Session V-FXCS

VFX – Von DBC zu SQL mit CA

Uwe Habermann, Venelina Jordanova

Einführung

Egal ob Sie ein Neueinsteiger in die Client/Server-Programmierung oder ein „alter Hase“ sind, in dieser Session wird C/S-Entwicklung auf dem neuesten technischen Stand gezeigt. Mit der leistungsfähigen Builder-Unterstützung des RAD-Frameworks VFX 9.5 wird in nur 75 Minuten eine komplette Anwendung erstellt, die ohne Programmänderungen sowohl mit einer FoxPro-Datenbank als auch mit einer SQL Server-Datenbank lauffähig ist. Durch den Einsatz von Cursoradaptern und eines Verbindungs-Managers wird die erstellte Anwendung so zwischen DBC und SQL Server umschaltbar sein, dass bei der Installation beim Kunden entschieden werden kann, ob mit DBC oder SQL Server gearbeitet werden soll.

Die Verwendung der CursorAdapter-Klasse

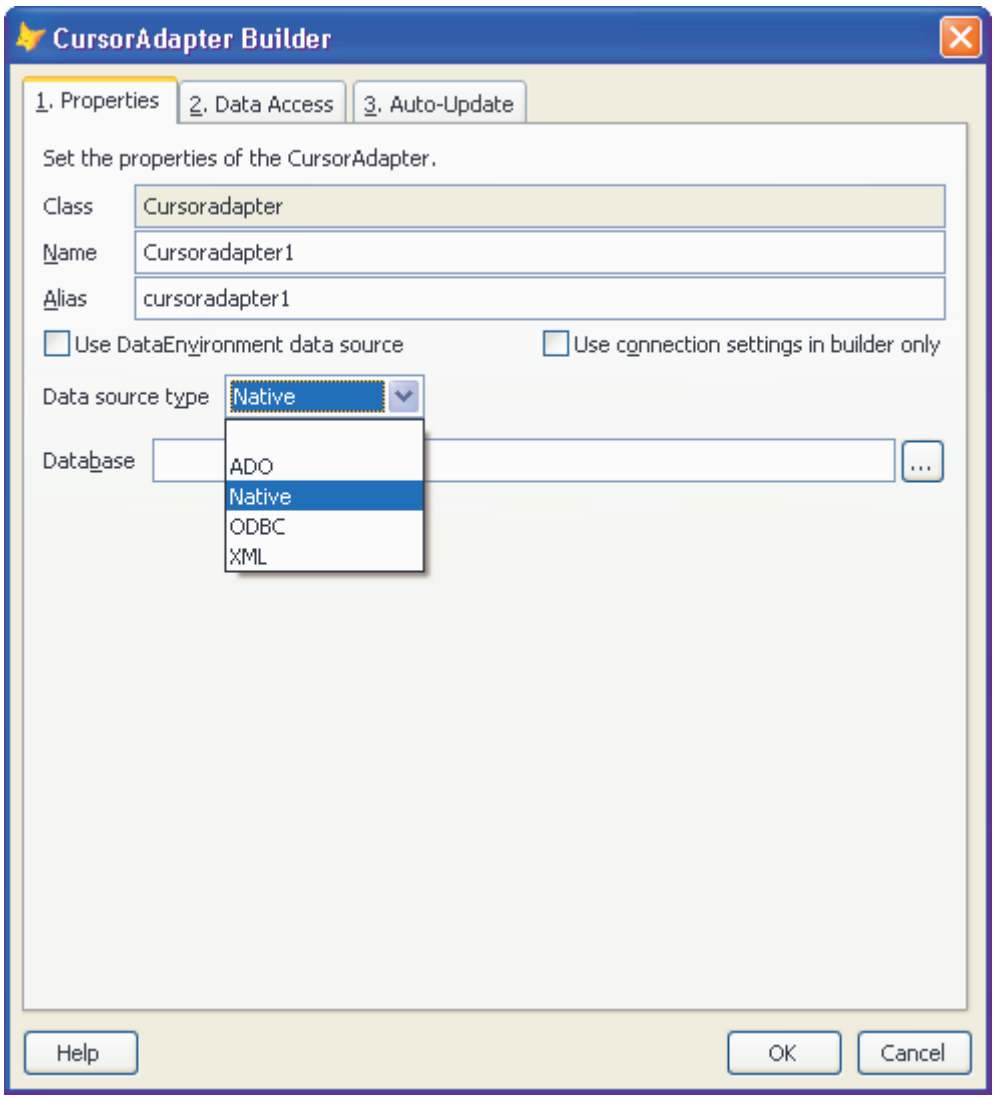
Die Klasse CursorAdapter gibt es in Visual FoxPro seit der Version 8. In VFP 9 wurden wichtige Eigenschaften und Features hinzugefügt, die den Einsatz von CursorAdaptern sehr erleichtern. Damit haben wir eine neue, leistungsfähige Möglichkeit um auf Daten zuzugreifen.

CursorAdapter bieten eine Möglichkeit um objektorientiert auf Daten zuzugreifen. CursorAdapter sind eine Klasse, so ähnlich wie eine Textbox oder auch ein Container. Wir können CursorAdapter in unseren Anwendungen genauso benutzen wie anderen Klassen auch. CursorAdapter haben Eigenschaften sowie Methoden, in die man Code einfügen kann und man kann CursorAdapter-Klassen vererben. Zur Laufzeit liefert uns ein CursorAdapter einen Cursor, der sich genauso verhält, wie ein Cursor, den eine Ansicht liefert. Wir können darin mit *LOCATE* nach Daten suchen, mit *SKIP* den Datensatzzeiger bewegen und mit *REPLACE* Daten ändern. Aktualisierungen können mit der Funktion *TABLEUPDATE()* in der Quell-Datenbank gespeichert werden.

Um CursorAdapter kennen zu lernen erstellen wir ein Beispiel mit reinen VFP-Mitteln. Anschließend werden wir sehen, mit welchen Features VFX die Verwendung von CursorAdaptern unterstützt.

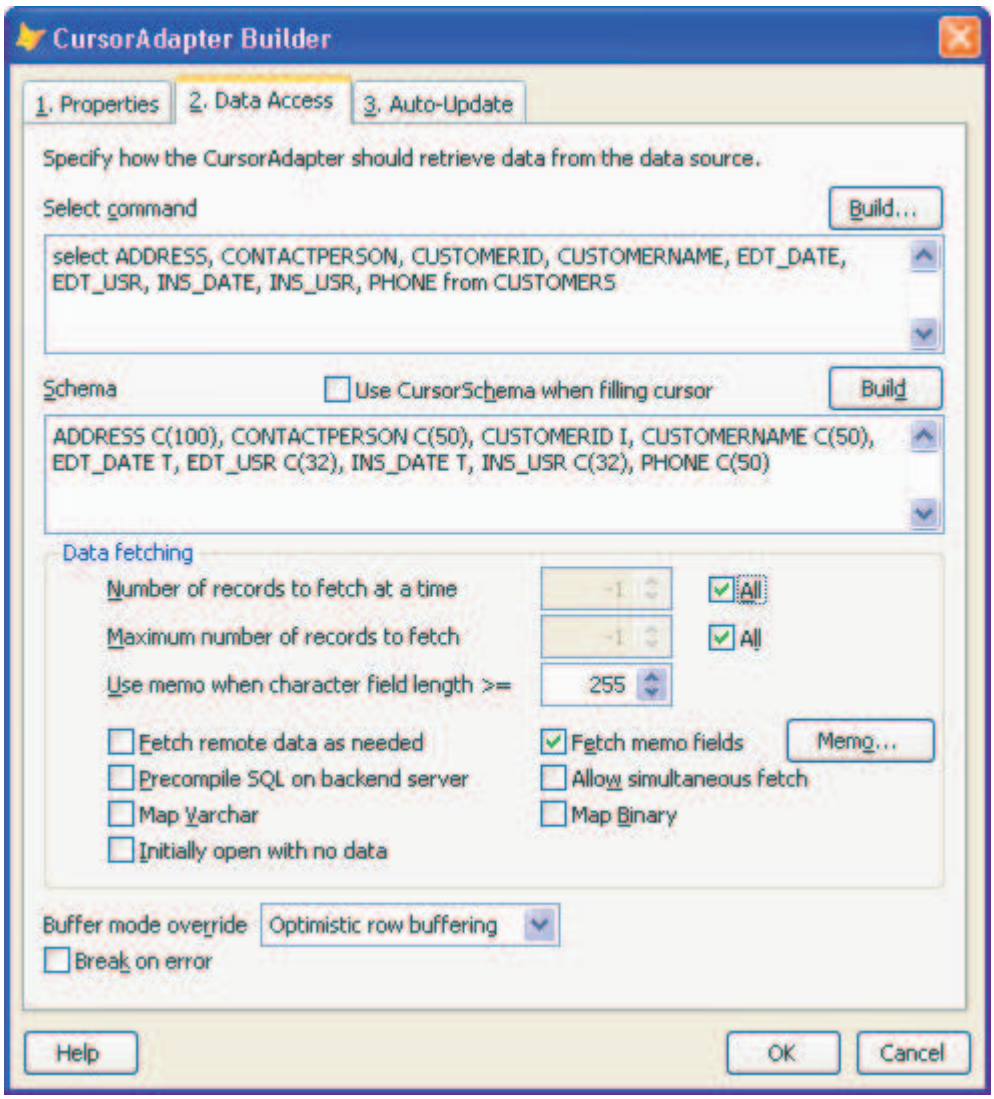
In VFP wird ein neues Formular basierend auf der VFP-Basisklasse Form angelegt. Um hier einen CursorAdapter zu verwenden, wird die Datenumgebung geöffnet und ein CursorAdapter wird über das Kontextmenü hinzugefügt. Im Eigenschaftsfenster können die zahlreichen Eigenschaften und Methoden der Klasse CursorAdapter gesehen werden. Hier alles manuell einzutragen, wäre viel Arbeit. Darum bietet uns VFP selbst einen Builder für CursorAdapter an, mit dem die wichtigsten Einstellungen gemacht werden können.

Um ein einfaches Beispiel zum Laufen zu bekommen, wählen wir als Datenquelle *Native* aus. Dadurch wird eine VFP-Datenbank verwendet. In der Textbox *Database* tragen wir den Pfad und Namen unserer Datenbank ein oder wählen über die Schaltfläche mit den drei Punkten die Datenbank aus.



VFP CusorAdapter Builder, Properties

Im zweiten Schritt des Builders wird der Datenzugriff eingestellt. Diese Seite bietet so ähnliche Einstellmöglichkeiten, wie wir sie aus dem Ansichts-Designer kennen. Hier kann ein SELECT-Befehl aufgebaut werden. Auch hierfür gibt es einen Builder, in dem zu verwendende Tabellen und Felder ausgewählt werden können. Damit haben wir einen SELECT-Befehl erstellt, der verwendet wird, um Daten aus der Datenquelle zu holen. Zusätzlich kann ein Schema verwendet werden, mit dem eine Typkonvertierung durchgeführt werden kann. Wichtig ist ein Häkchen in die Checkbox *All* bei *Number of records to fetch at a time* zu setzen. Dadurch werden bei jeder Abfrage alle Datensätze in den Cursor geholt und die Verbindung zur Datenbank wird sofort wieder freigegeben und kann von anderen CursorAdpaptern verwendet werden. Wenn dieses Häkchen nicht gesetzt wird, könnte die Verbindung geöffnet bleiben und der nächste CursorAdapter würde eine neue, zusätzliche Verbindung zur Datenbank herstellen.



VFP CusorAdapter Builder, Data Access

Auf der Seite *Auto-Update* werden die Aktualisierungskriterien eingestellt. Auch dies ist sehr ähnlich zu den Einstellmöglichkeiten im Ansichts-Designer. Wir wählen hier *Auto-update* und *Update all fields*. Damit haben wir eingestellt, dass Aktualisierungen grundsätzlich an die Datenbank gesendet werden und dass alle Felder aktualisiert werden sollen. Wir wissen, dass wir in der in unserem Beispiel verwendeten Kundentabelle ein ID-Feld haben. Dies ist das Feld *CustomerID*. Dieses Feld markieren wir als ID-Feld und als nicht aktualisierbar.

