# Access 2002. Projektowanie baz danych. Księga eksperta


Autorzy: Stephen Forte, Thomas Howe, Kurt Wall, Paul Kimmel, Russ Mullen
Tłumaczenie: Paweł Gonera
ISBN: 83-7197-669-0
Tytuł oryginału: Access 2002 Development Unleashed
Format: B5, stron: 666
Zawiera CD-ROM

Książka ta jest napisana przez programistów Accessa oraz dla programistów Accessa. Obecnie wiele książek pisanych jest przez zawodowych instruktorów i pisarzy. Autorzy tej książki zajmują się tym samym co Ty — tworzą w Accessie aplikacje dla swoich użytkowników. W książce tej dzielimy się naszym doświadczeniem. Skupiamy się na tych właściwościach Accessa, których najprawdopodobniej będziesz używał – nie opisujemy w nieskończoność każdego przycisku i każdej zakładki okna dialogowego. Niezależnie od tego, czy piszesz zawodowo aplikacje w Accessie, zmieniasz narzędzie z Visual Basica lub innego języka programowania, myślisz o zastosowaniu Accessa jako interfejsu użytkownika dla serwera bazy danych SQL Server, Oracle, Informix lub Sybase, bądź też jesteś po prostu niedoświadczonym użytkownikiem Accessa, chcącym poznać jego szczegóły techniczne — jest to książka dla Ciebie.
Jest to jedyna książka o Accessie, która tak dokładnie opisuje pracę z bazą Oracle. Za pierwszym razem, gdy musiałem stworzyć aplikację Accessa korzystającą z Oracle, byłem bardzo zawiedzony brakiem literatury na ten temat. Doświadczenie Jamesa Ralstona w łączeniu Accessa z bazą Oracle zostało wykorzystane w jednym z rozdziałów. Ponadto znajdziesz tu dwa pełne rozdziały na temat współdziałania Accessa z SQL Serverem oraz wiele fragmentów innych rozdziałów, opisujących ten sam problem.
Najlepiej opisuje wykorzystanie Accessa w Internecie. Rozwój popularności Internetu wpłynął nie tylko na wiele głównych firm przemysłu komputerowego — w tym Microsoftu – ale dzięki niemu niespodziewanie zmienił się także sposób tworzenia oprogramowania. Wiele książek o Accessie nadal traktuje Internet jako problem do przemyślenia; ta książka opisuje Sieć obok innych środowisk programowania. Istotnie, Internet jest kolejną platformą programowania, a Access jest bazą danych, pozwalającą na tworzenie aplikacji ukierunkowanych na dane. Opisaliśmy również sposoby stworzenia od podstaw serwera WWW na komputerze z Windows 2000/NT i korzystanie z Windows 95 z zainstalowanym Personal Web Server. Zagadnienia opisane w tej książce obejmują wiele problemów związanych z zastosowaniami Accessa w Internecie: począwszy od XML, ASP, DAP i komponentów sieciowych Office, na najnowszych trendach w programowaniu skończywszy.

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

# Spis treści

# Rozdział 7.
# Zaawansowane ADO

**W tym rozdziale:**

- Użycie dostawcy OLE DB dla Accessa 2002
- Uzyskanie dostępu do danych w bazach nierelacyjnych poprzez ADO
- Zaawansowana obróbka danych przy użyciu ADO
- Definiowanie danych przy użyciu ADOX

Teraz, kiedy już umiesz używać obiektów ADO i porównałeś je z DAO, przyjrzyj się tym cechom ADO, które pomogą Ci tworzyć aplikacje w Accessie 2002.

## Użycie dostawcy OLE DB dla Jet w Accessie 2002

Gdy używasz ADO w aplikacjach Accessa, najprawdopodobniej będziesz korzystał z dostawcy OLE DB dla Jet (użycie dostawcy OLE DB dla SQL Server omówiono w rozdziale 15.). Dostawca ten pozwala na bezpośredni dostęp do plików typu MDB. Zdolność bezpośredniej komunikacji ze źródłem danych to wielki krok naprzód dla ADO. Aby użyć dostawcy OLE DB dla Jet, musisz podać jego nazwę i pełną ścieżkę dostępu do bazy danych jako część ciągu połączeniowego. Aby zachować przejrzystość, umieszczono w ADO właściwość `Provider`, której odpowiednie ustawienie umożliwi użycie dostawcy dla Jet. Jego unikatowym `CLASSID` w ciągu połączenia jest *Microsoft. Jet.OLEDB.4.0*. Przedstawiony tu przykład pokazuje, jak tworzyć połączenie do bazy Accessa przy użyciu dostawcy OLE DB dla Jet:

```
Dim Connection As ADODB.Connection
Set Connection = New ADODB.Connection

   Connection.Provider = "Microsoft.Jet.OLE.DB.4.0"
   Connection.ConnectionString = "data source= C:\databases\sample.mdb"
   Connection.Open
```

Jeśli zachodzi konieczność określenia innych informacji dotyczących połączenia (np. hasła bazy danych czy pliku grupy roboczej), dołącz je do ciągu połączeniowego, poprzedzając średnikiem. W tabeli 7.1 znajdziesz charakterystyczne dla dostawcy wła-

ściwości połączenia, których możesz użyć w ciągu połączeniowym. Inne charakterystyczne dla dostawcy właściwości, znajdujące się w zbiorze właściwości połączenia, zostaną zignorowane.

**Tabela 7.1.** *Właściwości ciągów połączenia charakterystyczne dla dostawcy OLE DB dla Microsoft Jet*

| Nazwa | Opis |
|---|---|
| `Compact ReclaimedSpace Amount` | Średnia ilość miejsca odzyskiwana podczas kompaktowania bazy danych |
| `Connection Control` | Kontroluje dostęp użytkownika do bazy danych |
| `Create System Database` | Wykorzystuje się ją do tworzenia systemowej bazy danych podczas tworzenia nowego źródła danych |
| `Database Locking Mode` | Tryb używany podczas blokowania bazy danych; więcej informacji na ten temat znajdziesz w rozdziale 21., dotyczącym obsługi wielu użytkowników. Zauważ, że baza może być otwarta jednocześnie tylko w jednym trybie. Użytkownik, który jako pierwszy otwiera bazę, określa jej tryb blokowania |
| `Database Password` | Hasło używane do otwarcia bazy danych. Od hasła użytkownika różni się tym, że hasło bazy danych dotyczy pliku, a nie konkretnego użytkownika. Więcej informacji na ten temat znajdziesz w rozdziale 21. |
| `Don't Copy Locale on Compact` | Właściwość wykorzystywana do sterowania kopiowaniem informacji narodowych podczas kompaktowania bazy |
| `Encrypt Database` | Wykorzystuje się ją do sterowania szyfrowaniem bazy w trakcie kompaktowania |
| `Engine Type` | Określenie silnika, który aktualnie jest wykorzystywany w celu uzyskania dostępu do danych |
| `Exclusive Asynch Delay` | Określa maksymalną długość przerwy w czasie zapisu asynchronicznego |
| `Flush Transaction Timeout` | Określa czas oczekiwania, po którym dane asynchroniczne są zapisywane na dysk |
| `Global Bulk Transactions` | Określa, czy masowe operacje SQL są wykonywane w formie transakcji. Właściwość ta wskazuje ustawienie domyślne dla wszystkich operacji w aktualnym połączeniu |
| `Global Partial Bulk Ops` | Właściwość ta określa zachowanie Jet w przypadku błędu w masowych operacjach SQL DML. Może zostać unieważniona przez ustawienie właściwości Jet na `OLEDB:Partial Bulk Ops` |
| `Implict Commit Synch` | Definiuje wewnętrzny sposób zapisu transakcji — synchroniczny lub asynchroniczny |
| `Lock Delay` | Określa opóźnienie (w milisekundach), po którym nastąpi ponowna próba wykonania blokady |
| `Lock Retry` | Określa ilość prób nałożenia blokady |
| `Max Buffer Size` | Podaje maksymalną ilość pamięci dla bufora Jet, przeznaczoną na zapisywane dane |
| `Max Locks Per File` | Definiuje maksymalną ilość blokad w bazie danych; wartością domyślną jest 9500 |

