otwarcia bazy danych. Właśnie do tego służy obiekt `System.Data.OleDb.OleDbConnection`. Przykład użycia tego obiektu zamieszczono na wydruku 10.2.

Wydruk 10.2. *Otwieranie połączenia z bazą danych za pomocą obiektu OleDbConnection*

```
1:  dim strŁańcuchPołączenia as string = _
2:     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
3:     "Data Source=C:\ASPNET\dane\banking.mdb")
4:  dim Połączenie as new OleDbConnection( _
5:     strŁańuchPołączenia)
6:  Połączenie.Open()
7:  ...
8:  Połączenie.Close()
```

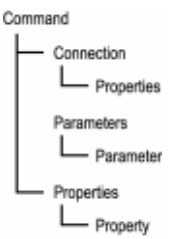
I to wszystko! Połączenie z bazą danych zostało właśnie otwarte. W wierszu 1. zadeklarowano łańcuch połączenia. Łańcuch ten jest wykorzystany przez obiekt `OleDbConnection` do połączenia z bazą danych (wiersz 4.). Za pomocą metody `Open` połączenie to jest otwierane (wiersz 6.), za pomocą metody `Close` — zamykane. Nie wolno zapominać o zamykaniu połączenia, kiedy nie jest już potrzebne.

W większości przypadków to wszystko, co robi obiekt `OleDbConnection` — otwiera i zamyka połączenie z bazą danych. Inne kruczki z zastosowaniem tego obiektu zostaną przedstawione w rozdziale 12. „Zastosowanie zaawansowanych technik obsługi danych".

Obiekt OleDbCommand

Po otwarciu połączenia z bazą danych można za pomocą poleceń wykonywać operacje dotyczące tej bazy, na przykład zapisanie danych w obiekcie `DataSet` lub aktualizację rekordów. Na rysunku 10.6 zamieszczono fragment modelu obiektu `OleDbCommand`.

Rysunek 10.6. *Model obiektu OleDbCommand*



Polecenia dotyczące bazy danych mają postać wyrażeń SQL, które są już znane. Wszystko, co trzeba zrobić, to przypisać dane wyrażenie SQL do obiektu `OleDbCommand`. Przykład zamieszczono na wydruku 10.3.

Wydruk 10.3. *Tworzenie i inicjalizacja obiektu OleDbCommand*

```
1:  'podaj wyrażenie SQL
2:  dim strSQL as string = "SELECT * FROM tblUsers"
3:
4:  'utwórz obiekt i nadaj wartości atrybutom
5:  dim objCmd as new OleDbCommand()
6:  objCmd.Connection = Conn
7:  objCmd.CommandText = strSQL
8:
9:  'lub
10: 'dim objCmd as new OleDbCommand(strSQL, Conn)
11:
12: 'lub
13: 'dim objCmd as new OleDbCommand(strSQL, _
14: strConnectionString)
```

Analiza Obiekt `OleDbCommand` może być inicjalizowany na wiele sposobów przez podanie odpowiednich parametrów. Parametrami może być wyrażenie SQL i obiekt `OleDbConnection` (wiersz 10.) lub utworzony przez programistę łańcuch połączenia (wiersz 13.). Jednak podanie tylko, które polecenie ma być wykonane, nie wystarczy. Konieczne jest również

wykonanie polecenia za pomocą jednej z metod `Execute`. Wybór metody zależy od przeznaczenia zwracanych danych. Na przykład, aby zapisać dane w obiekcie `OleDbDataReader` (opis w następnym punkcie), należy napisać, co następuje:

```
'utwórz obiekt DataReader
dim objReader as OleDbDataReader
objReader = objCmd.ExecuteReader
```

Aby wysłać zapytanie, które nie będzie zwracać danych, należy napisać:

```
ObjCmd.ExecuteNonQuery
```

Więcej wiadomości na temat metod `Execute` w dalszej części książki.

Obiekt OleDbDataReader

Obiekt `OleDbDataReader` jest prostym obiektem umożliwiającym prosty dostęp do danych zapisanych w magazynie danych. W gruncie rzeczy jest to obiekt `DataSet` wyświetlający dane w postaci strumienia (strumień `DataSet`). Więc po co stosować obiekty `OleDbDataReader`, jeśli do dyspozycji są obiekty `DataSet`?

Dane są odczytywane z bazy danych za pomocą obiektu `DataSet` i przechowywane w pamięci, dopóki nie zostaną wydane inne dyspozycje. Umożliwia to wykonywanie różnych operacji na takim odłączonym magazynie danych. Można, na przykład, modyfikować dane, nie przejmując się tym, co robią inni użytkownicy i dokonać przekształcenia na inny format. Jednak po rozpoczęciu pobierania dużej ilości danych z bazy danych napotyka się na ograniczenie pamięci, ponieważ w całości zapisany w niej jest obiekt `DataSet`. Jeśli użytkowników są tysiące, a każdy z nich ma własny obiekt `DataSet`, pojawiają się poważne problemy. (Należy zwrócić uwagę, że jest to przypadek skrajny, ale na jego przykładzie jasno widać potrzebę stosowania obiektów mniejszych niż `DataSet`).

Obiekt `OleDbDataReader` umieszcza w pamięci jednorazowo tylko jeden wiersz danych. Na żądanie tworzy *strumień* danych z magazynu danych. Zapobiega to występowaniu wielu problemów związanych z dostępną pamięcią, co daje w wyniku poprawę wydajności systemu. Niestety, z powodu tego, że dane mają postać strumienia, obiekt `OleDbDataReader` ma mniejszy zbiór funkcji niż obiekt `DataSet`. Obiekt `OleDbDataReader` jest obiektem tylko do odczytu; nie można również powracać do rekordów, które zostały już przetworzone.

Po zapisaniu danych w obiekcie `OleDbDataReader` przechodzenie do kolejnych rekordów jest bardzo łatwe. Należy po prostu wywołać metodę `Read`. Przykład zamieszczono na wydruku 10.4.

Wydruk 10.4. *Przechodzenie pomiędzy kolejnym rekordami zapisanymi w obiekcie OleDbDataReader*

```
1:  <%@ Page Language="VB" %>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
```

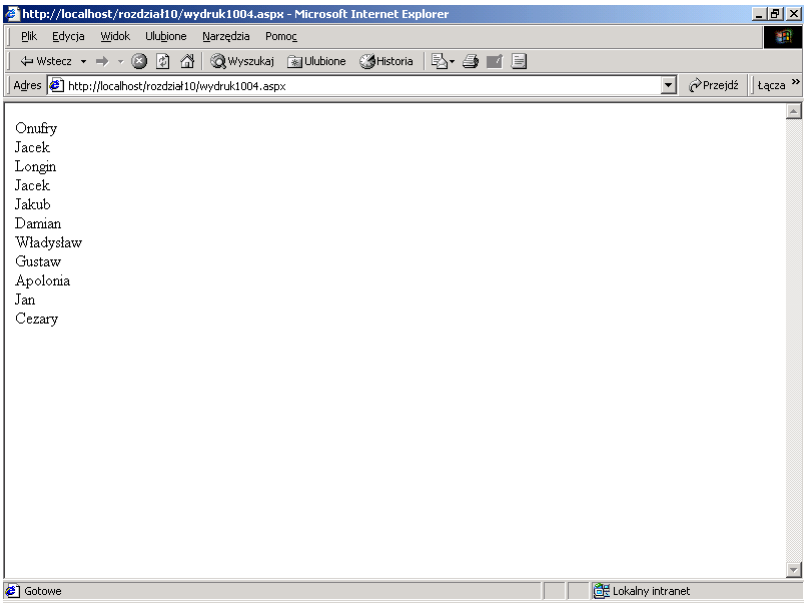
```
6:     sub Page_Load(obj as object, e as eventargs)
7:        dim objConn as new OleDbConnection _
8:           ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
9:        "Data Source=C:\ASPNET\dane\banking.mdb")
10:
11:       dim objCmd as new OleDbCommand _
12:          ("select * from tblUsers", objConn)
13:
14:       dim objReader as OleDbDataReader
15:
16:       objConn.Open
17:       objReader = objCmd.ExecuteReader
18:
19:       while objReader.Read
20:          Response.Write(objReader.GetString(0) & "<br>")
21:       end while
22:       objConn.Close
23:    end sub
24: </script>
25:
26: <html><body>
27:
28: </body></html>
```

