Kapitel 6
Verbesserter Datenzugriff

Es gibt mehrere Möglichkeiten, in VFP-Anwendungen auf andere Daten als die von VFP zuzugreifen, beispielsweise auf den SQL Server oder Oracle: Remote Ansichten, SQL Pass-Through, ADO und XML. VFP 8 führt eine überraschende neue Technologie mit Namen CursorAdapter ein, die den Zugriff auf remote Daten im Vergleich zu früheren Versionen deutlich vereinfacht.

Mehr und mehr VFP-Entwickler speichern ihre Daten nicht in Tabellen von VFP, sondern beispielsweise im SQL Server oder Oracle. Dafür gibt es viele Gründe, einschließlich der Zerstörbarkeit von VFP-Tabellen, der Sicherheit, der Größe der Datenbank und der Firmenstandards. Microsoft hat den Zugriff auf Daten, die nicht in VFP gespeichert sind, mit jeder Version vereinfacht und ermutigt auch dazu, indem mit VFP MSDE (Microsoft Data Engine, eine abgespeckte Version des SQL Server) ausgeliefert wird. Dies geschieht seit der Version 7.

Allerdings war der Zugriff auf eine Backend-Datenbank nie so einfach wie der Einsatz von VFP-Tabellen. Zusätzlich gibt es unterschiedliche Mechanismen für den Zugriff:

- Remote Ansichten, die auf ODBC-Verbindungen basieren.
- Funktionen von SQL Pass-Through (SPT), beispielsweise SQLCONNECT(), SQLEXEC() und SQLDISCONNECT(), die auch auf ODBC-Verbindungen basieren.
- ActiveX Data Objects (ADO), die OLE DB Providern für Datenbank-Engines ein objektorientertes Frontend bereitstellen.
- XML, ein schlanker, plattformunabhängiger Mechanismus für den Datentransport.

Wenn Sie mit diesen Mechanismen gearbeitet haben, werden Sie bemerkt haben, dass sich keine zwei Mechanismen gleichen. Das bedeutet, dass Sie sich in jeden einzelnen Mechanismen einarbeiten müssen und die Konvertierung einer bestehenden Anwendung von einem Mechanismus auf einen anderen ist keine einfache Aufgabe.

Um dieses Problem zu beseitigen, hat Microsoft in VFP 8 eine neue Basisklasse eingeführt: CursorAdapter. CursorAdapter ist eines der wichtigsten neuen Features in VFP.

- Die Klasse erleichtert den Einsatz von ODBC, ADO oder XML, auch wenn Sie mit diesen Technologien nicht vertraut sind.
- Sie bietet eine konsistente Schnittstelle zu remoten Daten, unabhängig vom von Ihnen gewählten Zugriffsmechanismus.
- Sie erleichtert den Wechsel von einem Zugriffsmechanismus zum anderen.

Hier ein Beispiel zum letzten Punkt. Nehmen Sie an, Sie haben eine Anwendung, die ODBC mit CursorAdapter einsetzt, um auf Daten des SQL Servers zuzugreifen. Stattdessen wollen Sie ADO einsetzen. Alles, was Sie dafür tun müssen, ist der Wechsel des DataSourceType in CursorAdapter, die Änderung der Verbindung zur Backend-Datenbank und Sie sind fertig. Der Rest der Komponenten in der Anwendung weiß davon nichts; sie sehen immer noch den gleichen Cursor, unabhängig vom Datenzugriff.

Der Name CursorAdapter ist gut gewählt; die Klasse fungiert als Adapter zwischen einer Datenquelle und einem Cursor von VFP. CursorAdapter erstellt und verwaltet einen Cursor von VFP. Aber Ihre Formulare, Berichte und der Code arbeiten immer noch auf dem Cursor, um Daten anzuzeigen und zu aktualisieren.

Die Eigenschaften, Ereignisse und Methoden von CursorAdapter

Lassen Sie uns am Beginn der Betrachtung von CursorAdapter die Eigenschaften, Ereignisse und Methoden untersuchen.

DataSourceType

Diese Eigenschaft ist sehr wichtig; sie entscheidet über das Verhalten der Klasse insgesamt sowie darüber, welche Arten von Werten in einige andere Eigenschaften geschrieben werden müssen. Die gültigen Auswahlmöglichkeiten, die den Mechanismus anzeigen, den Sie für den Datenzugriff verwenden wollen, sind: „Native“ (was anzeigt, dass Sie die nativen Tabellen von VFP einsetzen), „ODBC“, „ADO“ oder „XML“.

DataSource

Damit ist der Datenzugriff gemeint. VFP ignoriert diese Eigenschaft, wenn DataSourceType auf „Native“ oder „XML“ gesetzt ist. Für ODBC setzen Sie DataSource auf eine gültige ODBC-Verbindung (beachten Sie, dass Sie die Verbindung selbst verwalten müssen). Wenn Sie mit ADO arbeiten, muss

DataSource ein ADO Recordset enthalten, dessen Eigenschaft ActiveConnection auf ein geöffnetes ADO Connection-Objekt verweist (auch dies müssen Sie selbst verwalten).

Alias

Wie bei normalen Cursorobjekten enthält diese Eigenschaft den Alias des Cursors, der mit dem CursorAdapter verbunden ist.

UseDEDataSource

Diese Eigenschaft legt fest, ob CursorAdapter die Eigenschaften DataSourceType und DataSource der Klasse DataEnvironment nutzt. Ist diese Eigenschaft auf True gesetzt (Vorgabewert ist False), können Sie die Eigenschaften DataSourceType und DataSource ignorieren, da CursorAdapter stattdessen die Eigenschaften von DataEnvironment einsetzt (VFP 8 fügt auch der Klasse DataEnvironment die Eigenschaften DataSourceType und DataSource hinzu). Wenn Sie vorhaben, dass alle CursorAdapter in einer DataEnvironment die gleiche ODBC-Verbindung nutzen, würden Sie diese Eigenschaft auf True setzen und die Eigenschaften DataSourceType und DataSource von DataEnvironment nutzen, um die ODBC-Verbindung anzugeben.

SelectCmd

Dies ist der Befehl zum Empfangen der Daten. Bei allen Arten der DataSource mit Ausnahme von XML handelt es sich um einen Befehl SQL SELECT (beispielsweise SELECT * FROM CUSTOMERS) oder um eine gespeicherte Prozedur (beispielsweise EXEC GetCustomersByID 'ALFKI'). Im Fall von XML kann es sich entweder um einen gültigen XML-String oder um einen Ausdruck (beispielsweise um eine Funktion oder Methode) handeln, der einen gültigen XML-String zurückliefert. In jedem Fall nutzt VFP einen internen Aufruf von XMLTOCURSOR(), um XML in einen Cursor von VFP umzuwandeln.

CursorSchema

Diese Eigenschaft enthält die Struktur des Cursors im gleichen Format, das Sie auch in einem Befehl CREATE CURSOR(alles zwischen den Klammern in einem solchen Befehl) einsetzen würden. Hier ein Beispiel: CUST_ID C(6), COMPANY C(30), CONTACT C(30), CITY C(25). Obwohl es möglich ist, dies leer zu lassen und CursorAdapter mitzuteilen, die Struktur beim Erstellen des Cursors selbst festzulegen, ist es besser, CursorSchema auszufüllen. Auf der einen Seite erhalten Sie, wenn CursorSchema leer oder falsch ist, entweder

Fehlermeldungen, wenn Sie das DataEnvironment eines Formulars zu öffnen, oder Sie sind nicht in der Lage, Felder von CursorAdapter auf das Formular zu ziehen, um Steuerelemente zu erstellen. Außerdem haben Sie die Möglichkeit, exakt festzulegen, wie die Struktur des Cursors aussehen soll. Wenn Sie beispielsweise vorhaben, ein Feld vom Typ DateTime des SQL Servers in ein Feld vom Typ Datum im Cursor von VFP umzuwandeln, geben Sie für dieses Feld in CursorSchema „D" für den Datentyp ein.

Da VFP die Werte von Eigenschaften, die in das Eigenschaften-Fenster eingegeben werden, auf 255 Zeichen begrenzt, müssen Sie eventuell den Wert für diese Eigenschaft im Code eingeben. Glücklicherweise erledigt dies der CursorAdapter Builder, der mit VFP ausgeliefert wird (siehe den Abschnitt „Generatoren" weiter unten in diesem Kapitel) für Sie automatisch.

AllowDelete, AllowInsert, AllowUpdate und SendUpdates

Diese Eigenschaften, die als Vorgabe alle auf True stehen, entscheiden darüber, ob Löschungen, Einfügungen und Änderungen vorgenommen werden können und ob die Änderungen an die Datenquelle gesandt werden.

KeyFieldList, Tables, UpdatableFieldList und UpdateNameList

Diese Eigenschaften dienen dem gleichen Zweck wie die gleich benannten Eigenschaften von CURSORSETPROP() für Ansichten. Sie sind erforderlich, wenn Sie wollen, dass VFP die Datenquelle mit den im Cursor gemachten Änderungen automatisch aktualisiert. KeyFieldList ist ein kommaseparierte Liste von Feldern (ohne Aliasnamen), die den Primärschlüssel für den Cursor bilden. Tables ist eine kommaseparierte Liste von Tabellen, auf denen der Cursor basiert. UpdatableFieldList ist eine kommaseparierte Liste von Feldern (ohne Aliasnamen), die geändert werden können. UpdateNameList ist eine kommaseparierte Liste die die Feldnamen des Cursors mit denen in der Tabelle abgleicht. Das Format für UpdateNameList ist wie folgt: CUSORFIELDNAME1 TABLE. FIELDNAME1, CURSORFIELDNAME2 FIELD.FIELDNAME2,... Beachten Sie auch, dass, wenn UpdateFieldList den Namen den Primärschlüssels der Tabelle nicht enthält (da Sie nicht wollen, dass dieses Feld geändert wird), der Primärschlüssel trotzdem in UpdateNameList enthalten sein muss, oder die Aktualisierungen schlagen fehl.