Damit haben wir auch schon alle Einstellungen gemacht, dir erforderlich sind, um diesen CursorAdapter zum Laufen zu bekommen. Nach einem Klick auf *OK* im Builder sehen wir im Eigenschaftsfenster, dass der Builder die Werte von einigen Eigenschaften eingestellt hat und in einigen Methoden ist auch Code zu finden, zum Beispiel im *Init*-Ereignis. In diesem Code sehen wir, dass der Pfad zur Datenbank hard-codiert als vollständiger Pfadname eingetragen wurde. In unserer Entwicklungsumgebung funktioniert dies wunderbar, aber wenn wir unsere Anwendung beim Kunden installieren, wird es diesen Pfad dort wahrscheinlich nicht geben. Hier wäre schon die erste Stelle, an der wir in das Verhalten des CursorAdapters eingreifen müssten, wenn wir die VFP-CursorAdapter-Basisklasse verwenden würden. Aber dies soll nur ein einfaches Beispiel für die Verwendung von CursorAdaptern ohne VFX sein.

Aus der Datenumgebung können jetzt Felder auf das Formular im VFP Formular-Designer gezogen werden. Dabei werden Labels und Textboxen mit Controlsource angelegt. Damit sollte das Formular schon grundsätzlich funktionieren und Daten aus der Datenbank anzeigen können.

Vorteile der CursorAdapter

Wir haben gesehen, dass es Eigenschaften gibt, in denen die Datenquelle eingestellt werden kann. Diese Eigenschaften stehen natürlich auch zur Laufzeit der Anwendung zur Verfügung. Dadurch können wir zur Laufzeit einstellen, ob die Anwendung auf einer DBC-Datenbank oder auf einer SQL Server-Datenbank arbeiten soll.

Wie wir schon im CursorAdapter Builder gesehen haben, ermöglicht der CursorAdapter eine Typkonvertierung. Wenn eine Remote Datenbank Feldtypen verwendet, die VFP nicht direkt unterstützt, kann hier eine geeignete Typkonvertierung durchgeführt werden.

Nachteil von Ansichten

Ansichten sind im Datenbank-Container gespeichert. Remote Ansichten basieren auf einer Verbindung, die ebenfalls im Datenbank-Container gespeichert sein muss. Da der Datenbank-Container nichts anderes als eine DBF-Tabelle mit einer anderen Namenserweiterung ist, könnten sich Power User, zum Beispiel mit Excel, Zugriff auf diese Datenbankinformationen verschaffen. Die Daten werden dadurch ggf. manipulierbar.

Konzept des Datenzugriffs

In VFX wurde ein Konzept des Datenzugriffs implementiert, das weitgehend sicher vor Manipulationen ist und eine Umschaltung der Datenquelle beim Kunden ermöglicht.

Wir haben eine Klasse für CursorAdapter. Dies ist die Klasse *CBaseDataAccess* in der Klassenbibliothek *Vfxctrl.vcx*. Diese Klasse beinhaltet die gesamte Funktionalität der CursorAdapter in VFX.

Die Klasse *CBaseDataAccess* verwendet einen Verbindungs-Manager. Der Verbindungs-Manager stellt die Verbindung zu Remote Datenbanken her. Der Verbindungs-Manager ist die programmatisch erstellte Klasse *CConnectionMgr* und in *Vfxfunc.prg* zu finden. Wer Interesse an der Funktion des Verbindungs-Managers hat, kann sich den Quelltext ansehen.

Das Ziel der Verwendung der Klasse *CBaseDataAccess* ist, eine Austauschbarkeit der Datenquelle zur Laufzeit zu erreichen. Letztendlich soll beim Kunden entschieden werden, ob eine DBC-Datenbank oder eine SQL Server-Datenbank zum Einsatz kommen soll.

Die Zugriffsinformationen zur Datenbank werden hierzu in einer Datei mit dem Namen *Config.vfx* gespeichert. Diese Datei ist verschlüsselt und damit vor unbefugtem Zugriff und vor Manipulationen gut geschützt.

In Formularen werden als Datenquelle ausschließlich Ableitungen der Klasse *CBaseDataAccess* verwendet.

Wenn wir eine neue Anwendung mit VFX planen und wir nicht sicher sind, ob diese Anwendung vielleicht einmal mit einer Remote Datenbank laufen soll, sollten wir von vornherein grundsätzlich nur mit CursorAdaptern arbeiten. Alle Funktionen von VFX, alle Formularklassen, Auswahlfelder usw. können CursorAdapter als Datenquelle verwenden. Es muss niemals direkt auf Tabellen oder mit Ansichten gearbeitet werden. Der gesamte Datenzugriff kann über CursorAdapter erfolgen und damit sind wir von der Datenbank völlig unabhängig. Wir müssen nur darauf achten, dass alle CursorAdapter auf der Klasse *CBaseDataAccess* oder auf einer Ableitung davon basieren, damit der Verbindungs-Manager von VFX verwendet wird.

Der Verbindungs-Manager von VFX hat den Vorteil, dass er in der Regel nur eine Verbindung zur Remote Datenbank benötigt. Wir hatten schon beim Beispiel mit dem CursorAdapter basierend auf der VFP-Basisklasse eingestellt, dass alle Daten geholt werden sollen. Damit wird die Verbindung sofort wieder freigegeben und steht anderen CursorAdaptern zur Verfügung.

MSDE und SQL Server

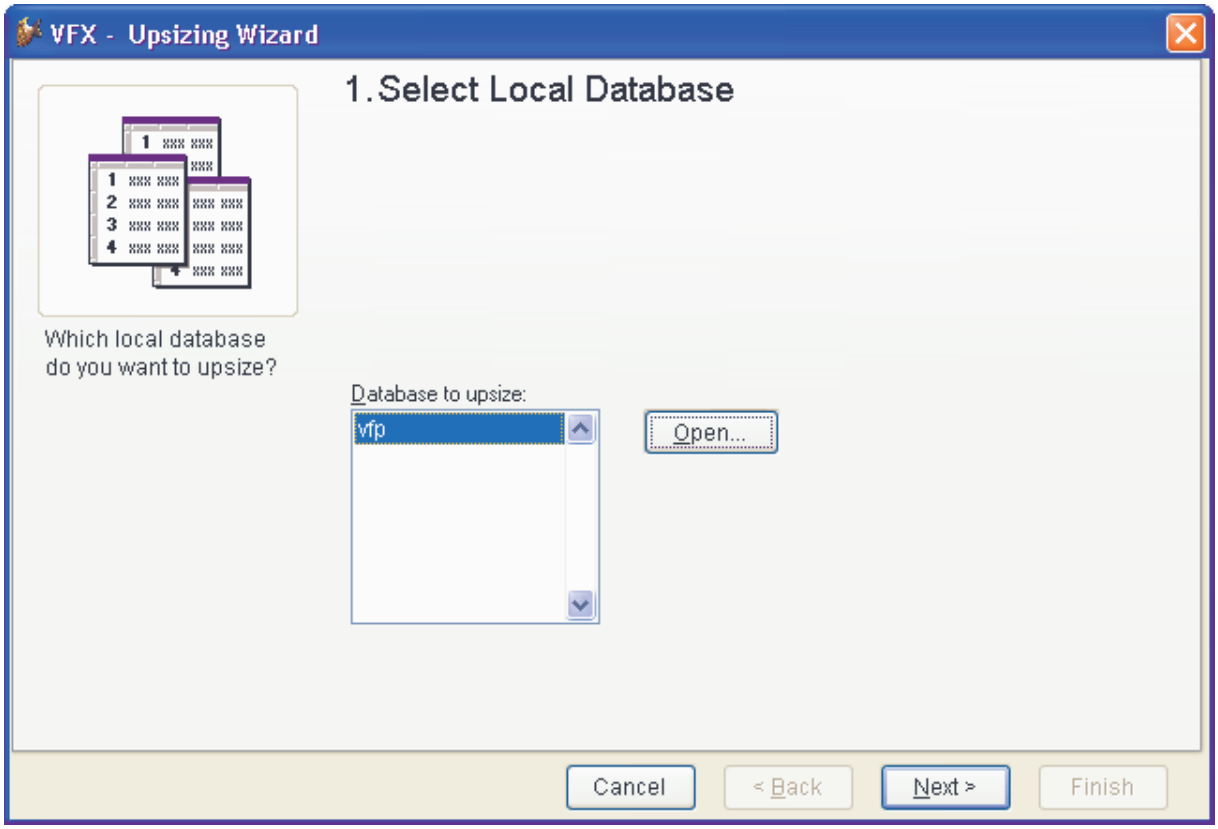
Warum setzen wir eigentlich einen SQL Server ein? SQL Server Datenbanken sind skalierbar und können fast beliebig groß werden. Die Datenbanken bieten eine hohe Sicherheit, weil der Zugriffsschutz im SQL Server selbst implementiert ist. Es können Benutzer angelegt und Rechte für verschiedene Aufgaben erteilt werden. Der SQL Server ist gut wartbar und bietet unter anderem eine Möglichkeit eine Datensicherung durchzuführen, während auf die Daten zugegriffen wird. Diese Vorteile können wir mit einer VFP-Datenbank nicht bieten.

Alle VFP-Entwickler haben den SQL Server zur Verfügung. Er befindet sich in der einfachen, kostenlosen Version MSDE auf jeder VFP-CD-ROM und kann von dort installiert werden. Entwickler, die noch keine Erfahrung mit dem SQL Server haben, können sich diese MSDE installieren und haben damit einen SQL Server mit fast vollständigem Funktionsumfang zur Verfügung. Ein Nachteil ist, dass eine MSDE-Datenbank nur 2 GB groß werden kann. Ein weiterer Nachteil ist, dass mit der MSDE keine Administrations-Werkzeuge geliefert werden. Dafür bietet uns VFP 9 aber den Data-Explorer, der es uns ermöglicht sowohl auf DBC-Datenbanken, als auch auf Remote Datenbanken zuzugreifen. Es können Daten angezeigt werden und es kann die Struktur von Tabellen verändert werden.

VFX – Upsizing Wizard

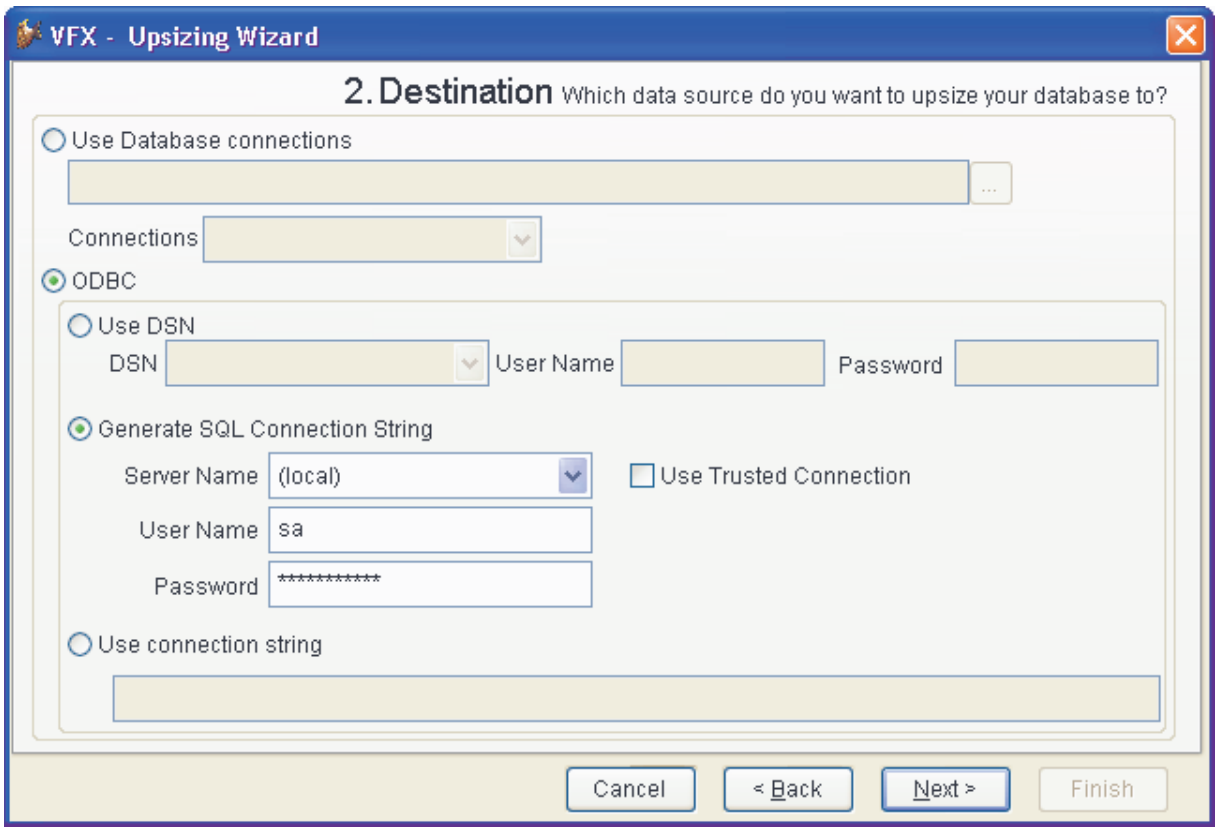
Wie kommen unsere Datenbanken nun auf den SQL Server? Dafür gibt es den Upsizing Wizard von VFP. VFX 9.5 bietet auch einen Upsizing Wizard. Der Vorteil des VFX Version des Upsizing Wizard ist, dass er funktioniert.

