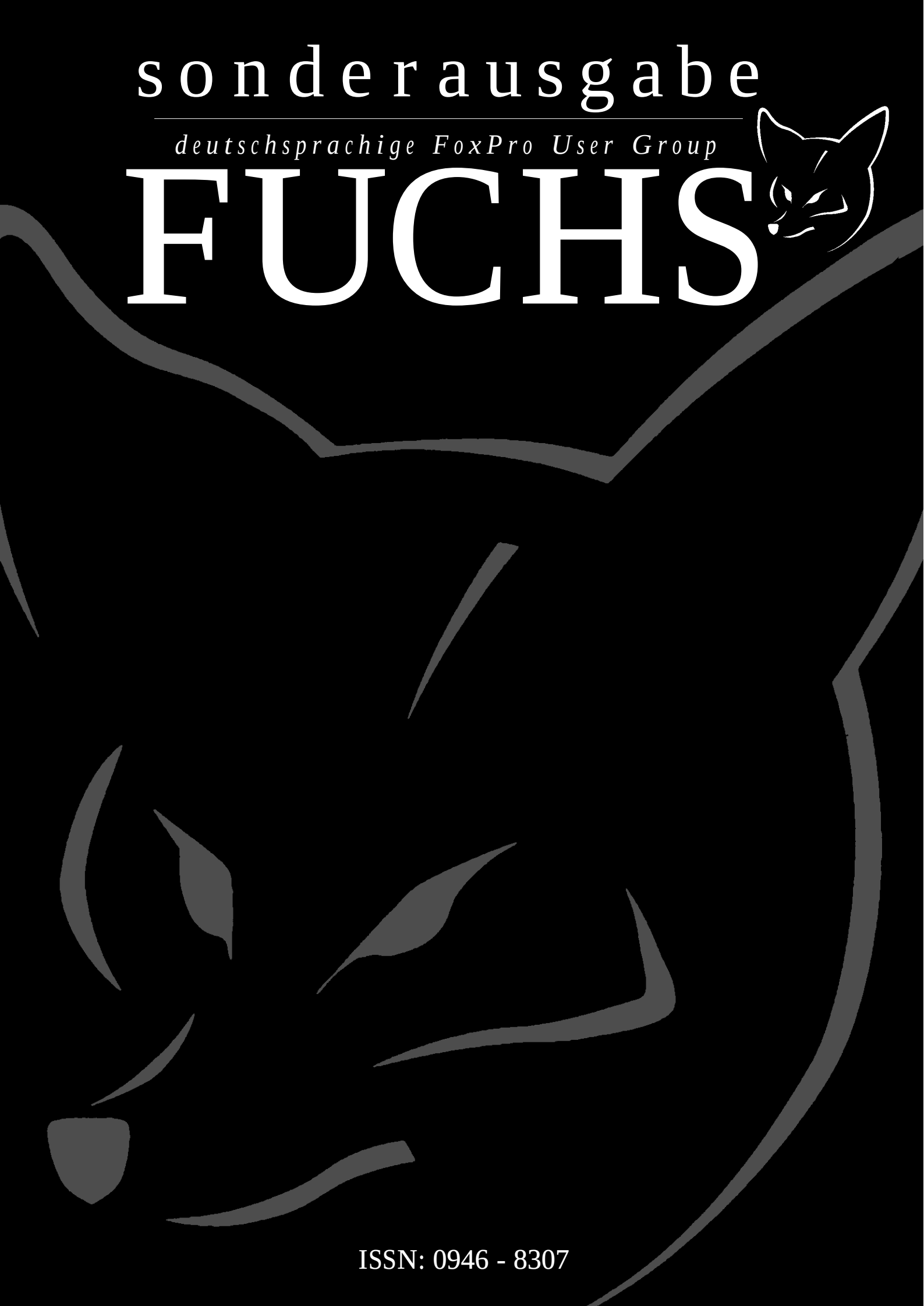
ANZEIGE

sonderausgabe

deutschsprachige FoxPro User Group

FUCHS



ISSN: 0946 - 8307

Vorwort

In dieser Sonderausgabe des Fuchs behandeln wir ausschliesslich einen Artikel von Colin Nicholls. Er beschreibt darin die Entwicklung einer einfachen Datenbankanwendung, die wahlweise auf VFPs Datenhaltung oder den SQL Server zugreift.

Nur – warum sollte man eigentlich den SQL Server einsetzen? Microsoft betont doch immer, daß sich Visual FoxPro in allen drei Schichten einer 3-Tier-Anwendung nutzen läßt.

Ein Grund könnte das Handbuch sein, da es bereits in der neuen Rechtschreibung verfaßt ist. So kann man sich beim Lesen schon mal daran gewöhnen.

Grundsätzlich wird man den SQL Server nur bei umfangreichen Anwendungen einsetzen. Bei einem Programm für den Handwerker um die Ecke, mit dem nur der Chef und sein Buchhalter arbeiten, werden Sie auch weiterhin FoxPros Datenbank nutzen.

Anders ist es, wenn Sie eine Anwendung für eine größere Firma entwickeln. Dort sehen Sie sich in der Regel mit umfangreichen Sicherheitsanforderungen konfrontiert, die VFP von Haus aus nicht bietet und die nur kompliziert, wenn überhaupt, zu programmieren sind. Mit dem SQL Server sind diese Funktionalitäten nur einige Mausklicks entfernt.

Wird die Anwendung noch umfangreicher steigen auch die Anforderungen an die Verfügbarkeit. Haben Sie eine Vorstellung von der Arbeit, die es bedeutet, eine Online-Datensicherung einer VFP-Datenbank durchzuführen? Auch dies ist eine Aufgabe, die der SQL Server automatisch verwaltet.

Beispiele dafür sind die Vergabe von Zugriffsrechten auf Feldebene, die Online-Sicherung von Daten, Datenmengen im Terabytebereich und die parallele Ausführung einer Abfrage auf mehreren Prozessoren. Weitere Vorteile des SQL Servers sind die einfache Administrierung, die ereignisgesteuerte Ausführung von Aufgaben und die automatische Alarmierung des Administrators, wenn Probleme auftreten...

Es würde dieses Vorwort und den Artikel sprengen, alle Vorteile des SQL Servers zu behandeln. Der Artikel soll lediglich aufzeigen, daß mit Visual FoxPro die Erstellung von Datenbanken für den SQL Server relativ problemlos möglich ist. Wenn Sie in der Zukunft vor dem Beginn eines umfangreichen Projekts stehen, sollten Sie sich schon Gedanken machen, ob es für Sie sinnvoll ist, den SQL Server einzusetzen. ■

Unterstützung unterschiedlicher Backends

Einführung

Visual FoxPro beherrscht den transparenten Zugriff auf unterschiedliche Backend-Datenbanken über Ansichten nun bereits seit mehreren Jahren. Aber bis vor etwa einem Jahr mußte ich diese Technik nie in einer realen Anwendung einsetzen.

Meine Firma erhielt von einem neuen Kunden den Auftrag, seine erfolgreich laufende DOS-Anwendung, die nicht in xBase erstellt war, auf 32-Bit-Windows zu portieren. Die meisten seiner Anwender waren damit einverstanden, ihre Daten in normalen XBase-Tabellen zu speichern. Ein Anwender, mit dem unser Auftraggeber die Hälfte seines Umsatzes erzeugte, machte jedoch zur Bedingung, daß die Daten im SQL Server gespeichert werden müssen. Anderenfalls hätte er sich einen anderen Lieferanten gesucht.

Es überrascht nicht, daß unser Auftraggeber bei der Suche nach einer entsprechenden Entwicklungsplattform auf Visual FoxPro stieß. Schließlich wird diese Fähigkeit von Microsoft in den Werbematerialien für VFP immer herausgestellt. Nun hatte unser Auftraggeber aber nie in einer Windows-Umgebung programmiert. So kamen wir ins Spiel. Wir wurden beauftragt, mit unserem Wissen unserem Auftraggeber zu helfen, die Windows-Version seiner Software zu erstellen, einschließlich des transparenten Datenzugriffs auf VFP-Tabellen oder einen SQL Server.

Bei diesem Projekt erlebten wir alle Abenteuer dieser Art der Anwendungsentwicklung: es dauerte länger als geplant, während die Arbeit fortschritt, fanden wir bessere Lösungen für Teilprobleme, und manche Funktionalitäten mußten mehrfach entwickelt werden. Wir hatten noch nie mit anderen Programmierern gearbeitet und viele der vorgebrachten Ideen waren uns fremd. Ich möchte darauf aber nicht weiter eingehen, da das nicht der Sinn dieses Artikels ist. Wir haben die Entwicklung der Anwendung geleitet, die inzwischen erfolgreich mit dem SQL Server und Visual FoxPro läuft.

Ungeachtet der Tatsache, daß ich diesen Artikel als Fallstudie bezeichne, möchte ich hier nicht näher auf das Projekt eingehen. Die Lösungsansätze und Strategien, die ich in diesem Artikel beschreibe, stellen lediglich einen möglichen Weg der Problemlösung dar. Ich erhebe nicht den Anspruch, daß es sich um die einzige Möglichkeit einer Cross-Plattform-Entwicklung mit Visual FoxPro handelt. Die beschriebenen Strategien haben unsere besondere Problemstellung gelöst, es können aber auch bessere Wege existieren. Andere Projekte können andersgeartete Erfordernisse aufweisen, auf die man dann anders reagiert.

Als ich das Projekt verwirklichte, habe ich mit Visual FoxPro 5.0a und dem SQL Server 6.5 gearbeitet. Für das Verfassen dieses Artikels habe ich Visual FoxPro 6.0 und die Beta 3 des SQL Server 7.0 verwendet. Daher können sich einige Unterschiede ergeben. Das könnte aber auch eine Chance sein, die Vor- und Nachteile der Portierung auf diese späteren Versionen zu erkennen.

In diesem Artikel werde ich

- das grundsätzliche Vorgehen des Programmierteams bei der Entwicklung der Anwendung erläutern und
- dieses Vorgehen am Beispiel einer einfachen Anwendung nachvollziehen, die in der Lage ist, sowohl mit einer FoxPro-Datenbank als auch mit dem SQL Server zu arbeiten, ohne daß der Anwender den Unterschied bemerkt.

Dabei werde ich auch einige spezielle Probleme aufzeigen, die auftraten, als wie die Anwendung entwickelten.

Das Konzept

Ich weiß nicht, welche Vorkenntnisse Sie haben. Sind Sie mit den Ansichten von Visual FoxPro nicht vertraut, oder haben Sie noch nicht mit dem SQL Server gearbeitet, lesen Sie bitte erst im Anhang A nach. Ich habe dort einen Schnelldurchlauf durch die Grundfunktionen erstellt. In diesem Text gehe ich davon aus, daß Sie

- einige Erfahrungen mit dem Datenbankcontainer von Visual FoxPro (DBC) und
- mit der Verwendung des SQL SELECT haben, sowie
- die Datenpufferung von VFP verstehen.

Entscheidung für eine Strategie

Nachdem wir die Möglichkeiten, mit Ansichten auf Remote Daten zuzugreifen, des Zugriffs auf Daten des SQL Server aus VFP heraus und die Fähigkeiten des Upsizing-Assistenten überprüft hatten, setzte sich unser Entwicklungsteam zusammen und stellte eine Reihe von Regeln und Richtlinien auf, an die wir uns während der Entwicklung unserer Anwendung hielten:

- Für die Änderung von Daten werden Ansichten, für die Änderung von Abfragen wird SQL Passthrough verwendet.
- Für lokale und Remote-Operationen werden unterschiedliche DBCs verwendet
- Der DBC wird als Container für plattformspezifische Ressourcen verwendet
- Für die Anmeldung beim SQL Server wird ein anwendungsspezifisches Login verwendet
- Ansichten werden programmatisch im Load() geöffnet, nicht in der Datenumgebung.

Ansichten für Änderungen der Daten, SPT für Abfragen

Wir entschieden uns, die Anwendung nur über Ansichten auf die Daten zugreifen zu lassen. Dabei überlegten wir uns, daß wir zwei Ansichten für jede Tabelle benötigten: Eine vertikale Ansicht für Auswahllisten und eine horizontale Ansicht mit nur einem Datensatz für die Änderung der Daten (Vgl. Abbildung 1).

Als wir dann die Anwendung entwickelten, stellten wir fest, daß wir in den meisten Fällen mehr als zwei Ansichten pro Tabelle benötigten. Daher beschäftigten wir uns mit SQL Passthrough (SPT) und kamen zu dem Ergebnis, daß SPT bei vielen Abfragen, bei denen Daten nicht geändert werden sollten, erheblich schneller arbeitete als die entsprechenden Remote-Ansichten. Wir erreichten die beste Performance durch den Einsatz von SPT für die verschiedenen Abfragen und Remote-Ansichten für die Änderung der Daten. Ansichten haben gegenüber SPT viele Vorteile, die ich später noch behandle.

Namenskonventionen

Wir haben die Ansichten nach den Tabellen benannt, die sie in erster Linie abfragen und ihnen ein Prefix vorangestellt, das deren Verwendung anzeigt: _rec für horizontale Ansichten und _lst für vertikale Ansichten (vgl. Abb. 1).

Account_lst_by_name ist ein typisches Beispiel.