*Cmd, *CmdDataSource, *CmdDataSourceType

Wenn Sie steuern wollen, wie VFP Datensätze in der Datenquelle löscht, einfügt oder aktualisiert, können Sie diesen Eigenschaftengruppen die entsprechenden Werte zuweisen (ersetzen Sie den Stern durch Delete, Insert oder

Update). Wenn Sie beispielsweise zum Löschen von Datensätzen eine gespeicherte Prozedur einsetzen, setzen Sie DeleteCmd auf etwas wie „EXEC DeleteCustomer CustomerID“ und stellen DeleteCmdSource und DeleteCmdDataSource Type auf die entsprechenden Werte für die Verbindung.

ConversionFunc

Diese Eigenschaft gibt Umwandlungsfunktionen an, die den Feldern hinzugefügt werden, wenn eine automatische Aktualisierung durchgeführt wird. Das Format für ConversionFunc ist folgendes: CURSORFIELDNAME1 FUNCTION, CURSORFIELDNAME3 FUNCTION,... Fügen Sie die Klammern am Ende des Funktionsnamens nicht ein. Jede Funktion muss den Feldnamen als einzigen Parameter akzeptieren. Bei der Funktion kann es sich um eine native Funktion von VFP oder um eine benutzerdefinierte Funktion handeln.

Wenn Sie beispielsweise für die Felder customername und city den Datentyp VarChar des SQL Server einsetzen, der über keine abschließenden Leerzeichen verfügt, werden Sie für die Felder des Servers von VFP zunächst die Funktion TRIM einsetzen. Dafür geben Sie in ConversionFunc „COMPANY TRIM, CITY TRIM“ an.

Andere Eigenschaften

CursorAdapter verfügt über verschiedene Eigenschaften, die mit denen von CURSORSETPROP() identisch sind: AllowSimultaneousFetch, BatchUpdateCount, CompareMemo, FetchAsNeeded, FetchMemo, FetchSize, MaxRecords, Prepared, UpdateType, UseMemoSize und WhereType. Außerdem gibt es noch einige andere Eigenschaften:

- BufferModeOverride entscheidet über den Puffermodus des Cursors (Datensatz oder Tabelle).
- UpdateGram enthält die am Cursor vorgenommenen Änderungen im Updategram-Format, wenn die Eigenschaften *DataSourceType auf „XML“ gesetzt sind. Diese Eigenschaft wird nicht ausgefüllt, wenn die Änderungen vorgenommen werden, sondern wenn die Aktualisierung vorgenommen werden soll (beispielsweise wenn ein TABLEUPDATE() ausgeführt wurde).
- Die Eigenschaft Flags enthält Einstellungen, die genutzt werden, wenn VFP das Updategram erstellt; sie hat die gleichen Werte wie die Parameter Flags der Funktion XMLUPDATEGRAM().

- UpdateGramSchemaLocation enthält den Namen und Speicherort für das Schema, wenn Sie eines nutzen wollen; wie bei den Flags funktioniert diese Eigenschaft wie die entsprechenden Parameter in XMLUPDATEGRAM().
- BreakOnError legt fest, was VFP tut, wenn im Code von VFP in einem der Ereignisse des CursorAdapter ein Fehler auftritt. Steht diese Eigenschaft auf True (Vorgabe ist False) zeigt VFP unmittelbar nach Auftreten eines Fehlers eine Fehlermeldung an, andernfalls wird die normale Fehlerbehandlung vorgenommen.

CursorFill(UseCursorSchema, lNoData, nOptions, oSource)

Diese Methode erstellt den Cursor und füllt ihn mit Daten aus der Datenquelle (auch wenn Sie als Parameter lNoData True übergeben können, um einen leeren Cursor zu erstellen). Übergeben Sie True als ersten Parameter, um das in CursorSchema angegebene Schema einzusetzen oder False, um VFP die passende Struktur von der Datenquelle erstellen zu lassen. Diese beiden Parameter behandeln wir später in diesem Kapitel.

MULTILOCKS muss eingeschaltet sein oder diese Methode schlägt fehl. Schlägt CursorFill aus irgendeinem Grund fehl, gibt es False zurück, statt einen Fehler aufzurufen. Nutzen Sie AERROR(), um festzustellen, was schief gelaufen ist (dabei müssen Sie aber für eine Suche gerüstet sein, da die Fehlermeldungen häufig nicht spezifisch genug sind, um die Quelle des Problems exakt festzustellen).

CursorRefresh()

Diese Methode entspricht der Funktion REQUERY(): sie aktualisiert die Inhalte des Cursors.

CursorAttach(cAlias, lInheritCursorProperties) und CursorDetach()

Diese Methoden ermöglichen es Ihnen, einen bestehenden Cursor einem CursorAdapter-Objekt zuzuordnen oder ihn wieder freizugeben. Zuordnen eines Cursors bedeutet, dass er sich unter der Steuerung des CursorAdapters befindet – Updates werden durch den CursorAdapter abgehandelt, der Cursor wird geschlossen, wenn der CursorAdapter freigegeben wird usw. Wenn Sie wollen, dass ein Cursor auch nach der Freigabe des CursorAdapters weiterhin existiert und ihn daher vom CursorAdapter getrennt wird, setzen Sie CursorDetach() ein.

Before*() und After*()

CursorAdapter verfügt über viele Hooks, die es ermöglichen, das Verhalten des CursorAdapters den eigenen Anforderungen anzupassen. Im Fall der Before-Ereignisse können Sie False zurückgeben, um zu verhindern, dass die ausgelöste Aktion ausgeführt wird (dies entspricht den Datenbankereignissen). Tabelle 1 zeigt eine Liste dieser Ereignisse.

Tabelle 1. CursorAdapter verfügt über verschiedene Ereignisse Before und After, was es Ihnen ermöglicht, zusätzliches Verhalten hinzuzufügen, wenn unterschiedliche Operationen auf den Cursor ausgeführt werden.

Ereignis	Wird ausgeführt
BeforeCursorAttach, AfterCursorAttach	Vor bzw. nach der Methode CursorAttach.
BeforeCursorClose, AfterCursorClose	Bevor bzw. nachdem der Cursor geschlossen wurde.
BeforeCursorDetach, AfterCursorDetach	Vor bzw. nach der Methode CursorDetach.
BeforeCursorFill, AfterCursorFill	Vor bzw. nach der Methode CursorFill.
BeforeCursorRefresh, AfterCursorRefresh	Vor bzw. nach der Methode CursorRefresh.
BeforeCursorUpdate, AfterCursorUpdate	Vor bzw. nach TABLEUPDATE(). Diese Methoden werden nicht ausgelöst, wenn das Backend implizit aktualisiert wird, beispielsweise wenn der Datensatzzeiger bewegt wird und satzweise Pufferung eingestellt ist. Sie umschließen die Methoden Before/AfterDelete, Insert oder Update, je nachdem, was mit den Daten geschieht. Wird beispielsweise eine Update-Operation mit TABLEUPDATE() ausgeführt, lautet die Reihenfolge der Ereignisse für jeden aktualisierten Datensatz BeforeCursorUpdate, BeforeUpdate, AfterUpdate und AfterCursorUpdate.
BeforeDelete, AfterDelete	Vor bzw. nach dem Senden einer Löschoperation an die Datenbank (auch wenn das Löschen implizit geschieht). Wird einmal je Datensatz ausgelöst; wird nicht ausgelöst, wenn BatchUpdateCount größer als 1 ist.
BeforeInsert, AfterInsert	Vor bzw. nach dem Senden einer Einfügeoperation an die Datenbank (auch, wenn das Einfügen implizit ausgeführt wird). Wird einmal je Datensatz ausgelöst; wird nicht ausgelöst, wenn BatchUpdateCount größer als 1 ist.
BeforeUpdate, AfterUpdate	Vor bzw. nach dem Senden einer Updateoperation an die Datenbank (auch, wenn das Update implizit ausgeführt wird). Wird einmal je Datensatz ausgelöst; wird nicht ausgelöst, wenn BatchUpdateCount größer als 1 ist.

Einige dieser Ereignisse sind sehr interessant und hilfreich. So können Sie beispielsweise in AfterCursorFill Indizes für den Cursor erstellen, so dass er für die Anweisung SEEK verfügbar wird, oder damit Relationen zwischen Cursorn aufgebaut werden können. Die Ereignisse Before und After für die Lösch-, Einfüge- und Updateoperationen (z. B. BeforeInsert) empfangen Parameter, die beschreiben, was mit den Daten geschieht, inklusive der Stati für das Feld und den Datensatz (der gleiche Wert, den Sie von GETFLDSTAT(-1) erhalten), ob die Änderungen erzwungen werden sollen oder nicht, den Befehl SQL UPDATE oder INSERT, der an die Datenbank gesandt wird (sehr hilfreich beim Debuggen), und wenn Updates erledigt werden, indem die alten Datensätze gelöscht und neue eingefügt werden (UpdateType=2), der Befehl SQL DELETE, der in den Ereignissen Before an die Datenbank gesandt werden, geben Ihnen die totale Kontrolle darüber, wie Updates ausgeführt werden.

CursorAdapter zum Arbeiten bringen

Hier ein Beispiel, dass verschiedene Kunden aus Brasilien aus der Tabelle Customers der Datenbank Northwind empfängt, die mit dem SQL Server ausgeliefert wird. Der Cursor ist nicht schreibgeschützt, so dass Sie, wenn Sie im Cursor Änderungen vornehmen, ihn schließen und dann das Programm erneut ausführen, feststellen werden, dass Ihre gemachten Änderungen im Backend gespeichert sind.

```
local loCursor as CursorAdapter, ;
  laErrors[1]
set multilocks on
loCursor = createobject('CursorAdapter')
with loCursor
  .Alias              = 'Customers'
  .DataSourceType     = 'ODBC'
  .DataSource         = sqlstringconnect('driver=SQL Server;' + ;
                        'server=(local);database=Northwind; ' + ;
                        'uid=sa;pwd=;trusted_connection=no')
  .SelectCmd          = "select CUSTOMERID, COMPANYNAME, " + ;
          " CONTACTNAME from CUSTOMERS where COUNTRY = 'Brazil'"
  .KeyFieldList       = 'CUSTOMERID'
  .Tables             = 'CUSTOMERS'
  .UpdatableFieldList = 'CUSTOMERID, COMPANYNAME, CONTACTNAME'
  .UpdateNameList     = 'CUSTOMERID CUSTOMERS.CUSTOMERID, ' + ;
                        'COMPANYNAME CUSTOMERS.COMPANYNAME, ' + ;
                        'CONTACTNAME CUSTOMERS.CONTACTNAME'
  if .CursorFill()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
```

Der Entwicklerdownload für dieses Kapitel, das unter www.hentzenwerke.com verfügbar ist, enthält diesen Code in CursorAdapterExample.PRG. Sie müssen eventuell die Einstellung der Eigenschaft DataSource ändern, damit sie die richtige Verbindungsinformation enthält, beispielsweise den Benutzernamen und das Passwort.