**Tabela 7.1.** *Właściwości ciągów połączenia charakterystyczne dla dostawcy OLE DB dla Microsoft Jet (ciąg dalszy)*

| Nazwa | Opis |
|---|---|
| `New Database Password` | Określa nowe hasło bazy danych |
| `ODBC Command Time Out` | Podaje czas (w milisekundach), po którym wykonanie zdalnego polecenia ODBC nie powiedzie się |
| `Page Locks to Table Lock` | Określa ilość blokad strony, zanim blokady te zostaną zamienione na blokadę tabeli |
| `Page Timeout` | Ilość milisekund po których baza danych uważa dane z bufora za nieaktualne. |
| `Recycle Long-Valued Pages` | Określa, czy strony BLOB będą „agresywnie" odzyskiwane |
| `Registry Path` | Ścieżka dostępu do klucza rejestru dla Jet. Nie zawiera znacznika *HKEY_LOCAL_MACHINE*. Wartość ta może zostać zamieniona na drugorzędną lokalizację w celu przechowywania wartości rejestru aplikacji, które nie są współdzielone z innymi aplikacjami korzystającymi z Jet na tym komputerze. Przykładowo, ustawienie dla Accessa 2002 to: *SOFTWARE\Microsoft\Office\9.0\Access\Jet\4.0\Engines* |
| `Reset ISAM Stats` | Wskazuje, czy liczniki wydajności powinny być zerowane po każdym pobraniu danych |
| `Shared Asynch Delay` | Określa opóźnienie (w milisekundach) zapisu asynchronicznego w przypadku otwarcia bazy danych w trybie wielodostępu |
| `System database` | Podaje lokalizację systemowej bazy Jet, która ma być używana przez określonych użytkowników. Powoduje to unieważnienie wartości ustalonej w rejestrze lub klucza rejestru *systemdb*, używanego wraz z Jet — `OLEDB:Registry Path`. Właściwość ta może zawierać ścieżkę dostępu do pliku |
| `Transaction Commit Mode` | Określa, czy zapis podczas zatwierdzania transakcji jest asynchroniczny, czy synchroniczny |
| `User Commit Sync` | Określa, czy zmiany w transakcji są zapisywane w sposób synchroniczny, czy asynchroniczny |

Oprócz właściwości umieszczonych w tabeli 7.1 silnik bazy danych Microsoft Jet posiada kilka opcji, których ustawienie określa zachowanie silnika. Opcje te często mają bezpośredni wpływ na jego wydajność. Domyślnie, po uruchomieniu Jet używane są wartości znajdujące się w rejestrze, w kluczu *\HKEY_LOCAL_MACHINES\Software\Microsoft\Jet*. Istnieje jednak możliwość tymczasowego unieważnienia tych ustawień. W przypadku ADO wartości te stanowią część ciągu połączenia. Stałe ciągu połączenia znajdują się w tabeli 7.2.

Następujący fragment kodu powoduje otwarcie zabezpieczonej bazy, określając jej hasło i plik grupy roboczej:

```
Dim Connection As ADODB.Connection
Set Connection = New ADODB.Connection

Connection.Provider = "Microsoft.Jet.OLE.DB.4.0"
Connection.ConnectionString = "data source= C:\databases\sample.mdb"&_
  ";Jet OLEDB:Database Password=supersecret " & _
  "; OLEDB:System database=" & "c:\windows\system\system.mdw"
Connection.Open
```

**Tabela 7.2.** *Opcje ciągu połączenia ADO*

| DAO | ADO |
|---|---|
| dbPageTimeout | Jet OLEDB:Page Timeout |
| dbSharedAsyncDelay | Jet OLEDB:Shared Async Delay |
| dbExclusiveAsyncDelay | Jet OLEDB:Exclusive Async Delay |
| dbLockRetry | Jet OLEDB:Lock Retry |
| dbUserCommitSync | Jet OLEDB:User Commit Sync |
| dbImplicitCommitSync | Jet OLEDB:Implicit Commit Sync |
| dbMaxBufferSize | Jet OLEDB:Max Buffer Size |
| dbMaxLocksPerFile | Jet OLEDB:Max Locks Per File |
| dbLockDelay | Jet OLEDB:Lock Delay |
| dbRecycleLVs | Jet OLEDB:Recycle Long-Valued Pages |
| dbFlushTransactionTimeout | Jet OLEDB:Flush transaction Timeout |

## CurrentProject.Connection

Gdy chcesz uzyskać dostęp do aktualnej bazy poprzez kod ADO, Access umożliwia Ci „pójście na skróty", oddając do Twojej dyspozycji odnośnik do obiektu, `Connection` — `CurrentProject.Connection`. Używając tej składni, możesz z łatwością przydzielić aktualnej bazie obiekt `Connection`. Odnośnik ten pozwala zaoszczędzić czas, jeśli chcesz otworzyć zestaw rekordów.

```
Dim Recordset As ADODB.Recordset
Set Recordset = New ADODB.Recordset
Recordset.Open "Customers", CurrentProject.Connection
```


Zalecamy ostrożność podczas korzystania z `CurrentProject.Connection`. Pamiętaj, że polecenie to nie może być używane w innym środowisku niż Access 2002. Jeśli zamierzasz wykorzystać swój kod w innej aplikacji niż Access (np. Visual Basic), do utworzenia połączenia musisz użyć dłuższej składni.

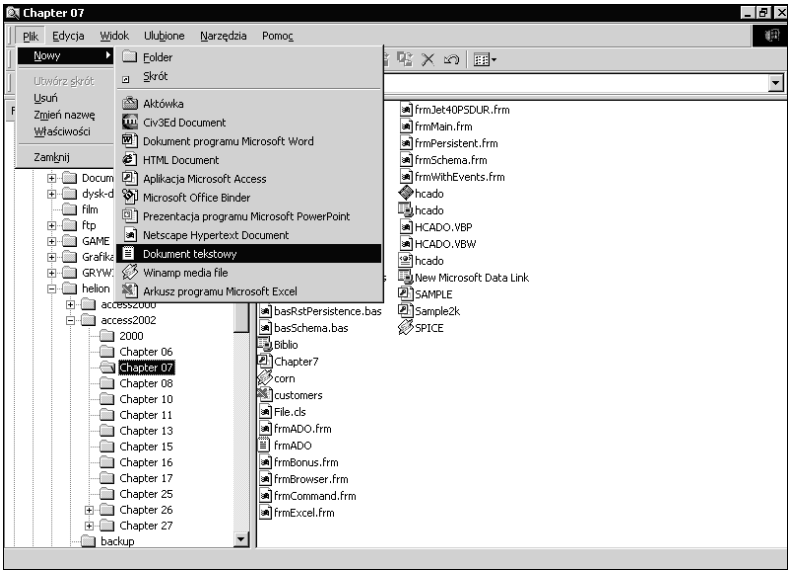
## Wykorzystanie Microsoft Data Links do podłączenia się do bazy danych

Dziś, gdy obiekty ADO są podstawowym sposobem uzyskiwania dostępu do danych, twórcy Accessa uznali, że programiści potrzebują standardowego sposobu obsługiwania, ładowania, otwierania i zarządzania informacjami o połączeniu OLE DB, przypominającego dawne narzędzie do zarządzania i administrowania sterownikami ODBC. Możliwość tę dają Microsoft Universal Data Links (UDL). Mechanizm UDL umożliwia zapisywanie informacji o połączeniu w pliku UDL, a następnie otwarcie obiektu `Connection` w ADO, na podstawie informacji zapisanych w tym pliku. Możliwość ta będzie bardzo przydatna, gdy będziesz chciał przetestować podłączenie bazy danych do różnych komputerów bez zmiany kodu.