Analiza Fragment powyższego wydruku od wiersza 7. do wiersza 12. powinien być zrozumiały: utworzony zostaje obiekt `OleDbConnection`, obiekt `OleDbCommand` i wykonywane jest wyrażenie języka SQL. Następnie utworzony zostaje nowy obiekt `OleDbDataReader`. W przypadku zastosowania obiektu `OleDbDataAdapter` metoda `Fill` automatycznie otwiera i zamyka połączenie z bazą danych. Jednak w przypadku zastosowania obiektu `OleDbDataReader` trzeba umieścić odpowiednie metody w kodzie strony (wiersz 16. i 22.). Wiersz 17. zawiera wywołanie wyrażenia SQL i tworzy strumień danych za pomocą obiektu `OleDbDataReader`. W wierszach 19. – 21. następuje odczytywanie kolejnych rekordów obiektu `OleDbDataReader`.

Metoda `Read` przechodzi automatycznie do kolejnego rekordu, zatrzymując się na ostatnim. W wierszu 20. za pomocą metody `GetString`, która dokonuje konwersji typu zwracanych danych na typ `string`, odczytywane jest pierwsze pole każdego z rekordów. (Nieco dalej metoda ta zostanie opisana bardziej szczegółowo). Następnie odczytane i przekształcone dane są wyświetlane za pomocą metody `Response.Write`. Wynik wykonania kodu z powyższego wydruku zamieszczono na rysunku 10.7.

Sprawdzając wartość atrybutu `HasMoreRows`, można sprawdzić, czy program dotarł już do ostatniego rekordu. Jeśli atrybut ten ma wartość logiczną `true`, oznacza to, że obiekt `OleDbDataReader` zawiera więcej rekordów. W przeciwnym przypadku atrybut ten ma wartość logiczną `false`.

Obiekt `OleDbDataReader` zawiera również szereg metod `Get`, które zwracają dane, dokonując równocześnie konwersji na określony typ (`GetByte`, `GetInt32`, `GetString` itd.). Metody te stosuje się, aby uniknąć konieczności rzutowania typów danych, kiedy dane te są przekazywane poza obiekt `OleDbDataReader`.



Rysunek 10.7. *Odczytanie rekordów obiektu OleDbDataReader za pomocą metody Read*

Bardzo ważne jest, aby zamykać obiekt `OleDbDataReader`, kiedy nie jest już potrzebny. Taki sam skutek ma zamknięcie obiektu `OleDbConnection`.

Tak	Nie
Stosuj obiekt `OleDbDataReader` wszędzie tam, gdzie kluczową sprawą jest wydajność systemu i wystarczy tylko wyświetlić dane odczytane z bazy danych.	Nie stosuj obiektu `OleDbDataReader`, kiedy należy wykonać pewne operacje na danych przed lub po wyświetleniu ich.

Wyrażenia SQL Update, Insert oraz Delete

Ponieważ obiekt `OleDbDataReader` jest obiektem tylko do odczytu, nie umożliwia modyfikowania danych. Zamiast tego konieczne jest użycie odpowiednich wyrażeń języka SQL (`Update`, `Insert` i `Delete`) jako parametru obiektu `OleDbCommand`.

Jednak żadne z tych wyrażeń nie zwraca rekordów, tak jak to czyni wyrażenie `Select`. Z tego względu należy stosować metodę `ExecuteNonQuery`, która zwraca wartość typu `integer`, określającą, ile rekordów zostało zmienionych za pomocą któregoś z wymienionych powyżej wyrażeń. Przykład zamieszczono na wydruku 10.5.

Wydruk 10.5. *Zastosowanie metody ExecuteNonQuery*

```
1:  dim I as integer
2:
3:  'wyrażenie SQL
4:  dim strSQL as string = "DELETE FROM tblUsers" & _
5:  "WHERE Identyfikator = 5"
```

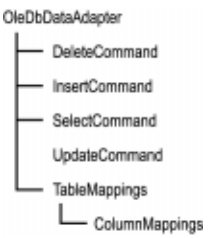
```
6:  'utwórz obiekt i nadaj wartość atrybutom
7:  dim objCmd as new OleDbCommand(strSQL, Conn)
8:
9:  I = objCmd.ExecuteNonQuery()
```

Ponieważ pole `Identyfikator` jest polem identyfikującym (identity field), za pomocą powyższego wyrażenia znaleziony zostanie co najwyżej jeden rekord, o ile w tabeli jest wiersz zawierający taki identyfikator. Sprawdzając wartość zmiennej `I` można przekonać się, czy wykonanie tego wyrażenia miało jakiś skutek. Wyrażenia `update` oraz `insert` działają podobnie (więcej informacji na temat wyrażeń SQL można znaleźć w rozdziale 8.).

Obiekt OleDbDataAdapter

Tak samo jak obiekt `OleDbCommand` służył do obsługi obiektów `OleDbDataReader`, obiekt `OleDbDataAdapter` współpracuje z obiektami `DataSet`. Głównym przeznaczeniem obiektu `OleDbDataAdapter` jest pobieranie danych z magazynu danych do obiektu `DataSet` oraz zapisywanie danych z powrotem z obiektu `DataSet` do magazynu danych. Model tego obiektu znajduje się na rysunku 10.8. Obiekt ten zawiera cztery metody `Command`, które służą do usuwania, wstawiania, wybierania i aktualizacji danych w obiekcie `DataSet`. Kolekcja `TableMappings` określa, w jaki sposób tabele i kolumny ze źródła danych są odwzorowane w obiektach `DataSet`.

Rysunek 10.8. *Model obiektu OleDbDataAdapter*



Obiekt `OleDbDataAdapter` ma duże możliwości. Chociaż jego głównym przeznaczeniem jest tylko odczytywanie danych, można za jego pomocą tworzyć nowe tabele z istniejących danych lub przedstawiać dane w postaci XML. Bardziej zaawansowane metody zostaną opisane w dalszej części niniejszego rozdziału, najpierw należy zapoznać się z podstawowymi informacjami na temat obiektu `OleDbDataAdapter`.

Tworzenie obiektu `OleDbDataAdapter` przypomina tworzenie obiektu `ADOCommand` (wydruk 10.6).