Genau wie der VFP – Upsizing Wizard, führt uns auch der VFX – Upsizing Wizard durch sechs Schritte.



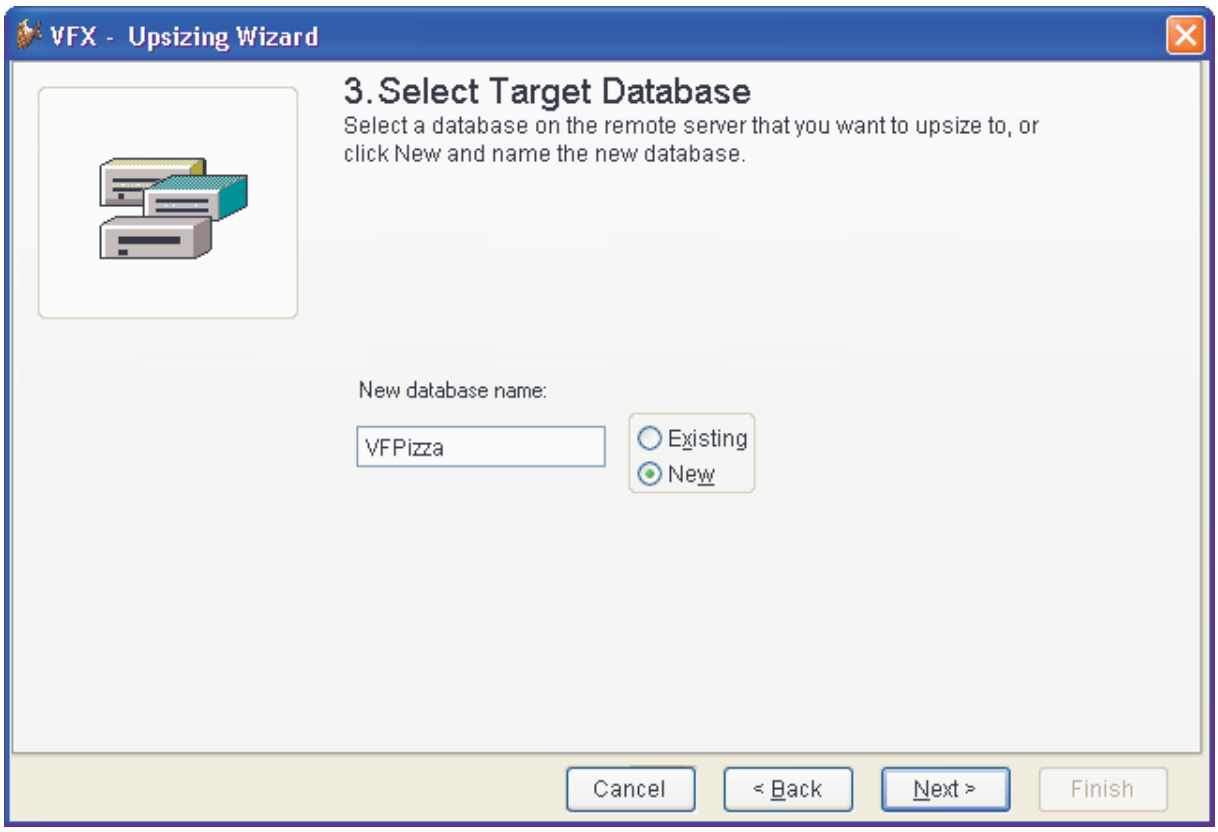
VFX – Upsizing Wizard, Select Local Database

- **Select Local Database** – Zunächst muss die lokale Datenbank ausgewählt werden, die auf den SQL Server portiert werden soll.



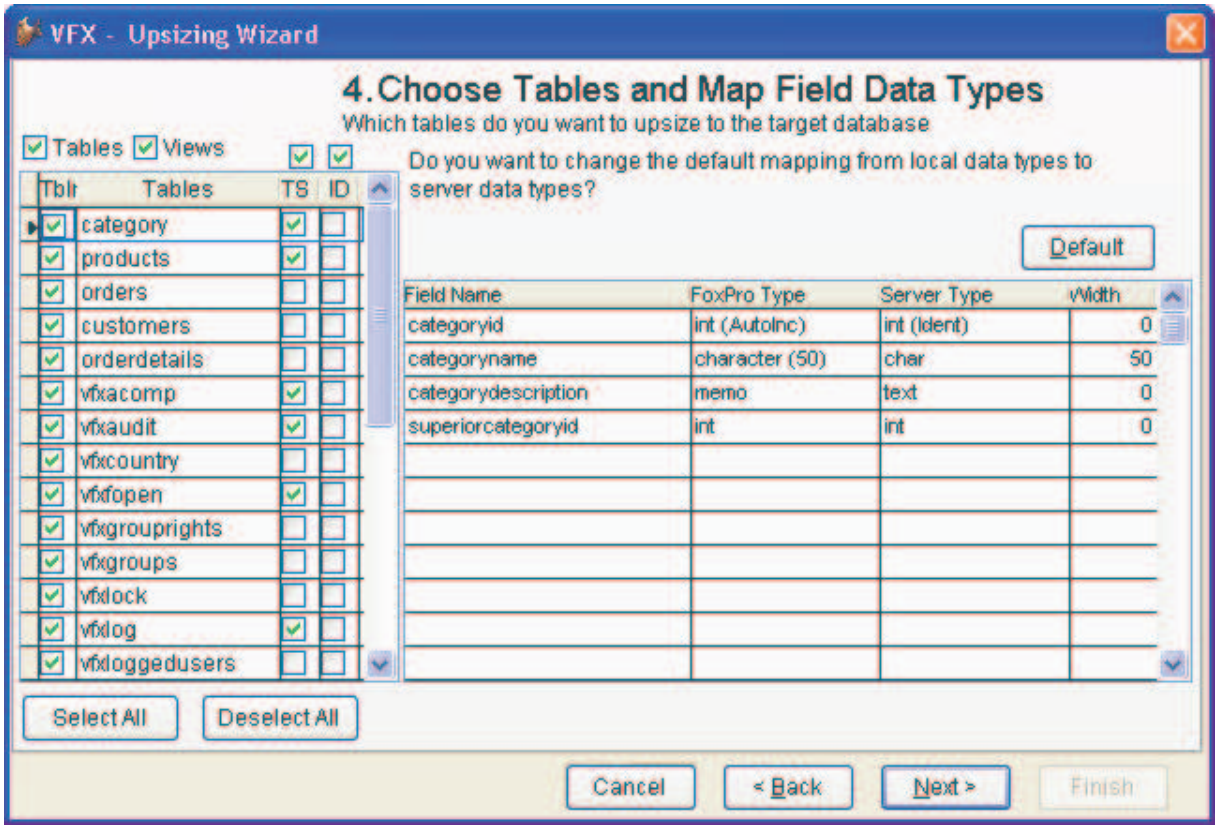
VFX – Upsizing Wizard, Destination

- **Destination** – Im zweiten Schritt wird ausgewählt, auf welchen Datenbank-Server die Datenbank portiert werden soll. Der VFX – Upsizing Wizard unterstützt alle Versionen des SQL Servers und der MSDE sowie dazu weitgehend kompatible Systeme, wie zum Beispiel MySQL, aber auch der Oracle Datenbank-Server wird unterstützt. Es gibt verschiedene Möglichkeiten eine Verbindung zur Remote Datenbank herzustellen. Es kann eine Verbindung aus der VFP-Datenbank verwendet werden, es kann eine DSN verwendet werden oder es kann eine Verbindungszeichenfolge verwendet werden. Zum Erstellen einer Verbindungszeichenfolge steht wiederum ein Wizard zur Verfügung. Es braucht nur ein Server ausgewählt werden und die Anmeldeinformation muss angegeben werden. Wenn die Anmeldung fehlschlägt, versucht der VFX – Upsizing Wizard automatisch eine „Trusted Connection“ herzustellen, also die Windows-Anmeldeinformation zu verwenden. Im Idealfall brauchen also keine Eintragungen zur Herstellung der Verbindung gemacht werden. Durch einen Mausklick auf *Next* wird in der Regel eine Verbindung zum lokal installierten SQL Server hergestellt.



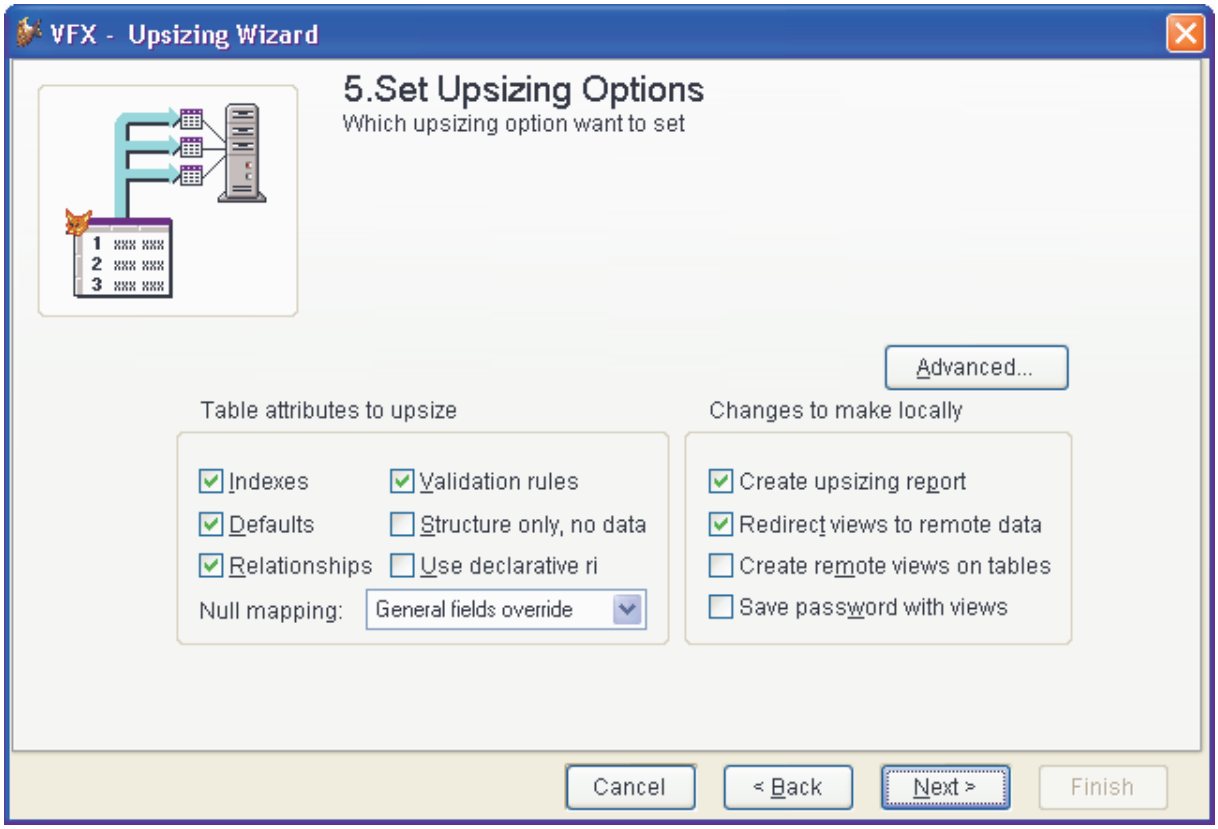
VFX – Upsizing Wizard, Select Target Database

- **Select Target Database** – Im dritten Schritt wird der neu zu erstellenden Datenbank ein Name gegeben.