DataEnvironment, Form und andere Änderungen

Um die neue Klasse CursorAdapter zu unterstützen, wurden an den Klassen DataEnvironment und Form sowie den entsprechenden Designern verschiedene Änderungen vorgenommen.

Zunächst einmal verfügt die Klasse DataEnvironment jetzt, wie bereits früher bemerkt, über die Eigenschaften DataSource und DataSourceType. Sie nutzt diese Eigenschaften nicht selbst, sondern sie werden von anderen Teilen des CursorAdapters genutzt, wenn UseDEDataSource auf True gesetzt ist. Außerdem können Sie von DataEnvironment jetzt im Klassen-Designer visuell Instanzen ableiten.

Wie bei den Formularen können Sie jetzt durch die Werte in den Eigenschaften DEClass und DEClassLibrary eine von DataEnvironment abgeleitete Klasse angeben, die eingesetzt werden soll. Wenn Sie dies tun, geht alles, was Sie mit der bestehenden Datenumgebung erstellt haben (Cursor, Code usw.) verloren, aber zumindest werden Sie vorher gewarnt.

CURSORGETPROP('SourceType') gibt einen neuen Wertebereich zurück: Wurde der Cursor mit CursorFill erstellt, ist der Wert 100 + der alte Wert (beispielsweise 102 für remote Daten). Wurde ein bestehender Cursor mit CursorAttach dem CursorAdapter zugeordnet, beträgt der Wert 200 + der alte Wert. Ist die Datenquelle ein ADO Recordset, ist der Wert 104 (CursorFill) oder 204 (CursorAttach). Eine Referenz auf das ADO Recordset, das mit einem Cursor verbunden ist, erhalten Sie mit CURSORGETPROP('ADORecordset').

Die Generatoren

VFP 8 enthält neue Generatoren für DataEnvironment und CursorAdapter, die es vereinfachen, mit diesen Klassen zu arbeiten.

Der DataEnvironment Builder wird auf die übliche Weise aufgerufen, indem mit der rechten Maustaste auf die Datenumgebung eines Formulars oder im Klassen-Designer auf eine von DataEnvironment abgeleitete Klasse geklickt und im Kontextmenü Builder ausgewählt wird. In der Seite „Data Source“

des DataEnvironment Builder (siehe Abbildung 1) geben Sie die Informationen zur Datenquelle ein. Wählen Sie den erforderlichen Typ der Datenquelle und von wo die Datenquelle kommt. Wenn Sie „Use existing connection handle“ (ODBC) oder „Use existing ADO Recordset“ (ADO) auswählen, geben Sie einen Ausdruck an, der die Datenquelle enthält (beispielsweise „goConnectionMgr.nHandle“). Sie können auch einen der DSNs auf Ihrem System oder einen Verbindungsstring auswählen. Die Schaltfläche Build, die nur aktiv ist, wenn Sie „Use connection String“ für ADO auswählen, zeigt den Dialog Data Link Properties, den Sie einsetzen können, um den Verbindungsstring visuell zu erstellen. Wenn Sie entweder „Use DSN“ oder „Use connection String“ auswählen, erstellt der Generator in der Methode BeforeOpenTables von DataEnvironment Code, um die erforderliche Verbindung aufzubauen. Wählen Sie „Native“ aus, können Sie als Datenquelle einen Datenbankcontainer von VFP auswählen; in diesem Fall stellt der generierte Code sicher, dass die Datenbank geöffnet ist (Sie können als Datenquelle auch freie Tabellen angeben).

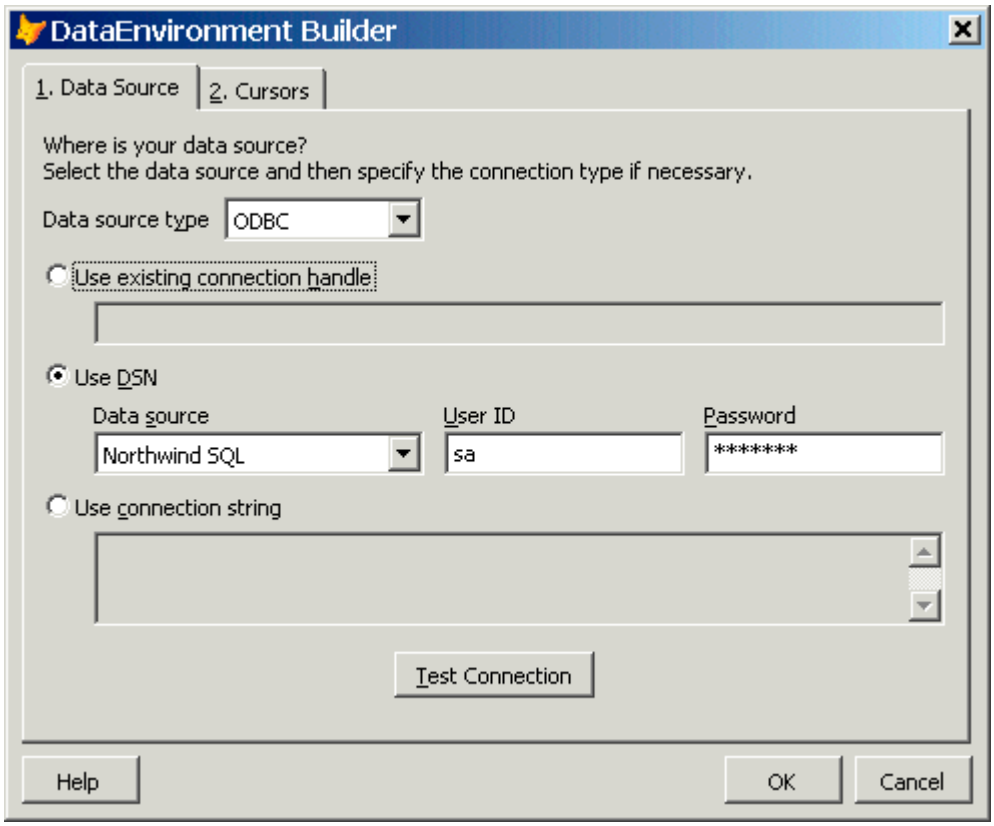

Abbildung 1. Die Seite Data Source des DataEnvironment Builder ermöglicht Ihnen die Angabe, welche Art der Datenquelle eingesetzt und womit sie verbunden werden soll.

Die Seite Cursors, die in Abbildung 2 gezeigt wird, ermöglicht Ihnen die Wartung der CursorAdapter in der DataEnvironment (Cursorobjekte erscheinen im Generator nicht, sie können auch nicht hinzugefügt werden). Die Schaltfläche Add ermöglicht es Ihnen, DataEnvironment eine von CursorAdapter

abgeleitete Klasse hinzuzufügen, obwohl New eine neue Basisklasse CursorAdapter erstellt; in jedem Fall wird der CursorAdapter Builder automatisch aufgerufen, so dass Sie mit dem neuen Objekt arbeiten können. Remove löscht den markierten CursorAdapter und Builder ruft den CursorAdapter Builder für den markierten CursorAdapter auf. Sie können den Namen des CursorAdapter-Objekts ändern, benötigen aber den CursorAdapter Builder, um die anderen Eigenschaften einzustellen.

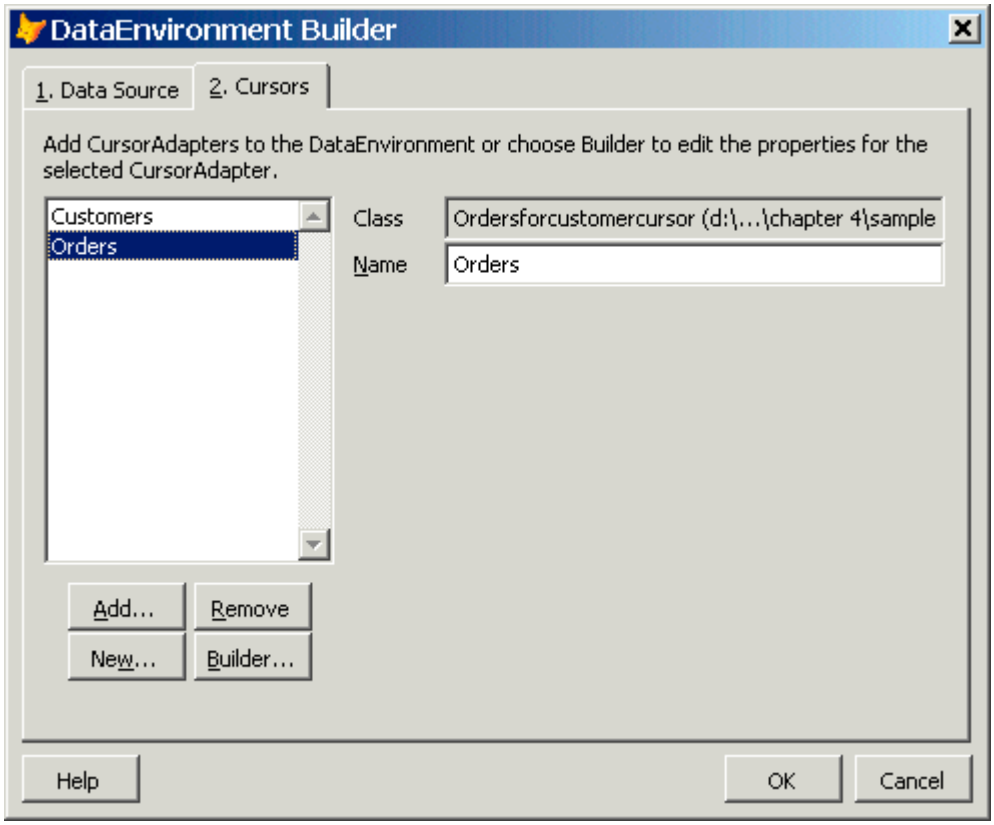

Abbildung 2. Auf der Seite Cursors können Sie CursorAdapter-Objekte hinzufügen oder löschen oder den CursorAdapter Builder für den markierten CursorAdapter aufrufen.