Wydruk 10.6. *Tworzenie obiektu OleDbDataAdapter*

```
1:  'podaj wyrażenie SQL
2:  dim strSQL as string = "SELECT * FROM tblUsers"
3:
4:  'utwórz obiekt i nadaj wartość atrybutom
5:  dim objCmd as new OleDbDataAdapter()
6:  objCmd.SelectCommand.Connection = Conn
7:  objCmd.SelectCommand.CommandText = strSQL
```

```
8:
9:  'lub
10: 'dim objCmd as new OleDbDataAdapter _
11: '   (strSQL, Conn)
12:
13: 'lub
14: 'dim objCmd as new OleDbDataAdapter( _
15: '   strSQL, strConnectionString)
```

Analiza Jak pokazano na rysunku 10.8, obiekt `OleDbDataAdapter` zawiera cztery metody `Command`. Każda z tych metod jest w rzeczywistości obiektem `OleDbCommand` z własnymi atrybutami `Connection` oraz `CommandText` (wiersze 6. i 7.). W powyższym przykładzie metodą określoną przez podanie wyrażenia SQL w trakcie inicjalizacji obiektu (wiersze 10. i 14.) jest metoda `SelectCommand`. Jeśli ma być zastosowana inna metoda `Command`, na przykład `Insert`, należy odpowiednio zmodyfikować program.

> Należy pamiętać o tym, że, stosując metody `Update`, `Insert` i `Delete`, w rzeczywistości nie zmienia się zawartości odnośnego magazynu danych, ale obiekt `DataSet`, odłączony od źródła danych. Zmiany zostaną wprowadzone w źródle danych dopiero po wywołaniu metody `OleDbDataAdapter.Update`.

Zapisywanie danych w obiektach DataSet

Poniżej zamieszczono przykład zapisywania danych w obiekcie `DataSet` (wydruk 10.7). Metody obiektu `OleDbDataAdapter` omówione zostaną w dalszej części niniejszego rozdziału.

Wydruk 10.7. *Zapisywanie danych w obiekcie DataSet*

```
1:  dim strConnectionString as String = _
2:  "Provider=Microsoft.Jet.OLEDB.4.0;" & _
3:  "Data Source=C:\ASPNET\dane\banking.mdb")
4:  dim ds as DataSet = New DataSet("MójDataSet")
5:  dim strSQL as String = "SELECT * FROM tblUsers"
6:
7:  dim objCmd as new OleDbDataAdapter(strSQL, _
8:  strConnectionString)
9:
10: objCmd.Fill(ds, "tblUsers")
```

Analiza Utworzono pusty obiekt `DataSet` pod nazwą „MójDataSet” (wiersz 4.). Potem utworzono nowy obiekt `OleDbDataAdapter` (wiersz 7.), którego polecenie `Select` zostało określone w postaci wyrażenia SQL zamieszczonego w wierszu 5. Wynikiem Wykonania powyższego kodu przykładowego jest obiekt `DataSet` zawierający tabelę (obiekt `DataTable`) pod nazwą „Użytkownicy”, która z kolei zawiera wszystkie rekordy tabeli `tblUsers`.

Jak to się stało? Przecież utworzono pusty obiekt `DataSet`. Metoda `Fill` obiektu `OleDbDataAdapter` pobiera ze źródła danych schemat (tabele, kolumny, definicje kluczy pierwotnych itd.), którego jeszcze nie zawiera obiekt `DataSet` i tworzy ten schemat

automatycznie. Następnie wypełnia tabelę (obiekt `DataTable`) „Użytkownicy" kolumnami odczytanymi ze źródła danych. Podobnie, gdyby obiekt `DataTable` zawierał już niektóre z kolumn, to metoda `FillDataSet` utworzy brakujące. Jeśli utworzone zostaną wszystkie kolumny, metoda ta tylko wypełni tabelę danymi. Jest to bardzo ważna cecha obiektu `OleDbDataAdapter`, która daje duże możliwości, a — jak okaże się w kolejnych podrozdziałach — będzie jeszcze lepiej.

Aktualizowanie źródeł danych

W poprzednim rozdziale został przedstawiony sposób wykonywania operacji na danych zapisanych w obiekcie `DataSet` przez uzyskanie dostępu do pól i wartości za pomocą kolekcji. Co jednak dzieje się po zmodyfikowaniu danych w taki sposób?

Po dokonaniu zmian można za pomocą odpowiednich metod obiektu `OleDbDataAdapter` wprowadzić te zmiany do magazynu danych. Polecenia te dotyczą tylko danych, które już zostały zmienione. Nie można, na przykład, tylko umieścić w kodzie strony wyrażenia `Insert` i oczekiwać, że w bazie danych pojawi się automatycznie nowy rekord. Wyrażenie `Insert` musi odwoływać się do nowego wiersza, który został już utworzony w obiekcie `DataSet`. Na wydruku 10.8 przedstawiono przykład nadawania odpowiednich wartości atrybutowi `UpdateCommand` obiektu `OleDbDataAdapter`.

Wydruk 10.8. *Wykonywanie operacji na danych zapisanych w obiekcie DataSet za pomocą obiektu OleDbDataAdapter*

```
1:  sub Page_Load(obj as object, e as EventArgs)
2:     'utwórz połączenie
3:     dim Conn as new OleDbConnection( _
4:        "Provider=Microsoft.Jet.OLEDB.4.0;" & _
5:        "Data Source=C:\ASPNET\dane\banking.mdb")
6:
7:     'utwórz obiekty DataSet oraz OleDbDataAdapter
8:     dim ds as new DataSet("MójDataSet")
9:     dim objCmd as new OleDbDataAdapter("SELECT * FROM " & _
10:       "tblUsers WHERE Identyfikator < 10", Conn)
11:
12:    'zapisz dane w obiekcie DataSet
13:    objCmd.Fill(ds, "tblUsers")
14:
15:    'zmień dane
16:    ds.Tables("tblUsers").Rows(2)(3) = "ASPVille"
17:
18:    dim dr as DataRow = ds.Tables("tblUsers").NewRow()
19:    dr(0) = "Greg"
20:    dr(1) = "Smith"
21:    dr(2) = "434 Maple Apt B"
22:    dr(3) = "Minneapolis"
23:    dr(4) = "MN"
24:    dr(5) = "12588"
25:    dr(6) = "5189876259"
26:    ds.Tables("tblUsers").Rows.Add(dr)
27:
28:    'podaj polecenie SQL oraz aktywne połączenie
29:    objCmd.UpdateCommand = new OleDbCommand
30:    objCmd.UpdateCommand.CommandText = "UPDATE tblUsers " & _
```

```
31:        "SET Miasto='ASP Górne' WHERE identyfikator=3"
32:     objCmd.UpdateCommand.Connection = Conn
33:
34:     'podaj inne polecenie SQL i inne aktywne połączenie
35:     objCmd.InsertCommand = new OleDbCommand
36:     objCmd.InsertCommand.CommandText = "Insert INTO " & _
37:        "tblUsers (Imię, Nazwisko Adres, Miasto, " & _
38:        "Województwo, Kod pocztowy, Telefon) VALUES 'Jacek', " & _
39:        "'Placek', 'Olejna 123', 'Tłuszcz', 'mazowieckie',  " & _
40:        "'00-098', '6783215565'"
41:     objCmd.InsertCommand.Connection = Conn
42:
43: end sub
```

Analiza W powyższym przykładzie najpierw tworzy się obiekty `OleDbConnection` oraz `OleDbDataAdapter` (wiersze 3. – 10.), a następnie za pomocą wyrażenia `Select` języka SQL wpisuje się dane do obiektu `DataSet` (wiersz 13.). Proces przebiega podobnie jak w przypadku zastosowania obiektu `OleDbCommand`. Kod w wierszach 16. – 26. służy do wprowadzania zmian do obiektu `DataSet` — edytowania wartości pojedynczej zmiennej (wiersz 16.) oraz dodania nowego wiersza (wiersze 18. – 26.). Następnym tworzonym obiektem jest obiekt `OleDbCommand` (wiersz 29.), który użyty jest razem z poleceniem `Update` obiektu `OleDbDataAdapter`. W wierszu 30. zamieszczono wyrażenie języka SQL, zgodnie z którym dane będą aktualizowane. Podobnie w wierszu 36. podano wyrażenie `Insert`. Należy zwrócić uwagę, że polecenia `Insert` oraz `Update` odnoszą się do tylko do danych w modyfikowanych wierszach. Modyfikacja źródła danych nastąpi po wywołaniu metody `Update`.