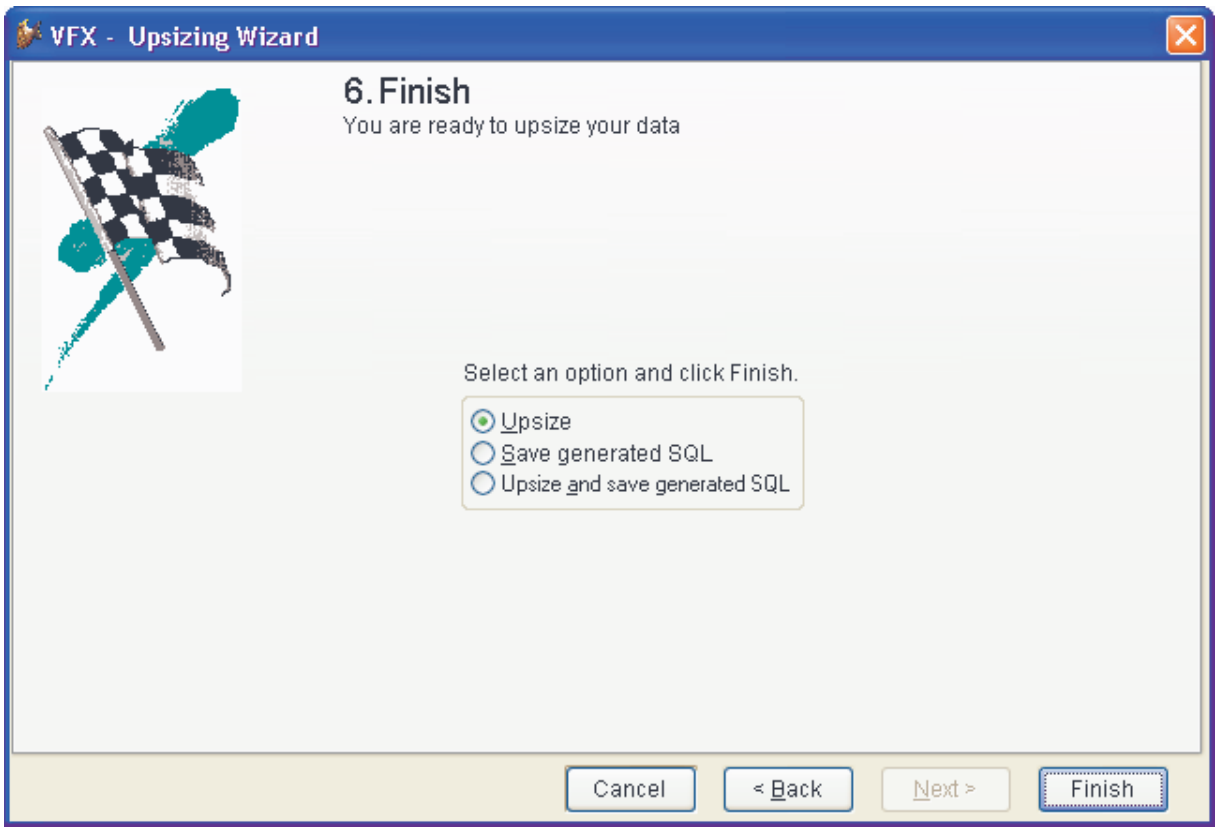
VFX – Upsizing Wizard, Choose Tables and Map Field Data Types

- **Choose Tables and Map Field Data Types** – Hier kann ausgewählt werden, welche Tabellen aus der lokalen Datenbank auf den SQL Server portiert werden sollen. Hier könnten einzelne Tabellen von der Portierung ausgeschlossen werden. Der VFX – Upsizing Wizard macht standardmäßig einen Vorschlag der sinnvoll zu portierenden Tabellen. Tabellen, die in das Projekt eingeschlossen sind, werden dabei automatisch nicht für die Portierung vorgeschlagen. So wird die Tabelle *Vfxmsg.dbf* zum Beispiel nicht auf den SQL Server portiert. Diese Tabelle wird in Exe-Dateien einbezogen. Hierin befinden sich die Texte für die Lokalisierung zur Laufzeit der Anwendung. Der Upsizing Wizard erkennt auch welchen Tabellen ein Timestamp-Feld hinzugefügt werden sollte, um eine effiziente Aktualisierung der Remote Daten zu ermöglichen. Bei Bedarf können Tabellen auch ID-Felder hinzugefügt werden. Auch eine Typkonvertierung von Feldern aus der Quelldatenbank ist in diesem Schritt möglich. Es können Tabellen und auch Ansichten in die Remote Datenbank portiert werden. In der Regel sind die Standardeinstellungen auf dieser Seite vernünftig und man kann mit einem Klick auf die Schaltfläche *Next* zum nächsten Schritt gelangen.



VFX – Upsizing Wizard, Set Upsizing Options

- **Set Upsizing Options** – Ähnlich dem VFP – Upsizing Wizard können hier noch weitere Optionen eingestellt werden. Die Empfehlung ist auch hier die Standardeinstellungen einfach so zu belassen.



VFX – Upsizing Wizard, Finish

- **Finish** – Im letzten Dialogschritt sehen wir die gleichen Optionen wie im VFP – Upsizing Wizard. Hier kann ausgewählt werden, ob die Portierung durchgeführt werden soll oder ob nur ein SQL Skript generiert werden soll. Mit einem Klick auf die Schaltfläche *Finish* wird die Portierung durchgeführt.

Nach einem kleinen Moment ist die Portierung beendet. Es wird ein kleines Projekt generiert, das einige Berichte mit Informationen zur durchgeführten Portierung enthält.

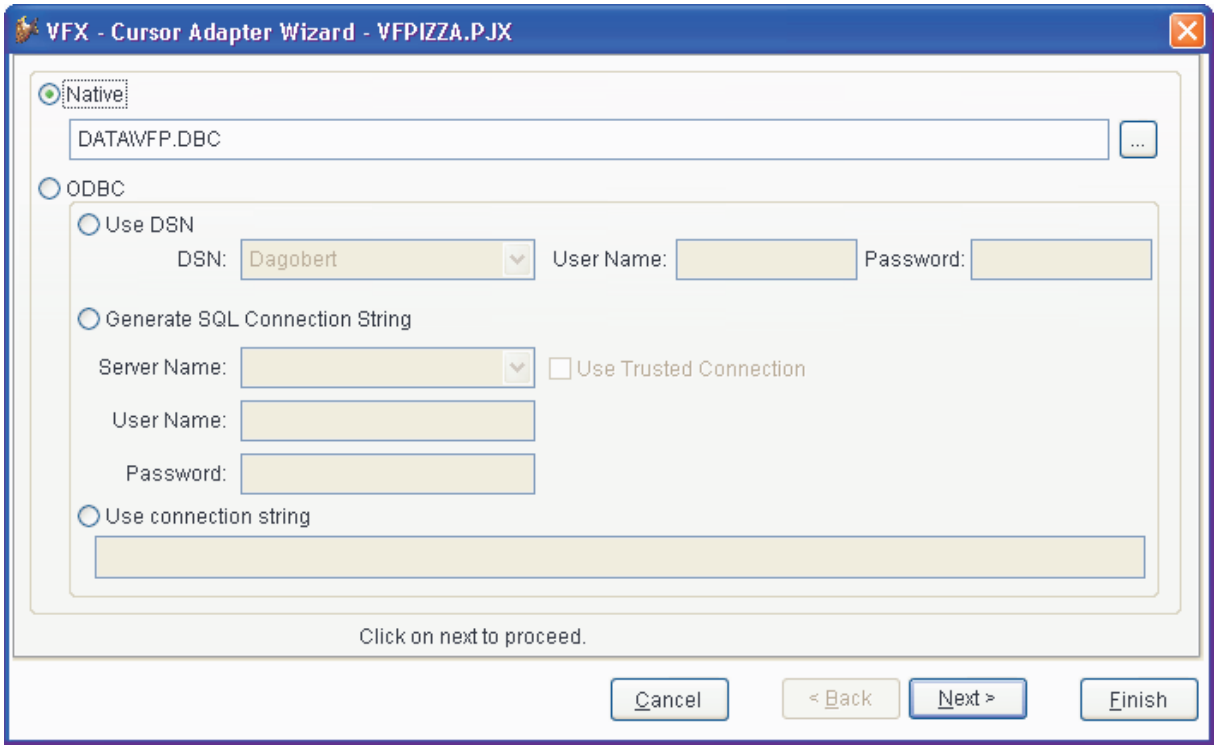
Der Data Explorer von VFP

Der Data Explorer kann aus der VFP Task Pane gestartet werden. Der Data Explorer ermöglicht den Zugriff auf VFP-Datenbanken und auf Remote Datenbanken. Durch die Auswahl im Treeview-Steuerelement im linken Teil des Data Explorer kann ein Datenbank-Server selektiert werden. Hier finden wir die mit dem VFX – Upsizing Wizard erstellten Datenbanken, auch wenn wir die Administrationswerkzeuge des SQL Server nicht zur Verfügung haben. In dem Treeview kann man den Knoten der Datenbank öffnen und sieht die einzelnen, in der Datenbank enthaltenen Tabellen. Zu den Tabellen kann die Struktur angezeigt werden. Im Kontextmenü der Tabelle kann mit *Browse* der Inhalt angezeigt werden. Wir sehen, dass die Daten vom VFX – Upsizing Wizard korrekt portiert wurden. Auch Schlüsselwerte finden wir in der Datenbank wieder. Der VFP – Upsizing Wizard unterstützt das Portieren von Schlüsselwerten leider nicht und vergibt in der Remote Datenbank neue Schlüssel.

VFX – CursorAdapter Wizard

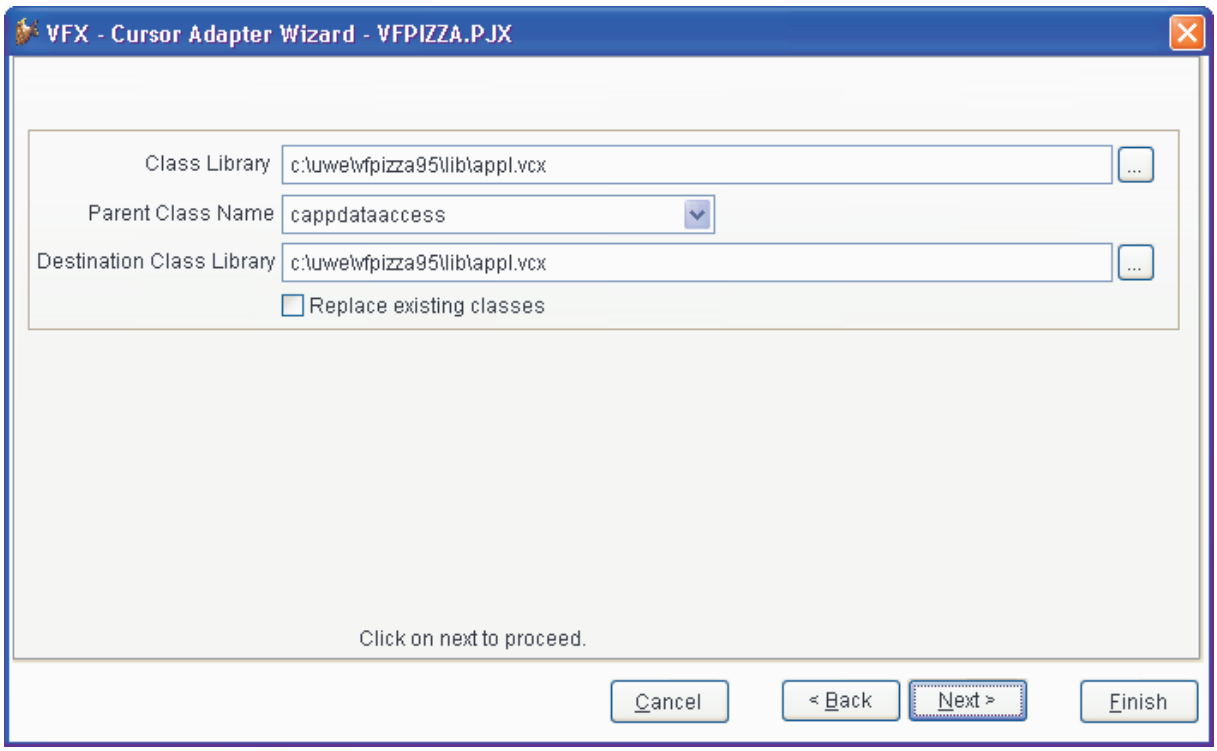
Wie kommen wir nun zu den CursorAdaptern? Wir hatten gesagt, dass wir in VFX eine Klasse *CBaseDataAccess* haben, die die Grundfunktionalität für unsere CursorAdapter zur Verfügung stellt. Nun brauchen wir aber für jede Tabelle in unserer Datenbank einen eigenen CursorAdapter, in dem der Zugriff auf diese Tabelle eingestellt ist. Dazu gehört insbesondere die Beschreibung der Struktur der Tabelle. In VFX haben wir einen CursorAdapter Wizard, der uns zu jeder Tabelle aus unserer Datenbank eine CursorAdapter-Klasse erstellt. Bei diesen Datenbanken kann es sich um Remote Datenbanken handeln, auf die mit einer DSN oder mit einer Verbindungszeichenfolge zugegriffen wird. Es kann aber auch eine DBC-Datenbank sein. Wenn wir unsere Anwendung zunächst mit einem DBC entwickeln und zu einem späteren Zeitpunkt auf SQL Server

wechseln möchten, können wir die CursorAdapter basierend auf unseren DBC-Tabellen erstellen und zu einem späteren Zeitpunkt teilen wir unserer Anwendung mit, dass jetzt nicht mehr auf dem DBC, sondern auf einer Remote Datenbank gearbeitet werden soll. Für die Generierung der CursorAdapter macht dies keinen Unterschied.