Mechanizm Microsoft Universal Data Links składa się z:

- graficznego interfejsu użytkownika, służącego do tworzenia połączeń OLE DB;
- interfejsu automatyzacji.

Zanim zaczniesz używać UDL w aplikacjach VB lub VBA, musisz najpierw je utworzyć. Tworzenie UDL nie jest skomplikowane. Uruchom Eksploratora Windows, z menu *Plik* wybierz *Nowy*, a następnie *Dokument tekstowy* (rysunek 7.1).



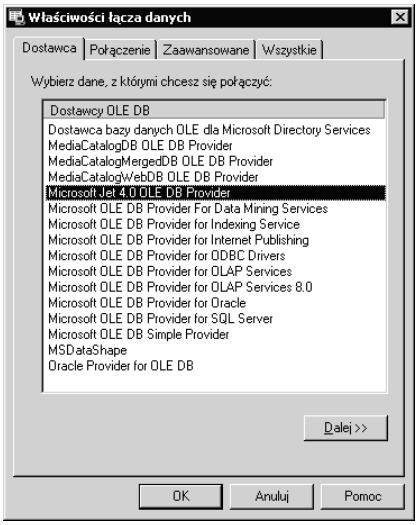

**Rysunek 7.1.** *Tworzenie nowego pliku UDL*

Po utworzeniu pliku UDL kliknij go dwukrotnie, aby rozpocząć edycję. Przedstawiony na rysunku 7.2 pierwszy ekran zawiera listę wszystkich zainstalowanych na Twoim komputerze dostawców OLE DB. Wybierz jednego z nich i kliknij *Dalej*, aby przejść do zakładki *Połączenie*. Teraz możesz określić wszystkie, charakterystyczne dla dostawcy informacje dotyczące połączenia (rysunek 7.3). W naszym przykładzie użyliśmy dostawcy OLE DB dla Jet, więc musieliśmy określić ścieżkę dostępu, identyfikator użytkownika i hasło (gdybyśmy wybrali SQL Server, musielibyśmy podać nazwę serwera i bazy danych). Następna zakładka, *Zaawansowane*, służy do określenia właściwości dostawcy, ustalanych zwykle przez takie właściwości, jak *CommandTimeout*. Ostatnia zakładka — *Wszystkie* — zawiera podsumowanie wszystkich wybranych właściwości.

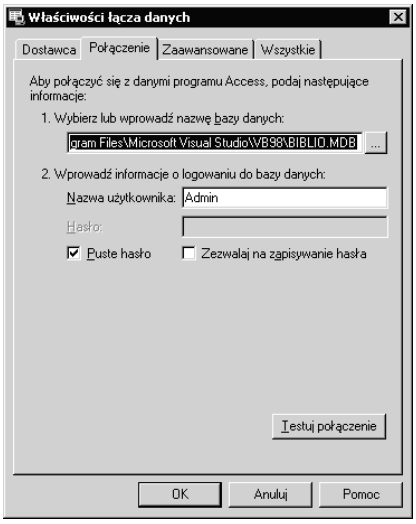
Teraz, gdy utworzyłeś już plik UDL, możesz umieścić go w kodzie VBA, aby utworzyć w ADO obiekt `Connection`. Służąca do tego składnia jest prosta. Utwórz obiekt `Connection` — tak, jak to zwykle robisz — i użyj metody `Open`. Następnie wskaż plik UDL za pomocą identyfikatora `File Name`:

```
Dim conn As ADODB.Connection
Set conn = New ADODB.Connection
Conn.Open "File Name=c:\chapter7.udl;"
```

**Rysunek 7.2.** *Wybór dostawcy OLE DB*




**Rysunek 7.3.** *Wprowadzanie informacji o połączeniu*





Fragment kodu na wydruku 7.1 znajduje się na dołączonej do książki płycie CD. Pokazuje on, jak utworzyć obiekt `Connection` oparty na pliku UDL, umieszczonym w folderze bazy danych Northwind. Łączysz się z bazą i wypełniasz okno zawartością tabeli *Customers*. Bardzo przydatną cechą tego kodu jest to, że jeśli będziesz chciał użyć bazy Northwind w wersji SQL, wystarczy zmienić właściwości pliku UDL i uruchomić kod ponownie.

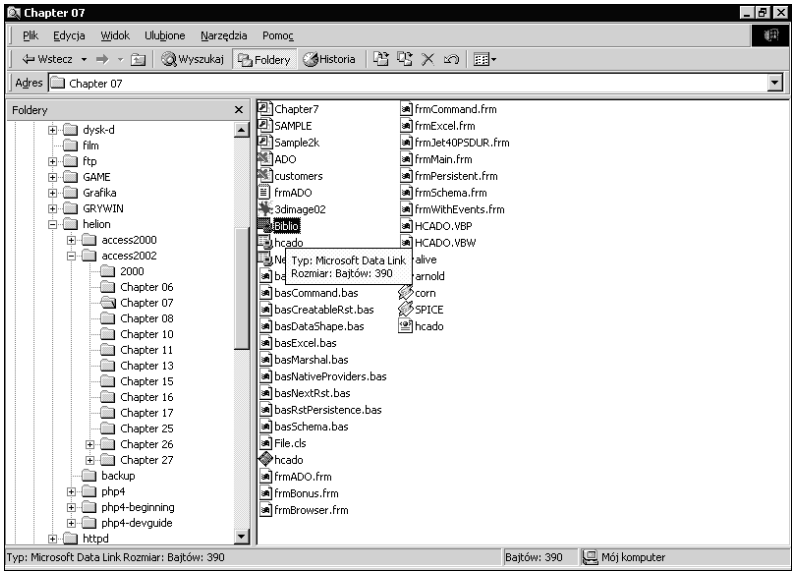
**Wydruk 7.1.** *Użycie UDL w kodzie VBA*

```
Private Sub OpenViaLink()

On Error GoTo Except

  Dim Connection As ADODB.Connection
  Set Connection = New ADODB.Connection
```

```
  Connection.Open "File Name=c:\temp\biblio.udl;"

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset
  Recordset.Open "Select * From publishers", Connection

  Do Until Recordset.EOF
    Debug.Print Recordset.Fields(1)
    Recordset.MoveNext
  Loop

  Recordset.Close
  Set Recordset = Nothing

  Connection.Close
  Set Connection = Nothing

Except:
  MsgBox Err.Description
End Sub
```

## Zarządzanie plikami UDL

Aby zmienić plik UDL, należy w programie Windows Explorer kliknąć dwukrotnie jego ikonę lub wpisać w oknie *Uruchom*, wywoływanym z menu *Start*, pełną ścieżkę do pliku UDL (rysunek 7.4).