Separate DBCs für lokale und Remote-Aufgaben

Wir hatten vor, in einem „lokalen“ DBC lokale Ansichten und in einem „remote“ DBC Remote-Ansichten mit identischen Namen zu definieren. Wir wollten dadurch der Anwendung die Möglichkeit geben, nur durch das Öffnen eines anderen DBC zwischen den Tabellen von FoxPro und denen des SQL Server umzuschalten.

account_lst

Account

Ac_id	Ac_account	Ac_name	Ac_balance	Ac_dob	Ac_inactiv	Ac_notes
1	012-040499	Jlioko, Y N	700.0000	10/23/79	F	Memo
2	012-040498	Avekkl, T A	9092.4265	07/07/78	F	Memo
3	012-040497	Vsiwai, I I	9163.4919	05/29/52	F	memo
4	012-040496	Oeeaod, I O	4449.7031	08/31/58	F	memo
5	012-040495	Wipyie, C O	7912.2299	02/17/88	F	memo
6	012-040494	Ciruoe, A H	1586.2554	09/30/56	F	memo
7	012-040493	Uoounu, I O	2285.7219	11/01/66	F	memo
8	012-040492	Dsvmii, F S	6802.2490	05/11/60	F	memo
9	012-040491	Tmceje, I I	9416.8248	02/09/68	F	memo
10	012-040490	Uhiaoc, E U	5138.5799	01/22/76	F	memo
11	012-040489	Eleypo, G E	6821.2764	07/06/80	F	memo
12	012-040488	Ufvljf, D I	1310.1018	04/13/53	F	memo
13	012-040487	Oiuoec, I N	2163.9315	03/08/87	T	memo
14	012-040486	Suinak, G E	2406.4320	12/16/84	F	memo
15	012-040485	Eevkva, H U	3308.1994	03/18/71	F	memo
16	012-040484	Rniofr, U U	2516.5850	07/14/53	F	memo
17	012-040483	Oibvio, A A	54.9363	11/05/53	F	memo
18	012-040482	Ieeioe, I E	1933.6846	08/23/59	F	memo
19	012-040481	Uoiuo, Y S	1477.0397	09/20/88	F	memo

Abbildung 1. Die Namenskonventionen

Verwenden des DBC als Container für plattformspezifische Ressourcen

Der plattformspezifische Code sollte in Form von gespeicherten Prozeduren im jeweiligen DBC gespeichert werden. Durch dieses Vorgehen verfügen wir über eine einheitliche Codebasis, mit der wir transparent auf die unterschiedlichen Datenbanken zugreifen können, ohne viele CASE...-Anweisungen rund um den plattformspezifischen Code schreiben zu müssen.

Ich habe eine kurze Dokumentation der wichtigsten 13 Funktionen im Anhang B zusammengefaßt. Wir werden später einige davon nutzen.

Ein anwendungsspezifisches Login für den SQL Server

Um die Installation und Administration der Anwendung auf unterschiedlichen Netzwerken zu vereinfachen, wollten wir versuchen, ein einheitliches Login für den SQL Server für alle Instanzen der Anwendung zu nutzen. Bis heute wurden uns keine Probleme mit dieser Strategie gemeldet.

Die Performance des Programms ist gut und es ist erheblich einfacher, die installierten Systeme zu administrieren (Die Sicherheitsfunktionen befinden sich in der mittleren Schicht der VFP-Anwendung).

Öffnen der Tabellen im Load(), nicht in der Datenumgebung

Unser Entwicklungsteam bestand sowohl aus erfahrenen VFP-Entwicklern als auch aus Entwicklern, die ihre Erfahrungen weder mit VFP noch mit objektorientierter Programmierung gesammelt hatten. Ich erläuterte den anderen Entwicklern die Datenumgebung. Außerdem behandelte ich die Probleme, die wir in der Vergangenheit hatten, unsere Formulare in die Lage zu versetzen, mit unterschiedlichen Daten zu arbeiten und die Probleme mit hart kodierten Pfaden zu den Tabellen. Nach einigen Diskussionen entschieden wir uns, die erforderlichen Ansichten im Ereignis Load() zu öffnen.

account_rec

Ich war zu dieser Zeit der Meinung, daß VFP 5.0 und 6.0 gegenüber der Version 3.0 erheblich verbessert wurden und hatte gute Erfahrungen mit dem Öffnen von Tabellen in der Datenumgebung. Würde ich heute wieder vor einem vergleichbaren Problem stehen, würde ich empfehlen, die Datenumgebung zu nutzen.

Daher werde ich in der Beispielsanwendung die Ansichten auch mit der Datenumgebung und nicht programmatisch öffnen. Dabei werde ich aufzeigen, wie die Objekte der Datenumgebung zur Laufzeit konfiguriert werden müssen, um auf den richtigen DBC zu verweisen.

Umsetzen der Strategie

Lassen Sie uns ein Beispiel entwickeln. Wir werden dabei

- einige VFP-Beispieltabellen erstellen
- einen DBC erstellen, der die lokalen Ansichten für diese Tabellen enthält
- den Prozeß des Upsizing für den SQL Server durchspielen
- einen DBC erstellen, der die Remote-Ansichten auf die Daten enthält
- ein einfaches Formular erstellen, das transparent mit den lokalen und entfernten Daten arbeiten kann.

Erstellen einiger VFP-Beispieldaten

Lassen Sie uns mit einer VFP-Datenbank beginnen, die Beispieldaten enthält.

Im Verzeichnis sample\data finden Sie das Programm CreateSampleData.prg. Sie können damit die Beispieldaten generieren, die wir in diesem Artikel nutzen. Ich habe dieses Programm geschrieben, weil ich eine angemessene Anzahl Beispieldaten benötigte, damit dieses Beispiel nicht zu einfach wird. Nebenbei macht es Spaß, sensible, aber sinnlose Daten zu generieren.

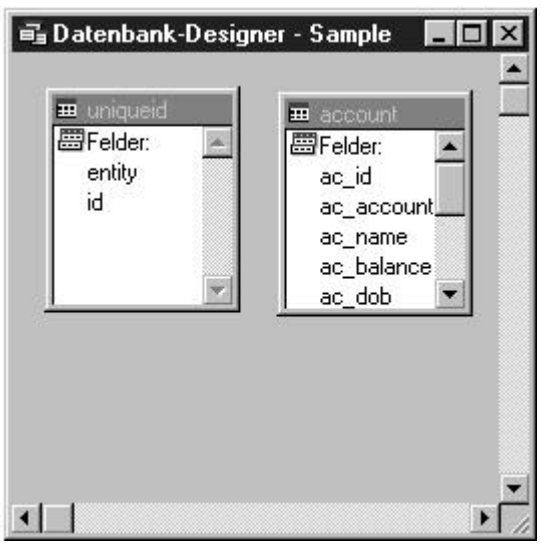


Abbildung 2. Die Beispieldaten

Die in unserer Beispielsanwendung benutzten Daten werden in Abbildung 2 dargestellt. Unsere Beispielsdatenbank – passenderweise Sample genannt – enthält eine Tabelle account, die Konteninformationen enthält. Unser Primärschlüssel ist ein numerisches Feld. Außerdem verfügen wir über eine Tabelle uniqueid, die nur einen Datensatz enthält, und die dazu dient, einen eindeutigen Wert für neue Datensätze zu erhalten. Viel einfacher geht es nicht!

L_SAMPLE, der DBC mit den lokalen Ansichten

Lassen Sie uns unseren DBC mit den lokalen Ansichten L_SAMPLE nennen (später werden wir den DBC mit den Remote-Ansichten R_SAMPLE nennen). Aufgrund unserer Strategie wissen wir, daß wir viele Ansichten benötigen.

Account_rec

Wir definieren uns die Ansicht account_rec, die wir einsetzen, um die Inhalte der Tabelle account anzuzeigen und uns von Datensatz zu Datensatz zu bewegen (vgl. Abb. 3). Wir parametrisieren die Ansicht durch die Angabe einer Filterklausel: WHERE Account.AC_ID = ?ipvAC_ID.

Um sicherzustellen, daß die verschiedenen Programmierer erkennen, welche Variablen Parameter einer Ansicht sind, haben wir folgende Namenskonvention entwickelt: Wir haben dem Feldnamen das Präfix xpv vorangestellt, wobei x den Datentyp anzeigt und pv für „parameterised view“ steht.

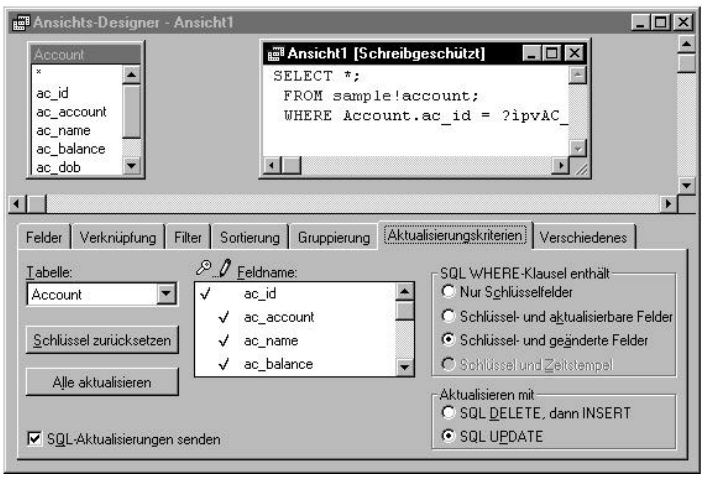


Abbildung 3. Erstellen der Ansicht Account_Rec

Account_lst

Lassen Sie uns eine andere Ansicht erstellen, die wir für eine Auswahlliste nutzen können. Dafür benötigen wir zwar alle Datensätze, aber nur einige Felder: den Primärschlüssel und ein beschreibendes Feld. Erstellen wir die Ansicht programmatisch:

```
open database l_sample
create sql view account_lst as ;
   select ac_id, ac_account from account order by ac_account
```

Die lokalen gespeicherten Prozeduren

An diesem Punkt müssen wir, außer der Überprüfungen auf der Ebene des DBC und ähnlicher Routinen, dem DBC eine neue gespeicherte Prozedur hinzufügen: sp_GetNewID().

Diese Prozedur gibt den nächsten verfügbaren Primärschlüssel zurück, der in einen neuen Datensatz eingefügt wird. Im lokalen DBC öffnen wir die Tabelle exklusiv, um die Sicherheit bei Mehrbenutzerbetrieb zu gewährleisten. In sample\sp\local.prg finden Sie die Implementierung, die wir in unserem Projekt einsetzten. Anschließend können wir beispielsweise folgenden Code benutzen, um neue Datensätze anzulegen:

```
open database l_sample
use account_rec nodata
insert into account_rec fields ;
   (ac_id, ac_account, ac_name, ac_balance, ;
    ac_dob, ac_inactiv, ac_notes ) ;
   values ;
   ( sp_GetNewId('account'), '', '', 0.00, {},.F.,'')
=tableupdate(TABLEUPDATE_CURRENTROW)
```

Daten auf den SQL Server portieren

Wenn Sie noch nicht mit dem SQL Server gearbeitet haben, empfehle ich Ihnen, zunächst den Abschnitt „Einführung in die SQL Server-Daten“ in Anhang A zu lesen. Es ist kein Ersatz für ein gutes technisches Buch, kann Ihnen aber helfen, den nächsten Schritten dieses Artikels zu folgen.

Wie weit kann uns der Upsizing-Assistent helfen?

Ich war wirklich überrascht über die Möglichkeiten des Upsizing-Assistenten von VFP 5.0. Natürlich hat auch er seine Haken und Ösen. Wir haben ihn aber immer als Basis genommen, Daten auf den SQL Server zu verschieben, wenn wir unsere Konvertierungsroutinen getestet haben.

Wir müssen einige Schritte durchlaufen:

- Erstellen und Vorbereiten einer leeren Datenbank im SQL Server für die Daten unserer Anwendung,
- Erstellen eines Benutzernamens, den unsere VFP-Anwendung nutzt, um auf die Datenbank zuzugreifen,
- Einrichten des ODBC-Kanals, über den die VFP-Anwendung auf die SQL Server-Datenbank zugreift,
- Ausführen des Upsizing-Assistenten, um die Tabellen auf dem SQL Server zu erstellen und mit unseren Daten zu füllen.

Ich weiß, daß der Upsizing-Assistent jeden dieser vier Schritte für Sie erledigen kann, aber meine erste Regel lautet: „Lasse niemals einen Assistenten etwas für Dich tun, wenn Du beim Selbermachen noch etwas lernen kannst.“ Der Upsizing-Assistent ist prima für wiederkehrende Teile der Aufgabe, aber in dieser Situation war ich froh, beim eigenhändigen Erledigen der Aufgaben im SQL Server noch etwas Neues zu entdecken.

Einführung in den SQL Server

Ich habe auf der Maschine, auf der ich diesen Artikel schreibe, die Desktop-Version des SQL Server 7.0 installiert. Dieser Prozeß ging vollkommen glatt ab. Daher will ich hier nicht weiter darauf eingehen. Nur ein Punkt: Der SQL Server fragt während der Installation nach der Sortierreihenfolge. Ich habe den Vorgabewert gewählt.

Nachdem Sie den SQL Server installiert haben und der SQL Server-Service läuft (vg. Abb.4), sollten Sie den SQL Server Enterprise Manager ausführen können. In der Version 7.0 ist er in die neue Microsoft Management Konsole eingefügt (vgl. Abb. 5).

Abbildung 4. Der SQL Server läuft als Service unter Windows NT 4.0

Vorbereiten einer leeren SQL Server-Datenbank

Im SQL Server 6.5 mußten wir mit Dingen, „Devices“ genannt, umgehen. Dabei handelte es sich um Einheiten, in die der SQL Server Datenbanken und andere Dateien speicherte. Dabei benutzte er sein eigenes Dateisystem. In der Version 7.0 gibt es diese Devices nicht mehr, so daß ich darauf nicht näher eingehe.