VFX – CursorAdapter Wizard, Schritt 1

Im ersten Schritt wird die Quelldatenbank ausgewählt. CursorAdapter können basierend auf einer DBC-Datenbank oder auf einer Remote Datenbank erstellt werden.

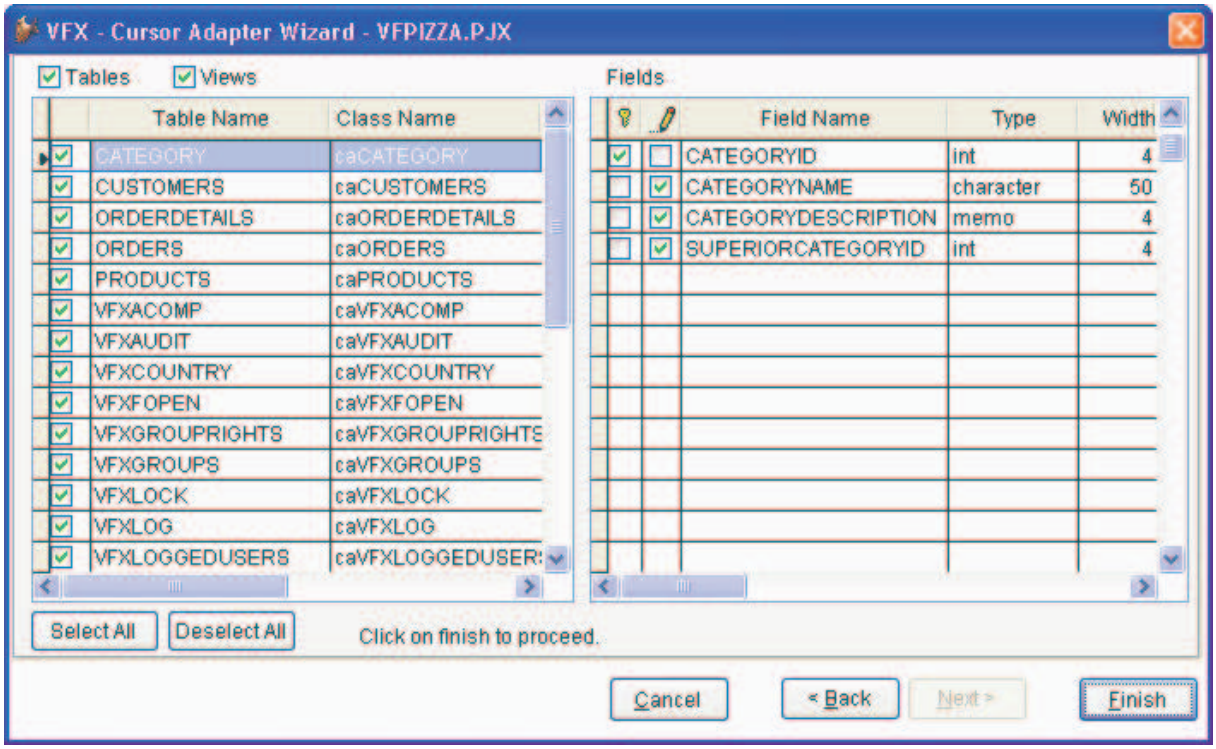


VFX – CursorAdapter Wizard, Schritt 2

Im zweiten Schritt erscheint eine Klassenauswahl. Die Klasse, auf der die generierten CursorAdapter basieren sollen, ist in der Regel *CAppDataAccess*. Dies ist eine 1:1-Ableitung der Klasse *CBaseDataAccess*. Entwickler, die den Funktionsumfang der Klasse *CBaseDataAccess* erweitern oder verändern wollen, können ihre Ände-

rungen also in der Klasse *CAppDataAccess* machen. Die Klasse *CAppDataAccess* ist in der Klassenbibliothek *Appl.vcx* gespeichert. Diese Klasse ist die Basisklasse für die zu generierenden CursorAdapter.

Die generierten CursorAdapter werden standardmäßig in der Klassenbibliothek *Appl.vcx* gespeichert. Bei Bedarf kann hier eine andere Klassenbibliothek eingestellt werden. In der Regel brauchen in diesem Schritt des Assistenten aber keine Änderungen gemacht zu werden.



VFX – CursorAdapter Wizard, Schritt 3

Im letzten Schritt kann eingestellt werden, für welche Tabellen CursorAdapter generiert werden sollen. Standardmäßig sind hier alle Tabellen ausgewählt. Zu jeder Tabelle wird die Struktur angezeigt. Der CursorAdapter Wizard erkennt automatisch Schlüsselfelder. Alle Felder, außer den Schlüsselfeldern, werden als aktualisierbar vorgeschlagen. Auch hier bietet der Assistent in der Regel eine sinnvolle Einstellung an und man kann einfach auf *Finish* klicken, um die CursorAdapter-Klassen erstellen zu lassen.

Anschließend sehen wir auf der Seite *Klassen* des VFP Projekt-Managers in der Klassenbibliothek *Appl.vcx* die neu erstellten Klassen. Wir haben jetzt CursorAdapter zur Verfügung, die so direkt in der Datenumgebung von Formularen verwendet werden können.

Manage Config.vfx

Das Ziel ist es den Datenzugriff so zu gestalten, dass zur Laufzeit der Anwendung zwischen verschiedenen Datenquellen umgeschaltet werden kann. Wie bereits erwähnt sind die Informationen zum Datenzugriff in der Datei *Config.vfx* gespeichert. Diese Datei ist verschlüsselt. Das Kennwort befindet sich in der Eigenschaft *goProgram.cconfigpassword*. Diese Eigenschaft ist standardmäßig leer. In diesem Fall verwendet VFX intern ein Kennwort. Die Datei *Config.vfx* ist also in jedem Fall verschlüsselt. Entwickler, die ein individuelles Kennwort verwenden möchten, können dies in dieser Eigenschaft eintragen. Die verschlüsselte Datei bietet einen guten Schutz vor unbefugtem Zugriff.

In der Entwicklungsumgebung finden wir im VFX-Menü den Menüpunkt *Data, Manage Config.vfx*. Hierüber kann der Inhalt dieser verschlüsselten Datei bearbeitet werden. Über die Schaltfläche *Add* können neue Datensätze hinzugefügt werden. In der Spalte *Connection Type* kann der Typ der Datenquelle ausgewählt werden. Bei der Verwendung von Remote Datenbanken steht auch hier der Wizard für Verbindungszeichenfolgen zur Verfügung. In diesem Wizard kann nach Eingabe der Anmeldeinformation die Datenbank ausgewählt werden.

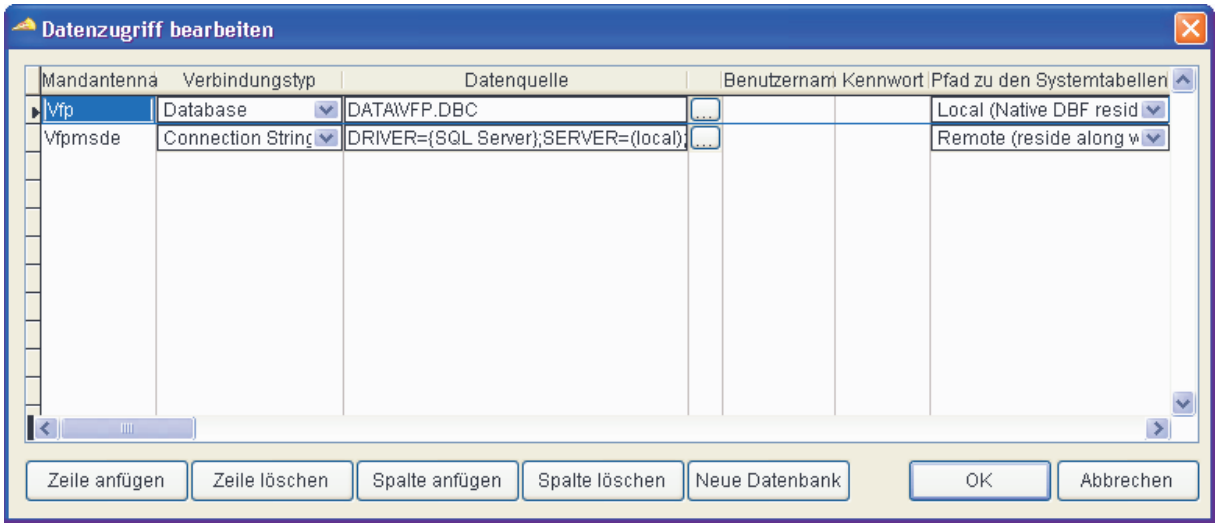
In der Spalte *System Table Location* kann angegeben werden wo sich die VFX-Tabellen befinden. Diese Tabellen können auch bei Verwendung von Remote Datenbanken als freie Tabellen vorliegen. Die VFX-Tabellen können aber auch in der Remote Datenbank gespeichert werden. Alternativ können die VFX-Tabellen

in einer eigenen Remote Datenbank gespeichert werden. So kann eine Gruppe von VFX-Tabellen für mehrere Remote Datenbanken verwendet werden.

Im Dialog *Manage Config.vfx* sind außerdem alle Spalten zu sehen, die wir auch aus der Tabelle *Vfxpath.dbf* kennen. Damit ist *Config.vfx* voll kompatibel zum bisherigen Konzept der Mandantenfähigkeit. Der Datei *Config.vfx* können über die Schaltfläche *Add Column* zusätzliche Felder hinzugefügt werden. So kann die Struktur beliebig erweitert werden.

In der Spalte *Client Name* wird der Name der Datenquelle angezeigt, wie er beim Start der Anwendung im Mandantenauswahldialog angezeigt wird. Der Mandantenauswahldialog erscheint automatisch beim Start einer Anwendung, wenn sich in der Datei *Config.vfx* bzw. in der Tabelle *VFXPath.dbf* mehr als ein Datensatz befindet.

Der gleiche Dialog zur Bearbeitung der Datenquellen steht auch unseren Endanwendern zur Verfügung.



Manage Config.vfx lokalisiert zur Laufzeit einer Anwendung

Wer schon in früheren VFX-Versionen mit mandantenfähigen Anwendungen gearbeitet hat, kennt die Tabelle *VFXPath.dbf*, in der der Zugriff auf mehrere VFP-Datenbanken eingetragen werden kann. Das neue Konzept der *Config.vfx* ist hierzu voll kompatibel. In vorhandenen Anwendungen, in denen bisher mit der *VFXPath.dbf* gearbeitet wurde, kann problemlos auf die Verwendung von *Config.vfx* umgeschaltet werden. Wenn beide Dateien in einem Anwendungsordner vorhanden sind, wird bevorzugt die Datei *Config.vfx* verwendet. Wenn *Config.vfx* nicht vorhanden ist, sucht die Anwendung automatisch nach *VFXPath.dbf*.

Bevor wir den so vorbereiteten Zugriff auf die Remote Datenbank nutzen können, müssen wir Formulare erstellen, die CursorAdapter als Datenquelle verwenden.