Należy zwrócić uwagę, że wymienione powyżej polecenia w rzeczywistości nie modyfikują żadnych danych. Przekazują po prostu obiektom ADO.NET instrukcje, w jaki sposób zapisać z powrotem dane, które zostały właśnie zmienione. Gdyby, na przykład, nie wprowadzono żadnych zmian za pomocą wierszy 16. – 26., to metody `Command` nie zostaną wykonane, bez względu na to, jakie wyrażenia języka SQL zamieszczono w programie.

Metoda `Update` obiektu `OleDbDataAdapter` wpisuje zmiany dokonane w obiekcie `DataSet` do źródła danych. Program zamieszczony na wydruku 10.8 zostanie zmodyfikowany w ten sposób, aby zawierał polecenie `Update`. W tym celu po wierszu 41. należy dopisać następujący wiersz:

```
objCmd.Update(ds, "tblUsers")
```

Parametrami tej metody są: obiekt `DataSet`, zawierający modyfikowane dane oraz tabela, z której dane będą odczytywane. Tabelę można pominąć, ale wtedy konieczne jest odwzorowywanie tabel (table mapping).

Powyższy sposób wygląda na bardzo kłopotliwy — konieczne jest ręczne wprowadzenie zmian, a potem jeszcze trzeba utworzyć wyrażenie języka SQL, które pokazuje, że dane zostały zmienione. Na szczęście, technologia ADO.NET przychodzi z pomocą. Jeśli nie określono odpowiednich poleceń, a wiersze w obiekcie `DataSet` zostały zmienione, to wtedy obiekt `OleDbDataAdapter` może za pomocą obiektu `OleDbCommandBulider` wygenerować potrzebne polecenia automatycznie. Oto przykład:

```
dim ds as new DataSet("MójDataSet")
dim objCmd as new OleDbDataAdapter _
   ("select * from tblUsers", Conn)
dim objAutoGen as new OleDbCommandBuilder(objCmd)
...
'odczytaj i modyfikuj
'dane
...
objCmd.Update(ds, "tblUsers")
```

Pierwsze trzy wiersze są takie jak poprzednio. Wiersz 4. służy do utworzenia obiektu `OleDbCommandBuilder`, którego parametrem jest obiekt `OleDbDataAdapter`. Obiekt `OleDbCommandBuilder`, po wywołaniu metody `Update`, sprawdza różnice pomiędzy obiektem `DataSet` a źródłem danych i generuje odpowiednie wyrażenia języka SQL, aby dostosować zawartość danego obiektu `DataSet` do tego źródła danych. Przyjrzyjmy się następującemu wierszowi kodu:

```
ds.Tables("tblUsers").Rows(3).Delete
```

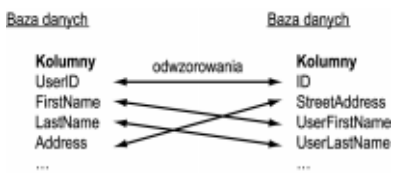
W połączeniu z obiektem `OleDbCommandBulider` i metodą `Update` zostanie wygenerowane następujące wyrażenie SQL:

```
DELETE FROM tblUsers WHERE Identyfikator = 4
```

Do określenia, który wiersz ma zostać usunięty, obiektowi `OleDbDataAdapter` służą wartości klucza pierwotnego. W powyższym przykładzie wartość klucza pierwotnego czwartego wiersza kolumny `Identyfikator` (`Rows(3)`) wynosi `4`. Automatyczne generowanie wyrażeń możliwe jest tylko wtedy, kiedy istnieje klucz pierwotny lub kolumna, w której wierszach wartości się nie powtarzają (unique column).

Odwzorowania (Mappings)

Odwzorowanie tabel i kolumn umożliwia przyporządkowanie tabeli lub kolumnie obiektu `DataSet` tabeli lub kolumny z magazynu danych. Odwzorowanie takie stosowane jest w technologii ADO.NET, kiedy dane są przenoszone z jednego źródła danych do innego. Umożliwia to korzystanie z kolumn lub tabel pod różnymi nazwami, zależnymi od lokalizacji, lub nawet odwzorowanie wzajemne kolumn lub tabel. Idea ta została przedstawiona na rysunku 10.9.



Rysunek 10.9. *Odwzorowanie łączące niepowiązane z pozoru kolumny*

Odwzorowania takie zapisane są w kolekcji `TableMappings` obiektu `OleDbDataAdapter`. Przykład zamieszczono na wydruku 10.9.

Wydruk 10.9. *Tworzenie odwzorowanie tabel dla bazy danych banking*

```
1:  dim ds as DataSet = New DataSet()
2:  dim Conn as New OleDbConnection( _
3:     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
4:     "Data Source=C:\ASPNET\dane\banking.mdb")
```

```
5:  dim objCmd as New OleDbDataAdapter _
6:     ("SELECT * FROM tblUsers", Conn)
7:
8:  'dodaj odwzorowania
9:  objCmd.TableMappings.Add("Tabela", "tblUsers")
10: with objCmd.TableMappings(0).ColumnMappings
11:    .Add("Identyfikator", "Numer")
12:    .Add("NazwiskoUżytkownika", "Nazwisko")
13:    .Add("ImięUżytkownika", "Imię")
14:    .Add("Telefon", "Telefon")
15:    .Add("Adres", "AdresUżytkownika")
16:    .Add("Miasto", "Miasto")
17:    .Add("Województwo", "Województwo")
18:    .Add("Kod", "Kod")
19: end With
20: objCmd.Fill(ds)
```

Analiza W powyższym przykładzie wykorzystano bazę danych utworzoną zgodnie z opisem zamieszczonym w rozdziale 8. i odwzorowano kolumny tabeli `tblUsers` na kolumny zapisane w obiekcie `DataSet`.

Fragment kodu w wierszach 1. – 6. powinien być już zrozumiały. Utworzone i zainicjalizowane zostały obiekty `DataSet`, `OleDbConnection` oraz `OleDbDataAdapter`. Następnie dokonywane jest odwzorowanie tabeli z magazynu danych na tabelę Użytkownicy w obiekcie `DataSet`. Pierwszym parametrem jest tabela źródłowa, z której odczytywane są dane, a drugim parametrem jest tabela docelowa, w której dane mają być zapisane. Ponieważ w deklaracji obiektu `OleDbDataAdapter` zamieszczono wyrażenie `Select`, wiadomo, że wynik wykonania tego wyrażenia ma być tabelą źródłową.

Nazwa `Table`, umieszczona w wierszu 9., w środowisku ADO.NET jest nazwą specjalną. Kiedy polecenia `Fill` lub `Update` uruchamiane są bez podania parametru, którym jest tabela z obiektu `DataSet`, wtedy odwzorowanie pod nazwą `Table` określa, skąd odczytać dane. Na przykład, jeśli użyte zostanie odwzorowanie z wydruku 10.9 i wywołana zostanie następująca metoda:

```
objCmd.Update(ds)
```

to dane zostaną odczytane z tabeli `Użytkownicy` obiektu `DataSet`, ponieważ tak podaje odwzorowanie `Table`. Zwykle, aby dokonać aktualizacji danych, wywołuje się metodę w następującej postaci:

```
objCmd.Update(ds, "tblUsers")
```

We fragmencie kodu od wiersza 10. do wiersza 20. do zdefiniowanego właśnie odwzorowania `TableMapping` dodawane są odwzorowania `ColumnMapping`. Odwzorowania te zastępują niektóre nazwy kolumn bardziej poręcznymi, a pozostałe odwzorowują bez zmian.