Führen Sie nun den Enterprise Manager des SQL Server aus, klappen den Punkt „Datenbanken“ auf und wählen nach einem Klick mit der rechten Maustaste „Neue Datenbank“ aus dem Kontextmenü (vgl. Abb. 5).

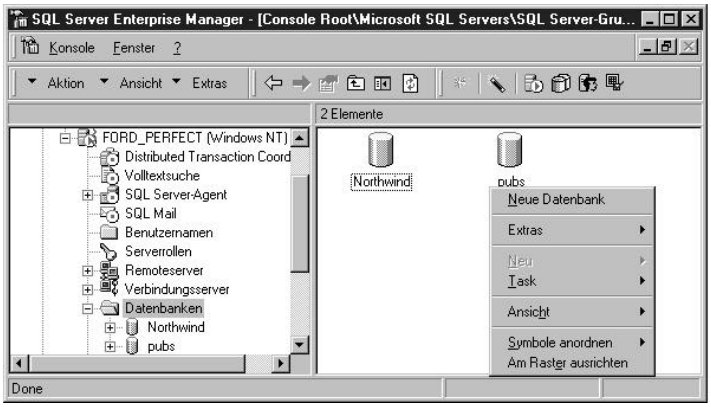


Abbildung 5. Erstellen einer neuen Datenbank im Enterprise Manager

Daraufhin erscheint der Dialog „Datenbankeigenschaften“ für eine neue Datenbank. Wir sollten sie „Sample“ nennen, genau wie unsere VFP-Datenbank heißt. Ansonsten belassen wir alle Werte zunächst auf ihren Vorgabewerten.

Nach der Betätigung der Schaltfläche „OK“ benötigt der SQL Server einige Sekunden, um die Datenbankdateien auf der Festplatte zu erstellen. Nach dieser Operation sollten wir das Symbol für die neue Datenbank im rechten Fenster des Enterprise Managers sehen.

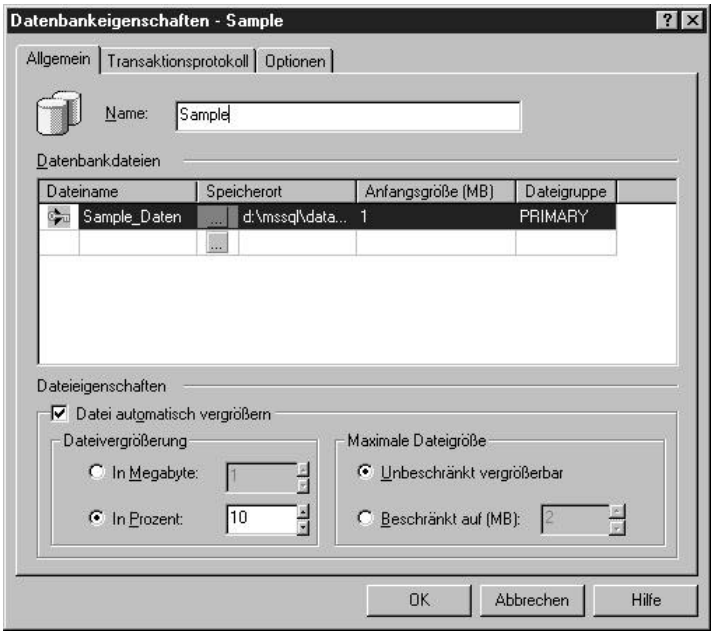


Abbildung 5. Der Dialog „Datenbankeigenschaften“

Erstellen eines Benutzernamens für den Server

Eine Bemerkung über die Sicherheitsmodelle des SQL Server

Auf einem NT-Server sollten wir die „integrierte Sicherheit“ benutzen, bei der der SQL Server die Rechte der Anwender in NT prüft und danach entscheidet, wie den Anwendern der Zugriff auf die SQL-Datenbanken ermöglicht werden soll. In diesem Artikel benutze ich das Standard-Sicherheitsmodell des SQL Server. Das hat verschiedene Gründe:

- Auf einer Standalone-Maschine unter Windows 95/98 haben wir keine andere Wahl
- Wichtiger: Wir arbeiten an unserer Anwendung und wir sollten jetzt keine Vermutungen anstellen, wie der SQL Server später mit einer NT-Domäne verbunden wird.
- Wir gehen jetzt mal davon aus, daß die integrierte Sicherheit einfacher zu verwalten ist. Daher benutzen wir jetzt die Standard-Sicherheit, um uns das Leben interessanter zu machen.

Logins oder Anwender?

Der SQL Server unterscheidet zwei Arten von Benutzerrechten. Da ist einmal der Benutzername, mit dem Sie sich beim SQL Server anmelden und der datenbankspezifische Benutzer. Die Rechte des Benutzers an der Datenbank bestimmen Ihre Rechte innerhalb dieser Datenbank unabhängig von Ihren Rechten an anderen Datenbanken. Dadurch haben Anwender die Möglichkeit, innerhalb unterschiedlicher Datenbanken unterschiedliche Rollen zu spielen. Im Falle der integrierten NT-Sicherheit ist der Benutzername im SQL Server derjenige, mit dem Sie sich bei NT angemeldet haben. Sie benötigen aber zusätzlich eine Berechtigung für jede Datenbank, auf die Sie innerhalb des SQL Servers zugreifen.

Als Vorgabe hat der Benutzername des Administrators ad kein Kennwort. Wenn Sie nicht gerade in einer Situation sind, in der die Sicherheit keine Rolle spielt, empfehlen alle Experten, möglichst schnell ein Paßwort einzurichten. Da mein Computer Barad-dur heißt, war es logisch, das Paßwort für sa auf „sauron“ zu ändern.

Unter dem Standard-Sicherheitsmodell ist es vielleicht nicht sinnvoll, sich die Anwendung als sa anmelden zu lassen. Daher werden wir für unsere Beispielsanwendung eine spezielle LoginID erstellen. Dafür öffnen wir den Punkt „DB-Benutzernamen“ und klicken auf „Neuer Benutzername“ in der Toolbar des Enterprise Managers (vgl. Abb. 7).

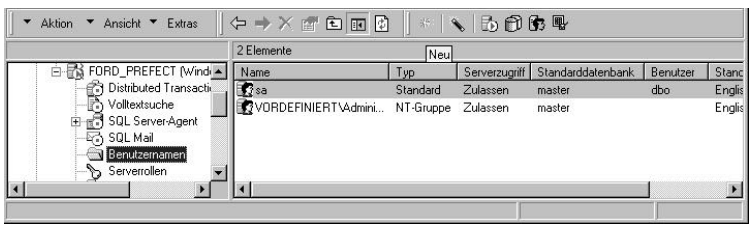


Abbildung 5. Erstellen eines neuen Benutzernamens

Dadurch öffnen wir den Dialog „Neuer Benutzername“ (vgl. Abb. 8)

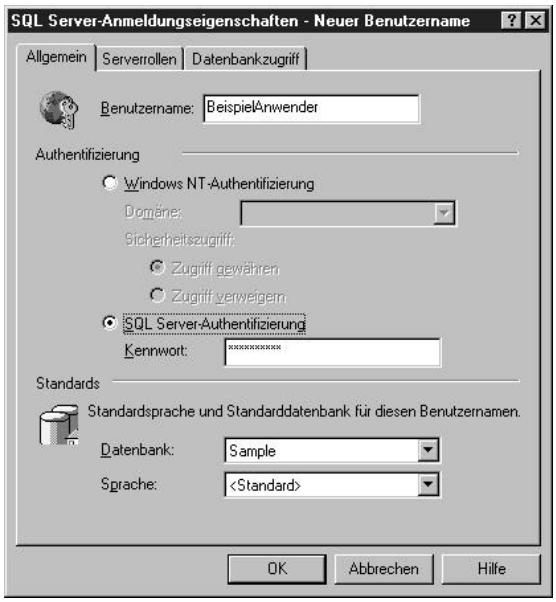


Abbildung 7. Der Dialog „Neuer Benutzername“

Hier geben wir den Benutzernamen – „SampleClient“ – und das Kennwort ein. Um bei Tolkin zu bleiben, benutze ich das Kennwort „frodo“. Wir können auch angeben, daß dieser Benutzername auch der Datenbank Sample zugeordnet werden soll.

Bevor wir diesen Dialog schließen, müssen wir noch einige Einstellungen auf der Seite „Datenbankzugriff“ ändern.

Durch Markieren der Datenbank Sample können wir automatisch den Benutzernamen „SampleClient“ für diese Datenbank anlegen. Lassen wir die anderen Einstellungen wie sie sind und klicken auf OK, damit der Benutzer erstellt wird. Sie erhalten noch ein Dialogfeld, in dem Sie das Kennwort bestätigen müssen.

Sie werden bemerken, daß der neue Benutzername im Fenster des Enterprise Managers erscheint.

Abbildung 8. Die Seite „Datenbankzugriff“

Erstellen einer ODBC-Datenquelle

Lassen Sie uns jetzt eine ODBC-Datenquelle für unsere neue SQL Server-Datenbank organisieren. Dieser Schritt muß später auf jeder Maschine, auf der unsere Anwendung läuft, ausgeführt werden. Wir gehen jetzt diesen Prozeß manuell durch, obwohl sich auch dieser Schritt automatisieren läßt. Vergleichen Sie dazu die Dokumentation der Win32 API, besonders die Funktion SQLConfigDataSource in der ODBCCP32.DLL.

Ausführen des ODBC-Datenquellen-Administrators

Sie finden dieses Werkzeug als Icon auf dem Desktop von Windows. Haben Sie das Visual Studio 6.0 installiert, finden Sie es auch im Startmenü in der Progammguppe „Datenzugriffskomponenten“.

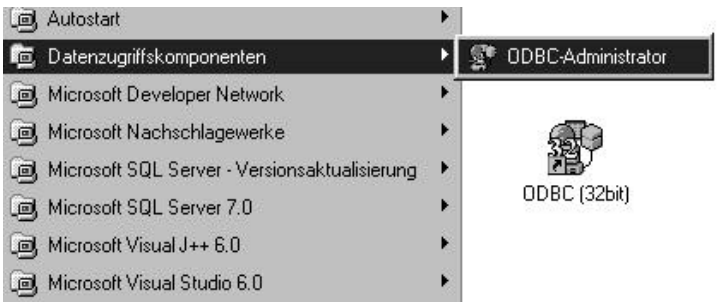


Abbildung 10. Der ODBC-Administrator

Sie führen den ODBC-Datenquellen-Administrator aus, wechseln auf die Seite „System-DSN“ und klicken auf „Hinzufügen...“. Aus irgendeinem Grund besitzt der ODBC-Treiber des SQL Server als einziger nicht das Präfix „Microsoft“, so daß er sich am Ende der Liste befindet. Markieren Sie ihn und klicken auf „Fertig stellen“. Nun wird der Assistent „Neue Datenquelle für SQL Server erstellen“ aufgerufen.

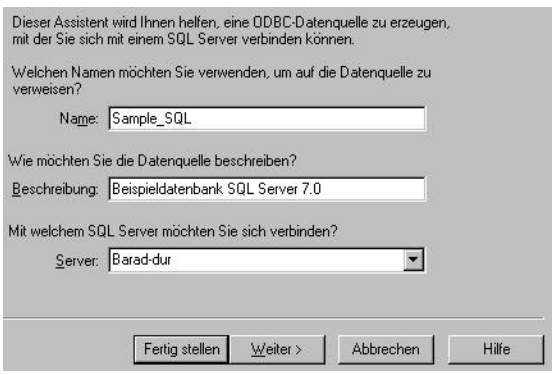


Abbildung 11. Der Assistent „Neue Datenquelle für SQL Server erstellen (1)

Ich glaube nicht, daß die Beschreibung, die wir hier eingeben können, irgendeine tiefere Bedeutung hat. Wichtiger ist der Eintrag im Feld „Name“, den wir

benutzen, wenn wir den DSN (Data Source Name) referenzieren. Ich benutze hier den Namen Sample_SQL. Wenn Sie den SQL Server auf Ihrer lokalen Workstation ausführen, enthält die Combobox „Server“ den Eintrag „(local“). Ich bin mir nicht sicher, ob es einen Unterschied macht, aber ich bevorzuge den Namen des Rechners, auf dem der Server läuft, in meinem Fall Barad-dur. Jetzt können wir auf die Schaltfläche „Weiter“ klicken.

Abbildung 12. Der Assistent „Neue Datenquelle für SQL Server erstellen (2)

Jetzt muß der Assistent den SQL Server aufrufen, um den System-DSN zu konfigurieren. Dafür ist ein Benutzername erforderlich. Daher teilen wir dem Assistenten mit, daß wir die „Standardsicherheit“ nutzen, und daß der Benutzername „sa“ und das Kennwort ... ist. Diese Anmeldeinformation wird nicht mit dem DSN gespeichert; sie wird lediglich durch den Assistenten für die Konfiguration des DSN genutzt. Nach einem Klick auf die Schaltfläche „Weiter“ kommen wir zu der Seite, die in Abb. 13 gezeigt wird.