CDataFormPage-Formulare mit CursorAdapter

Wir wollen nun ein Formular basierend auf der Formularklasse *CDataFormPage* erstellen und als Datenquelle einen CursorAdapter verwenden. Wir verwenden dazu den VFX – CDataFormPage Builder, den wir bereits bei der Arbeit mit Tabellen kennen gelernt haben. Genau wie bei der Arbeit mit Tabellen wird das Formular zunächst mit dem VFX – Form Wizard angelegt. Hier geben wir dem Formular den Namen. Anschließend startet der VFX – Data Environment Builder, in dem wir einen CursorAdapter hinzufügen. Bei einem Klick auf die Schaltfläche *Add CA* öffnet sich der Klassenauswahldialog und es ist die Klassenbibliothek *Appl.vcx* aus dem aktuellen Projekt geöffnet. Hier kann eine vom VFX – CursorAdapter Wizard generierte Klasse ausgewählt werden. An der ausgewählten CursorAdapter-Klasse und an der Datenumgebung brauchen keine weiteren Einstellungen gemacht werden. Mit einem Klick auf die Schaltfläche *Next* kommen wir zum VFX – CDataFormPage Builder. Im Form Builder stehen genau die gleichen Optionen, wie bei der Arbeit mit Tabellen zur Verfügung.

Dem Formular werden Steuerelemente für die Bearbeitungsseiten und Spalten für das Such-Grid hinzugefügt. Weitere Einstellungen sind nicht erforderlich und das Formular kann durch einen Klick auf *OK* generiert werden. Im VFP Formular-Designer kann das generierte Formular kontrolliert und anschließend gespeichert werden.

Zum Test des Formulars starten wir *Vfxmain.prg*. Im Mandantenauswahldialog wählen wir die SQL Server-Datenbank aus. Das neu erstellte Formular kann aus dem Öffnen-Dialog gestartet werden. Das Formular funktioniert zur Laufzeit genauso wie das Formular, das auf Tabellen als Datenquelle basiert. Wir machen nun eine Änderung im ersten Datensatz und speichern die Daten. Jetzt beenden wir die Anwendung und starten erneut, wählen diesmal aber die DBC-Datenbank aus. Wir starten unser neu erstelltes Formular und sehen jetzt die Daten aus der DBC-Datenbank.

Damit haben wir unser Ziel erreicht und ein Formular erstellt, bei dem die Datenquelle zur Laufzeit der Anwendung eingestellt werden kann.

Entwickler, die Erfahrung mit dem SQL Server haben, werden noch eine Eigenschaft an unserem Formular bemängeln. Der Nachteil ist im Moment, dass bei jedem Laden des Formulars alle Datensätze aus der SQL Server Basistabelle in einen lokalen Cursor für unser Formular übertragen werden. Wir wollen den Zugriff auf die Tabelle jetzt noch so optimieren, dass beim Laden des Formulars nicht alle Datensätze vom SQL Server geholt werden.

Where-Klauseln

Den optimierten Zugriff auf die Daten können wir in einem CursorAdapter durch eine Where-Klausel erreichen. CursorAdapter haben eine Eigenschaft *SelectCMD*, in die ein SELECT-Befehl eingetragen wird. Dies ist ein ganz normaler SELECT-Befehl, der unter anderem auch eine Where-Klausel enthalten kann. In VFX 9.5 hat die CursorAdapter-Klasse eine zusätzliche Eigenschaft *cwhereclause*, in die nur die Where-Klausel eingetragen wird. Dadurch kann die Eigenschaft *SelectCMD* unverändert bleiben.

In der Klassenbibliothek *Appl.vcx* sehen wir bei den CursorAdapter-Klassen die Eigenschaften, die der VFX – CursorAdapter Wizard eingetragen hat. Hier ist der generierte Select-Befehl in der Eigenschaft *SelectCMD* zu sehen. Dieser Select-Befehl wird ausgeführt, wenn das Formular geladen wird.

In unserem Formular sehen wir in der Datenumgebung bei diesem CursorAdapter im Eigenschaftsfenster, dass alle Eigenschaften ihren Standardwert haben. Wenn wir hier der Eigenschaft *SelectCMD* eine Where-Klausel hinzufügen würden, hätte das den Nachteil, dass wir die Vererbung durchbrechen würden.

Wenn die Struktur der Basistabelle später geändert wird, können wir den VFX – CursorAdapter Wizard erneut laufen lassen, um die geänderte Tabellenstruktur in die Eigenschaften des CursorAdapters zu übernehmen. Wenn wir nun die Eigenschaft *SelectCMD* direkt im CursorAdapter auf dem Formular ändern würden, hätten wir keine Möglichkeit von den geänderten Eigenschaften der CursorAdapter-Klassen in der Klassenbibliothek *Appl.vcx* profitieren zu können. Die Eigenschaft *SelectCMD* müsste manuell nachgearbeitet werden. Bei sehr vielen CursorAdaptern und sehr vielen Formularen könnte dies eine aufwändige Arbeit werden.

Darum gibt es in VFX 9.5 bei CursorAdaptern die Eigenschaft *cwhereclause*, in der die Where-Klausel unabhängig vom Select-Befehl gespeichert werden kann. Zur Laufzeit wird diese Where-Klausel automatisch dem Select-Befehl hinzugefügt. Der Vorteil ist, dass *SelectCMD* weiterhin aus der Vererbung stammt. Zu jeder Zeit kann der VFX – CursorAdapter Wizard erneut gestartet werden, um eine geänderte Datenbankstruktur in die CursorAdapter-Klassen zu übernehmen.

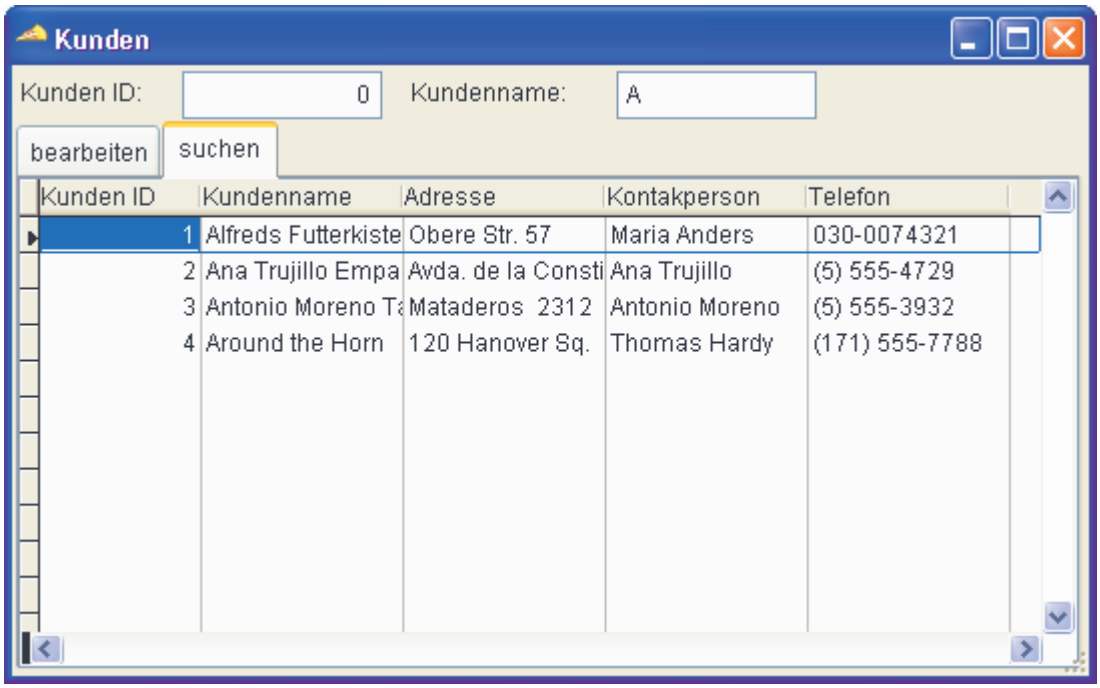
Eine Where-Klausel könnte so aussehen:

```
customername LIKE ?thisform.customername
```

Die Where-Klausel kann im Eigenschaftsfenster des CursorAdapters eingetragen werden, es kann aber auch der VFX – CDataFormPage Builder hierfür verwendet werden. Im VFX – Data Environment Builder gibt es eine Spalte *Where Clause*. Hier kann die Where-Klausel, genau wie oben im Beispiel angegeben, eingetragen werden.

Im nächsten Schritt, im Form Builder, wählen wir die Seite *View Parameters*. Hier fügen wir das Feld hinzu, das dem Ansichtsparameter entspricht, also *customername*. Als Parametername wird vorgeschlagen *tcustomername* zu verwenden. Dies wäre ein gut geeigneter Variablenname. Wir wollen jedoch eine Formulareigenschaft für den Parameter verwenden. Daher tragen wir als *Parameter Name* ein: *thisform.customername*, so wir es auch in der Where-Klausel geschrieben haben. Weitere Schritte sind nicht erforderlich. Wir haben dem CursorAdapter eine Where-Klausel gegeben und haben auf dieser Seite des Form Builders das dazugehörige Feld ausgewählt (linker Teil der Where-Klausel) sowie den Namen des Parameters wiederholt (rechter Teil der Where-Klausel). Detailliertere Hinweise zu den Eingabemöglichkeiten auf der Seite View Parameters des Form Builders finden Sie in den Session Notes zur Session V-FXDB.

Bei einem Klick auf *OK* fügt der Form Builder dem Formular am oberen Rand eine Textbox sowie ein Label hinzu, in der der Benutzer zur Laufzeit einen Wert für den Parameter eintragen kann. Der Form Builder fügt dem Formular auch automatisch die Formulareigenschaft hinzu, die für den Wert des Parameters benötigt wird. Der Form Builder fügt der Textbox für die Eingabe des Parameters Code in den Ereignissen *Init* und *Valid* hinzu. Wer an der Funktionsweise näheres Interesse hat, kann sich diesen Code ansehen und bei Bedarf auch entsprechend seinen Wünschen anpassen.



Formular zur Kundenbearbeitung mit Eingabefeldern für Parameter

Beim Test des Formulars fällt auf, dass das Formular ohne Daten geladen wird. Es sind nur das Label und die Textbox zur Eingabe des Parameters am oberen Formularrand sichtbar. VFX läd CursorAdapter mit einer Where-Klausel automatisch ohne Daten. Wir geben in der Textbox jetzt einen Kundennamen ein. Uns fällt auf, dass in der Standard-Symbolleiste jetzt die Schaltfläche zur Aktualisierung der Ansicht enabled ist. Durch einen Klick auf diese Schaltfläche wird bei dem CursorAdapter die *CursorRefresh()* Methode aufgerufen und das Formular wird mit Daten gefüllt. Damit haben wir einen sehr gut optimierten Zugriff auf die Daten erreicht. Es werden optimal wenige Daten aus der Datenbank geholt und es entsteht auch optimal wenig Datenverkehr im Netzwerk.