Der CursorAdapter Builder kann über das Kontextmenü eines CursorAdapters oder des DataEnvironment Builders aufgerufen werden. Die Seite Properties (siehe Abbildung 3) zeigt die Klasse und den Namen des Objekts (Name kann nur geändert werden, wenn der Generator von einer DataEnvironment aufgerufen wird, da er für eine von CursorAdapter abgeleitete Klasse schreibgeschützt ist), den Alias des Cursors, der erstellt wird, ob die Datenquelle von DataEnvironment eingesetzt werden soll oder nicht, und falls nicht, die zu verwendende Verbindungsinformation. Wie beim Generator DataEnvironment generiert auch der CursorAdapter Generator Code, um die erforderliche Verbindung zu erstellen (in diesem Fall in der Methode CursorFill), wenn Sie entweder „Use DSN“ oder „Use connection string“ auswäh-

len. Sie können auch eine Verbindung angeben, die der Generator temporär nutzt. In diesem Fall erstellt der Generator keinen Code für die Verbindung.

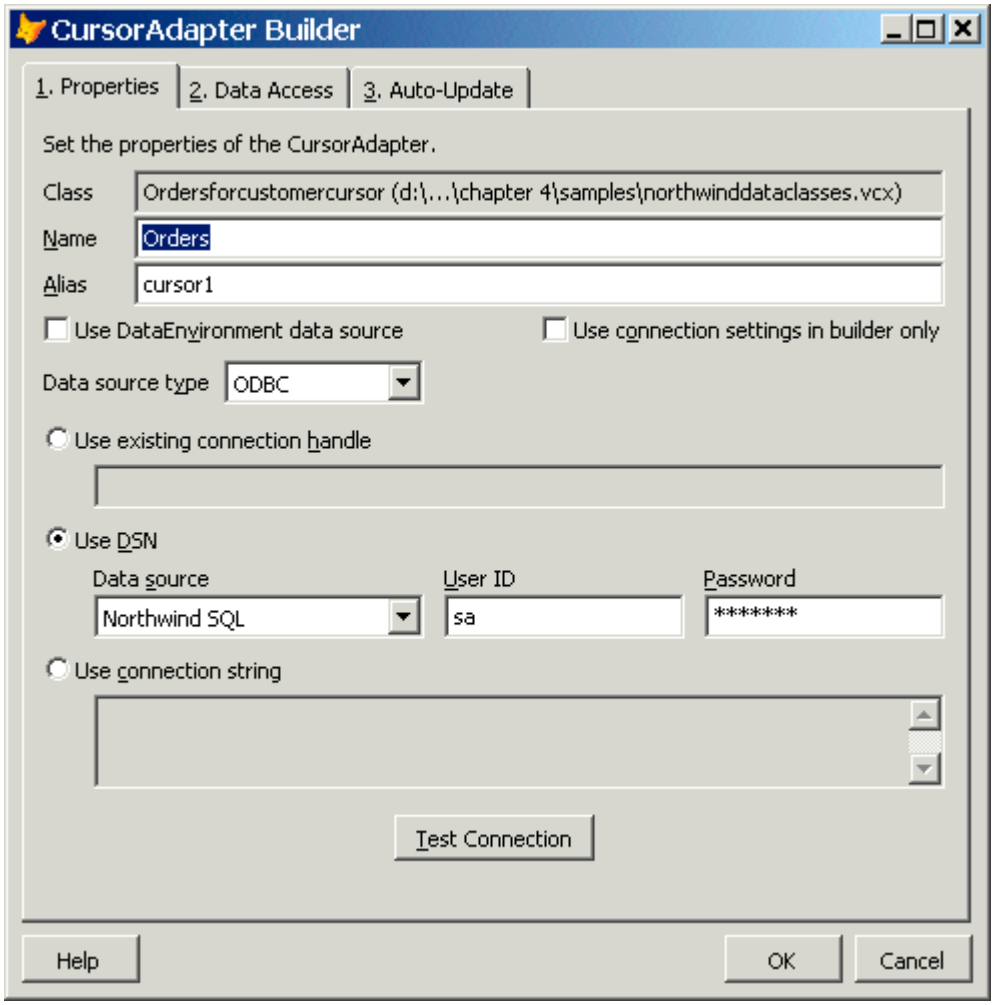

Abbildung 3. Nutzen Sie die Seite Properties des CursorAdapter Generators, um verschiedene Einstellungen für einen CursorAdapter vorzunehmen, einschließlich der Art, wie er mit der Datenbank verbunden wird.

Die Seite Data Access, die Sie in Abbildung 4 sehen, ermöglicht Ihnen die Angabe von SelectCmd, CursorSchema und anderer Eigenschaften. Wenn Sie die Verbindungsinformation angegeben haben, können Sie für SelectCmd auf die Schaltfläche Build klicken, um den Select Command Generator aufzurufen, der es vereinfacht, das SelectCmd zu erstellen. Das Schema muss weniger als 255 Zeichen lang sein, oder Sie erhalten nach einem Klick auf die Schaltfläche OK einen Warnhinweis.

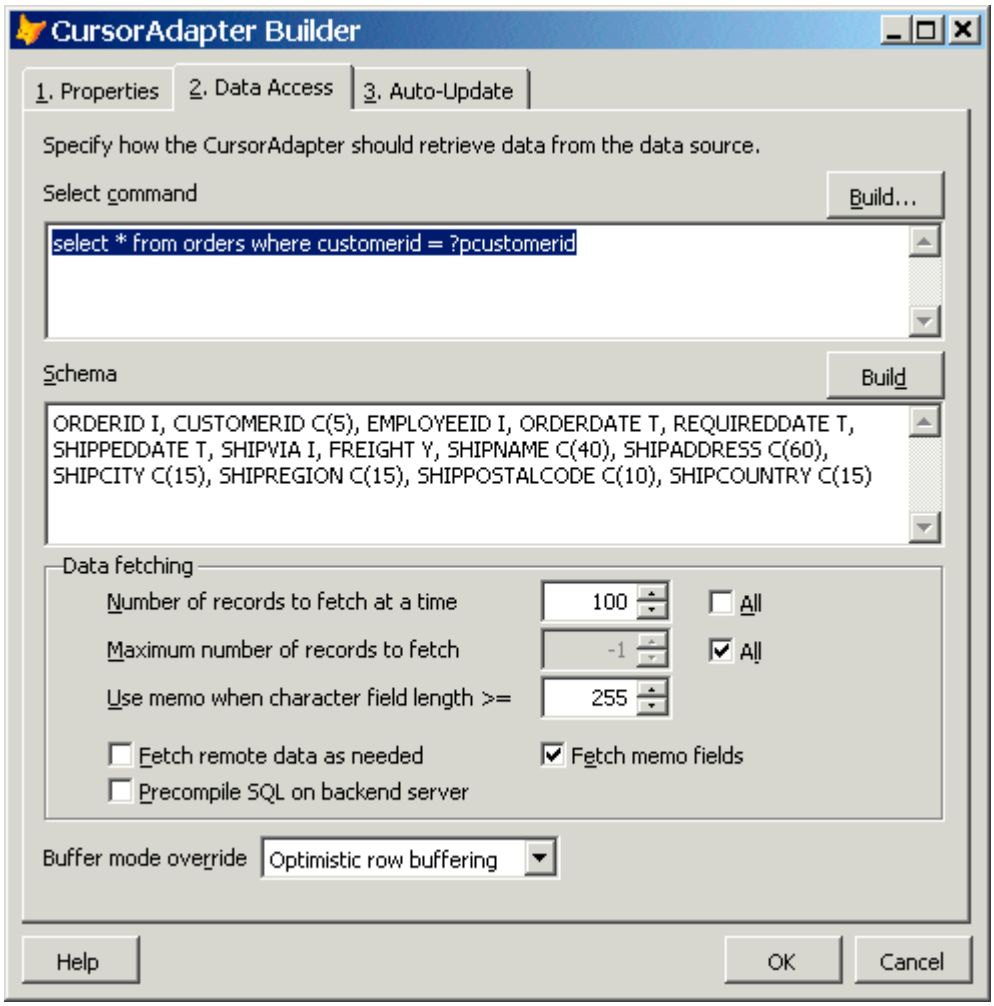

Abbildung 4. Die Seite Data Access erleichtert die Eingabe der Eigenschaften SelectCmd und CursorSchema sowie die Angabe, wie der Datenzugriff funktionieren soll.

Der Select Command Generator, der in Abbildung 5 dargestellt wird, erstellt einen einfachen Befehl SELECT. Wählen Sie in der Dropdown-Liste table die erforderliche Tabelle und verschieben anschließend die entsprechenden Felder auf die Seite der ausgewählten Felder. Wenn Sie mit einer nativen Datenquelle arbeiten, können Sie der Combobox Table Tabellen hinzufügen (beispielsweise, wenn Sie freie Tabellen einsetzen wollen). Wenn sie auf OK klicken, wird SelectCmd mit dem entsprechenden Befehl SELECT SQL gefüllt.

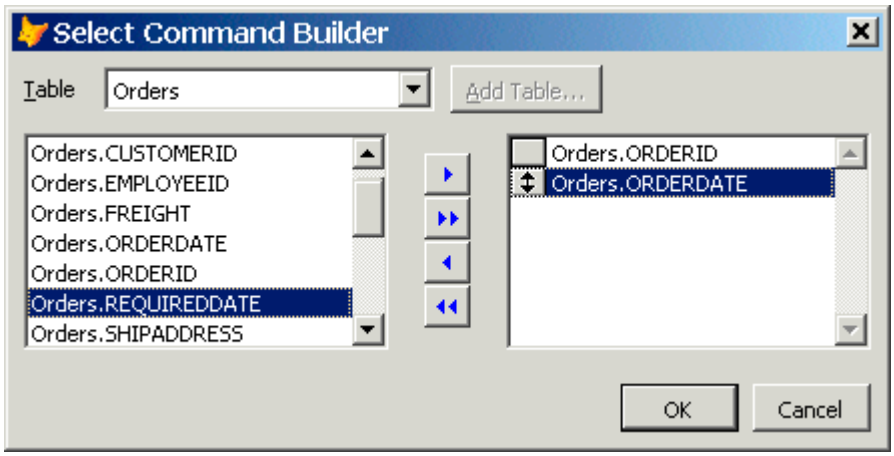

Abbildung 5. Der Generator Select Command ist ein visuelles Werkzeug für die Erstellung eines Befehls SQL SELECT.