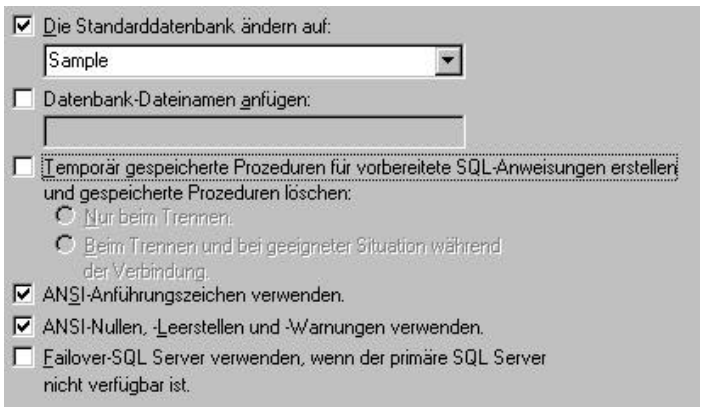


Abbildung 13. Der Assistent „Neue Datenquelle für SQL Server erstellen (3)

Hier können wir die Standarddatenbank ändern. Das ist zwar nicht unbedingt notwendig, aber wir wollen die Verbindung nutzen, um auf unsere Beispieldatenbank zuzugreifen. Daher ist es hier sinnvoll. Im Moment können wir die anderen Einstellungen auf dieser Seite auf ihren Vorgabewerten belassen.

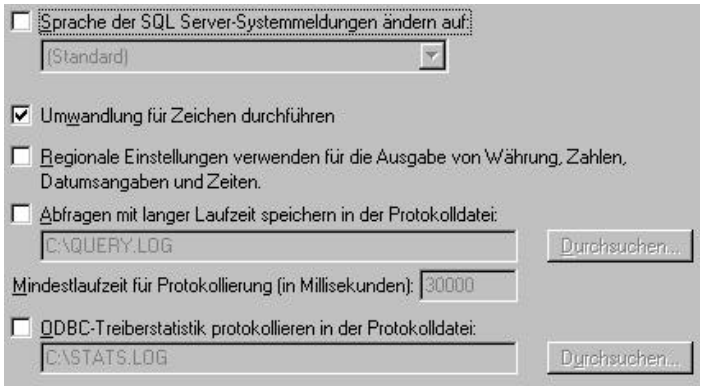


Abbildung 14. Der Assistent „Neue Datenquelle für SQL Server erstellen (4)

Abbildung 14 zeigt die vierte Seite des Assistenten. Wir lassen diese Einstellungen auf ihren Vorgabewerten. Einige der Einstellungen sehen zwar interessant aus, sind aber jetzt nicht wichtig. Nun können wir die Schaltfläche „Fertigstellen“ betätigen und den neuen DSN im Hauptfenster des ODBC-Administrators überprüfen (vgl. Abb. 15).

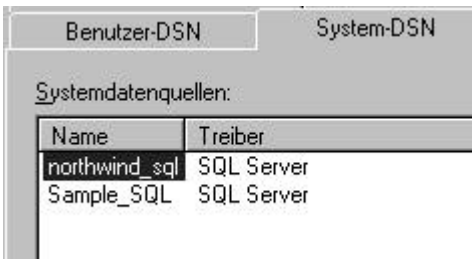


Abbildung 15. Der neu erstellte DSN Sample_SQL

Das ist schon alles, was wir tun müssen, bevor VFP auf die Datenbank des SQL Server zugreifen kann. Wir können den ODBC-Administrator schließen und VFP ausführen.

Ausführen des Upsizing-Assistenten

Als nächsten Schritt rufen wir den Upsizing-Assistenten von Visual FoxPro auf, um unsere VFP-Daten auf den SQL Server zu portieren. Wählen Sie dafür den Punkt „Upsizing“ aus dem Menü Extras – Assistenten. Es erscheint der in Abbildung 16 gezeigte Dialog.

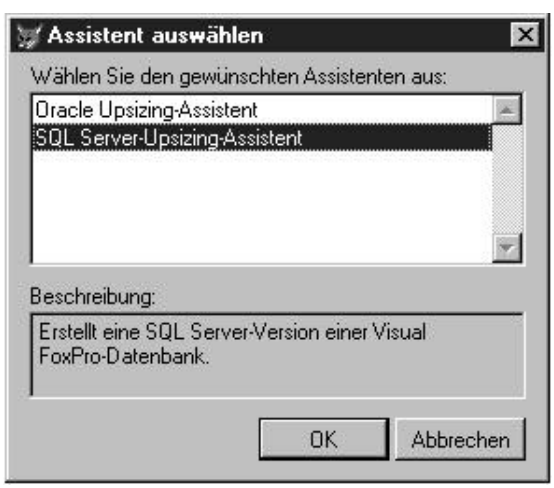


Abbildung 16. Auswählen eines Upsizing-Assistenten

Wir wählen natürlich den Upsizing-Assistenten für den SQL Server.

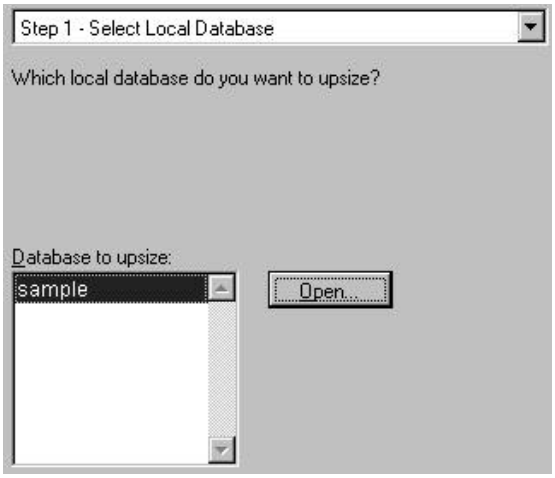


Abbildung 17. Öffnen der zu portierenden Datenbank

Unsere Beispieldaten sind in SAMPLE.DBC gespeichert. Nachdem wir sichergestellt haben, daß die Datenbank geöffnet ist, können wir zum nächsten Schritt kommen.

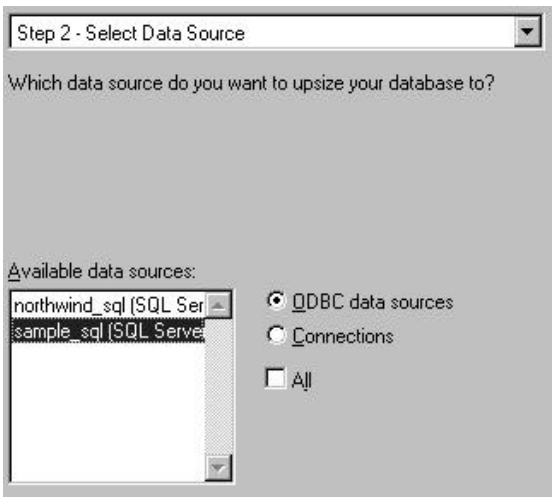


Abbildung 18. Auswahl eines DSN

Hier zeigt uns der Upsizing-Assistent eine Liste der Namen der vorher definierten Datenquellen, einschließlich sample_sql, den wir gerade erstellt haben. Da der Assistent den DSN benutzt, um sich mit dem SQL Server zu verständigen, werden wir nach einem Benutzernamen und dem Kennwort gefragt. Wir wollen Tabellen erstellen und sollten daher einen Namen wählen, der uns die notwendigen Rechte verleiht. In diesem Fall ist das „sa“.

Abbildung 19. Der Assistent muß sich mit dem SQL Server verbinden

Nach der Anmeldung beim SQL Server fragt uns der Assistent, welche Tabellen der VFP-Datenbank portiert werden sollen (vgl. Abb. 20). Die richtige Antwort ist „alle“.

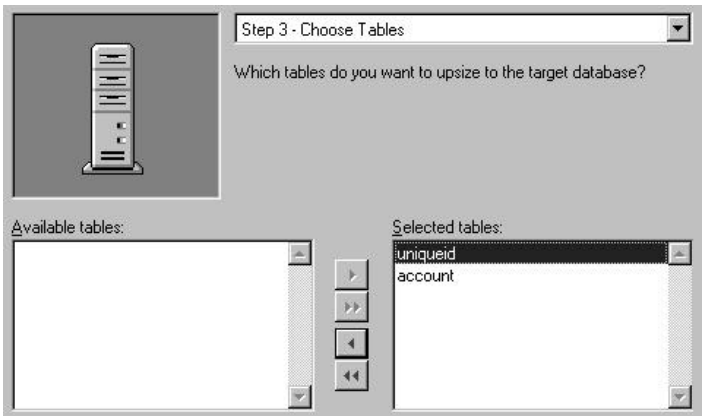


Abbildung 20. Auswahl der benötigten Tabellen

Der Assistent analysiert die Tabellen, was einen Augenblick dauern kann, und kommt dann zum nächsten Schritt, in dem die Datentypen der Felder in der Datenbank des SQL Server festgelegt werden. Er ist intelligent genug, uns Vorgabewerte anzubieten, die meist auch richtig sind.

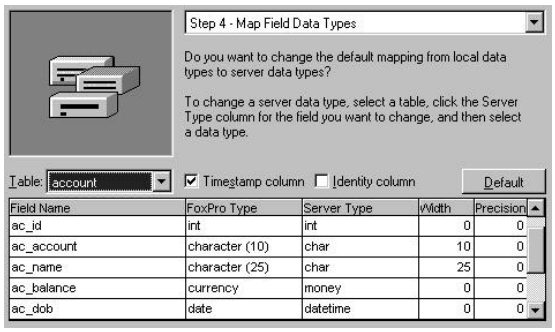


Abbildung 21. Anpassen der Datentypen

Hier ist es wichtig zu verstehen, daß es einige Unterschiede in den verfügbaren Datentypen in VFP und dem SQL Server gibt. So unterstützt der SQL Server beispielsweise nicht den Datentyp DATE. Daher werden Datumsfelder in den Datentyp DATETIME konvertiert. Das kann zu einigen Problemen führen, die ich aber später behandele. VFPs Datentyp CURRENCY entspricht im Wesentlichen dem Datentyp MONEY des SQL Server. Logische Felder werden in BIT-Felder des SQL Server umgewandelt; MEMO-Felder in TEXT-Felder.

Im SQL Server 6.5 war es nicht einfach, die Struktur einer Tabelle nach ihrer Erstellung zu ändern. Das hat sich in der Version 7.0 geändert. Trotzdem ist es natürlich besser, den Feldern bereits vor der Portierung die richtigen Datentypen zuzuordnen.

Die Zeitstempel-Felder des SQL Server

Sie haben wahrscheinlich die beiden Kontrollkästchen „Timestamp column“ und „Identity column“ in Abbildung 21 bemerkt. Auf die Identity columns gehe ich weiter hinten noch ein, aber lassen Sie uns bereits jetzt die Zeitstempel-Felder untersuchen.

Der SQL Server bietet die Möglichkeit, den Tabellen ein Zeitstempel-Feld hinzuzufügen. Der Zeitstempel ist ein eindeutiger Wert, der vom SQL Server während einer Update-Operation generiert wird.

Dadurch ist es dem SQL Server möglich, Update-Konflikte schneller zu behandeln, indem er den Zeitstempel des neuen Datensatzes mit dem des bereits existierenden vergleicht. Sind die Zeitstempel identisch, weiß der SQL Server, daß er den Datensatz ändern kann. Sind die Zeitstempel nicht identisch, wurde der Datensatz eventuell seit dem letzten Lesen für die Änderung geändert.

Der Upsizing-Assistent empfiehlt, ein Zeitstempel-Feld in allen Tabellen zu nutzen, die ein Memofeld enthalten. Die Vergleiche der Inhalte von Text- oder Image-Feldern benötigen einige Zeit, so daß der Vergleich der Zeitstempel erheblich schneller ist. Für Tabellen ohne Memofelder kann ein Zeitstempel-Feld eventuell unnötig sein.

Jetzt lassen wir die Empfehlungen des Upsizing-Assistenten so stehen wie sie sind und gehen zum nächsten Schritt. Unsere Anwendung soll einfach bleiben – deshalb berücksichtigen wir dieses Feld lediglich in der Ansichten-Definition, wo die Tabellen, die ein Zeitstempel-Feld enthalten. Dort sollten aus Gründen der Geschwindigkeit die Zeitstempel in der WHERE-Klausel der SQL-Anweisung genutzt werden.

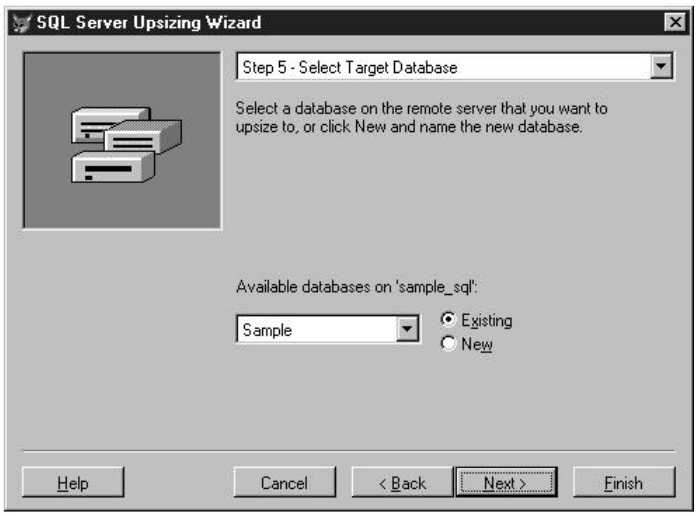


Abbildung 22. Auswahl der zu portierenden Datenbank

In Schritt 5 werden wir aufgefordert, eine SQL Server-Datenbank auszuwählen, die wir benutzen wollen. Wir wählen natürlich Sample. An diesem Punkt hätten wir auch angeben können, daß der Assistent die Datenbank für uns erstellen soll. Ich bevorzuge es jedoch, mich mit dem Enterprise Manager vertraut zu machen und hoffe dabei, durch die manuelle Erstellung der Datenbank mehr zu lernen.