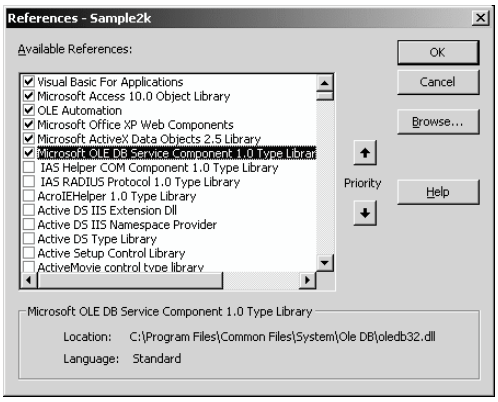



**Rysunek 7.4.** *Dwukrotne kliknięcie pliku UDL uruchamia okno właściwości łącza danych*

## Programowe zarządzanie plikami UDL

Ponieważ pliki UDL posiadają swój własny interfejs automatyzacji, możesz nimi zarządzać i kontrolować je programowo. Aby użyć UDL w kodzie VBA, ustaw odwołanie na

typ biblioteki o nazwie Microsoft OLE DB Service Component 1.0 Type Library (rysunek 7.5).




**Rysunek 7.5.** *Ustawianie odnośnika na Microsoft OLE DB Service Component 1.0 Type Library*

Po ustawienie odwołania możesz rozpocząć zarządzanie plikami UDL. Dodatkowo możesz poprosić użytkownika o podanie informacji o połączeniu OLE DB poprzez okna dialogowe UDL (możliwość ta jest przydatna w prototypowych aplikacjach lub wersjach demonstracyjnych produktów przeznaczonych dla użytkowników zaznajomionych z funkcjonowaniem Microsoft Data Links). Możesz tego dokonać, wykorzystując metodę `PromptNew` obiektu `DataLinks`. Kod na wydruku 7.2 prosi użytkownika o podanie informacji o połączeniu OLE DB poprzez okno dialogowe UDL, a następnie łączy się z bazą danych i otwiera zestaw rekordów.

**Wydruk 7.2.** *Programowe UDL*

```
Private Sub OpenUDLDialog()

On Error GoTo Except

  Dim UDL As MSDASC.DataLinks
  Set UDL = New MSDASC.DataLinks

  Dim Connect As String
  Connect = UDL.PromptNew

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  ' Zakładamy, że użytkownik wybrał bazę danych Biblio.mdb
  Recordset.Open "Select * From Publishers", Connect
  MsgBox Recordset.Fields(1)
  Exit Sub

Except:
  MsgBox Err.Description
End Sub
```

# Uzyskanie poprzez ADO dostępu do danych w bazach nierelacyjnych

Nawiązując do omówionego w rozdziale 6. wykorzystania Accessa do komunikowania się z danymi nierelacyjnymi (np. danymi arkusza Excel) trzeba powiedzieć w tym miejscu, że ADO posiada kilka zaawansowanych opcji, które ułatwią Ci pracę z tym typem danych. Przyjrzymy się tym opcjom. Są nimi:

- lista użytkowników Jet;
- tworzenie zestawów rekordów;
- kształtowanie danych.

## Lista użytkowników Jet

Dostawca OLE DB dla Jet umożliwi Ci otwarcie zestawu rekordów, zawierającego nazwy użytkowników aktualnie zalogowanych do bazy, której nazwę określisz w połączeniu. Dzięki temu nie będziesz już musiał korzystać z dostarczanego z poprzednimi wersjami Jet narzędzia LDBView. Kod na wydruku 7.3 przedstawia sposób utworzenia zestawów rekordów z aktualnymi użytkownikami bazy danych. Użyliśmy w nim metody `OpenSchema` obiektu `Connection` z parametrem charakterystycznym dla aparatu JET/JOLT. Zauważ, że ADO nie podpowiada Ci żadnych stałych. Będziesz musiał ręcznie wprowadzić wartość GUID (`{947bb102-5d43-11d1-bdbf-00c04fb92675}`). Pozostaje nam mieć nadzieję, że niedogodność ta zostanie poprawiona w kolejnej wersji obiektów ADO.

**Wydruk 7.3.** *Użycie listy użytkowników*

```
Sub UserRoster()

  On Error GoTo Except

  Dim Connection As ADODB.Connection
  Set Connection = New ADODB.Connection
  Connection.Provider = "Microsoft.Jet.OLEDB.4.0"
  Connection.ConnectionString = "file name=c:\temp\biblio.udl"
  Connection.Open

  Dim Recordset As ADODB.Recordset
  Set Recordset = Connection.OpenSchema(adSchemaProviderSpecific, , _
  "{947bb102-5d43-iidl-bdbf-00c04fb92675}")

  Do Until Recordset.EOF
    Debug.Print Recordset!COMPUTER_NAME
    Debug.Print Recordset!LOGIN_NAME
    Debug.Print Recordset!CONNECTED
    Debug.Print Recordset!SUSPECTED STATE
```

```
    Recordset.MoveNext
  Loop

  Exit Sub
Except:
  MsgBox Err.Description
End Sub
```

## Tworzone zestawy rekordów

Obiekty ADO dają Ci możliwość tworzenia zestawów rekordów „z niczego". Możesz teraz korzystać z metod `Fields.Append` i `AddNew`, aby dodawać pola i dane do zestawu rekordów. Technika ta nadaje się świetnie do tworzenia procedur pobierających dane z nierelacyjnych źródeł (np. folderu z plikami) i zwracających zestaw rekordów. Dzięki ADO aplikacja może przez cały czas współpracować z zestawami rekordów, a jej kod jest dużo prostszy. Poniższy przykład przedstawia sposób utworzenia zestawu rekordów, zawierającego nazwy plików znajdujących się w określonym folderze. Jeśli umieścisz ten kod w module, będziesz go mógł użyć w wielu aplikacjach. My na przykład wykorzystaliśmy go do sporządzenia listy wszystkich plików typu ZIP w katalogu serwera FTP, a następnie przejrzenia ich i utworzenia strony, z której można te pliki pobrać. Przewagą tej techniki nad dostawcą NT Active OLE DB jest fakt, iż działa ona w systemie Windows 95. Wydruk 7.4 pokazuje, w jaki sposób można otworzyć zestaw rekordów oparty na liście nazw plików w folderze.

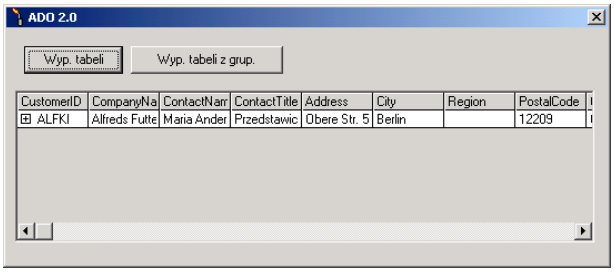
**Wydruk 7.4.** *Tworzenie zestawu rekordów z nierelacyjnych danych*

```
Sub CreatableRst_Files(Optional Path As String = "c:\temp")

On Error GoTo Except

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Recordset.CursorLocation = adUseClient
  Recordset.Fields.Append "FileName", adVarChar, 255, adFldRowID
  Recordset.Fields.Append "Extention", adChar, 3, adFldFixed
  Recordset.Open , , adOpenStatic, adLockBatchOptimistic

  If Right(Path, 1) <> "\" Then Path = Path & "\"
  Path = Dir(Path & "*.*", vbNormal)

  Do While strPath > ""
    Recordset.AddNew Array("FileName", "Extention"), _
        Array(strPath, Right(strPath, 3))

      Path = Dir
  Loop

  Recordset.MoveFirst

  Do Until Recordset.EOF
    Debug.Print Recordset!FileName
    Recordset.MoveNext
```