Odwzorowanie oprócz zastępowania nazw kolumn bardziej wygodnymi ma również inne zastosowania. Po dokładniejszym zapoznaniu się z wyrażeniami języka SQL da się zauważyć, że niektóre z nich nie zwracają pól ani kolumn — po prostu zwracają wartość. Sytuacje takie nie zdarzają się jednak zbyt często w trakcie tworzenia stron ASP.NET. Więcej informacji na ten temat można znaleźć w rozdziale 12.

Zastosowanie obiektów ADO.NET w środowisku ASP.NET

Pora, aby napisać aplikację ASP.NET. Utworzona zostanie w pełni funkcjonalna, choć prosta, aplikacja obsługująca bazę danych, która umożliwia użytkownikowi podgląd i modyfikowanie danych. Potrzebne jest sprawdzenie nowo nabytych umiejętności.

Dla tej aplikacji wystarczy jedna strona. Na niej za pomocą obiektu sterującego `DataGrid` wyświetlone zostaną dane z tabeli użytkowników. Do wprowadzenia zmian zostaną wykorzystane funkcje edycyjne obiektu sterującego `DataGrid`. Następnie dane zostaną z powrotem zapisane w źródle danych za pomocą obiektu `OleDbCommand`.

Najpierw należy utworzyć interfejs użytkownika. Program zamieszczony na wydruku 10.10 zawiera standardowy obiekt `DataGrid`, w którym znajdują się kolumny powiązane (bound columns) dla każdego z pól bazy danych, obiekty `EditCommandColumn` oraz `ButtonCommandColumn`, umożliwiające usuwanie danych. Do wyświetlania dodatkowych pól wejściowych, umożliwiających wstawianie do bazy danych nowych wierszy, służy obiekt `Panel`. Utworzone zostanie także obiekt sterujący `Label` (etykieta), wyświetlający komunikaty dla użytkownika.

Wydruk 10.10. *Interfejs użytkownika przykładowej aplikacji obsługującej bazę danych*

```
<html><body>
   <asp:Label id="lblMessage" runat="server"/>

   <form runat="server">
      <asp:DataGrid id="dgData" runat="server"
         BorderColor="black" GridLines="Vertical"
         cellpading="4" cellspacing="0" width="100%"
         AutoGenerateColumns="False"
         OnDeleteCommand="dgData_Delete"
         OnEditCommand="dgData_Edit"
         OnCancelCommand="dgData_Cancel"
         OnUpdateCommand="dgData_Update"
         OnPageIndexChanged="dgData_PageIndexChanged" >

         <Columns>
            <asp:TemplateColumn HeaderText="L.p.">
               <ItemTemplate>
                  <asp:Label id="Nazwa" runat="server"
                     Text='<%# Container.DataItem
                     ➥("Identyfikator")%>' />
               </ItemTemplate>
            </asp:TemplateColumn>

            <asp:BoundColumn HeaderText="Imię"
               DataField="Imię" />
            <asp:BoundColumn HeaderText="Nazwisko"
               DataField="Nazwisko" />
            <asp:BoundColumn HeaderText="Adres"
               DataField="Adres" />
            <asp:BoundColumn HeaderText="Miasto"
```

```
30:                DataField="Miasto" />
31:             <asp:BoundColumn HeaderText="Województwo"
32:                DataField="Województwo" />
33:             <asp:BoundColumn HeaderText="Kod pocztowy"
34:                DataField="Kod pocztowy" />
35:             <asp:BoundColumn HeaderText="Telefon"
36:                DataField="Telefon" />
37:
38:             <asp:EditCommandColumn
39:                EditText="Edycja"
40:                CancelText="Anuluj"
41:                UpdateText="Aktualizuj"
42:                HeaderText="Edycja"/>
43:
44:             <asp:ButtonColumn HeaderText="" text="Delete"
45:                CommandName="delete" />
46:          </Columns>
47:       </asp:DataGrid><p>
48:
49:       <asp:Panel id="AddPanel" runat="server">
50:          <table>
51:          <tr>
52:             <td width="100" valign="top">
53:                Imię i nazwisko:
54:             </td>
55:             <td width="300" valign="top">
56:                <asp:TextBox id="tbFName" runat="server"/>
57:                <asp:TextBox id="tbLName" runat="server"/>
58:             </td>
59:          </tr>
60:          <tr>
61:             <td valign="top">Adres:</td>
62:             <td valign="top">
63:                <asp:TextBox id="tbAddress" runat="server" />
64:             </td>
65:          </tr>
66:          <tr>
67:             <td valign="top">Miasto, stan, kod:</td>
68:             <td valign="top">
69:                <asp:TextBox id="tbCity"
70:                   runat="server" />
71:                <asp:TextBox id= "tbState" runat="server"
72:                   size=2 /> 
73:                <asp:TextBox id="tbZIP" runat="server"
74:                   size=5 />
75:             </td>
76:          </tr>
77:          <tr>
78:             <td valign="top">Telefon:</td>
79:             <td valign="top">
80:                <asp:TextBox id="tbPhone" runat="server"
81:                   size=11 /><p>
82:             </td>
83:          </tr>
84:          <tr>
85:             <td colspan="2" valign="top" align="right">
86:                <asp:Button id="btSubmit" runat="server"
```

```
87:                   text="Dodaj" OnClick="Submit" />
88:             </td>
89:          </tr>
90:          </table>
91:       </asp:Panel>
92:    </form>
93: </body></html>
```

Analiza Powyższy kod zawiera po prostu serwerowe obiekty sterujące z wieloma różnymi parametrami. Właściwe zadanie wypełnia blok deklarowania kodu, co okaże się w dalszej części. Najważniejszą częścią powyższego programu jest obiekt `DataGrid` i procedury obsługi zdarzeń, które zawiera (wiersze 9. – 13.). Każdy z obiektów sterujących umożliwia użytkownikowi wprowadzenie nowych danych do bazy; dane te są zapisywane w bazie po naciśnięciu przycisku *Dodaj* (wiersz 86.).

Na wydruku 10.11 zamieszczono blok deklarowania kodu. Zawiera on między innymi procedurę zapisywania danych w obiekcie `DataGrid`, procedurę obsługi zdarzenia obiektu `DataGrid` oraz procedurę aktualizacji źródła danych po wypełnieniu formularza przez użytkownika i naciśnięciu przez niego przycisku *Wyślij*.