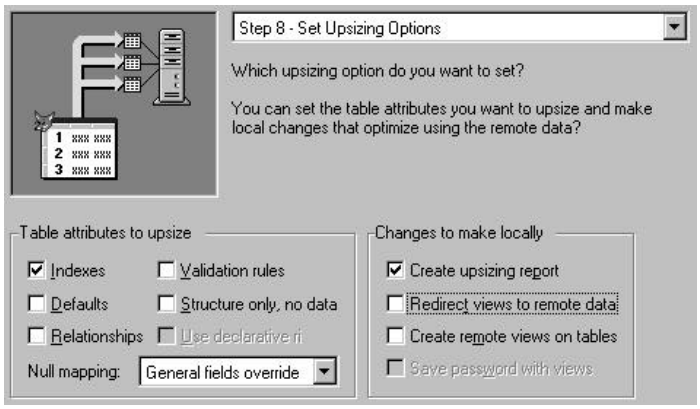


Abbildung 23. Optionen für das Upsizing festlegen

Da wir eine bereits erstellte Datenbank ausgewählt haben, springt der Assistent direkt zu Schritt 8, wo wir die Optionen für die Portierung festlegen. In unserem einfachen Beispiel (und auch während der Entwicklung der Anwendung) haben wir keine Vorgabewerte, Relationen oder Prüfungen auf Feld- oder Satzgültigkeit. Daher ändern wir die vorgegebenen Werte wie in Abbildung 23 dargestellt ab.

Auf den ersten Blick sieht es so aus, als wäre „Redirect views to remote data“ genau das, was wir benötigen. Aber Vorsicht: SAMPLE.DBC enthält keine Ansichten. Die lokalen Ansichten befinden sich in L_SAMPLE.DBC.

Wir lassen den Upsizing-Assistenten auch keine Remote Ansichten in SAMPLE.DBC erstellen, da er unparametrisierte Ansichten mit den allen Datensätzen erstellen würde. Diese Ansichten wären nutzlos und langsam.

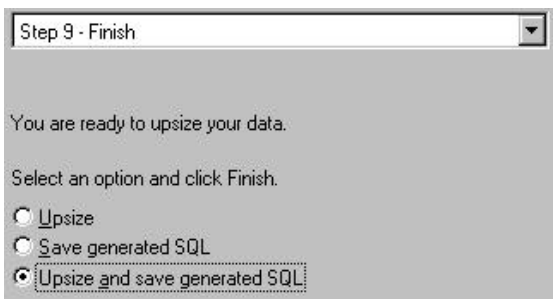


Abbildung 24. So, das war's

Normalerweise würden wir das Skript, das der Assistent erzeugt, nicht speichern, aber wenn Sie ein solches Skript noch nicht gesehen haben, ist es doch ganz interessant, einmal einen Blick darauf zu werfen. Jedenfalls sind wir jetzt

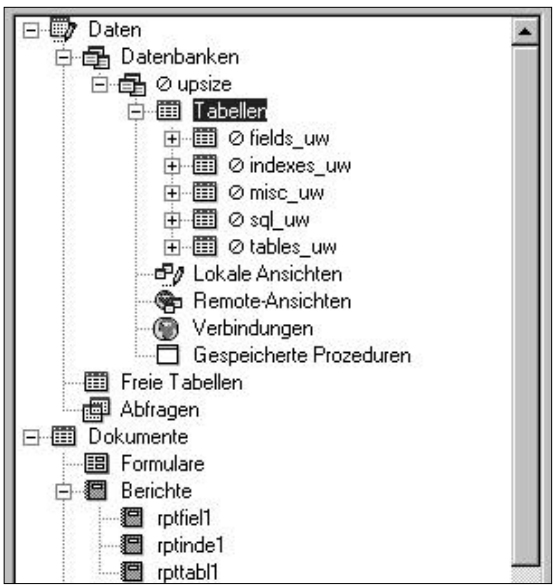
so weit, daß wir auf die Schaltfläche „Finish“ klicken können. Es erscheint eine Fortschrittsanzeige, auf der Sie die Fortschritte des Prozesses verfolgen können.

Anschließend erscheint ein Dialog „Upsizing complete“ und ein Projekt mit dem Namen REPORT.PJX wird geöffnet, das die Datenbank „UPSIZE“ und einige Berichte enthält.

Die Datenbank enthält verschiedene Tabellen, die Informationen enthalten, die während des Upsizing-Prozesses generiert wurden.

Nach meiner Erfahrung sind die Berichte recht nutzlos, es sei denn, während der Portierung wären Fehler aufgetreten, die Sie suchen müßten, um sie zu beseitigen.

Abbildung 25. REPORT.PJX



In unserem Fall haben wir die Portierungs-Funktionen, die häufig Fehler hervorrufen, nicht ausgeführt. Wären Fehler aufgetreten, würde die Datenbank noch die Tabelle errors_uw enthalten.

Die Skript-Tabelle SQL_UW

Es ist sinnvoll, einmal die Inhalte des Memofelds in der Tabelle sql_uw zu überprüfen. Das Feld enthält das SQL-Skript des Upsizing-Assistenten für die Portierung:

```
/* This zero default is bound to one or more fields. */
CREATE DEFAULT UW_ZeroDefault AS 0

/* Create table 'uniqueid' */
CREATE TABLE uniqueid (entity char(8) NOT NULL, id int NOT NULL)

/* Create table 'account' */
CREATE TABLE account (ac_id int NOT NULL, ac_account char(10) NOT
NULL, ac_name char(25) NOT NULL, ac_balance money NOT NULL,
ac_dob datetime NOT NULL, ac_inactiv bit NOT NULL, ac_notes text
NOT NULL, timestamp_column timestamp)

/* Index code */
CREATE  INDEX ac_id ON account (ac_id)
CREATE  INDEX ac_account ON account (ac_account)
CREATE  INDEX uac_name ON account (ac_name)

/* Default code */
sp_bindefault UW_ZeroDefault, 'account.ac_inactiv'
```

Dieses Skript kann nicht direkt im SQL Server ausgeführt werden. Es wird während des Upsizing-Prozesses erstellt, enthält aber nicht alle Schritte, die der Assistent währenddessen ausführt. Nachdem wir den Upsizing-Prozeß beendet haben, können wir die Werkzeuge benutzen, die mit dem SQL Server ausgeliefert werden, um ein Skript zu erstellen, mit dem wir die Datenbank erneut konstruieren können.

In dem Skript können Sie einen Unterschied zwischen dem SQL Server und FoxPro sehen: dem logischen Feld wird ein Vorgabewert von 0 zugeordnet.

Logische Werte im SQL Server

Logische Werte können im SQL Server keinen NULL-Wert enthalten, sondern nur 0 oder 1 (nicht „TRUE“ oder „FALSE“, sondern einen 0/1-Schalter). VFP dagegen ermöglicht den NULL-Wert in logischen Feldern. Um mit diesem Unterschied umzugehen, ordnet der Upsizing-Assistent jedem logischen Feld den Vorgabewert 0 zu. Anders gesagt: Wenn Sie dem logischen Feld nicht explizit einen Wert zugewiesen haben, wenn Sie das TABLEUPDATE durchführen, enthält es 0 statt NULL.

Vorgabewerte, NULL-Werte und der ODBC-Treiber des SQL Server

Vorgabewerte verhalten sich im SQL Server anders als in FoxPro. In FoxPro werden die Vorgabewerte zugeordnet, wenn Sie einen neuen Datensatz anlegen. Der SQL Server schreibt die Vorgabewerte, nachdem der Datensatz in die Tabelle eingefügt wurde. Zusammen mit einem Feature des ODBC-Treibers kann das weitreichende Auswirkungen auf Ihre Anwendung haben:

Ist im SQL Server ein Feld mit „NOT NULL“ markiert und Sie haben dem Feld in der INSERT-Anweisung nicht explizit einen Wert zugewiesen, versucht der SQL Server, dem Feld den Wert NULL zuzuweisen. Dadurch wird ein Fehler erzeugt, den Ihre Anwendung abfangen muß.

Als Regel können Sie entweder:

- *sicherstellen, daß alle Felder, die keinen Anfangswert aufweisen, den NULL-Wert aufnehmen können,*

oder

- *immer einen Anfangswert für alle Werte in Ihren INSERT-Anweisungen angeben.*

Indizes auf DELETED()

Wenn Sie einen Index auf DELETED() gelegt haben, wie es für FoxPro-Systeme empfohlen wird, erhalten Sie in der Tabelle error_uw des Upsizing-Assistenten dafür Fehlermeldungen, denn der SQL Server verfügt nicht über ein Äquivalent zu DELETED(). Wenn Sie dort einen Datensatz löschen, wird er physikalisch gelöscht und Sie können ihn nicht mit RECALL wiederbeleben. Daher gibt es beim SQL Server keinen Grund, einen Index auf DELETED() zu erzeugen, denn nicht vorhandene Datensätze braucht man auch bei der Abfrage nicht zu beachten.

Prüfen der neuen Tabellen im SQL Server

Wenn wir jetzt den Enterprise Manager ausführen, können wir uns die Tabellen unserer Datenbank anzeigen lassen:

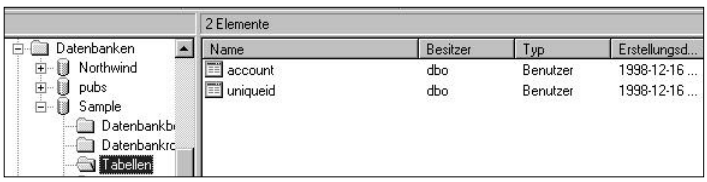


Abbildung 26. Die vom Upsizing-Assistenten neu erstellten Tabellen

Generierung eindeutiger Schlüssel im SQL Server

Sicher gibt es mehrere Möglichkeiten im SQL Server eindeutige Werte für Primärschlüssel zu generieren. Für den Anfang unterstützen die Tabellen des SQL Server „Identity Columns“. Dabei handelt es sich um Felder, die einen automatisch hochgezählten Integerwert, der generiert wird, wenn neue Datensätze angelegt werden. Wenn Ihre Datenbank mit numerischen Primärschlüsseln arbeitet, können Sie eventuell diese „Identity Columns“ einsetzen. Wenn Ihre Primärschlüssel noch anderen Erfordernissen genügen müssen, sind die „Identity Columns“ nicht die richtige Wahl, so daß Sie sich einen anderen Mechanismus einfallen lassen müssen.

Der Upsizing-Assistent unterstützt die „Identity Columns“ nicht – aber wenn Sie in das Kontrollkästchen Identity Ihrer portierten Tabellenstruktur klicken, wird ein weiteres Feld in Ihrer Tabelle erzeugt. Ich bin mir nicht sicher, wie sinnvoll das ist, solange dafür keine unterstützende Logik existiert. Auf jeden Fall kann bereits Ihre Originaldatenbank funktionierende Primär- und Fremdschlüssel haben.

In unserem einfachen Beispiel hat die Tabelle account einen numerischen Primärschlüssel, aber um die ganze Geschichte etwas interessanter zu gestalten, wollen wir jetzt keine „Identity Columns“ benutzen. Lassen Sie uns die vertraute Technik nutzen, die wir bereits in der Situation lokale Ansicht/VFP-Datenbank eingesetzt haben (Vergleichen Sie dazu den Abschnitt L_Sample.dbc weiter oben in diesem Artikel).

In unserem lokalen DBC mit den Ansichten haben wir die gespeicherte Prozedur sp_GetNewId() abgelegt, die auf die Tabelle uniqueid zugreift. Es liegt in der Natur der Sache, daß gespeicherte Prozeduren immer plattformspezifisch sind. Im Falle von sp_GetNewId() muß die Prozedur in zwei Teile getrennt werden: ein Teil wird auf dem SQL Server, der andere innerhalb der FoxPro-Anwendung ausgeführt.

Erstellen einer gespeicherten Prozedur im SQL Server

Wir klappen im Enterprise Manager Datenbanken\ Sample\ Gespeicherte Prozeduren auf, klicken mit der rechten Maustaste in das rechte Fenster und wählen im Kontextmenü „Neue gespeicherte Prozedur...“. Damit öffnen wir den Dialog „Neue gespeicherte Prozedur“ (Abb. 27).

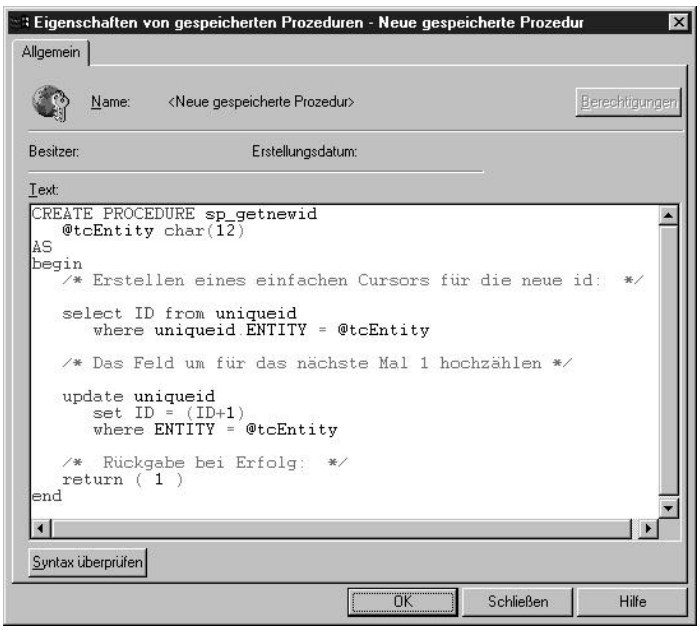


Abbildung 27. Der Dialog „Neue gespeicherte Prozedur“ des SQL Server

Jetzt können wir das SQL-Skript eingeben, um die gespeicherte Prozedur zu erstellen ...

```
CREATE PROCEDURE sp_getnewid
  @tcEntity char(12)
AS
begin
  /* Erstellen eines einfachen Cursors für die neue id: */
```

```
  select ID from uniqueid
    where uniqueid.ENTITY = @tcEntity
 /* Das Feld um für das nächste Mal 1 hochzählen */

  update uniqueid
    set ID = (ID+1)
    where ENTITY = @tcEntity

  /*  Rückgabe bei Erfolg:  */
  return ( 1 )
end
```

... und auf „OK“ klicken. Jetzt sollte die neue Prozedur im Enterprise Manager erscheinen (Abb. 28).