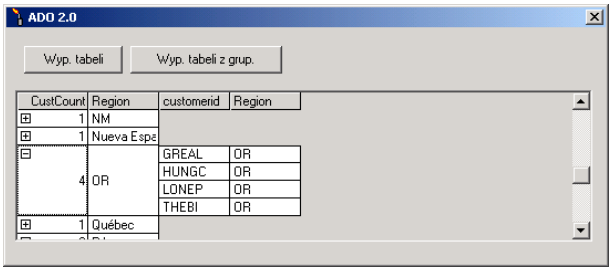
```
    Loop

    Exit Sub

Except:
    MsgBox Err.Description
End Sub
```

## Kształtowanie danych

Umożliwienie przeglądania danych w zhierarchizowany sposób jest w obiektach ADO bardzo proste. Możesz użyć nowej zdolności ADO, o nazwie *Data Shaping* (ang. kształtowanie danych), aby przeglądać nadrzędne i podrzędne rekordy w jednym zestawie. Możesz użyć dostawcy MSDataShape OLE DB do tworzenia hierarchii opartych na relacjach i hierarchii grupowania. Tworzenie tego typu hierarchicznych zestawów rekordów może ułatwić dość pracochłonną obróbkę hierarchicznych danych. Przykładowo, na rysunku 7.6 znajduje się grupowa hierarchia wszystkich zamówień klientów z bazy Northwind, przeglądana w nowej siatce, o nazwie Microsoft Hierarchical FlexGrid (wersja OLAP popularnej FlexGrid). Formant ten dołączony jest do pakietu Office 2002 w wersji Developer Edition oraz do Visual Basic 6 (nasze rysunki i listingi zostały stworzone w VB 6 i używają bazy danych Accessa jako źródła danych). Zwróć uwagę na znak plus, znajdujący się przy każdym z rekordów. Umożliwia on rozszerzanie ich i przeglądanie ich szczegółów, co pokazuje rysunek 7.7. Zanim powstał dostawca MSDataShape OLE DB, utworzenie takiej siatki wymagało napisania wielu linijek kodu.




**Rysunek 7.6.** *Zestaw rekordów uzyskany poprzez dostawcę MSDataShape OLE DB w Visual Basic*




**Rysunek 7.7.** *Zestaw rekordów GrupujWedług w Visual Basic*

**Wydruk 7.5.** *Tworzenie rekordu z ukształtowanymi danymi i wypełnianie siatki MSHFlexGrid w VB*

```
Sub CubeGroupHierarchy()

  Dim ConnectionString As String
```

```
  ConnectionString = "Provider=MSDataShape" & _
       ";data provider=Microsoft.Jet.OLEDB.4.0" & _
       ";data source=" & App.Path & "\sample.mdb"

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Recordset.Source = "shape {Select customerid, " & _
        "Region from Customers} rst1 " & _
        "COMPUTE COUNT (rst1.customerid) AS CustCount, rst1 By Region "

  Recordset.Open , ConnectionString
  Set frmADO.MSHFlexGrid1.Recordset = Recordset

End Sub
```

Jak to widać w tym fragmencie kodu, musisz użyć specjalnej, powiązanej z *Data Shaping* składni SQL. Kompletną listę zastosowań tej składni znajdziesz w dostarczanym wraz z Visual Basic 6 i pakietem Office Developer 2000 dokumentacji MSDN oraz na stronie WWW o adresie *http://www.microsoft.com/data*. Możesz również użyć tego kodu do operacji *Data Shaping* i przeniesienia danych do programu Microsoft Excel. Kod na wydruku 7.6 przedstawia sposób, w jaki — za pomocą poprzedniego przykładu — można przenieść dane do arkusza programu Excel.

**Wydruk 7.6.** *Użycie Data Shaping w programie Microsoft Excel*

```
Function GetDatabase() As String
  ' Zmień tę funkcję tak, aby zwracała ścieżkę do przykładowej bazy danych
  Getatabase = "c:\temp\sample.mdb"
End Function

Sub CUBRelation()

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Dim ConnectionString As String
  ConnectionString = "Provider=MSDataShape;" & _
    ";data provider=Microsoft.Jet.OLEDB.4.0" & _
    ";data source=" & GetDatabase()

  Recordset.Source = _
    "shape {Select * from customers where Customerid='ALFKI'}"&_
    " Append ({Select * From Orders} As rsOrders " & _
     "RELATE customerid to customerid)"

  Recordset.ActiveConnection = ConnectionString

  Recordset.Open , , adOpenStatic, adLockBatchOptimistic

  ShowRecordset Recordset

End Sub

Sub ShowRecordset(ByVal Recordset As ADODB.Recordset, _
        Optional SkipClear As Boolean = False)
```

```
    Dim Child As ADODB.Recordset
    Dim Col As ADODB.Field
    Dim CurrentColumn As Integer
    Dim CurrentRow As Integer


    If Not SkipClear Then
      Cells.Select
      Selection.ClearContents
    End If

    Do Until Recordset.EOF
      CurrentColumn = CurrentColumn + 1
      If Not SkipClear Then
        CurrentRow = 1
      Else
        CurrentRow = 2
      End If


      For Each Col In Recordset.Fields
        If col.Type <> adChapter Then
          Cells(intCurrentColumn, intCurrentRow) = _
                col.Name & ": " & col.Value
        Else
          Set Child = col.Value
          ShowRecordset Child, True
        End If

        CurrentRow = CurrentRow + 1
      Next

      Recordset.MoveNext
    Loop

    Cells(1, 1).Select

  End Sub
```

# Zaawansowana obróbka danych przy użyciu ADO

W ostatnim rozdziale pokazaliśmy, jak tworzyć zestawy rekordów zawierające dane i jak korzystać z obiektów `Command`. Teraz pokażemy kilka dodatkowych możliwości obiektów ADO. Przyjrzymy się kolejno:

- modyfikowaniu danych w zestawie rekordów;
- trwałym zestawom rekordów.

## Modyfikowanie danych w zestawie rekordów

Aby modyfikować dane w zestawie rekordów, musisz jedynie przejść do niego i rozpocząć edycję. Po przejściu do kolejnego rekordu lub użyciu metody `Update` dokonane przez Ciebie zmiany zostaną zapisane. Jest to duży krok naprzód w porównaniu z obiektami DAO, gdzie musiałeś najpierw użyć polecenia `Edit`, a następnie obowiązkowo użyć polecenia `Update`. Kod na wydruku 7.7 pokazuje, w jaki sposób można dokonać edycji danych.

**Wydruk 7.7.** *Edycja rekordu*

```
Sub EditRecords()

  On Error GoTo Except

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Dim SQL As String
  SQL = "Select * From Customers Where CustomerID=1"

  Recordset.Open SQL, CurrentProject.Connection, _
                 adOpenKeyset, adLockOptimistic

  Recordset!CompanyName = "Nowa nazwa"
  Recordset.Update

  Exit Sub

Except:
  MsgBox Err.Description
End Sub
```


Możesz otworzyć zestaw rekordów bazujący na wyrażeniu SQL wymagającym klucza podstawowego; zajmie to znacznie mniej czasu, niż otwieranie całej tabeli i w niej dopiero wyszukiwanie żądanego rekordu.