Klicken Sie auf die Schaltfläche Build, damit das CursorSchema für Sie automatisch ausgefüllt wird. Dafür erstellt der Generator ein neues CursorAdapter-Objekt, setzt die Eigenschaften passend und ruft CursorFill auf, um den Cursor zu erstellen. Wenn Sie mit der Datenquelle nicht verbunden sind oder CursorFill schlägt aus irgendeinem Grund fehl (beispielsweise aufgrund eines ungültigen SelectCmd), funktioniert dies selbstverständlich nicht.

Die Seite Auto-Update sehen Sie in Abbildung 6. Nutzen Sie diese Seite, um die Eigenschaften einzustellen, die VFP benötigt, um automatisch Updateanweisungen für die Datenquelle zu generieren. Die Eigenschaft Tables wird automatisch durch die in SelectCmd angegebenen Tabellen ausgefüllt und der Grid Fields aufgrund der Felder in CursorSchema. Wie im Ansichts-Designer können Sie die Schlüsselfelder und die änderbaren Felder auswählen, indem Sie die entsprechende Spalte im Grid markieren. Sie können auch andere Eigenschaften einstellen, beispielsweise Funktionen, um die Daten in verschiedenen Feldern des Cursors zu konvertieren, bevor diese an die Datenquelle gesandt werden.

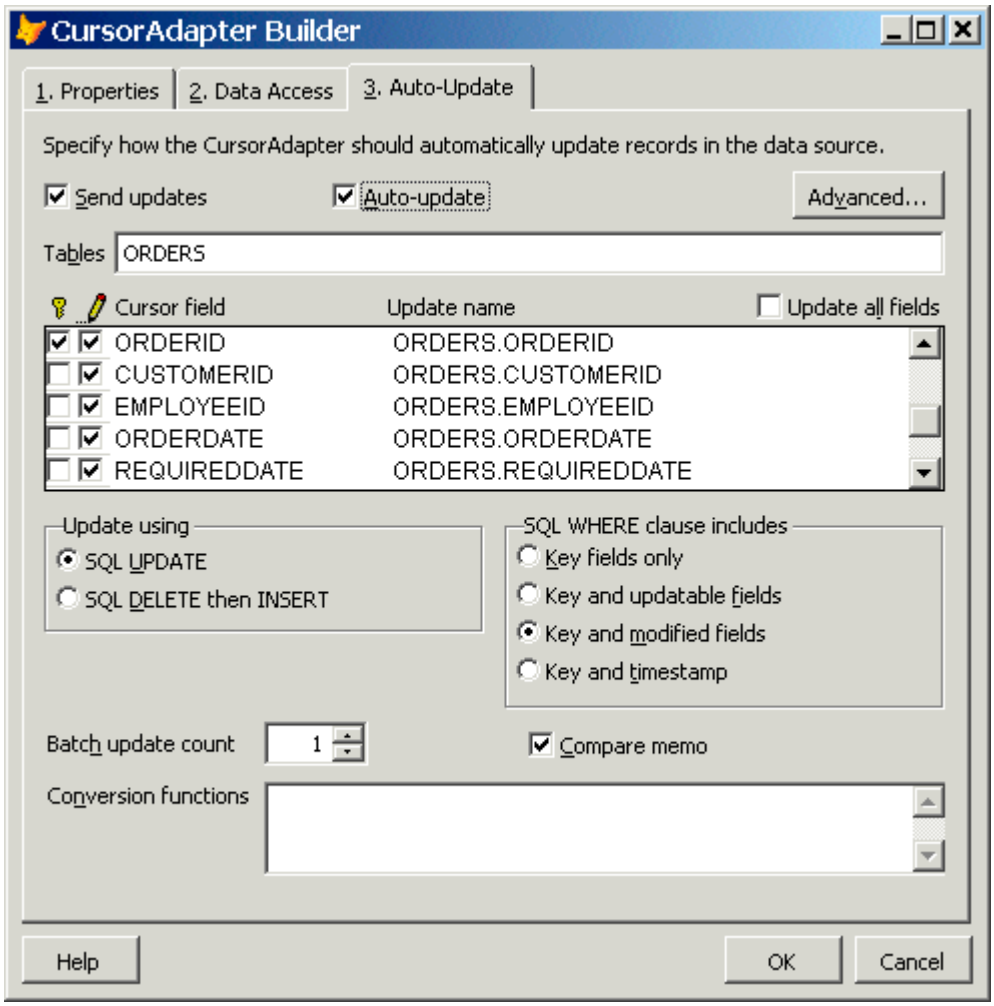

Abbildung 6. Nutzen Sie die Einstellungen auf der Seite Auto-Update, um anzugeben, wie VFP die Updateanweisungen für den Cursor generiert.

Wenn Sie eine erhöhte Kontrolle über die Updates benötigen, klicken Sie auf die Schaltfläche Advanced, um den Dialog Advanced Update Properties aufzurufen, den Sie in Abbildung 7 sehen. Die Seiten Update, Insert und Delete haben fast das gleiche Aussehen. Sie ermöglichen Ihnen, Werte für die Eigenschaften Update, Delete und Insert anzugeben. Dies ist besonders für XML wichtig, da VFP Updateanweisungen nicht automatisch generieren kann.

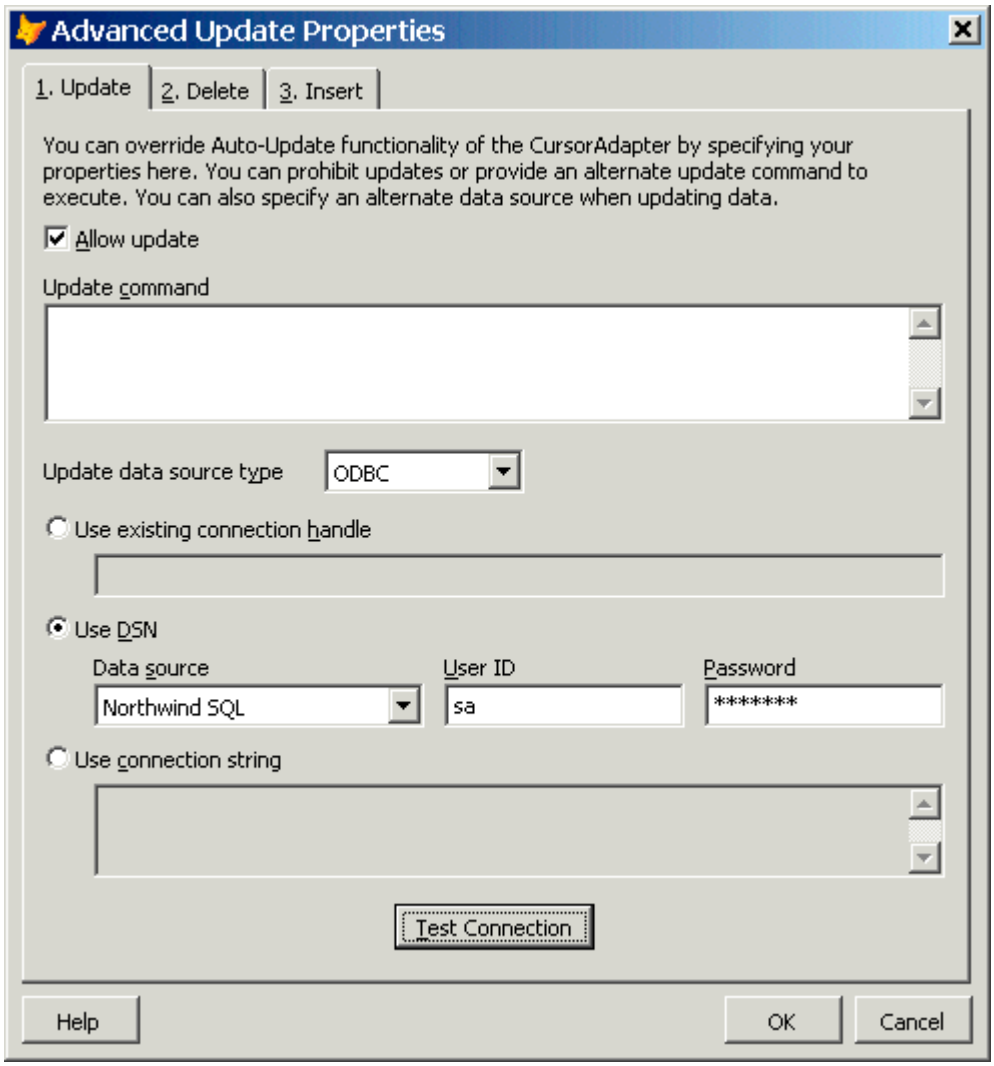

Abbildung 7. Der Dialog Advanced Update Properties ermöglicht Ihnen die Kontrolle darüber, wie Updates an die Datenbank gesendet werden.

Besonderheit der Datenquellen

Die Eigenschaft DataSourceType zeigt an, welchen Mechanismus der CursorAdapter einsetzt, um mit der Datenbankengine zu kommunizieren: native (für VFP-Daten), ODBC, ADO oder XML. Jeder dieser Typen hat seine eigenen Regeln, mit denen er arbeitet. Lassen Sie uns jetzt die Details jedes einzelnen Typen betrachten.

Einsatz nativer Daten

Auch wenn CursorAdapter für einen standardisierten und einfachen Zugriff auf Daten gedacht ist, die nicht von VFP kommen, können Sie ihn als Ersatz für den Cursor einsetzen, wenn Sie DatatSourceType auf „Native“ stellen. Weshalb sollten Sie dies tun? Meist als einen Blick in die Zukunft, für den Fall, dass Ihre Anwendung auf eine Backend-Datenbank konvertiert wird.