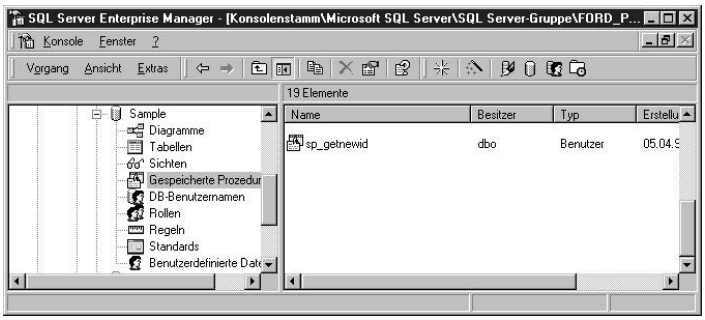


Abbildung 28. Die neue gespeicherte Prozedur sp_GetNewId()

R_SAMPLE, der DBC mit den remote Ansichten

Nachdem wir nun unsere Beispieldaten in der Datenbank des SQL Server untergebracht haben, können wir einen alternativen DBC (R_Sample.dbc) erstellen, der die Ansichten enthält, die unsere Anwendung nutzt. Die entfernten Ansichten erhalten exakt die gleichen Namen, die wir bereits in L_Sample.dbc verwendet haben. Auch die gespeicherten Prozeduren haben identische Namen.

Gehen wir zu VFP zurück und geben ein:

```
create database r_sample
```

Erstellen einer Verbindungsdefinition in der Datenbank

Um eine wiederholte Anmeldeaufforderung durch den ODBC-Treiber des SQL Server zu vermeiden, können wir diese Informationen in der VFP-Datenbank als Verbindung speichern. Anschließend können wir unsere remote Ansichten so erstellen, daß sie die Verbindungsdefinition referenzieren.

Wir haben ja bereits eine spezielle LoginID für unsere Anwendung erstellt – jetzt werden wir sie nutzen:

```
create connection con_sample_sql ;
datasource „Sample_SQL“ ;
userid „SampleClient“ ;
password „frodo“ ;
database „sample“
```

Beachten Sie, daß Sie bei dieser Anweisung auf Groß- und Kleinschreibung achten müssen. Ansonsten müssen Sie damit rechnen, daß die Verbindung auf eine falsche Datenquelle zugreift. Wenn Sie jetzt zum ersten Mal eine Definition einer Verbindung erstellen, wäre es sinnvoll, mit einem MODIFY DATABASE die Verbindung zu prüfen (vgl. Abb. 29).

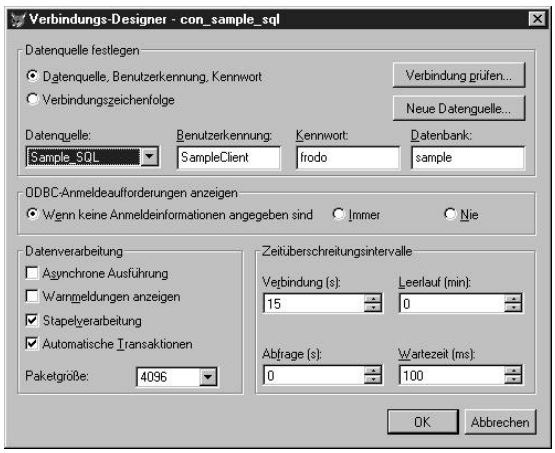


Abbildung 29. Prüfen der Verbeindungseinstellungen

Sie können die Verbindung auch prüfen, indem Sie sie einfach benutzen:

```
? sqlconnect('con_sample_sql')
7

? sqlexec(7,'select * from account')
-1

=aerror(x)
? x[1,2]
Connectivity-Fehler: [Microsoft][ODBC SQL Server Driver]
[SQL Server]SELECT permission denied on object 'account',
database 'Sample', owner 'dbo'.
```

Haben wir etwa etwas vergessen?

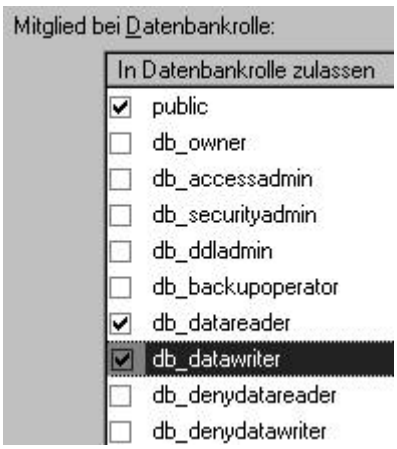
Der Benutzerkennung Zugriffsrechte zuordnen

Ja, natürlich. Wir müssen sicherstellen, daß unser Benutzer SampleClient die notwendigen Rechte besitzt, die Daten in den Tabellen unserer Datenbank zu lesen und/oder zu schreiben. Dies wird (Sie ahnen es bereits) im Enterprise Manager angegeben.

Abbildung 30. Die Eigenschaften einer UserID festlegen

Öffnen Sie den Ordner Datenbanken\Sample\Datenbankbenutzer im linken Fenster und klicken Sie anschließend mit der rechten Maustaste auf unseren SampleClient und wählen Sie „Eigenschaften...“ aus dem Kontextmenü.

Damit öffnen Sie den Dialog „Datenbankbenutzer – Eigenschaften“. Hier müssen wir lediglich unseren Anwender für die Datenbankrollen db_datareader und da_datawriter zulassen. Die Rollen auf diese Weise zuzulassen ist neu im SQL Server 7.0. Damit habe wir ein einfaches Werkzeug, einzelnen Benutzern individuelle Zugriffsrechte zu erteilen. Viele von uns haben in der Vergangenheit rollenbasierte Sicherheitsmodelle entwickelt, und es ist gut zu sehen, dies im SQL Server 7.0 implementiert zu sehen.



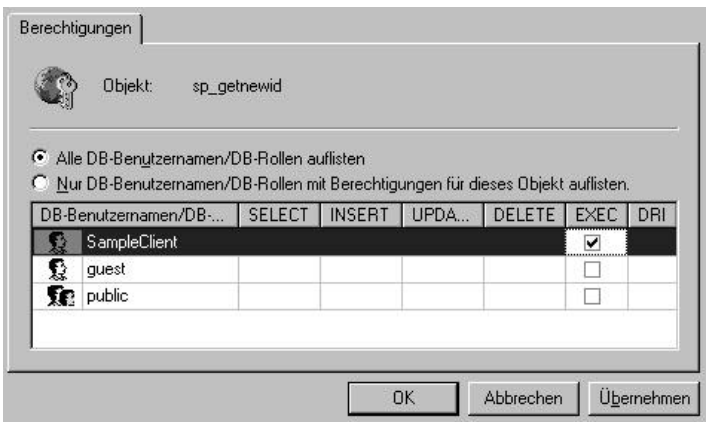


Abbildung 31/31a. Der UserID Rollen und Ausführungsrechte zuordnen

Wir ermöglichen dem Anwender den Zugriff auf die Rollen, indem wir in die Kontrollkästchen neben db_datawriter und da_datareader klicken. Nachdem wir die Änderungen gespeichert haben, können wir die Verbindung erneut testen – und diesmal sollte es klappen.

Wir müssen nun noch unserem SampleClient das Recht einräumen, die gespeicherte Prozedur sp_GetNewId auszuführen. Anderenfalls könnte unsere Anwendung sie nicht aufrufen.

Erstellen der remote Ansichten

Nun können wir die remoten Ansichten, die wir benötigen, erstellen:

```
open database r_sample

create sql view account_rec remote ;
   connection con_sample_sql shared ;
   as ;
   select * from account where account.ac_id = ?ipvAC_ID

create sql view account_lst remote ;
   connection con_sample_sql share ;
   as ;
   select ac_id, ac_account from account order by ac_account
```

Obwohl ich hier den Code für die Erstellung der Ansichten zeige, würden wir wahrscheinlich im Normalfall den Ansichten-Designer für diese Aufgabe benutzen, da wir noch das Schlüsselfeld und die Aktualisierungskriterien (vgl. Abb. 3, weiter vorne im Artikel) festlegen müssen. Diese Aufgaben sind im Ansichtsdesigner einfacher zu erledigen.

Sie können die Aufgaben aber mit der Funktion DBSETPROP() auch programmatisch erledigen. Im Angang A habe ich dafür Beispiele aufgeführt.

Gemeinsam genutzte Verbindungen

Sie werden bemerken, daß ich bei der Ansichtsdefinition die Klausel „SHARED“ verwendet habe. Dadurch werden die Ansichten angewiesen, sich bestehende Verbindungen zu teilen statt jeweils eine eigene Verbindung einzurichten. Experten empfehlen dieses Vorgehen dringend, weil es auf dem Server schneller und effizienter ist, eine bereits geöffnete Verbindung mehrfach zu nutzen. Dazu aber später mehr.

Betrachten wir die Arbeitsweise der Ansichten:

```
open data r_sample
ipvAC_ID = 2
use account_rec
edit noedit
```

Wahrscheinlich enthält das BROWSE-Fenster, das jetzt aufgeht, Daten. Wir können sagen, daß diese Daten von der Datenbank des SQL Server kommen, da der Datensatz das Feld TIMESTAMP_COLUMN enthält (das wir bereits behandelt haben) und da das Feld AC_DOB nun den Datentyp DATETIME besitzt.

Dadurch können Probleme mit unserer Anwendung auftreten. Wir wollen ja eigentlich nicht, daß die Anwender, die auf die remoten Daten zugreifen, Werte vom Typ DATETIME statt DATE bearbeiten müssen, oder wir müssen diesen Punkt in unserer Anwendung unterschiedlich behandeln.

Einführung in die Datenzuordnung

Da der SQL Server nicht über den Datentyp DATE verfügt, können wir dieses Problem umgehen, indem wir eine gute Fähigkeit der Remote-Ansichten nutzen, die Datenzuordnung. Der Ansichten-Designer besitzt auf dem Tab „Felder“ die Schaltfläche „Eigenschaften...“, die einen Dialog aufruft, in dem wir verschiedene Einstellungen für jedes Feld vornehmen können. Eine dieser Einstellungen ist der Datentyp. Abbildung 32 zeigt die Änderung des Datentyps des Feldes AC_DOB von DATETIME auf DATE.

In VFP 6.0 erledigt bereits der neue Upsizing-Assistent diese Aufgabe für uns, indem er die Felder, die in der original VFP-Datenbank den Datentyp DATE besitzen, als solche kennzeichnet, so daß sie später in den Ansichten wiederum als DATE erscheinen.

Wenn Sie in VFP 5.0 über dieses Problem gestolpert sind, können Sie sicher sein, daß es in der Version 6.0 nicht mehr auftreten wird. Wenn Sie mit der Version 6.0 das erste mal den Upsizing-Assistenten benutzen, werden Sie das Problem nicht haben.

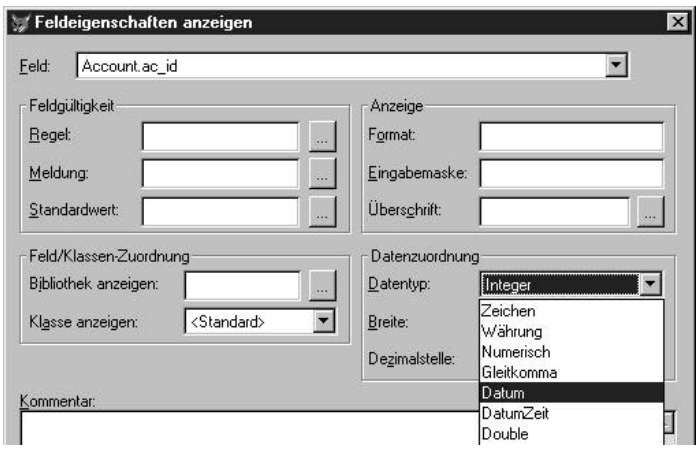


Abbildung 32.
Ändern des Datentypen eines Feldes in einer Remote-Ansicht

Gespeicherte Prozeduren in Remote-Ansichten

Der letzte Teil der Erstellung der Datenbank R_Sample.dbc mit den Remote-Ansichten ist das Schreiben der gespeicherten Prozeduren.

Wie Sie sich erinnern werden, haben wir uns entschieden, in beiden DBCs gespeicherte Prozeduren mit einheitlichen Namen einzusetzen. Diese gespeicherten Prozeduren ermöglichen es unserer Anwendung, die beiden DBCs abwechselnd zu benutzen, ohne daß sie dafür speziellen Code benötigt. Wir haben all unseren plattformabhängigen Code isoliert und in gespeicherten Prozeduren abgelegt. Außerdem haben wir einige generische gespeicherte Prozeduren für lokale und entfernte DBCs. Vergleichen Sie sample\sp\remote.prg für den Sourcecode unserer gespeicherten Prozeduren in R_Sample.dbc.