Ponieważ zmiany dokonywane w zestawie rekordów uwzględniane są bez użycia polecenia `Update` (wystarczy do tego przejście do kolejnego rekordu), zalecamy ostrożność podczas edycji danych.

## Dodawanie rekordu

Dodawanie rekordu przebiega w prawie identyczny sposób, jak jego edycja. Musisz otworzyć zestaw rekordów i użyć metody `AddNew`. W przypadku operacji `Edit` zastosowanie mają te same zasady, co w przypadku `Update`. Po dodaniu rekordu znajdziesz się w aktualnym rekordzie. Wydruk 7.8 zawiera przykład dodawania rekordu do tabeli:

**Wydruk 7.8.** *Dodawanie rekordu do tabeli*

```
Sub AddRecords()

  On Error GoTo Except
```

```
  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Recordset.Open "Customers", CurrentProject.Connection, _
    adOpenDynamic, adLockOptimistic, adCmdTableDirect

  Recordset.AddNew
  ' Należy zdefiniować algorytm generowania unikatowego identyfikatora
  Recordset!CustomerID = 3
  Recordset!Name_Pre = "Pan"
  Recordset!Name_First = "Jan"
  Recordset!Name_Last = "Nowak"
  Recordset.Update
  MsgBox "ID nowego autora: " & Recordset!CustomerID

  Recordset.Close
  Set Recordset = Nothing
  Exit Sub
Except:
  MsgBox Err.Description
End Sub
```

Być może zauważyłeś, że otwarłeś tabelę ze stałą `adCmdTableDirect`. Jeśli użyjesz tej stałej z typem kursora `adOpenDynaset` i blokowaniem optymistycznym, osiągniesz taki sam efekt, jak przy użyciu polecenia `dbOpenTable` w DAO. Tabela 7.3 zawiera wszystkie opcje kursorów i blokowania ADO oraz ich odpowiedniki w DAO:

**Tabela 7.3.** *Kombinacje obiektów ADO dla Accessa*

| Typ kursora | Opcje i typy blokowania | Odpowiednik w DAO |
|---|---|---|
| adOpenForwardOnly | adLockReadOnly | dbOpenSnapShot<br>dbForwardOnly |
| AdOpenKeySet | adLockReadOnly | Brak odpowiednika |
| AdOpenKeySet | adLockPessimistic | dnOpenDynaset |
| AdOpenKeySet | adLockOptimistic | dnOpenDynaset |
| AdOpenKeySet | adLockBatchOptimistic | dnOpenDynaset |
| AdOpenStatic | adLockReadOnly | dnOpenDynaset |
| adOpenDynamic | adLockOptimistic,<br>adCmdTableDirect | dbOpenTable |

## Trwałe zestawy rekordów

Jeśli musisz zadbać również o obsługę użytkowników nie podłączonych do sieci, pojęcie trwałości rekordów stanowić będzie dla Ciebie fantastyczny podarunek od firmy z Redmond. Teraz masz bowiem możliwość zapisania zestawu rekordów na dysku i obrabiania go w aplikacji VB lub VBA. Później będziesz mógł ponownie włączyć ten zestaw do bazy. Aby tego dokonać, otwórz zestaw rekordów ADO z typem blokowania `adLockBatchOptimistic`, używając kursora `client`. Następnie użyj metody `Save` obiektu `Recordset`. Musisz określić nazwę pliku i format, w jakim ma być zapisany. ADO oferuje następujące możliwości:

- adPersistADTG;
- adPersistXML.

Jeśli wybierzesz format adPersistADTG lub adPersistXML, będziesz mógł później z łatwością otworzyć ten plik w ADO. Wydruk 7.9 pokazuje, jak otworzyć zestaw rekordów, zmienić rekord, zapisać zmiany do pliku, otworzyć rekord ponownie i włączyć go z powrotem do bazy.

**Wydruk 7.9.** *Zapisywanie zestawu rekordów na dysku, jego ponowne otwarcie oraz włączenie do bazy danych*

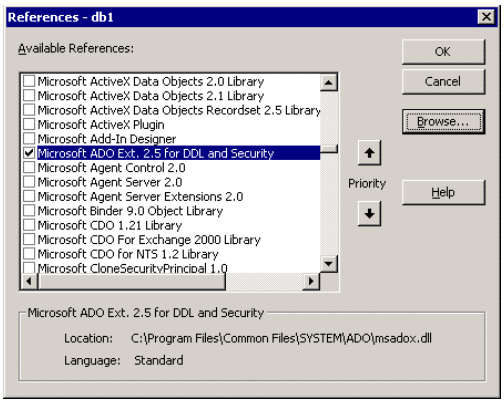
```
Sub Persist()

  On Error GoTo Except

  Dim SQL As String
  Dim Msg As String
  Dim NewValue As String


  SQL = "Select * From Customers Where CustomerID='ALFKI'"

  Dim Connection As ADODB.Connection
  Set Connection = New ADODB.Connection
  Connection.Provider = "Microsoft.Jet.OLEDB.3.51"
  Connection.ConnectionString = "data source=" & _
    App.Path & "\sample.mdb"

  Connection.Open

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  Recordset.CursorLocation = adUseClient
  Recordset.Open SQL, _
    Connection, adOpenStatic, adLockBatchOptimistic

  Msg = "Proszę podać nową nazwę firmy dla: " & _
        Recordset!CompanyName & vbNewLine & _
        "Identyfikator klienta=" & Recordset!CustomerID

  NewValue = InputBox(Msg)

  Recordset.Update "CompanyName", NewValue

  On Error Resume Next
    Kill App.Path & "\hcado.dat"

  On Error GoTo Except
  Recordset.Save App.Path & "\hcado.dat", adPersistADTG

  Recordset.Close
  Set Recordset = Nothing

  MsgBox "Zapisane do pliku: " & App.Path & "\hcado.dat", vbInformation
```

```
  Exit Sub

Except:
  MsgBox Err.Description
End Sub

Sub Resynch()

  On Error GoTo Except

  Dim SQL As String
  SQL = "Select * From Customers Where CustomerID='ALFKI'"

  Dim Recordset As ADODB.Recordset
  Set Recordset = New ADODB.Recordset

  ' Otwarcie recordsetu zapisanego w pliku
  Recordset.Open App.Path & "\hcado.dat", , _
                 adOpenStatic, adLockBatchOptimistic, adCmdFile

  Dim Msg As String
  Msg = "Poprzednie wartości: " & _
        Recordset!CompanyName.OriginalValue

  Msg = Msg & vbNewLine & _
    "Wartości w pliku       : " & Recordset!CompanyName

  MsgBox Msg, vbInformation, "Poprzednie i zapisane wartości"

  Dim Connection As ADODB.Connection
  Set Connection = New ADODB.Connection
  Connection.Provider = "Microsoft.Jet.OLEDB.3.51"
  Connection.ConnectionString = "data source=" & _
              App.Path & "\sample.mdb"
  Connection.Open

  Recordset.ActiveConnection = Connection
  Recordset.UpdateBatch

  MsgBox "Baza danych została zsynchronizowana!", vbInformation

  Exit Sub

Except:
  MsgBox Err.Description
End Sub
```

# Definiowanie danych przy użyciu ADOX

Korzystając z obiektów ADO, możesz z łatwością dokonywać definiowania danych. Aby tworzyć i modyfikować tabele, kwerendy i inne obiekty związane z danymi, na przykład indeksy, musisz skorzystać ze specjalnej biblioteki ADO o nazwie ADOX.