COneToMany-Formulare mit CursorAdapter

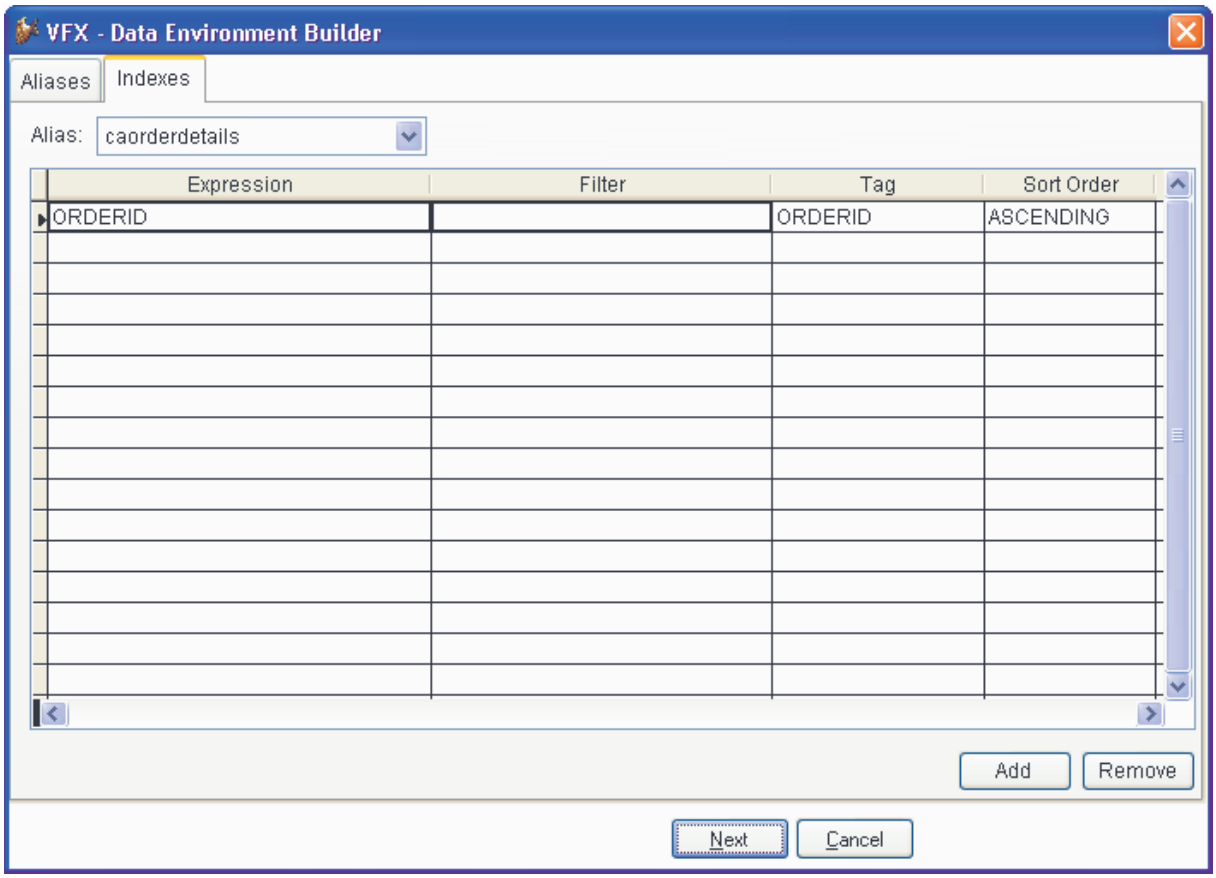
Eine etwas kompliziertere Formularklasse ist die Klasse *cOneToMany*. Hiermit werden 1:n-Beziehungen in einem Formular abgebildet. Die Parent-Daten werden in einem Seitenrahmen bearbeitet, so ähnlich wie dies in Formularen basierend auf der Klasse *cDataFormPage* möglich ist. Unterhalb des Seitenrahmens können Child-Daten bearbeitet werden. Child-Daten können wahlweise in Grids oder in beliebigen anderen Steuerelementen bearbeitet werden.

Um dieses Formular zu erstellen, verwenden wir den VFX – COneToMany Builder. Alle Einstellungen in diesem Builder werden genauso gemacht, als wenn mit Tabellen als Datenquellen gearbeitet würde. Der Unterschied ergibt sich in der Datenumgebung. Hier verwenden wir CursorAdapter statt Tabellen.

Wir starten den VFX – Form Wizard, geben dem Formular einen Namen und wählen die Klasse *COneToMany* aus. Nach einem Klick auf *Next* erscheint der VFX – Data Environment Builder. Zunächst wollen wir die Datenumgebung so aufbauen, wie wir es auch mit Tabellen tun würden. Anschließend werden wir zeigen, wie der Zugriff auf die Daten optimiert werden kann.

Im Unterschied zu Tabellen haben CursorAdapter keine Indexschlüssel. Wir können aber im VFX – Data Environment Builder temporäre Indexdateien für CursorAdapter erstellen. Wir brauchen für den Child-CursorAdapter einen Indexschlüssel, über den die Beziehung zum Parent-CursorAdapter hergestellt wird. Auch die Beziehung kann im VFX – Data Environment Builder erstellt werden.

Zunächst müssen im VFX – Data Environment Builder die Datenquellen für das Formular ausgewählt werden. Für ein 1:n-Formular benötigen wir zwei Datenquellen, in diesem Fall zwei CursorAdapter. Über die Schaltfläche *Add CA* erscheint ein Öffnen-Dialog, der die Klassenbibliothek *Appl.vcx* geöffnet. In der Klassenauswahlliste sehen wir alle CursorAdapter-Klassen, die wir mit dem VFX – CursorAdapter Wizard generiert haben. Als erstes wird die Klasse *caOrders* ausgewählt. Dies ist der CursorAdapter für den Parent-Teil des Formulars. Als nächstes wird der CursorAdapter für die Child-Daten im Formular ausgewählt. Dies ist der CursorAdapter *caOrderDetails.*



Erstellen von Indexschlüsseln für CursorAdapter

Auf der Seite *Indexes* des VFX – Data Environment Builder können Indexschlüssel angelegt werden. Bei CursorAdaptern gibt es keine Indexschlüssel. Um eine Beziehung zwischen den beiden ausgewählten CursorAdaptern herstellen zu können, muss hier für den CursorAdapter *caOrderDetails* ein Indexschlüssel angelegt werden. Dazu wird der CursorAdapter *caOrderDetails* aus der Combobox *Alias* ausgewählt. Der zu verwendende Indexausdruck *OrderID* wird in der Spalte *Expression* eingetragen. In der Spalte *Tag* wird der Name des Indexschlüssels eingetragen. Auch hier geben wir *OrderID* ein. Damit sind alle Angaben gemacht, um zur Laufzeit einen temporären Indexschlüssel anzulegen. Der VFX – Data Environment Builder fügt eine Zeile Code in der Methode *CreateIndexes* des gewählten CursorAdapters ein. Zu jedem CursorAdapter können beliebig viele Indexschlüssel angelegt werden.

Auf der Seite *Aliases* wird jetzt die Beziehung zwischen den beiden CursorAdaptern hergestellt. Dafür wird in der Zeile mit dem CursorAdapter *caOrderDetails* die Order *OrderID* ausgewählt. Für die Beziehung muss ein Parent-Alias in der Spalte *Parent* ausgewählt werden. Hier wählen wir den CursorAdapter *caOrders* aus. Alle Comboboxen sind bereits mit den sinnvoll auswählbaren Werten vorbelegt. Als Beziehungsausdruck wird *OrderID* in der Spalte *Rel Expression* gewählt. Das ist das Feld im Parent-Cursor, auf das die Beziehung zeigen soll. Auf allen Formularklassen von VFX wird zur Laufzeit ein Objekt *oRelationMgr* instanziiert. In diesem Objekt gibt es die Methode *CreateRelations*. Der VFX – Data Environment Builder trägt in diese Methode eine Zeile Code für jede angelegte Beziehung an. Damit ist die Datenumgebung fertig.

Mit einem Mausklick auf die Schaltfläche *Next* wird der VFX – COneToMany Builder gestartet. Genau wie bei 1:n-Formularen basierend auf Tabellen können hier dem Formular Steuerelemente hinzugefügt werden. Auf der Seite *Grid Page* können dem Such-Grid Spalten hinzugefügt werden. Auf der Seite *Children* wird der Child-Teil des Formulars erstellt. In der Combobox *Child Table* wird die Datenquelle für die Child-Daten ausgewählt. Dies ist der CursorAdapter *caOrderDetails.* Die Relation haben wir bereits im VFX – Data Environment Builder eingestellt. Durch die Auswahl von *caOrderDetails* wechselt die Ansicht im Field

Assistant und hier uns jetzt alle Felder des CursorAdapters *caOrderDetails* angezeigt. Die Felder, die im Child-Grid zur Verfügung stehen sollen, können im Field Assistant mit einem Doppelklick ausgewählt werden.

Mit einem Klick auf die Schaltfläche *OK* wird das Formular generiert. Auch dieses Formular wird von VFX automatisch in die Tabelle *Vfxfopen* und damit in den Öffnen-Dialog eingetragen. Damit ist das Formular fertig und kann gespeichert werden.

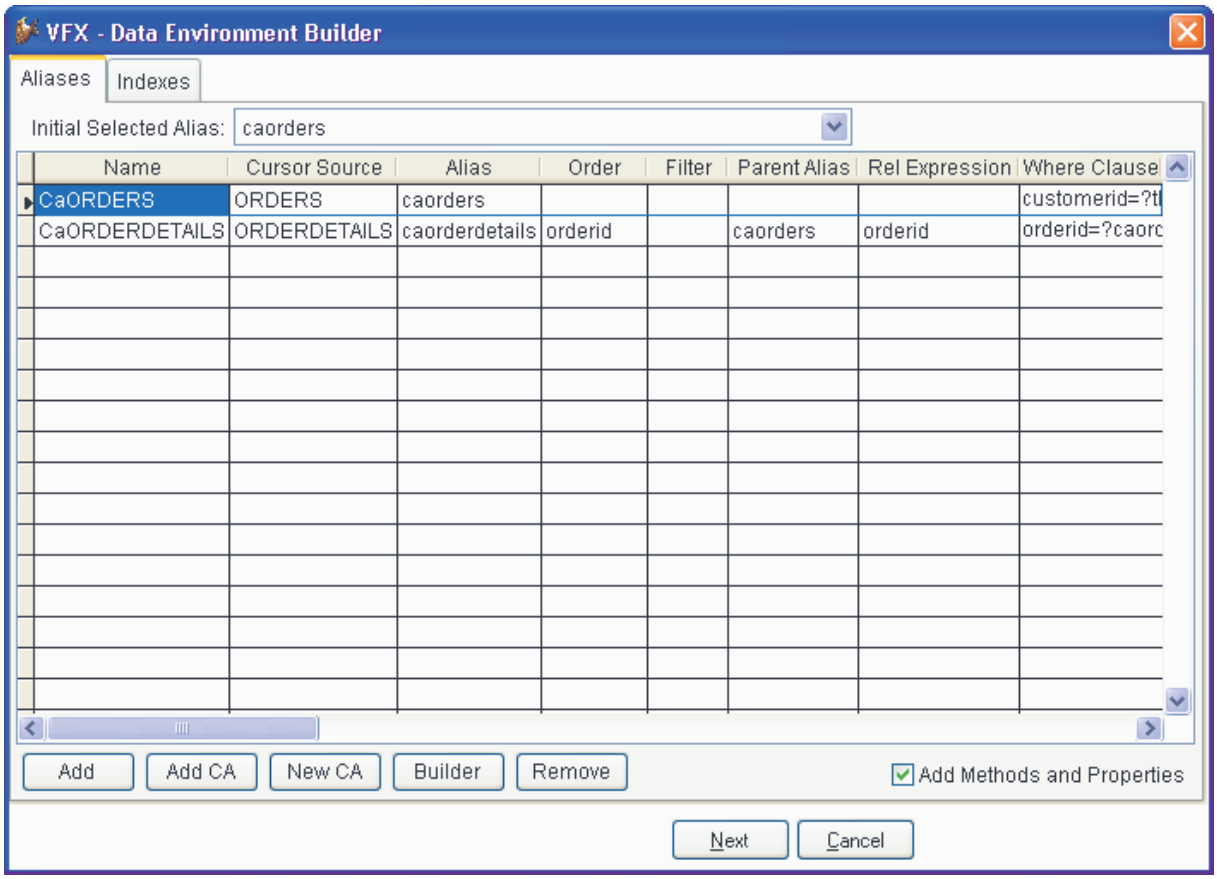
Zum Test starten wir *Vfxmain.prg* und wählen die VFP-Datenbank aus. Wir sehen im Öffnen-Dialog das neu erstellte 1:n-Formular. Wir sehen, dass im Child-Teil des Formulars, die zum dem aktuellen Parent-Datensatz gehörenden Datensätze angezeigt werden. Wenn der Datensatzzeiger im Parent-Teil bewegt wird, wird die Ansicht im Child-Teil des Formulars automatisch aktualisiert. Die im VFX – Data Environment Builder erstellte Relation funktioniert also.

Wenn wir mit sehr großen Datenmengen umzugehen haben, hat dieses Formular aber noch einen Nachteil. Beim Laden des Formulars werden alle Datensätze aus der Tabelle *Orders* in den CursorAdapter *caOrders* geladen. Auch für den unteren Teil des Formulars mit den Child-Daten werden gleich beim Laden des Formulars alle Child-Datensätze aus der Tabelle *OrderDetails* geholt. Dieses Verfahren erzeugt beim Laden des Formulars im Netzwerk viel Verkehr. Das sollte möglichst vermieden werden. Darum wollen wir das Formular jetzt optimieren.

Dazu öffnen wir das Formular erneut im VFP Formular-Designer, um das Formular auf den optimierten Datenzugriff vorzubereiten. Über einen Rechtsklick wird aus dem Kontextmenü der VFX – Data Environment Builder gestartet. Im CursorAdapter *caOrderDetails* sollen jetzt nur noch die Datensätze geladen werden, die der aktuellen *OrderID* aus dem CursorAdapter *caOrders* entsprechen. Dazu muss beim CursorAdapter *caOrderDetails* eine Where-Klausel eingetragen werden. Die Where-Klausel muss lauten:

```
orderid=?caorders.orderid
```

Durch diese Where-Klausel werden in den CursorAdapter *caOrderDetails* nur die erforderlichen Datensätze geladen. Das ist die einzige Änderung, die im VFX – Data Environment Builder gemacht werden muss.