Wydruk 10.11. *Kod ASP.NET do wydruku 10.10*

```
1:  <%@ Page Language="VB" %>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
6:     'zadeklaruj połączenie
7:     dim Conn as new OleDbConnection( _
8:        "Provider=Microsoft.Jet.OLEDB.4.0;" & _
9:        "Data Source=C:\ASPNET\dane\banking.mdb")
10:
11:    sub Page_Load(obj as Object, e as EventArgs)
12:       if Not Page.IsPostBack then
13:          FillDataGrid()
14:       end if
15:    end sub
16:
17:    sub Submit(obj as object, e as eventargs)
18:       'wstaw nowe dane
19:       dim i, j as integer
20:       dim params(7) as string
21:       dim strText as string
22:       dim blnGo as boolean = true
23:
24:       j = 0
25:
26:       for i = 0 to AddPanel.Controls.Count-1
27:          if AddPanel.controls(i).GetType Is _
28:             GetType(TextBox) then
29:             strText = Ctype(AddPanel.Controls(i), _
30:                TextBox).Text
31:             if strText <>"" then
32:                params(j)= strText
33:             else
```

```
34:                blnGo = false
35:                lblMessage.Text = lblMessage.Text & _
36:                   "Nie podano wartości dla " & _
37:                   AddPanel.Controls(i).ID & "<p>"
38:                lblMessage.Style("ForeColor") = "Red"
39:             end if
40:             j = j + 1
41:          end if
42:       next
43:
44:       if not blnGo then
45:          exit sub
46:       end if
47:
48:       dim strSQL as string = "INSERT INTO tblUsers " & _
49:          "(Imię, Nazwisko, Adres, Miasto, Województwo, " & _
50:          "Kod pocztowy, Telefon) VALUES (" & _
51:          "'" & params(0) & "'," & _
52:          "'" & params(1) & "'," & _
53:          "'" & params(2) & "'," & _
54:          "'" & params(3) & "'," & _
55:          "'" & params(4) & "'," & _
56:          "'" & params(5) & "'," & _
57:          "'" & params(6) & "')"
58:
59:       ExecuteStatement(strSQL)
60:
61:       FillDataGrid()
62:    end sub
63:
64:    sub dgData_Edit(obj as object, e as DataGridCommandEventArgs)
65:       FillDataGrid(e.Item.ItemIndex)
66:    end sub
67:
68:    sub dgData_Delete(obj as object, e as DataGridCommandEventArgs)
69:       dim strSQL as string = "DELETE FROM tblUsers " & _
70:          "WHERE Identyfikator = " & e.Item.ItemIndex + 1
71:
72:       ExecuteStatement(strSQL)
73:
74:       FillDataGrid()
75:    end sub
76:
77:    sub dgData_Update(obj as object, e as DataGridCommandEventArgs)
78:       if UpdateDataStore(e) then
79:          FillDataGrid(-1)
80:       end if
81:    end sub
82:
83:    sub dgData_Cancel(obj as object, e as DataGridCommandEventArgs)
84:       FillDataGrid(-1)
85:    end sub
86:
87:    sub dgData_PageIndexChanged(obj as object, e as
       ➥DataGridPageChangedEventArgs)
88:       dgData.DataBind()
89:    end sub
90:
```

```
91:    function UpdateDataStore(e as
       ➥DataGridCommandEventArgs) as boolean
92:
93:       dim i, j as integer
94:       dim params(7) as string
95:       dim strText as string
96:       dim blnGo as boolean = true
97:
98:       j = 0
99:
100:      for i = 1 to e.Item.Cells.Count - 3
101:         strText = Ctype(e.Item.Cells(i).Controls(0), _
102:            TextBox).Text
103:         if strText <> "" then
104:            params(j) = strText
105:            j = j + 1
106:         else
107:            blnGo = false
108:            lblMessage.Text = lblMessage.Text & _
109:               "Nie podano wartości <p>"
110:         end if
111:      next
112:
113:      if not blnGo then
114:         return false
115:         exit function
116:      end if
117:
118:      dim strSQL as string = "UPDATE tblUsers SET " & _
119:         "Imię = '" & params(0) & "'," & _
120:         "Nazwisko = '" & params(1) & "'," & _
121:         "Adres = '" & params(2) & "'," & _
122:         "Miasto = '" & params(3) & "'," & _
123:         "Województwo = '" & params(4) & "'," & _
124:         "Kod pocztowy = '" & params(5) & "'," & _
125:         "Telefon = '" & params(6) & "'," & _
126:         "WHERE Identyfikator = " & Ctype(e.Item.Cells(0). _
127:            Controls(1), Label).text
128:
129:      ExecuteStatement(strSQL)
130:      return blnGo
131:   end function
132:
133:   sub FillDataGrid(Optional EditIndex as integer=-1)
134:      'otwórz połączenie
135:      dim objCmd as new OleDbCommand _
136:         ("select * from tblUsers", Conn)
137:      dim objReader as OleDbDataReader
138:
139:      try
140:         objCmd.Connection.Open()
141:         objReader = objCmd.ExecuteReader()
142:      catch ex as Exception
143:         lblMessage.Text = "Wystąpił błąd odczytu " & _
144:            "z bazy danych."
145:      end try
146:
147:      dgData.DataSource = objReader
```

```
148:       if not EditIndex.Equals(Nothing) then
149:          dgData.EditItemIndex = EditIndex
150:       end if
151:
152:       dgData.DataBind()
153:
154:       objReader.Close
155:       objCmd.Connection.Close()
156:
157:    end sub
158:
159:    function ExecuteStatement(strSQL)
160:       dim objCmd as new OleDbCommand(strSQL, Conn)
161:
162:       try
163:          objCmd.Connection.Open()
164:          objCmd.ExecuteNonQuery()
165:       catch ex as Exception
166:          lblMessage.Text = "Wystąpił błąd aktualizacji bazy danych."
167:       end try
168:
169:       objCmd.Connection.Close()
170:    end function
171: </script>
```

Analiza Jest to spory fragment programu, ale naprawdę nie jest taki straszny, jak na to wygląda. Najpierw omówione zostaną metody, które odczytują dane z bazy danych i zapisują dane w obiekcie `DataGrid`. Dalej opisane zostaną metody aktualizujące bazę danych.

W wierszu 7., poza jakąkolwiek metodą, zadeklarowano obiekt `OleDbConnection`. Ponieważ obiekt ten nie przynależy do żadnej konkretnej metody (dotyczy raczej całej strony), może być wykorzystywany w kodzie dowolnej metody. Pierwszą wykonywaną metodą jest metoda `Page_Load` (wiersze 11. – 15.). Metoda ta sprawdza, czy dany formularz został już przesłany z powrotem; jeśli nie, to wywoływana jest kolejna metoda, `FillDataGrid`, która służy do odczytania danych z bazy danych i powiązania ich z obiektem `DataGrid`. Kod metody `FillDataGrid` znajduje się w wierszach 133. – 157. Należy zwrócić uwagę na opcjonalny parametr `EditIndex`; znaczenie tego parametru zostanie wyjaśnione w dalszej części niniejszego rozdziału. W wierszach 135. i 137. znajdują się deklaracje obiektów `OleDbCommand` oraz `OleDbDataReader`, w których zastosowano obiekt `OleDbConnection` (zadeklarowany w wierszu 7.).