Aby użyć ADOX, musisz ustawić odnośnik na *Microsoft ADO Ext. 2.5 For DDL and Security*, wybierając z menu *Tools* pozycję *References* (rysunek 7.8).




**Rysunek 7.8.** *Microsoft ADO Ext. 2.5 For DDL and Security*

Gdy ustawiłeś już ten odnośnik, jesteś gotów do korzystania z ADOX.

## Obiekt Catalog

Jeśli korzystasz z ADOX, wszystko, co robisz, związane jest z obiektem `Catalog`. Obiekt ten oznacza bazę danych. Pracując w Accessie, musisz jedynie ustawić jej właściwość — `ActiveConnection` — na prawidłowy obiekt `Command`.

```
Dim Catalog As ADOX.Catalog
Set Catalog = New ADOX.Catalog
'Wskaż obiektowi Catalog
'aktualną bazę danych
Catalog.ActiveConnection = CurrentProject.Connection
```

## Tworzenie bazy danych

Aby utworzyć nową, pustą bazę danych przy użyciu ADOX, musisz użyć metody `Create` obiektu `Catalog` i podać ścieżkę dostępu do nowej bazy (wydruk 7.10).

**Wydruk 7.10.** *Tworzenie bazy danych przy użyciu ADOX*

```
Sub CreateDataBase()

  On Error GoTo Except

    Dim Catalog As ADOX.Catalog
    Set Catalog = New ADOX.Catalog
    Catalog.Create "provider=Microsoft.JET.OLEDB.4.0;" & _
                   "data source=C:\newdb.mdb"

    MsgBox "Baza danych utworzona!", vbInformation
    set Catalog = Nothing

    Exit Sub
```

```
Except:
    MsgBox Err.Description
End Sub
```

## Tworzenie tabel i pól

Aby utworzyć tabelę i jej pola, możesz utworzyć obiekt ADOX o nazwie `table` i dołączyć do niego kolumny (wydruk 7.11).

**Wydruk 7.11.** *Tworzenie tabel i pól*

```
Sub CreateTable_JOLT4O()

  On Error GoTo Except

  Dim Catalog As ADOX.Catalog
  Set Catalog = New ADOX.Catalog
  Catalog.ActiveConnection = CurrentProject.Connection

  Dim Table As ADOX.Table
  Set Table = New ADOX.Table

  Table.Name = "Customer_ADO"
  Table.Columns.Append "CustomerID", adInteger
  Table.Columns.Append "Name_Prefix", adWChar, 25.
  Table.Columns.Append "Name_First", adWChar, 2
  Table.Columns.Append "Name_Last", adWChar, 50

  Catalog.Tables.Append Table

  Set Table = Nothing
  Set Catalog = Nothing

  Exit Sub

Except:
  MsgBox Err.Description
End Sub
```

### Tworzenie dołączonej tabeli

Aby utworzyć w Accessie dołączoną tabelę, musisz utworzyć tabelę nie dołączając do niej żadnych pól, a następnie ustawić dwie właściwości:

- `Jet OLE DB:Link Datasource`;
- `JET OLE DB:Remote Table Name`.

Jako `Link Datasource` podaj ścieżkę dostępu do bazy danych, do której chcesz się dołączyć. Jako `Remote Table Name` ustaw nazwę tej bazy. Cały przykład znajduje się na wydruku 7.12.

**Wydruk 7.12.** *Tworzenie dołączonej tabeli*

```
Sub CreateAttachedJetTable()
  On Error GoTo Except

  Dim Catalog As ADOX.Catalog
  Set Catalog = New ADOX.Catalog
  Catalog.ActiveConnection = CurrentProject.Connection

  Dim Table As ADOX.Table
  Set Table = New ADOX.Table

  Table.Name = "Customers_Linked"
  Set Table.ParentCatalog = Catalog

  Table.Properties("Jet OLEDB:Link Datasource") =   "c:\temp\newdb.mdb"
  Table.Properties("Jet OLEDB:Remote Table Name") = "Customers"

  Catalog.Tables.Append Table
  Set Catalog = Nothing

  Exit Sub
Except:
  MsgBox Err.Description
End Sub
```

## Ustawianie właściwości pola

Kiedy utworzysz pole, możesz użyć w ADOX obiektu `Column`, aby ustawić właściwości charakterystyczne dla tego pola. Na przykład możesz ustawić typ pola na *Autonumerowanie*, nadając właściwości `AutoIncrement` wartość `True`:

```
Dim Column As ADOX.Column
Set Column = Table.Columns("CustomerID")
Column.Properties("Autincrement") = True
```

Istnieją także inne właściwości, które możesz ustawić dla obiektu `Column`. Umieściliśmy je w tabeli 7.4, wraz z ich odpowiednikami w DAO.

**Tabela 7.4.** *Właściwości obiektu Field*

| Pole DAO | | Obiekt Column w ADOX | |
|---|---|---|---|
| **Właściwość** | **Wartość** | **Właściwość** | **Wartość** |
| `Attributes` | `dbAutoIncrField` | `AutoIncrement` | `True` |
| `Attributes` | `dbFixedField` | `ColumnAttributes` | `adColFixed` |
| `Attributes` | `dbHyperlinkField` | `Jet OLEDB:Hyperlink` | `True` |
| `Attributes` | `dbSystemField` | | |
| `Attributes` | `dbVariableField` | `ColumnAttributes` | inna niż `adColFixed` |

## Tworzenie indeksu

Tworząc indeks w ADOX, będziesz musiał użyć obiektu klucza. Po jego utworzeniu musisz dołączyć do niego pole, a następnie określić, czy ma to być klucz podstawowy, czy unikatowy oraz jaka ma być kolejność sortowania:

```
Set Index = New ADOX.Index
Index.Name = 'Primary Key'
Index.Columns.Append "CustomerID"
Index.Columns("CustomerID").SortOrder = adSortDescending
Index.IndexNulls = adIndexNullsDisallow
Index.PrimaryKey = True
Table.Indexes.Append Index
```

Właściwość `IndexNulls` ustawiona jest domyślnie na `adIndexNullsDisallow`, co oznacza, że wartości `Null` nie są w indeksie dopuszczalne i w przypadku, gdy pole w indeksie będzie taką wartość zawierać, nie zostanie dokonany żaden wpis do indeksu. W tabeli 7.5 znajdują się wartości akceptowane przez właściwość `IndexNulls`, wraz z ich odpowiednikami w DAO.

**Tabela 7.5.** *Wartości dla właściwości IndexNulls*

| DAO | ADOX | Opis | |
|---|---|---|---|
| Wymagane `Ignore Nulls` | `Index Nulls` | | |
| `True` | `False` | `AdIndexNullsDisallow` | Wartość `Null` nie jest akceptowana w polu indeks; wpis do indeksu nie zostanie dokonany |
| `False` | `True` | `AdIndexNullsIgnore` | Wartość `Null` dopuszczona w polu indeks; wpis do indeksu nie zostanie dokonany |
| `False` | `False` | `AdIndexNullsIgnoreAny` | Wartość `Null` dopuszczona w polu indeks; wpis do indeksu zostanie dokonany |

Kod na wydruku 7.13 łączy te wszystkie przykłady, tworząc tabelę pola *Autonumerowanie* i indeks klucza głównego w tej tabeli.