Durch ein einfaches Austauschen des DataSourceType durch eine der anderen Auswahlmöglichkeiten (und in der Regel das Ändern einiger Eigenschaften wie der Verbindungsinformation), können Sie einfach auf eine andere Datenbank, beispielsweise den SQL Server, umschalten.

Ist DataSourceType auf „Native“ gesetzt, ignoriert VFP die Eigenschaft DataSource. SelectCmd muss eine SQL SELECT-Anweisung enthalten, keinen Befehl USE oder Ausdruck, was bedeutet, dass Sie immer mit dem Äquivalent einer lokalen Ansicht arbeiten, nicht direkt auf den Tabellen. Sie sind dafür verantwortlich, dass VFP alle Tabellen findet, die in der SELECT-Anweisung angegeben werden. Befinden sich die Tabellen nicht im aktuellen Verzeichnis, müssen Sie entweder einen Pfad setzen oder die Datenbank öffnen, zu der die Tabellen gehören. In der Regel müssen Sie die Update-Eigenschaften (KeyFieldList, Tables, UpdatableFieldList und UpdateNameList) setzen, wenn der Cursor nicht schreibgeschützt sein soll.

Der folgende Code erstellt einen beschreibbaren Cursor aus der Tabelle Customer in der Beispieldatenbank TestData von VFP:

```
local loCursor as CursorAdapter, ;
  laErrors[1]
set multilocks on
open database (_samples + 'data\testdata')
loCursor = createobject('CursorAdapter')
with loCursor
  .Alias              = 'customercursor'
  .DataSourceType     = 'Native'
  .SelectCmd          = "select CUST_ID, COMPANY, CONTACT" + ;
                        "from CUSTOMER where COUNTRY = 'Brazil'"
  .KeyFieldList       = 'CUST_ID'
  .Tables             = 'CUSTOMER'
  .UpdatableFieldList = 'CUST_ID, COMPANY, CONTACT'
  .UpdateNameList     = 'CUST_ID CUSTOMER.CUST_ID, ' + ;
        'COMPANY CUSTOMER.COMPANY, CONTACT CUSTOMER.CONTACT'
  if .CursorFill()
    browse
    tableupdate(1)
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
close databases all
```

Der Entwicklerdownload auf www.hentzenwerke.com enthält diesen Code in NativeExample.PRG.

Einsatz von ODBC

ODBC ist die einfachste der vier Einstellungen für DataSourceType. Mit DataSource öffnen Sie einen ODBC-Verbindungshandle, stellen die üblichen Eigenschaften ein und rufen CursorFill auf, um die Daten zu empfangen. Wenn Sie KeyFieldList, Tables, UpdatableFieldList und UpdateNameList ausfüllen, generiert VFP automatisch die entsprechenden Anweisungen UPDATE, INSERT und DELETE, um das Backend mit den Änderungen zu aktualisieren. Wenn Sie stattdessen eine gespeicherte Prozedur einsetzen wollen, setzen Sie die Eigenschaften *Cmd, *CmdDataSource und *CmdDataSourceType entsprechend.

Hier ein Beispiel, das die gespeicherte Prozedur CustOrderHist der Datenbank Northwind aufruft, um die Gesamtmenge der Produkte abzurufen, die an einen bestimmten Kunden verkauft wurden:

```
local loCursor as CursorAdapter, ;
  laErrors[1]
set multilocks on
loCursor = createobject('CursorAdapter')
with loCursor
  .Alias          = 'CustomerHistory'
  .DataSourceType = 'ODBC'
  .DataSource     = sqlstringconnect('driver=SQL
Server;server=(local);' + ;
    'database=Northwind;uid=sa;pwd=;trusted_connection=no')
  .SelectCmd      = "exec CustOrderHist 'ALFKI'"
  if .CursorFill()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
```

> *Die Entwicklerdownloads auf www.hentzenwerke.com enthalten diesen Code in ODBCExample.PRG.*

Einsatz von ADO

Wenn Sie ADO als Zugriffsmechanismus mit CursorAdapter einsetzen, müssen Sie einige Dinge mehr beachten als beim Einsatz von ODBC:

- DataSource muss auf ADO Recordset gesetzt sein, dessen Eigenschaft ActiveConnection auf ein geöffnetes ADO Verbindungsobjekt gesetzt ist.
- Wenn Sie bereits über ein geöffnetes ADO Recordset verfügen (beispielsweise eines, das von einem Objekt der mittleren Schicht einer

mehrschichtigen Anwendung zurückgegeben wurde), übergeben Sie dieses als vierten Parameter an CursorFill. Dadurch wird die Eigenschaft DataSource nicht beachtet. Auch CursorRefresh funktioniert in diesem Fall etwas anders: Sie sind dafür verantwortlich, dass die Parameterwerte aktualisiert werden, bevor CursorRefresh aufgerufen wird. Einzelheiten dazu erfahren Sie im Hilfeeintrag für CursorRefresh.

- Damit die Updates funktionieren, muss das ADO Recordset die Eigenschaft Bookmark unterstützen. Sie ist nur bei clientseitigen Cursorn verfügbar. Daher sollten Sie die Eigenschaft CursorLocation des Recordset auf 3 setzen.
- Wenn Sie eine parametrisierte Abfrage einsetzen wollen (was in der Regel die bessere Lösung ist als das Empfangen aller Datensätze), müssen Sie ein ADO Command-Objekt mit einer auf ein geöffnetes ADO Verbindungsobjekt eingestellten Eigenschaft ActiveConnection als vierten Parameter an CursorFill übergeben. VFP überwacht für Sie auf das korrekte Füllen der Collection Parameters des Befehlsobjekts (um die Parameter zu finden, wird SelectCmd geparst), aber natürlich müssen sich die Variablen, die die Werte der Parameter enthalten, in Sichtweite befinden.
- Wenn Sie statt des automatischen Update Ihre eigenen Befehle für das Löschen, Einfügen oder Aktualisieren angeben wollen, setzen Sie die entsprechende Eigenschaft *CmdDataSourceType auf „ADO“, *CmdDataSource auf ein Befehlsobjekt von ADO, dessen Eigenschaft ActiveConnection auf ein geöffnetes ADO-Verbindungsobjekt verweist und *Cmd auf den auszuführenden Befehl.
- Der Einsatz eines CursorAdapters mit ADO ist in einem DataEnvironment am einfachsten. Wenn Sie dies wünschen, setzen Sie UseDEDataSource auf True und setzen anschließend die Eigenschaften DataSource und SataSourceType des DataEnvironment, so wie Sie es auch mit CursorAdapter tun würden. Allerdings funktioniert dies nicht, wenn Sie über mehr als einen CursorAdapter im DataEnvironment verfügen. Der Grund dafür ist, dass das ADO Recordset, das durch DataEnvironment.DataSource referenziert wird, nur die Daten eines einzelnen CursorAdapters enthalten kann. Wenn Sie CursorFill für den zweiten CursorAdapter aufrufen, erhalten Sie den Fehler „Recordset is already open“. Daher müssen Sie, wenn Ihre DataEnvironment mehr als einen CursorAdapter enthält, UseDEDataSource auf False setzen und die Eigenschaften DataSource und DataSourceType jedes CursorAdapters selbst verwalten (oder eine von DataEnvironment abgeleitete Klasse einsetzen, die dies für Sie tut).

Das unten stehende Beispiel zeigt, wie Sie mit Hilfe des ADO Befehlsobjekts Daten über eine parametrisierte Abfrage empfangen. Dieses Beispiel zeigt auch, wie sie die neue strukturierte Fehlerbehandlung in VFP 8 einsetzen, die im Detail im Kapitel 12, „Fehlerbehandlung“, behandelt wird. Der Aufruf der Methode Open der ADO Connection ist in eine TRY- Struktur gepackt, um den COM-Fehler verfolgen zu können, den die Methode im Falle des Fehlschlagens auslöst. Am Ende wird der Einsatz von CursorRefresh demonstriert, um den Cursor zu aktualisieren, wenn sich die parametrisierte Abfrage ändert.

```
local loConn as ADODB.Connection, ;
  loCommand as ADODB.Command, ;
  loException as Exception, ;
  loCursor as CursorAdapter, ;
  lcCountry, ;
  laErrors[1]
set multilocks on
loConn = createobject('ADODB.Connection')
with loConn
  .ConnectionString = 'provider=SQLOLEDB.1;data source=(local);' + ;
    'initial catalog=Northwind;uid=sa;pwd=;trusted_connection=no'
  try
    .Open()
  catch to loException
    messagebox(loException.Message)
    cancel
  endtry
endwith
loCommand = createobject('ADODB.Command')
loCursor  = createobject('CursorAdapter')
with loCursor
  .Alias          = 'Customers'
  .DataSourceType = 'ADO'
  .DataSource     = createobject('ADODB.Recordset')
  .SelectCmd      = 'select * from customers where ' + ;
                    'country=?lcCountry'
  lcCountry       = 'Brazil'
  .DataSource.ActiveConnection = loConn
  loCommand.ActiveConnection   = loConn
  if .CursorFill(.F., .F., 0, loCommand)
    browse
    lcCountry = 'Canada'
    .CursorRefresh()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
```

Die Entwicklerdownloads auf www.hentzenwerke.com enthalten diesen Code in ADOExample.PRG.

Einsatz von XML

Hier einige Punkte, auf die Sie achten müssen, wenn Sie XML mit Cursor-Adapter einsetzen:

- Die Eigenschaft DataSource wird ignoriert.
- Die Eigenschaft SelectCmd muss auf eine XML-Quelle eingestellt werden. Beispielsweise können Sie einen Ausdruck einsetzen, der das XML für den Cursor zurückgibt, beispielsweise eine UDF oder die Methode eines Objekts. Sie können auch den Namen eines XML-Dokuments angeben; in diesem Fall übergeben Sie CursorFill als dritten Parameter 512 (wodurch CursorFill weiß, dass das XML aus einer Datei kommt, nicht aus einem String).
- Im Cursor vorgenommene Änderungen werden in ein Updategram konvertiert. Dabei handelt es sich um XML, dass die neuen und alten Werte für geänderte Felder und Datensätze enthält und das in der Eigenschaft UpdateGram platziert wird, wenn das Update erforderlich ist.
- Um Änderungen in die Datenquelle zurückzuschreiben, muss UpdateCmdDataSourceType auf XML gesetzt werden und UpdateCmd muss auf einen Ausdruck (auch hier in der Regel eine UDF oder die Methode eines Objekts) gesetzt werden. Wahrscheinlich wollen Sie der UDF „This.UpdateGram“ übergeben, so dass sie die Änderungen an die Datenquelle senden kann. Ist BufferModeOverride auf 5-optimistic table buffering gesetzt, wird die in UpdateCmd genannte Funktion für jeden geänderten Datensatz ein Mal aufgerufen und UpdateGram enthält nur die Änderungen des Datensatzes, der im Moment aktualisiert wird.