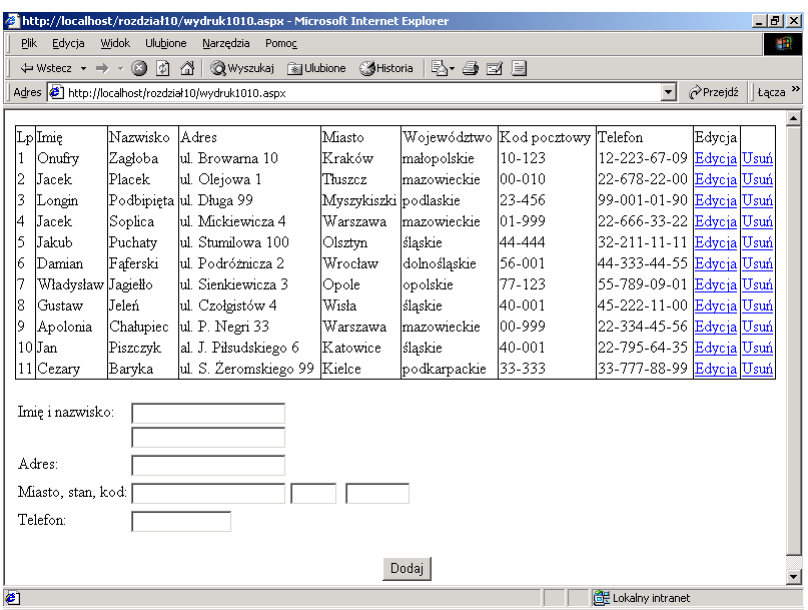
W następnej części programu zastosowano wyrażenie `try`. Wyrażenie to służy do wstępnego wykonania fragmentu kodu, pod warunkiem że nie wystąpią błędy. Jeśli błąd jednak wystąpi, można za pomocą wyrażenia `catch` (wiersz 142.) określić procedurę obsługi tego błędu i kontynuować wykonywanie metody. W przeciwnym razie nastąpiłaby awaria aplikacji. Do tej pory wymienione powyżej wyrażenia nie były stosowane, ponieważ przykładowe programy były proste. Jednak przy łączeniu się z dowolnym systemem, który nie należy do środowiska ASP.NET (w tym przypadku jest to baza danych), mogą wystąpić błędy. W przypadku profesjonalnych aplikacji konieczne jest zapobieganie, aby takie błędy nie powodowały awarii systemu lub co gorsza, by błędy te nie były zauważalne dla użytkownika.

Tak	Nie
Stosuj wyrażenia `try...catch...finally` w przypadku łączenia się z obiektem lub systemem, który nie należy do środowiska ASP.NET. Dotyczy to baz danych, komponentów zarządzanych (managed components), obiektów COM itp.	Nie licz na to, że nie stanie się nic złego ani nie sądź, że uwzględnione zostały już wszystkie możliwe problemy.

Jeśli powyższy opis wydaje się niezbyt jasny, nie należy się tym przejmować. Wyrażenie `try` zostało opisane dokładnie w rozdziale 20., „Testowanie stron ASP.NET".

W bloku wyrażenia `try` następuje łączenie się z bazą danych i próba zapisania danych w obiekcie `OleDbDataReader`. Jeśli próba nie powiedzie się, wyświetlony zostanie odpowiedni komunikat. W wierszu 147. podano źródło danych dla obiektu `DataGrid`, którym jest obiekt `DataReader`. Zanim jednak nastąpi powiązanie danych, należy jeszcze raz zwrócić uwagę na opcjonalny parametr `EditIndex`. Za każdym razem, kiedy użytkownik próbuje edytować daną pozycję, atrybutowi `EditItemIndex` obiektu `DataGrid` należy nadać odpowiednią wartość, a następnie powiązać dane. Sposób ten umożliwia nadawanie wartości atrybutowi `EditItemIndex`, dzięki czemu można uniknąć kłopotów związanych z przechodzeniem do trybu edytowania i koniecznością ponownego wiązania danych. Z tego powodu w wierszu 148. za pomocą wyrażenia `if` sprawdza się, czy podano parametr `EditIndex`. Jeśli tak, to atrybutowi `EditItemIndex` obiektu `DataGrid` nadawana jest odpowiednia wartość.

W końcowej części następuje zamknięcie obiektu `DataReader` i połączenia (wiersze 154. i 155.). W wyniku wykonania powyższego kodu strony okno przeglądarki powinno wyglądać podobnie jak na rysunku 10.10.



Rysunek 10.10. *Interfejs użytkownika przykładowej bazy danych*

Teraz należy dodać metody umożliwiające użytkownikowi edytowanie pozycji obiektu `DataGrid`:

- od wiersza 64. rozpoczyna się metoda obsługująca zdarzenie `Edit` obiektu `DataGrid`,
- od wiersza 68. rozpoczyna się metoda obsługująca zdarzenie `Delete` obiektu `DataGrid`,
- od wiersza 77. rozpoczyna się metoda obsługująca zdarzenie `Update` obiektu `DataGrid`,
- od wiersza 83. rozpoczyna się metoda obsługująca zdarzenie `Cancel` obiektu `DataGrid`,
- od wiersza 91. zaczyna się funkcja aktualizacji magazynu danych.

W przypadku poleceń `Edit` i `Cancel` wywołuje się po prostu metodę `FillDataGrid` z opcjonalnym parametrem, którym jest indeks edytowanej pozycji. Oto przykład:

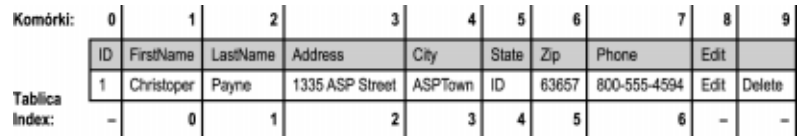
```
sub dgData_Edit(obj as object, e as DataGridCommandEventArgs)
   FillDataGrid(e.Item.ItemIndex)
end sub

sub dgData_Cancel(obj as object, e as DataGridCommandEventArgs)
   FillDataGrid(-1)
end sub
```

Metoda `Edit` przyporządkowuje atrybutowi `EditItemIndex` obiektu `DataGrid` pozycję wybraną przez użytkownika, metoda `Cancel` nadaje temu atrybutowi wartość `-1`, co oznacza zakończenie trybu edycji.

W przypadku polecenia `Update` trzeba być bardziej twórczym. Konieczne jest odczytywanie danych z pól tekstowych, które będą tworzone dynamicznie, kiedy użytkownik kliknie łącze `Edit`. W tym celu właśnie zastosowano pętlę `for` (wiersz 100.). W pętli tej odczytywana jest kolejno zawartość komórek wybranej pozycji obiektu `DataGrid` z pominięciem trzech ostatnich komórek (należy pamiętać, że komórki te nie zawierają danych — są to przyciski *Update*, *Cancel* oraz *Delete*), które potem zapisywane są w zmiennej `strText` (wiersz 101.). Jeśli odczytana wartość nie jest łańcuchem pustym, zapisuje się ją w tablicy, która będzie potem wykorzystana do aktualizowania bazy danych. Jeśli odczytana wartość jest łańcuchem pustym, procedura będzie zakończona i wyświetlony zostanie komunikat (wiersze 106. – 108.).

Należy zwrócić uwagę, że występują tu dwa wskaźniki pętli — `i` oraz `j`. Zmienna `i` jest wskaźnikiem kolejnych komórek danego wiersza, `j` jest wskaźnikiem tablicy parametrów. Użycie obydwu zmiennych jest konieczne, ponieważ wskaźniki komórek nie odpowiadają wskaźnikom tablicy. Na przykład, komórka „LastName” ma indeks 1, ale w tablicy parametrów ma mieć indeks 0. W przeciwnym przypadku wystąpią błędy typu „Indeks poza zakresem” (*index out of bounds*). Wartość zmiennej `j` jest to aktualny indeks tablicy. Powyższe zagadnienie ilustruje rysunek 10.11.



Komórki:	0	1	2	3	4	5	6	7	8	9
	ID	FirstName	LastName	Address	City	State	Zip	Phone	Edit	
	1	Christoper	Payne	1335 ASP Street	ASPTown	ID	63657	800-555-4594	Edit	Delete
Tablica Index:	–	0	1	2	3	4	5	6	–	–

Rysunek 10.11. *Niezgodność indeksów komórek i indeksów tablicy*

Jeśli którekolwiek z pól tekstowych byłoby puste, to wtedy zmiennej boolowskiej `blnGo` nadawana jest wartość `false` (wiersz 106.). Wtedy procedura aktualizowania magazynu danych zostanie zatrzymana, aby nie zapisać w bazie danych niepoprawnych. Do tego właśnie służy fragment kodu w wierszach 113. – 116:

```
if not blnGo then
   return false
   exit function
end if
```

Następnym etapem procesu aktualizowania bazy danych jest utworzenie odpowiedniego wyrażenia języka SQL. Ponieważ wszystkie nowe wartości są umieszczone w tablicy `param`, jest to łatwe, co widać w wierszach 118. – 127.