Relationen und Where-Klauseln im VFX – Data Environment Builder

Mit einem Klick auf *OK* kommen wir zum VFX – COneToMany Builder und mit einem Klick auf *OK* werden die erforderlichen Änderungen im Formular gemacht. Das Formular kann gespeichert und anschließend getestet werden.

Sobald zur Laufzeit der Datensatzzeiger im CursorAdapter *caOrders* bewegt wird, wird automatisch ein *CursorRefresh()* des CursorAdapters *caorderDetails* ausgeführt. In früheren Versionen von VFX war es erforderlich, in der Formularmethode *OnChildRequery()* manuell Code einzutragen, um die Child-Daten zu aktualisieren. In VFX 9.5 ist dies nicht mehr erforderlich. VFX sorgt für die Aktualisierung der Child-Daten automatisch selbst.

Zum Test starten wir erneut *Vfxmain.prg*, wählen diesmal aber die SQL Server-Datenbank aus. Wir sehen im Verhalten des Formulars keine Veränderung. Im Child-Grid werden bei jeder Bewegung des Datensatzzeigers, die erwarteten Child-Daten angezeigt. Die Ladezeit des Formulars hat sich verringert. Es werden nicht mehr alle Datensätze aus der Tabelle *OrderDetails* in den CursorAdapter *caOrderDetails* geholt, sondern nur noch die wenigen benötigten Datensätze, die zu einem Auftrag gehören.

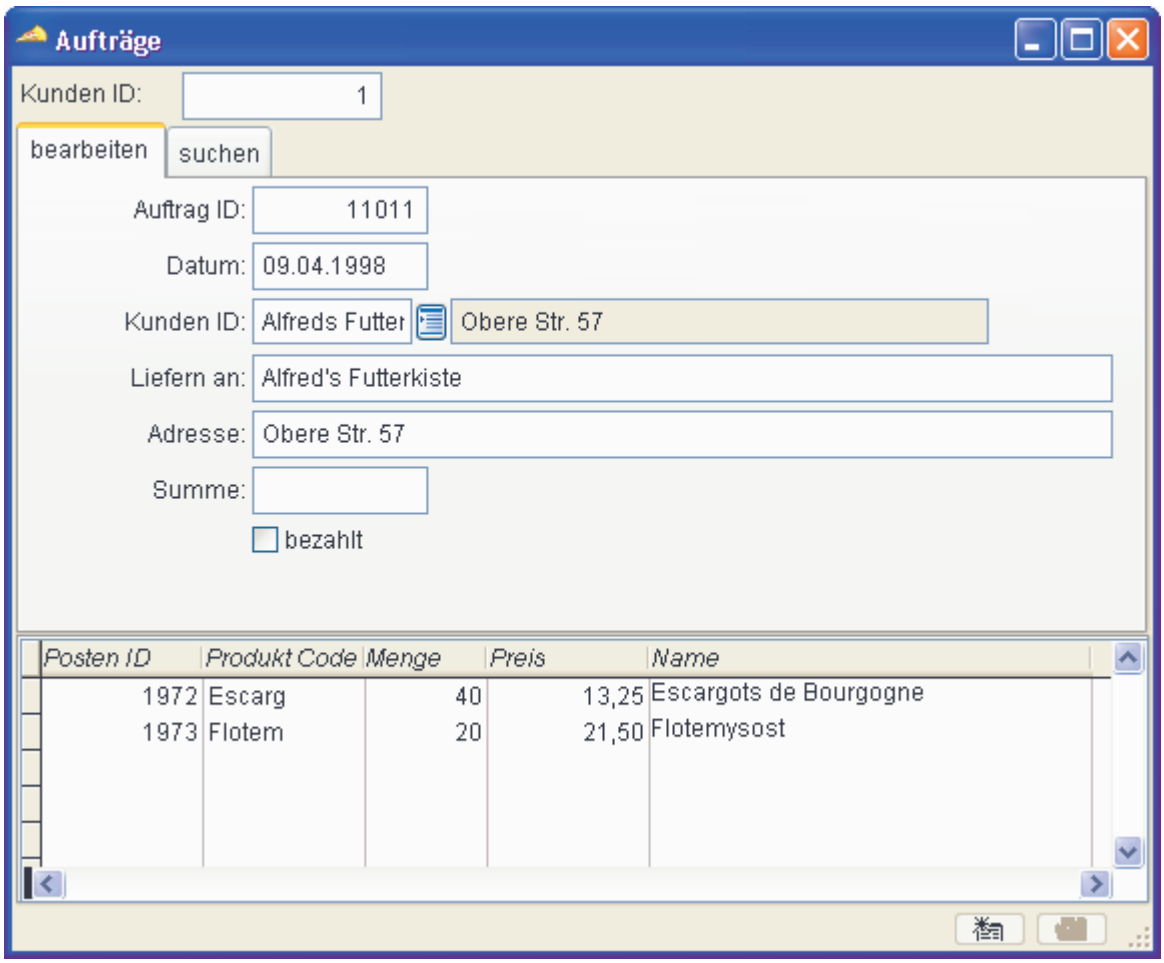
Jetzt haben wir immer noch den Nachteil in unserem Formular, dass beim Laden alle Datensätze aus der Tabelle *Orders* in den CursorAdapter *caOrders* geholt werden. Auch dafür wollen wir den Zugriff optimieren.

Dazu öffnen wir das Formular nochmals im VFP Formular-Designer. Im VFX – Data Environment Builder fügen wir dem CursorAdapter *caOrders* nun ebenfalls eine Where-Klausel hinzu. Es sollen nur noch die Datensätze angezeigt werden, die eine bestimmte *CustomerID* haben. Als Where-Klausel tragen wir ein:

```
customerid=?thisform.tcustomerid
```

In der Formulareigenschaft *thisform.tcustomerid* soll die *CustomerID* abgelegt werden, zu der die Aufträge angezeigt werden sollen. Im VFX – Data Environment Builder sind keine weiteren Einstellungen erforderlich. Mit einem Klick auf die Schaltfläche *Next* kommen wir zum VFX – COneToMany Builder. Hier gehen wir auf die Seite *View Parameters*. Wir fügen das Feld hinzu, das wir auch in der Where-Klausel verwendet haben, also *CustomerID* aus dem CursorAdapter *caOrders*. Als Parametername schlägt mit der Builder *tcustomerid* vor. Das wäre so, wie es der Builder vorschlägt, eine Variable. Wir wollen jedoch eine Formulareigenschaft als Parameter verwenden und schreiben daher noch *thisform.* davor. Damit sind wir mit dem VFX – COneToMany Builder fertig und können auf *OK* klicken.

Am oberen Rand des Formulars fügt der Builder ein Label und eine Textbox zur Eingabe des Parameters hinzu. Auch die Formulareigenschaft *tCustomerID* wird vom Builder dem Formular hinzugefügt. Damit sind alle Änderungen gemacht, um einen optimierten Zugriff auf die Auftragsköpfe zu ermöglichen.



Das Auftragsformular zur Laufzeit

Zum Test starten wir *Vfxmain.prg* und wählen die SQL Server-Datebank aus. Das Formular startet „leer“. Beim Laden des Formulars werden also tatsächlich keine Daten aus der Datenbank gelesen. Am oberen Rand des Formulars sehen wir die Textbox zur Eingabe einer *CustomerID*. Erst durch die Eingabe einer *CustomerID* und die Aktualisierung der Ansicht werden die Daten aus der Datenbank geholt. Es werden nur die Datensätze aus der Tabelle *Orders* geholt, deren *CustomerID* dem eingegebenen Wert entspricht. Für den Child-Teil des Formulars werden nur die Datensätze in den CursorAdapter *caOrderDetails* geholt, die dem aktuellen Parent-Datensatz zugeordnet sind. Damit ist dieses Formular sehr gut optimiert und es werden keine überflüssigen Daten aus der Datenbank geholt. Dadurch haben wir einen sehr schnellen Datenzugriff erreicht.

Parent/Child-Beziehungen mit CursorAdapter

Wir wollen nun eine Parent/Child-Beziehung zwischen Formularen mit CursorAdaptern als Datenquelle herstellen. Wir verwenden dazu die beiden Formulare *KundenCA.scx* und *AuftragCA.scx*. Um die Beziehung herzustellen benutzen wir den VFX – Parent/Child Builder, den wir bereits in der Session V-FXDB kennen gelernt haben. Hier wollen wir daher nur auf die Besonderheiten in Zusammenhang mit CursorAdaptern eingehen.

Der Aufruf des Child-Formulars erfolgt zur Laufzeit mit der Funktionstaste *F6*, genau wie bei der Arbeit mit Tabellen, ist es auch bei CursorAdaptern als Datenquelle möglich, mehrere Children aus einem Parent-Formular aufzurufen. Ein Child-Formular kann wiederum eigene Children haben, sodass eine hierarchische Verschachtelung von Formularen möglich ist.

Zur Einrichtung der Parent/Child-Beziehung wird das Formular *KundenCA.scx* im VFP Formular-Designer geöffnet. Aus dem VFX-Menü wird unter Forms der VFX – Parent/Child Builder gestartet. Im Builder wird eine neue Zeile hinzugefügt und als *Command Type* wird *Child Form* gewählt. Das Child-Formular kann über die Schaltfläche in der dritten Grid-Spalte ausgewählt werden. Wir wählen hier das Formular *AuftragCA.scx*. Genau wie bei der Arbeit mit Tabellen analysiert VFX die beiden zu verbindenden Formulare, um uns die Arbeit so weit wie möglich abzunehmen. Unmittelbar nach Auswahl des Child-Formulars stehen daher schon einige weitere Einträge zur Verfügung. Der Parent/Child Builder hat ermittelt, dass das *Parent Field* das Feld

cacustomers.customerid ist. Dies ist der Schlüssel im CursorAdapter *cacustomers*. Im Child-Formular wird auf der Datenquelle *caorders* gearbeitet. Auch dort gibt es das Feld *customerid*.

Bei der Erstellung des Formulars *AuftragCA.scx* haben wir dem CursorAdapter *caorders* eine Where-Klausel hinzugefügt, die auf die *customerid* wirkt. Dies ist erforderlich, damit die Parent/Child-Beziehung funktioniert. Immer, wenn zur Laufzeit der Satzzeiger im Parent-Formular bewegt wird, soll der CursorAdapter *caorders* aktualisiert werden. Dafür wird von VFX automatisch die *CursorRefresh()*-Methode von *caorders* aufgerufen. Da wir dort in der Where-Klausel die *customerid* haben, werden anschließend genau die richtigen Datensätze angezeigt. Die weiteren Einstellungen im VFX – Parent/Child Builder sind genauso wie bei der Arbeit mit Tabellen auszufüllen.

Mit einem Klick auf *OK* wird der für die Parent/Child-Beziehung erforderliche Code generiert. Wir können das Ergebnis testen.

Wir starten das Formular *KundenCA.scx*. Im Eingabefeld für *Customername* geben wir *A* ein und nach einem Klick auf *Aktualisieren* in der Symbolleiste erscheinen alle Kunden, deren Name mit A anfängt. Jetzt klicken wir in der Symbolleiste auf *Weitere Funktionen* und es wird das Formular *AuftragCA.scx* geöffnet. Tatsächlich werden hier nur die Kunden angezeigt, die zum aktuellen Kunden gehören. Wenn der Datensatzzeiger im Kundenformular bewegt wird, wird gleichzeitig die Ansicht im Auftragsformular aktualisiert und es werden die zum jeweils aktuellen Kunden gehörenden Aufträge angezeigt. Das Auftragsformular ist ein OneToMany-Formular. Immer, wenn der Datensatzzeiger im oberen Formularteil bewegt wird, muss VFX die dazu gehörenden Child-Daten im unteren Formularteil anzeigen. Wenn der Datensatzzeiger im oberen Teil des Auftragsformulars bewegt wird, werden die Daten im CursorAdapter *caorderdetails* neu geholt.

Damit haben wir 1:n:m-Beziehung über zwei Formulare aufgebaut, die einen voll optimierten Datenzugriff hat. Aus jeder der drei beteiligten Datenquellen, also *cacustomers*, *caorders* und *caorderdetails* werden nur die unbedingt erforderlichen Datensätze geholt.