Die XML-Quelle für den Cursor kann von unterschiedlichen Orten stammen. Beispielsweise könnten Sie eine UDF aufrufen, die mit CURSORTOXML() einen Cursor von VFP in XML umwandelt und die Ergebnisse zurückgibt:

```
use CUSTOMERS
cursortoxml('customers', 'lcXML', 1, 8, 0, '1')
return lcXML
```

Die UDF könnte einen Webdienst aufrufen, der eine Ergebnismenge als XML zurückliefert. Hier ein Beispiel für von einem Webdienst generierte IntelliSense (Die Details sind nicht wichtig, sondern es handelt sich um ein Beispiel eines Webdienstes. Vergleichen Sie Kapitel 10, „Erweiterungen durch COM und Webdienste“, um Einzelheiten zu Webdiensten zu erfahren).

local loWS as dataserver web service

```
loWS = NEWOBJECT("Wsclient",HOME()+"ffc\_webservices.vcx")
loWS.cWSName = "dataserver web service"
loWS = loWS.SetupClient("http://localhost/SQDataServer", ;
       "/dataserver.WSDL", "dataserver", "dataserverSoapPort")
lcXML = loWS.GetCustomers()
return lcXML
```

Dabei könnte SQLXML eingesetzt werden, um eine auf dem SQL Server in einer Template-Datei gespeicherte Abfrage auszuführen (nähere Informationen zu SQLXML finden Sie unter http://msdn.microsoft.com, wo Sie nach SQLXML suchen). Der folgende Code setzt das Objekt MSXML2.XMLHTTP ein, um via HTTP alle Datensätze der Tabelle Customers aus der Datenbank Northwind zu empfangen. Später behandeln wir dies noch im Detail.

```
local loXML as MSXML2.XMLHTTP
loXML = createobject('MSXML2.XMLHTTP')
loXML.open('POST', 'http://localhost/northwind/template/' + ;
           'getallcustomers.xml', .F.)
loXML.setRequestHeader('Content-type', 'text/xml')
loXML.send()
return loXML.responseText
```

Die Behandlung von Updates ist schwieriger. Die Datenquelle muss in der Lage sein, Updategrams zu akzeptieren und zu verarbeiten (wie es beim SQL Server 2000 der Fall ist), oder Sie müssen die Änderungen selbst feststellen und eine Serie von SQL-Anweisungen (UPDATE, INSERT und DELETE) absetzen, um die Updates auszuführen.

Hier ein Beispiel, das einen CursorAdapter mit einer XML-Datenquelle nutzt. Im Fall von SelectCmd wird der UDF GetNWXML, die wir uns gleich ansehen werden, der Name eines SQL Server XML Template und die Kundennummer übergeben. Für UpdateCmd übergibt VFP die Eigenschaft UpdateGram an SendNWXML, was wir uns ebenfalls später ansehen werden.

Der Entwicklerdownload, der unter www.hentzenwerke.com verfügbar ist, enthält allen für dieses Beispiel erforderlichen Code: XMLExample.PRG, GetNWXML.PRG, SendNWXML.PRG und CustomersByID .XML.

```
local loCustomers as CursorAdapter, laErrors[1]
set multilocks on
loCustomers = createobject('CursorAdapter')
with loCustomers
  .Alias          = 'Customers'
  .CursorSchema   = 'CUSTOMERID C(5), COMPANYNAME C(40), ' + ;
      'CONTACTNAME C(30), CONTACTTITLE C(30), ADDRESS C(60), ' + ;
      'CITY C(15), REGION C(15), POSTALCODE C(10), ' + ;
      'COUNTRY C(15), PHONE C(24), FAX C(24)'
  .DataSourceType = 'XML'
  .KeyFieldList   = 'CUSTOMERID'
  .SelectCmd      = 'GetNWXML([customersbyid.xml?customerid=ALFKI])'
  .Tables         = 'CUSTOMERS'
  .UpdatableFieldList = 'CUSTOMERID, COMPANYNAME, CONTACTNAME, ' + ;
                        'CONTACTTITLE, ADDRESS, CITY, REGION, ' + ;
                        'POSTALCODE, COUNTRY, PHONE, FAX'
  .UpdateCmdDataSourceType = 'XML'
  .UpdateCmd               = 'SendNWXML(This.UpdateGram)'
  .UpdateNameList          = 'CUSTOMERID CUSTOMERS.CUSTOMERID, ' + ;
    'COMPANYNAME CUSTOMERS.COMPANYNAME, ' + ;
    'CONTACTNAME CUSTOMERS.CONTACTNAME, ' + ;
    'CONTACTTITLE CUSTOMERS.CONTACTTITLE, ' + ;
    'ADDRESS CUSTOMERS.ADDRESS, CITY CUSTOMERS.CITY, ' + ;
    'REGION CUSTOMERS.REGION, POSTALCODE CUSTOMERS.POSTALCODE, ' + ;
    'COUNTRY CUSTOMERS.COUNTRY, PHONE CUSTOMERS.PHONE, ' + ;
    'FAX CUSTOMERS.FAX'
  if .CursorFill(.T.)
    browse
    tableupdate(.T.)
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
close tables
```

Das XML-Template CustomersByID.XML, das dieser Code referenziert, sieht folgendermaßen aus:

```
<root xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:header>
    <sql:param name="customerid">
    </sql:param>
  </sql:header>
  <sql:query client-side-xml="0">
    SELECT *
    FROM   Customers
    WHERE CustomerID = @customerid
    FOR XML AUTO
  </sql:query>
</root>
```

Platzieren Sie diese Datei in ein virtuelles Verzeichnis des IIS für die Datenbank Northwind (vgl. Anhang 1, „Einrichten des XML-Zugriffs im SQL Server 2000“).

Hier der Code für GetNWXML. Er nutzt ein MSXML2.XMLHTTP-Objekt, um XML via HTTP zu transferieren. Die Methode Open öffnet die Verbindung zu einer URL, die in diesem Fall ein XML-Template des SQL Server auf einem Webserver angibt. Die Methode SetRequestHeader teilt dem XMLHTTP-Objekt mit, welche Art an Daten übertragen wird. Die Methode Send sendet die Anfrage an den Server und stellt die Ergebnisse in die Eigenschaft ResponseText. Der Name des Templates (und optional der Abfrageparameter) wird dem Code als Parameter übergeben.

```
lparameters tcURL
local loXML as MSXML2.XMLHTTP
loXML = createobject('MSXML2.XMLHTTP')
loXML.open('POST', 'http://localhost/northwind/template/' + ;
           tcURL, .F.)
loXML.setRequestHeader('Content-type', 'text/xml')
loXML.send()
return loXML.responseText
```

SendNWXML sieht dementsprechend aus; außer dass es erwartet, ein Updategram übergeben zu erhalten, lädt das Updategram in ein MSXML2DOMDocument-Objekt und übergibt dieses Objekt dem Webserver, der es anschließend via SQLXML zur Verarbeitung an den SQL Server sendet.

```
lparameters tcUpdateGram
local loDOM as MSXML2.DOMDocument, ;
      loXML as MSXML2.XMLHTTP
loDOM = createobject('MSXML2.DOMDocument')
loDOM.async = .F.
loDOM.loadXML(tcUpdateGram)
loXML = createobject('MSXML2.XMLHTTP')
loXML.open('POST', 'http://localhost/northwind/', .F.)
loXML.setRequestHeader('Content-type', 'text/xml')
loXML.send(loDOM)
```

Um diesen Code bei der Arbeit zu sehen, führen Sie XMLExample.prg aus. Sie sollten in einem BROWSE-Fenster einen einzelnen Datensatz (den Kunden ALFKI) sehen. Ändern Sie den Wert eines Feldes, schließen das Fenster und führen das PRG erneut aus. Sie sollten feststellen, dass Ihre Änderung in das Backend geschrieben wurden.

Updates

Wenn sie die originale Datenquelle mit den Änderungen des Cursors aktualisieren wollen, der durch CursorAdapter erstellt wurde, verfügen Sie über ein großes Maß an Flexibilität. Die einfachste Art besteht darin, VFP die Updates durch Setzen der Eigenschaften KeyFieldList, Tables, UpdatableFieldList und UpdateNameList vornehmen zu lassen. Für eine verbesserte Kontrolle setzen Sie die Eigenschaften *Cmd, *DataSource und *DataSourceType (wobei der Stern „Delete“, „Insert“ oder „Update“ ersetzt. Auf eines sollten Sie aber achten, da Updatekonflikte nicht auf die gleiche Weise wie mit VFP-Daten behandelt werden.

Mit UpdateType 1 (dem Vorgabewert) und automatischen Updates sendet VFP einen SQL UPDATE-Befehl an die Datenquelle. Der Befehl UPDATE sieht in der Regel ähnlich wie das folgende Beispiel aus (in diesem Fall ist CUSTOMERID das Schlüsselfeld und der Wert des Feldes COMPANYNAME wurde geändert):

```
UPDATE CUSTOMERS SET COMPANYNAME=?customer.companyname
  WHERE CUSTOMERID=?OLDVAL('customerid','customer') AND
  COMPANYNAME=?OLDVAL('companyname','customer')
```

Was geschieht, wenn ein anderer Anwender den Firmennamen bereits geändert hat? In diesem Fall schlägt die Klausel WHERE fehl, da kein Datensatz mit dem früheren Namen gefunden wird. Dies ruft aber keinen Fehler hervor. Im Ergebnis sieht es so aus, als wäre das Update erfolgreich verlaufen – TABLEUPDATE() gibt True zurück.