**Wydruk 7.13.** *Tworzenie tabeli z polem Autonumerowanie i indeksem klucza głównego*

```
Sub CreateAutoNumberPK()

  On Error GoTo Except

  Dim Catalog As ADOX.Catalog
  Set Catalog = New ADOX.Catalog
  Catalog.ActiveConnection = CurrentProject.Connection

  Dim Table As New ADOX.Table
  Set Table = New ADOX.Table

  Table.Name = "Customer_ADO"
  Table.ParentCatalog = Catalog

  Table.Columns.Append "CustomerID", adInteger
  Table.Columns("CustomerID").Properties("Autoincrement") = True
```

```
  Table.Columns.Append "Name_Prefix", adWChar, 25
  Table.Columns.Append "Name_First", adWChar, 25
  Table.Columns.Append "Name_Last", adWChar, 50

  Catalog.Tables.Append Table

  Dim Index As ADOX.Index
  Set Index = New ADOX.Index

  Index.Name = "PrimaryKey"
  Index.Columns.Append "CustomerID"
  Index.Columns("CustomerID").SortOrder = adSortDescending

  Index.IndexNulls = adIndexNullsDisallow
  Index.PrimaryKey = True

  Table.Indexes.Append Index

  Set Table = Nothing
  Set Index = Nothing
  Set Catalog = Nothing

  Exit Sub

Except:
    MsgBox Err.Description
End Sub
```

## Tworzenie relacji w ADOX

Aby utworzyć relację w ADOX, musisz utworzyć obiekt `Key`, określić jego właściwości, a następnie dołączyć ten klucz do obiektu `Table`. Użyj następującej składni:

```
Catalog.Tables("Products").Keys.Append Key
```

Kod na wydruku 7.14 wykonuje wszystkie te czynności oraz tworzy relacje.

**Wydruk 7.14.** *Tworzenie relacji*

```
Sub CreateForeignKey()

  On Error GoTo Except

  Dim Catalog As ADOX.Catalog
  Set Catalog = New ADOX.Catalog
  Catalog.ActiveConnection = CurrentProject.Connection

  Dim Key As ADOX.Key
  Set Key = New ADOX.Key

  Key.Name = "CategoriesProducts"
  Key.Type = adKeyForeign
  Key.RelatedTable = "Categories"
  Key.Columns.Append "CategoryID"
  Key.Columns("CategoryID").RelatedColumn = "CategoryID"
  Key.UpdateRule = adRICascade
```

```
  Catalog.Tables("Products").Keys.Append Key
  Exit Sub

Except:
    MsgBox Err.Description
End Sub
```

# Tworzenie kwerend w ADOX

Aby utworzyć kwerendę w ADOX, musisz użyć pustego obiektu `Command` i ustawić jego właściwość `CommandText`, a następnie dołączyć go do aktualnego obiektu `catalog`:

```
Command.CommandText = "Select * FROM Categories"
Catalog.Views.Append "qryCategories", Command
```

Gdy odnosisz się do kwerendy w Accessie, musisz pamiętać, że Microsoft Jet 4.0 zarządza wszystkimi tabelami, kwerendami, polami i indeksami za Ciebie. Jet definiuje kwerendy Accessa na dwa sposoby — jako widoki lub procedury.

Przez *widok* rozumiemy zwracającą wiersze, nieparametryczną kwerendę, podczas gdy *procedura* oznacza wszystkie inne kwerendy.

## Tworzenie widoku

Microsoft Jet zajmie się za Ciebie definiowaniem kwerendy. Aby utworzyć widok, należy utworzyć kwerendę poprzez obiekt `Command` (wydruk 7.15).

**Wydruk 7.15.** *Tworzenie widoku w ADOX*

```
Sub CreateView()
  On Error GoTo Except

    Dim Command As New ADODB.Command
    Command.CommandText = "Select * FROM Categories"
    Dim Catalog As New ADOX.Catalog
    Catalog.ActiveConnection = CurrentProject.Connection

    Catalog.Views.Append "qryCategories", Command
    Set Catalog = Nothing
    Exit Sub
Except:
  MsgBox Err.Description
End Sub
```

## Tworzenie procedury

Tworzenie procedury jest równie proste, jak tworzenie widoku. Wystarczy zdefiniować w obiekcie `Command` właściwość `Commandtext` i w takiej postaci dołączyć go do obiektu `catalog` (wydruk 7.16).

**Wydruk 7.16.** *Tworzenie procedury*

```
Sub CreateProcedure()
  On Error GoTo Except

    Dim Catalog As New ADOX.Catalog
    Catalog.ActiveConnection = CurrentProject.Connection

    Dim Command As New ADODB.Command
    Command.CommandText = "Parameters [@Region] Text;" & _
      "Select * from Employees where Region = [@Region]"

    Catalog.Procedures.Append "qryEmployeesRegion", Command

    Set Command = Nothing
    Set Catalog = Nothing

    Exit Sub
Except:
  MsgBox Err.Description
End Sub
```

## Modyfikowanie wyrażenia SQL kwerendy

Aby zmodyfikować wyrażenie SQL kwerendy, musisz najpierw utworzyć obiekt `command`, oparty na aktualnej kwerendzie w bazie danych, a następnie zmodyfikować właściwość `commandtext` i zresetować obiekt `command` (wydruk 7.17).

**Wydruk 7.17.** *Zmiana wyrażenia SQL*

```
Sub ModifyQuerySQL()

  On Error GoTo Except

    Dim Catalog As New ADOX.Catalog
    Catalog.ActiveConnection = CurrentProject.Connection

    Dim Command As New ADODB.Command
    Set Command = Catalog.Procedures("qryEmployeesRegion").Command

    Command.CommandText = _
      "Parameters [forms]![frmOrder]![txtRegion] Text;" &_
      "Select * from Employees " & _
      "Where Region = [forms]![frmOrder]![txtRegion] "

    Set Catalog.Procedures("qryEmployeesRegion").Command = Command

    Set Command = Nothing
    Set Catalog = Nothing
    Exit Sub

Except:
  MsgBox Err.Description
End Sub
```

Zauważyłeś zapewne, że w kodzie ADO znajdującym się na wydruku 7.17 ustawienie właściwości `Command` obiektu `Procedure` na zmodyfikowany obiekt `Command` powoduje zapisanie zmian. Gdybyśmy nie umieścili tego ostatniego kroku, zmiany nie zostałyby umieszczone na stałe w bazie danych. Różnica ta wynika z tego, że obiekty `Command` zostały zaprojektowane jako tymczasowe. Fakt ten ma duże znaczenie, gdyż w DAO obiekty `Querydefs` uważane były za trwałe. Może Ci się wydawać, że przedstawione tutaj fragmenty kodu ADO są równoznaczne. Oto pierwszy z nich:

```
Set Command = Catalog.Procedures("qryEmployeesRegion").Command
Comman.CommandText = "Parameters [@Region] Text;" & _
  "Select * from Employees where Region = [@Region]"
Set Catalog.Procedures("qryEmployeesRegion").Command = Command
```

a to drugi:

```
Catalog.Procedures("qryEmployeesRegion").CommandText = _
  "Parameters [@Region] text;" & _
  "Select * from Employees where Region = [@Region] "
```

Mimo tego, że można odnieść wrażenie, iż działają one podobnie, drugi fragment kodu nie spowoduje rzeczywistej aktualizacji kwerendy w bazie. W drugim przykładzie możesz użyć obiektu `Command` z jego nowym wyrażeniem SQL, zapisanym w kodzie VBA. Jednak po zakończeniu pracy zmiany w SQL nie zostaną zapisane w bazie. W pierwszym przypadku natomiast — ponieważ nakazujesz zapisanie `Command` — zmiany zostaną uwzględnione w bazie danych.