Sie kennen bereits spGetNewId(), aber Sie werden sich wahrscheinlich wundern, welche anderen Funktionen benötigt werden könnten. Vielleicht läßt sich das am einfachsten demonstrieren, wenn wir uns die remote Implementierung von sp_GetNewId() einmal ansehen.

sp_GetNewId muß den nächsten verfügbaren Wert für ein Schlüsselfeld zurückgeben, indem es eine gespeicherte Prozedur in der Datenbank des SQL Server ausführt.

Weshalb haben wir den Code von GetNewId() als gespeicherte Prozedur im SQL Server abgelegt? Er soll in einer Multi-User-Umgebung so schnell und so sicher wie möglich sein. Der beste Platz dafür ist innerhalb der Datenbank des SQL Server. Anschließend schreiben wir in R_SAMPLE.DBC eine Prozedur, welche die Prozedur im SQL Server aufruft.

Um in der SQL Server-Datenbank direkt eine gespeicherte Prozedur auszuführen müssen wir SQLEXEC() benutzen, das ein Handle einer Verbindung benötigt, welches wir in der Regel durch die VFP-Funktion SQLCONNECT() erhalten. Wir haben die Prozedur sp_GetAvailableConnectionHandle() geschrieben, die nach einer geöffneten Ansicht sucht und sich ihr Handle mit Hilfe der Funktion CURSORGETPROP() „ausleiht“. Am Ende konnten wir an vielen Stellen Code wie im folgenden Beispiel verwenden:

```
lhConnection = sp_GetAvailableConnectionHandle()
if m.lhConnection > 0
   llCloseConnection = .F.
else
   lhConnection     = sp_GetNewConnectionHandle()
   llCloseConnection = (m.lhConnection > 0)
endif
```

```
if m.lhConnection > 0
   *//      irgendwelche Aktionen...

   if m.llCloseConnection
      =SQLDisconnect( m.lhConnection )
   endif
endif
```

Eine komplette Liste der Basisprozeduren und ihre Verwendung finden Sie im Anhang B.

Erstellen einer Beispielanwendung

Wir bauen unsere Beispielanwendung rund um ein einfaches Formular.

Definition eines einfachen Formulars

Lassen Sie uns ein einfaches Formular für unsere Anwendung definieren – wir benötigen jeweils zwei Schaltflächen für die Navigation und jeweils eine für die Neuanlage und Änderung von Datensätzen. Für den Anfang reicht das schon.

- Öffnen Sie die Datenbank data\L_sample.dbc
- Erstellen Sie ein leeres Formular mit einer privaten Datensitzung.
- Geben Sie als Include-Datei SAMPLE.H an.
- Öffnen Sie die Datenumgebung.
- Fügen Sie die Ansicht account_rec der Datenumgebung hinzu, und stellen Sie sicher, daß im Cursor NoDataOnLoad auf .T. steht.
- Ziehen Sie außer AC_ID alle Felder einzeln auf das Formular.
- Fügen Sie die Schaltflächen „Vorheriger“, „Nächster“, „Neu“ und „Ändern“ hinzu.

Unser Formular sollte jetzt in etwa wie Abbildung 33 aussehen.

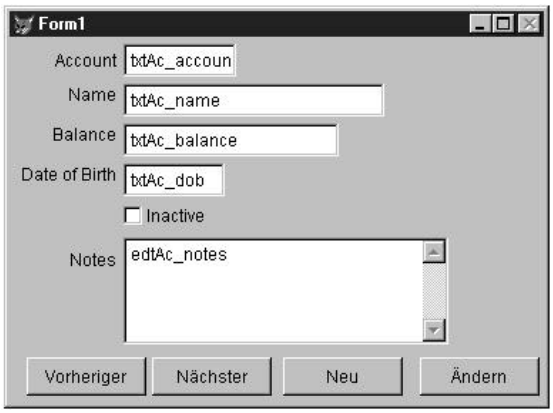


Abbildung 33. Unser Formular

Auffinden der richtigen Daten

Damit das Formular sowohl mit lokalen als auch mit remote Daten arbeiten kann, besteht der erste Schritt darin, sicherzustellen, daß die richtige Datenbank geöffnet ist und daß die Cursorobjekte auf sie verweisen. Dies erreichen wir, indem wir dem Ereignis BeforeOpenTables() der Datenumgebung einigen Code hinzufügen.

Um die ganze Sache einfach zu halten, fügen wir an dieser Stelle nur einen „Democode“ ein, der fragt, welcher DBC – lokal oder remote – vom Formular genutzt werden soll:

```
*//Wählen: lokale oder remote Daten

if IDYES=messagebox('Wollen Sie remote Daten bearbeiten?',;
        MB_YESNO+MB_ICONQUESTION)
   lcDBC = 'data\r_sample.dbc'
else
   lcDBC = 'data\l_sample.dbc'
   set path to data\            && nur lokal benötigt
endif
```

Normalerweise würden wir diese Einstellungen aus einer INI-Datei, einem Anwendungs-Objekt oder sonstwo auslesen.

Nachdem wir nun herausgefunden haben, welchen DBC wir verwenden, stellen wir sicher, daß er geöffnet ist und setzen in einer Schleife die Eigenschaft Database aller Cursor-Objekte in der Datenbank dementsprechend.

```
*//  Datenbank geöffnet?

if not dbused( m.lcDBC )
   open database (m.lcDBC)
else
   set database to (m.lcDBC)
endif

*//  Die richtige Datenbank in jedem Cursor-Objekt einstellen:

local array laMembers[1]
liCount = amembers(laMembers, THIS, 2)
for each lcCursorName in laMembers
   loMember = eval('THIS.'+lcCursorName)
   if upper(loMember.Baseclass) = „CURSOR“
      loMember.Database = m.lcDBC
   endif
endfor
```

Der Code im Formular

Lassen Sie uns das Formular vervollständigen, indem wir ihm den folgenden Code hinzufügen:

Init()

```
*// Platzhalter – ersten Datensatz wählen:
ipvAC_ID = 1
=Requery('account_rec')

*// Im Titel des Formulars anzeigen, welcher DBC genutzt wird:
THIS.Caption = sp_GetDBType()
```

cmdUpdate.Click()

```
*// Wenn Sie den Fehler „connection is busy“ erhalten,
*// die Verbindung mittels einer gespeicherten Prozedur prüfen:

* sp_WaitWhileConnectionBusy( 'account_rec' )

if not TableUpdate( TABLEUPDATE_CURRENTROW, ;
                    TABLEUPDATE_SKIPCONFLICTS, ;
                    'account_rec' )
    =TableRevert( TABLEREVERT_CURRENTROW )
  if messagebox('Ihre Änderungen konnten nicht gesichert werden.'+;
            ' Wollen Sie die letzten Daten sehen?', ;
            MB_YESNO+MB_ICONQUESTION)=IDYES

    *// Ansicht erneut abfragen:
    ipvAC_ID = account_rec.AC_ID
    =requery('account_rec')
    THISFORM.Refresh()
  endif
else
  =messagebox('Die Änderungen wurden gesichert.',;
            MB_ICONINFORMATION)
endif
```

cmdNext.Click()

```
*// Mit sp_SQLExecute() die letzte id erhalten:

local lcSQL
lcSQL = „select top 1 AC_ID from account“
lcSQL = lcSQL + „ where account.AC_ID >
„+trans(account_rec.AC_ID)
lcSQL = lcSQL + „ order by 1 „

if sp_SQLExecute( m.lcSQL, 'csrResult') > 0
  local ipvAC_ID
  ipvAC_ID = csrResult.AC_ID
  use in csrResult
  =requery('account_rec')
  THISFORM.Refresh()
endif
```

cmdPrevious.Click()

```
*// Mit sp_SQLExecute() die letzte id erhalten:

local lcSQL
lcSQL = „select top 1 AC_ID from account“
lcSQL = lcSQL + „ where account.AC_ID <
„+trans(account_rec.AC_ID)
lcSQL = lcSQL + „ order by 1 desc“

(sp_SQLExecute() as above...)
```

cmdNew.Click()

```
  insert into account_Rec ;
    ( ac_id ) ;
  values ;
    ( sp_GetNewId('account') )
  THISFORM.Refresh()
```

Und das war's auch schon. Sie können das Formular jetzt speichern (z.B. als ACCOUNT.SCX) und ausführen. Sie können es mehrfach sowohl auf den lokalen als auch auf den remoten Daten ausführen und beobachten, wie Einfügen, Pufferung und Konfliktbehandlung auf die gleiche Weise funktionieren.

Potentielle Probleme und wie man sie vermeidet

Wir sollten unser Formular noch um die Möglichkeit, einen bestimmten Datensatz anzuzeigen, erweitern. Lassen Sie uns dafür die Ansicht account_lst in einem Auswahldialog verwenden:

- Erstellen Sie ein neues Formular. Dieses Mal belassen Sie die DataSession auf „1- Default“, der WindowType ist Modal.
- Plazieren Sie ein Listenfeld auf dem Formular, stellen Sie dessen RowSourceType auf „6 – Fields“ und die RowSource auf „account_lst.AC_ACCOUNT“.
- Fügen Sie List1.Valid() folgenden Code hinzu:

```
Release THISFORM
```

- Speichern Sie das neue Formular unter chooser.scx.

Jetzt verbinden wir dieses Formular mit dem ersten:

- MODIFY FORM account.scx
- Öffnen Sie die Datenumgebung und fügen Sie die Ansicht account_lst hinzu.
- Fügen Sie dem Formular die Schaltfläche „Suchen...“ hinzu.
- Fügen Sie dem Ereignis cmdChoose.Click() folgenden Code hinzu:

```
do form chooser
ipvAC_ID = account_lst.AC_ID && zum aktuellen Datensatz
gehen
=Requery('account_rec')
THISFORM.Refresh()
```

› Jetzt können Sie das Formular speichern.

Das soll es schon gewesen sein. Starten Sie das Formular ... und Sie erhalten den in Abbildung 34 dargestellten Fehler.

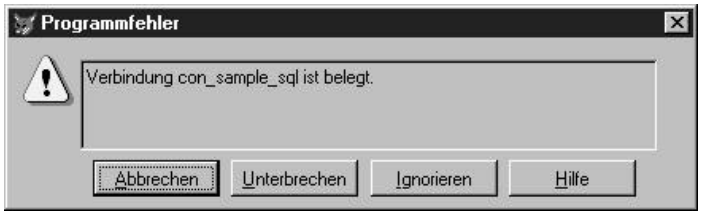


Abbildung 34. Der Fehler „Verbindung belegt“

Der Fehler wird durch das REQUERY() im .Init() des Formulars hervorgerufen. Warum ist die Verbindung aber plötzlich belegt?

Wir haben in der Ansicht account_lst, die wir der Datenumgebung hinzugefügt haben, NoDataOnLoad nicht auf .F. gesetzt. Daher empfängt die Ansicht weiterhin im Hintergrund Datensätze vom SQL Server, während unser Formular ausgeführt wird. Wenn das Formular nun in seinem Init das REQUERY() ausführt, tritt dieser Fehler auf, da auch das REQUERY() die Verbindung benötigt.

Aber Moment – haben wir nicht eine synchrone Verbindung erstellt?

Wenn Sie sich noch einmal die Abbildung 29 ins Gedächtnis zurückrufen, werden Sie jetzt wahrscheinlich einwenden, daß wir, als wir die Verbindung con_sample_sql im Datenbankcontainer R_SAMPLE.DBC erstellt haben, das Auswahlkästchen „Asynchrone Ausführung“ nicht markiert hatten. Warum wird also unser Code weiterhin ausgeführt, während die Daten empfangen werden? Bedeutet „synchron“ nicht, daß FoxPro die nächste Codezeile erst ausführt, wenn die Ansicht ihre Daten erhalten hat?

Eine einfache Lösung

Wenn wir in der Datenumgebung des Objekts account_lst NoDataOnLoad auf .T. setzen, müssen wir dem Ereignis .Click() der Schaltfläche „Suchen...“ ein =Requery('account_lst') hinzufügen. Vergessen wir dies, bleibt unsere Liste leer.

Aber – statt es uns so einfach zu machen – lassen Sie uns andere Möglichkeiten ausloten, aus einer solchen Situation herauszukommen.

Behandeln belegter Verbindungen

Während der Entwicklung unserer Anwendung stolperten wir ständig über den Fehler „belegte Verbindung“, bis wir im Dialog „Weitere Optionen“ des Ansichts-Designers etwas Feintuning durchführten sowie die gespeicherte Prozedur sp_WaitWhileConnectionBusy() schrieben und sie an einigen strategischen Stellen plazierten. Diese Prozedur fängt den Fehler ab, indem sie in einer Schleife den Status der Verbindung abfragt. Ist die Verbindung frei, wird dem aufrufenden Programm die Kontrolle zurückgegeben. Dafür benutzt die Prozedur die VFP-Funktion SQLGETPROP(h,'ConnectBusy').

Benutzen wir sie doch einmal, um zu sehen, wie sie arbeitet:

```
*// ersten Datensatz suchen:
ipvAC_ID = 1
sp_WaitWhileConnectionBusy('account_rec')
=Requery('account_rec')

*// Der Titel des Formulars zeigt, welcher DBC genutzt wird:
THIS.Caption = sp_GetDBType()
```

Wenn wir das Formular jetzt laufen lassen, läuft sp_WaitWhileConnectionBusy() in der Schleife, bis die Verbindung frei ist. Währenddessen wird die Nachricht „Connection is busy. Retrying...“ in der rechten oberen Bildschirmecke angezeigt. Nach kurzer Zeit erscheint das Formular, so daß mit ihm gearbeitet werden kann.

Soweit alles schön und gut. Wir bekommen keinen Fehler mehr. Nur – warum ist die Ladezeit so lang?