Wird UpdateType auf 2 gesetzt, kann diese Situation besser oder schlechter sein. In diesem Fall generiert VFP zunächst ein SQL DELETE und anschließend ein INSERT:

```
DELETE FROM CUSTOMERS ;
      WHERE CUSTOMERID=?OLDVAL('customerid','customer') ;
            AND COMPANYNAME=?OLDVAL('companyname','customer')
INSERT INTO CUSTOMERS ;
      (CUSTOMERID, COMPANYNAME, CONTACTNAME, CONTACTTITLE,;
      ADDRESS, CITY, REGION, POSTALCODE, COUNTRY, PHONE, FAX) ;
      VALUES ;
      (?customer.customerid, ?customer.companyname, ;
      ?customer.contactname, ?customer.contacttitle, ;
      ?customer.address, ?customer.city, ?customer.region, ;
      ?customer.postalcode, ?customer.country, ;
      ?customer.phone, ?customer.fax)
```

Im Falle eines Konflikts schlägt der Befehl DELETE fehl, ruft aber keinen Fehler hervor. Der Befehl INSERT schlägt mit einem Fehler aufgrund eines

doppelt vorhandenen Primärschlüssel fehl (was gut ist, da TABLEUPDATE() False zurückgibt). Ist kein Primärschlüssel vorhanden, wird der Datensatz eingefügt, was zu einem doppelten Datensatz führt (was schlecht ist).

Eine Möglichkeit, diese Situation zu behandeln, ist das Auswechseln der Befehle SQL UPDATE und DELETE, so dass sie einen Fehler hervorrufen, wenn sie fehlschlagen. Im Fall des SQL Server können Sie in der Methode BeforeUpdate des CursorAdapter den folgenden Code einsetzen, um die Befehle UPDATE und DELETE so zu ändern, dass sie einen Fehler hervorrufen, wenn der Prozess fehlschlägt:

```
lcErrorCode      = " if @@ROWCOUNT=0 " + ;
                   "RAISERROR('Update conflict!', 16, 1)"
cUpdateInsertCmd = cUpdateInsertCmd + lcErrorCode
cDeleteCmd       = iif(empty(cDeleteCmd), '', ;
                       cDeleteCmd + lcErrorCode)
```

Wiederverwendbare Datenklassen

Bereits seit langer Zeit haben die VFP-Entwickler Microsoft nach wiederverwendbaren Datenumgebungen gefragt. Sie könnten beispielsweise über ein Formular und einen Bericht mit exakt der gleichen Einrichtung der Daten verfügen, aber Sie müssen DataEnvironment bei beiden manuell füllen, da Datenumgebungen nicht wieder verwendbar sind. Einige Entwickler (wie auch die meisten Frameworkhersteller) vereinfachten die Erstellung wieder verwendbarer DataEnvironments, indem Sie sie im Code erstellten (sie können visuell nicht abgeleitet werden) und ein „Loader"-Objekt auf dem Formular einsetzten, um die von DataEnvironment abgeleitete Klasse zu instanziieren. Das war aber nicht sehr geradlinig und half nicht bei Berichten.

Jetzt in VFP 8 haben wir die Möglichkeit, wieder verwendbare Datenklassen zu erstellen, die Cursor aus jeder Datenquelle jedem bereitstellen können, der sie benötigt, sowie wieder verwendbare DataEnvironments, in denen die Datenklassen enthalten sind. Leider können Sie eine von DataEnvironment abgeleitete Klasse in einem Bericht nicht einsetzen, aber Sie können CursorAdapters oder davon abgeleitete Klassen im DataEnvironment des Berichts einsetzen, um hier Vorteile aus der Wiederverwendbarkeit zu ziehen.

Hier ein Beispiel. Wenn Sie einen CursorAdapter benötigen, der mit der Tabelle Customers der Datenbank Northwind arbeitet, ist es sinnvoller, dafür eine abgeleitete Klasse einzusetzen, statt separate Instanzen zu erstellen und jedes Mal, wenn Sie eine Instanz benötigen, die Eigenschaften auszufüllen. Tabelle 2 zeigt die Eigenschaften der von CursorAdapter abgeleiteten Klasse CustomersCursor.

Tabelle 2. Die Eigenschaften der Klasse CustomersCursor stellen einen CursorAdapter bereit, der weiß, wie auf die Tabelle Customers von Northwind zugegriffen und wie sie aktualisiert wird.

Eigenschaft	Wert
Alias	Customers
CusorSchema	CUSTOMERID C(5), COMPANYNAME C(40), CONTACTNAME C(30), CONTACTTITLE C(30), ADDRESS C(60), CITY C(15), REGION C(15), POSTALCODE C(10), COUNTRY C(15), PHONE C(24), FAX C(24)
KeyFieldList	CUSTOMERID
SelectCmd	Select * from customers
Tables	CUSTOMERS
UpdatableFieldList	CUSTOMERID, COMPANYNAME, CONTACTNAME, CONTACTTITLE, ADDRESS, CITY, REGION, POSTALCODE, COUNTRY, PHONE, FAX
UpdateNameList	CUSTOMERID CUSTOMERS.CUSTOMERID, COMPANYNAME CUSTOMERS.COMPANYNAME, CONTACTNAME CUSTOMERS.CONTACTNAME, CONTACTTITLE CUSTOMERS.CONTACTTITLE, ADDRESS CUSTOMERS.ADDRESS, CITY CUSTOMERS.CITY, REGION CUSTOMERS.REGION, POSTALCODE CUSTOMERS.POSTALCODE, COUNTRY CUSTOMERS.COUNTRY, PHONE CUSTOMERS.PHONE, FAX CUSTOMERS.FAX

Sie können den CursorAdapter Generator einsetzen, um den Großteil der Arbeit zu erledigen, besonders das Einstellen des CursorSchema und der Updateeigenschaften. Der Trick besteht darin, die Option „use connection settings in builder only" zu aktivieren, die Verbindungsinformation auszufüllen, so dass Sie über eine funktionierende Verbindung verfügen, SelectCmd auszufüllen und den Generator einsetzen, damit er den Rest der Eigenschaften für Sie erstellt.

Jedes Mal, wenn Sie jetzt Datensätze aus der Tabelle Customers der Datenbank Northwind benötigen, setzen Sie einfach die Klasse CustomersCursor ein. Selbstverständlich haben wir keine Verbindungsinformation definiert, aber dies ist ein gutes Vorgehen, da diese Klasse sich nicht um Dinge kümmern sollte, wie die Daten empfangen werden (ODBC, ADO oder XML) oder auch welche Datenbankengine eingesetzt werden soll (Versionen der Datenbank Northwind gibt es für den SQL Server, Access und, neu in Version 8, für VFP). Hier ein Beispiel, das die Klasse CustomersClass einsetzt. Beachten Sie, das es nur die Verbindungsinformation einstellen muss, um über ei-

nen vollständig aktualisierbaren Cursor der Northwind Customers zu verfügen.

```
loCursor = newobject('CustomersCursor', 'NorthwindDataClasses')
with loCursor
  .DataSourceType = 'ODBC'
  .DataSource     = sqlstringconnect('driver=SQL Server;' + ;
    'server=(local);database=Northwind;uid=sa;pwd=;' + ;
    'trusted_connection=no')
  if .CursorFill()
    browse
  else
    aerror(laErrors)
    messagebox(laErrors[2])
  endif
endwith
```

Die Dateien des Entwicklerdownloads auf www.hentzenwerke.com beinhalten diesen Code in der Datei TestCustomersCursor.PRG, zusätzlich NorthwindDataClasses.VCX, in der die Klassendefinition CustomersCursor enthalten ist. Sie finden dort auch weiteren Beispielcode, der hier nicht beschrieben wird, einschließlich einiger von CursorAdapter und DataEnvironment abgeleiteter Klassen mit den Namen SFCursorAdapter und SFDataEnvironment, die zusätzliche Funktionalitäten enthalten und als übergeordnete Klassen für spezialisierte Klassen dienen können.

Eine Sache, auf die Sie beim Ableiten von CursorAdapter achten sollten, ist die eindeutige Reihenfolge, in der die Ereignisse bei der Instanziierung ausgelöst werden. Wird ein CursorAdapter im Code instanziiert, wird das Ereignis Init vor allen anderen ausgelöst. Dies erwarten Sie auch so. Hier aber die Reihenfolge der Ereignisse für einen CursorAdapter in der DataEnvironment eines Formulars:

```
DataEnvironment.OpenTables
DataEnvironment.BeforeOpenTables
CursorAdapter.AutoOpen
CursorAdapter.BeforeCursorFill
CursorAdapter.CursorFill
CursorAdapter.AfterCursorFill
CursorAdapter.Init
DataEnvironment.Init
```

Obwohl Init nach CursorFill ausgelöst wird, können Sie dort keinen Code einfügen, der Eigenschaften wie DataSource und DataSourceType initialisiert – dieser Code würde zu spät ausgeführt. Stattdessen schreiben Sie diesen Code in BeforeCursorFill oder CursorFill.

Sie können der DataEnvironment eines Berichts die Basisklasse CursorAdapters hinzufügen, aber keine davon abgeleiteten Klassen, zumindest nicht visuell. Der Grund dafür ist, dass VFP in der FRX nur den Klassennamen speichert, nicht die Klassenbibliothek. VFP weiß nicht, wo die abgeleiteten Klassen definiert sind. Wenn Sie von CursorAdapter abgeleitete Klassen einsetzen wollen, setzen Sie im Init der DataEnvironment AddObject oder NewObject ein.

Zusammenfassung

CursorAdapter ist eine der größten und am meisten überraschenden Erweiterungen in VFP 8, da sie eine konsistente und einfach einzusetzende Schnittstelle zu remoten Daten darstellt und es vereinfacht, von einem Zugriffsmechanismus zu einem anderen zu wechseln und Ihnen den Einsatz wirklich wieder verwendbarer Datenklassen ermöglicht.

VFP 8 enthält noch eine Menge anderer Änderungen im Hinblick auf den Umgang mit den Daten. Diese werden wir in den nächsten beiden Kapiteln detailliert behandeln.