W wierszach 126. – 127. znajduje się klauzula `WHERE` wyrażenia `UPDATE`. Aktualizowane mają być tylko te pozycje, które zostały zmodyfikowane, więc w tym celu należy określić pole identyfikujące tych pozycji. Tak się dobrze składa, że w tym przypadku polem identyfikującym jest komórka 1. każdego z wierszy — „Identyfikator”. Odczytuje się zawartość tej komórki, wykonuje rzutowanie na typ `Label` (etykieta), a następnie odczytuje uzyskany tekst. Zatem, zgodnie z rysunkiem 10.10, jeśli edytowany jest pierwszy wiersz, to odpowiednie wyrażenie języka SQL ma postać następującą:

```
UPDATE tblUsers SET Imię = 'Christopher', Nazwisko =
➥'Payne', Adres = '1335 ASP Street', Miasto = 'ASPTown',
➥State = 'ID', ZIP = '83657', Telefon = '800-555-4594'
➥WHERE Identyfikator = 1
```

Na zakończenie należy wykonać wyrażenie aktualizujące. Robi się to za pomocą wywołania innej metody `ExecuteStatement`, którą zamieszczono w wierszach 159. – 170. Metoda ta to po prostu hermetyzacja kolejnych kroków tworzenia obiektu `OleDbCommand` i uruchamiania wyrażenia SQL.

Po raz kolejny użyte zostało wyrażenie `try...catch`, dzięki czemu wszystkie błędy, które mogą wystąpić, zostaną obsłużone. Aktualizację wykonuje się za pomocą metody `ExecuteNonQuery`.

Jako następna zostanie omówiona metoda obsługi zdarzenia `Delete` (wiersze 68. – 75.):

```
sub dgData_Delete(obj as object, e as _
   DataGridCommandEventArgs)
   dim strSQL as string = "DELETE FROM tblUsers " &_
      "WHERE Identyfikator = " & Ctype(e.Item.Cells(0). _
      Controls(1), Label).text

   ExecuteStatement (strSQL)

   FillDataGrid()
end sub
```

Metoda ta jest względnie prosta. Tworzy się wyrażenie języka SQL `DELETE`, w którym w klauzuli `WHERE` występuje pole `Identyfikator`, następnie wyrażenie jest wykonywane i ponownie zapisuje się dane w obiekcie `DataGrid`.

Pozostało tylko dodać nowe rekordy do bazy danych, kiedy użytkownik wypełni pola formularza i naciśnie przycisk *Dodaj*. Metoda `Submit` (wiersze 17. – 62.) przypomina opisaną już wcześniej metodę `Update`, z tą tylko różnicą, że zamiast polecenia `update` wykonywane jest polecenie `insert`.

Metoda `Submit` zostanie omówiona pokrótce. Należy przypomnieć tu umieszczanie wszystkich obiektów sterujących, które służą do wprowadzania danych, w obiekcie `Panel`, by można było odczytywać kolejno ich zawartość, tak samo jak było to w przypadku komórek obiektu `DataGrid`. Tym razem zamiast odczytywania kolejnych wartości z kolekcji `cells` obiektu `DataGridCommandEventArgs`, odczytuje się wartości z kolekcji `Controls` obiektu `Panel`. Również wyrażenie języka SQL ma trochę inną budowę. Poza tym, wszystko jest dokładnie takie samo, jak w przypadku metody `Update`.

Gratulujemy, aplikacja umożliwiająca użytkownikom aktualizowanie, dodawanie i usuwanie rekordy bazy danych jest gotowa. Kod aplikacji jest długi, ale w większości dość prosty.

Przy pisaniu aplikacji należy również dzielić program na moduły, co sprawia, że program jest bardziej przejrzysty, a modułów takich można używać wielokrotnie w różnych aplikacjach po niewielkich modyfikacjach.

Jak można było zauważyć, nastąpiła częściowa utrata kontroli nad sposobem wyświetlania pól w przypadku użycia metody `EditCommand`. Na przykład, pola tekstowe były dłuższe niż to konieczne. W przyszłości w przypadku stosowania obiektu sterującego `DataGrid` lepsze może być zastąpienie obiektów sterujących `BoundColumn` i metody `EditCommand` obiektami `TemplateColumns` i polami tekstowymi. Wtedy będzie mniej zdarzeń do obsłużenia, a kontrola nad wykonaniem programu będzie większa. Ale to już jest kwestia indywidualnego stylu programisty.

To nie jest ASP!

Programiści, którzy znają klasyczne środowiska ASP i ADO, mogą zastanawiać się, jak do tego wszystkiego pasuje stary dobry obiekt `Recordset`.

Chociaż obiekt `ADODB Recordset` należy już do przeszłości, idea jednak pozostała. W gruncie rzeczy, obiekt `DataReader` jest to po prostu obiekt `Recordset`, w którym kursor przesuwa się tylko w przód, podczas gdy `DataSet` jest to obiekt `Recordset`, w którym zastosowano kursor dynamiczny lub kursor, którego położenie można zmieniać (scrollable cursor). Oczywiście, nowe obiekty ADO.NET mają więcej funkcji, ale podstawowy zakres funkcji jest podobny.

Korzystanie z obiektu `Recordset`, niezależnie od rodzaju kursora, wymaga zablokowania innym użytkownikom dostępu do danych, ponieważ obiekt jest stale połączony z bazą danych. Jest to sposób raczej niezbyt efektywny, szczególnie w przypadku wielu użytkowników korzystających z bazy danych równocześnie. Obiekt `DataSet` jest magazynem danych całkowicie odłączonym (disconnected data store), to znaczy, że dane po prostu są odczytywane z bazy danych, po czym połączenie jest zamykane. Nie trzeba wtedy utrzymywać połączenia z bazą danych, nie trzeba blokować dostępu do danych. Oprócz tego, obiekt `DataSet` jest lepszym przedstawieniem magazynu danych, ponieważ, jak wiadomo z niniejszego rozdziału, może zawierać wiele tabel (co jest niemożliwe w przypadku obiektu `Recordset`), relacje oraz odwzorowania. Jest to nowy sposób przetwarzania danych, który może usunąć wiele kłopotów związanych z korzystaniem z baz danych w środowisku ADO. Ma to jednak swoją cenę. Dlatego też do dyspozycji jest obiekt `OleDbDataReader`, zbliżony do obiektu `Recordset`. Różnica jest taka, że obiekt `OleDbDataReader` jest całkowicie obiektowy i korzysta z mechanizmów mocnej kontroli typów danych (type-safe data).

Obiekty `OleDbConnection` i `OleDbCommand` są zbliżone do obiektów `ADODB Connection` oraz `Command` występujących w tradycyjnym środowisku ADO, więc zmiana powinna być bezbolesna. Takie pojęcia jak transakcje oraz obiekty `Parameter`, znane ze środowiska ASP, występują również w środowisku ASP.NET. Więcej szczegółów na ten temat w rozdziale 12.

Pomimo wielu różnic pomiędzy ADO i ADO.NET, występują także podobieństwa ułatwiające pracę programistom znającym środowisko ADO. Większość zmian wprowadzono w środowisku ADO.NET, aby zapewnić większą spójność, elastyczność i funkcjonalność modelu danych.