Die Antwort finden wir im Dialog „Erweiterte Optionen“ des Ansichts-Designers. Dieser Dialog ist gut versteckt; ich hatte schon ganz vergessen, daß es ihn gibt. Trotzdem ist er sehr wichtig. Lassen Sie uns die Ansicht account_lst ändern und den Dialog „Erweiterte Optionen...“ aus dem Menü Abfrage aufrufen, während der Ansichts-Designer aktiv ist.

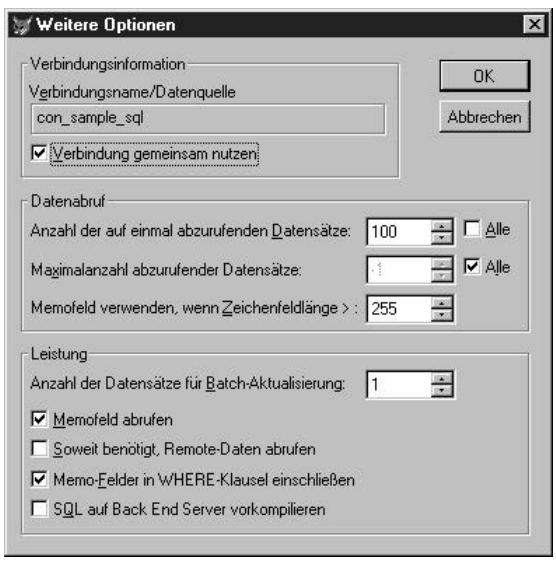


Abbildung 35. Der Dialog „Weitere Optionen“ der Ansicht

Unsere Ansicht account_lst ist so eingestellt, daß sie jeweils 100 Datensätze vom Server empfängt, bis alle Datensätze angekommen sind. Die 500 Datensätze unserer Tabelle ACCOUNT werden also in fünf getrennten Schritten übertragen.

Lassen Sie uns diese Einstellungen ändern, so daß unsere kleine Anwendung besser arbeitet:

- Maximalanzahl abzurufender Datensätze

Wenn wir diese Einstellung von Alle auf beispielsweise 100 ändern, holen wir immer nur die ersten 100 Datensätze in die Ansicht. Das geht zwar schneller, aber wir können dann nie eine Auswahl aus allen Datensätzen der Tabelle treffen.

- Anzahl der auf einmal abzurufenden Datensätze

Lassen Sie uns auch hier einmal die Option „Alle“ aktivieren, die Ansicht speichern und das Formular erneut ausführen.

Das Formular wird jetzt erheblich schneller gestartet. Klasse. Wir erhalten alle 500 Datensätze auf einmal, und es passiert nicht, daß die Verbindung längere Zeit belegt ist. Nur – was passiert, wenn die Tabelle ACCOUNT nicht 500 Datensätze enthält, sondern 500.000 oder 5 Millionen? Wird es dann wieder langsamer?

Die Antwort ist „Ja“. Die Geschwindigkeit, die Sie erreichen können, wird von vielen Faktoren beeinflußt, beispielsweise von der Geschwindigkeit des Servers, der Anzahl der Anwender, die gleichzeitig auf die Datenbank zugreifen usw. Sie sollten also grundsätzlich, sobald die Tabelle zu groß wird, um mit dieser Einstellung vernünftig zu arbeiten, andere Einstellungen durchprobieren.

Progressiver Datenabruf

Als „progressiven Datenabruf“ bezeichnet man das Verhalten, bei dem die Ansicht nur eine beschränkte Anzahl Datensätze erhält und anschließend wartet, bis Sie oder Ihre Anwendung weitere Daten benötigen – und sie dann abruft. Wir erreichen dieses Verhalten folgendermaßen:

- Gehen Sie in die Erweiterten Optionen für die Ansicht account_lst;
- Setzen Sie „Anzahl der auf einmal abzurufenden Datensätze“ auf 50;
- Setzen Sie „Maximalanzahl abzurufender Datensätze“ auf „Alle“;
- Aktivieren Sie „Soweit benötigt, Remote-Daten abrufen“.

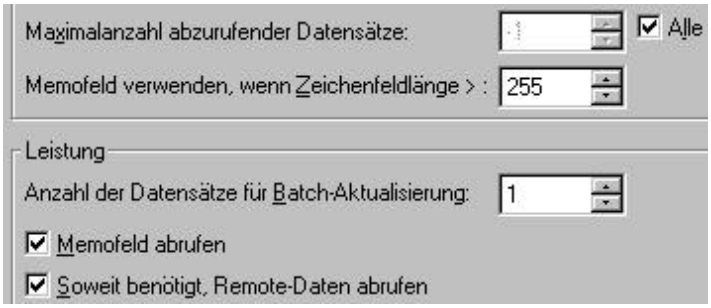


Abbildung 36. Einstellungen für den progressiven Datenabruf

Um den progressiven Datenabruf in Aktion zu sehen, können wir die Einstellungen für account_lst wie in Abbildung 36 gezeigt ändern und anschließend die folgenden Befehle ausführen:

```
set database to r_sample
use account_lst
browse
```

Werfen Sie nun einen Blick auf die Statusleiste.

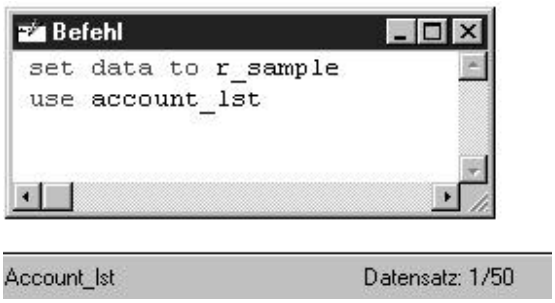


Abbildung 37. Der progressive Datenabruf in Aktion

Dort wird uns angezeigt, daß sich in der Ansicht 50 Datensätze befinden. Anschließend:

- Gehen Sie im BROWSE-Fenster nach unten. Sie werden feststellen, daß der Datensatzzähler auf 100 springt.
- Nach Ausführung des Befehls GO TO 175 springt der Datensatzzähler auf 175.
- Lassen Sie ein REQUERY() ausführen, geht der Datensatzzähler wieder auf 50 zurück.
- Führen Sie die Anweisung „LOCATE FOR AC_ACCOUNT ='012-0402'„ aus, springt der Datensatzzähler auf 150, der Datensatzzeiger steht auf 113.

Klasse! Die Ansicht empfängt lediglich die Daten, die sie benötigt, die von uns gegebenen Befehle auszuführen.

Aber Vorsicht! Einige Befehle umgehen den progressiven Datenabruf und erfordern den vollständigen Abruf aller Daten zu einem Zeitpunkt:

- Wenn Sie beispielsweise ? RECCOUNT() oder GO BOTTOM ausführen lassen, springt der Datensatzzähler auf 500 – oder wie viele Datensätze Ihre Tabelle gerade enthält.

Lassen Sie uns jetzt einmal versuchen, unser Formular ACCOUNT.SCX mit aktiviertem progressivem Datenabruf auf den Remote-Daten auszuführen.

Prompt stoßen wir auf das nächste Problem – wir sitzen wieder vor einer belegten Verbindung. Warum?

Der Progressive Datenabruf besitzt einen unangenehmen Nebeneffekt: die Verbindung bleibt belegt, bis alle Daten empfangen wurden. Da dies nur geschieht, wenn es unbedingt notwendig ist, bleibt die Verbindung ständig belegt. Es gibt nur eine Möglichkeit, anderen Ansichten ihrer Anwendung zu ermöglichen, Daten zu empfangen: Sie müssen jeder Ansicht, die den Progressiven Datenabruf nutzt, ihre eigene Verbindung zuordnen. Dies geschieht im Dialog Erweiterten Optionen (vgl. Abb. 35), indem Sie die Option „Verbindung gemeinsam nutzen“ deaktivieren.

Es ist naheliegend, daß wir noch einige Richtlinien für die Entwicklung der Anwendung aufgestellt haben:

- So selten wie möglich Daten nachladen.
- Ansichten, die den Progressiven Datenabruf nutzen, dürfen sich keine Verbindung teilen.

Lassen Sie uns als die Ansicht account_lst ändern, so daß sie die Verbindung nicht mehr teilt. Anschließend führen wir unser Formular ACCOUNT.SCX mit den remote Daten aus, um zu sehen, was passiert:

Alles geht gut: das Formular wird in einer akzeptablen Zeit geladen, der Auswahldialog zeigt die Datensätze der Ansicht account_lst an ... aber warum enthält sie nur 50 Datensätze?

Noch ein dummes Problem

Würden wir mit dem Progressiven Datenabruf arbeiten, wäre das ja kein Problem. Wenn wir aber jetzt ans Ende der Datensätze gehen, fordert das Listenfeld die Ansicht nicht auf, weitere Datensätze abzurufen. Wir erhalten immer nur die ersten 50 Sätze (oder wie viele Datensätze Sie auch immer angegeben haben).

Dieses Problem lösen wir, indem wir statt des Listenfeldes einen Grid einsetzen. Der Grid arbeitet mehr wie ein BROWSE. Wenn Sie sich innerhalb der Datensätze abwärts bewegen, empfängt die darunterliegende Ansicht bei Bedarf automatisch weitere Datensätze.

Einführung in den Ansichten-Manager

In den Sourcen zu diesem Artikel finden Sie ein kleines Hilfsprogramm, den Ansichten-Manager, den ich während der Entwicklung unseres Programms geschrieben habe. Ich habe das Programm Ansichten-Manager genannt, da es eine Reihe von Ansichten-Definitionen unabhängig vom DBC verwaltet. Außerdem ermöglicht er es Ihnen, DBCs zu generieren, die lokale und remote Versionen der Ansichten enthalten (vgl. Abb. 38).

Warum Sie den Ansichten-Manager einsetzen sollten

Ich habe in diesem Artikel bereits einige der Mängel des Upsizing-Assistenten und des Ansichts-Designers beschrieben. Es gibt aber auch noch weitere gute Gründe für den Einsatz eines separaten Werkzeugs für die Bearbeitung von Ansichten und die Generierung von DBCs:

- Meist ist es schwierig, Ansichten mit drei Joins im Ansichts-Designer zu bearbeiten, manche Funktionalitäten werden gar nicht unterstützt.
- Der Upsizing-Assistent von VFP 5.0 kann das Daten-Mapping nicht automatisch verwalten.
- Wenn Sie die Struktur der darunterliegenden Tabelle ändern und anschließend versuchen, eine Ansicht im DBC zu ändern, öffnet der Ansichts-Designer die Ansicht nicht richtig und einige Einstellungen gehen verloren.
- Der Upsizing-Assistent ist nicht in der Lage, remote Versionen ihrer lokalen Ansichten zu generieren, wenn Ihre lokalen Ansichten sich in einem anderen DBC als Ihre Tabellen befinden.
- Der Ansichten-Manager erstellt remote Versionen Ihrer lokalen Abfragen, die an den SQL Server angepaßt sind. So werden in der SQL-Anweisung „==“ in „=“, .T./.F. in 1 und 0 und doppelte Anführungszeichen in einfache umgewandelt. Dadurch ist es für Sie einfacher, lokale Ansichten zu erstellen, die „remote-kompatibel“ sind.
- Durch die Verwendung des Ansichten-Managers wird es für Sie einfacher, lokale und remote Ansichten nach einer Beschädigung wieder zu reparieren.

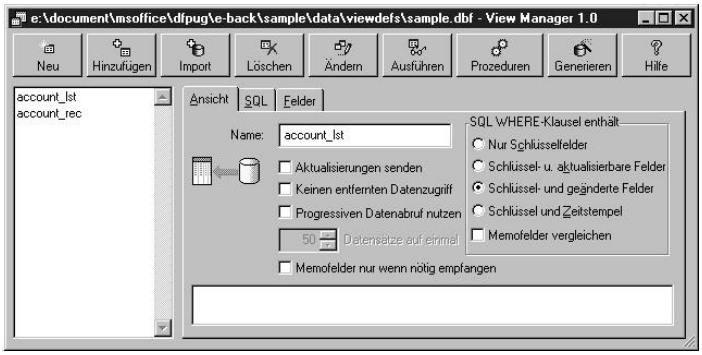


Abbildung 38. Der Ansichten-Manager in Aktion

Nachteile

Der Ansichten-Manager besitzt einen Nachteil: in seiner aktuellen Version unterstützt er nur eine benannte Verbindung im remoten DBC. Das stellte für mich kein Problem dar; es könnte aber in der Zukunft zu Problemen führen.

Dokumentation

Ich habe für den Ansichten-Manager auch eine detaillierte Dokumentation als Hilfedatei im HTML-Format geschrieben. Sie finden Sie unter vm\docs\default.htm.

Impressum

Herausgeber:

ISYS Softwareentwicklungs- und Verlagsgesellschaft mbH
Frankfurter Straße 21 b, D-61476 Kronberg i.Ts.
Telefon: 0 61 73-95 09 03
Telefax: 0 61 73-95 09 04
Hotline: 0 61 73-95 09 05
Mailbox: 0 61 73-95 09 08
http://www.dfpug.de
100024.1364@compuserve.com

ISYS LOGO

Redaktion Fuchs:

Rainer Becker, Uschi Benedict, Mathias Gronau

Gestaltung, Satz und Produktion:

U. Bartsch · Logic Consult, logic-consult@t-online.de

dFPUG & Fuchs:

Das Fuchs-Abonnement (Newsletter und Sonderhefte) beinhaltet die Mitgliedschaft in der deutschsprachigen FoxPro User Group (dFPUG) und deren Serviceangebote für 1 Jahr. Bestellung bei der dFPUG c/o ISYS GmbH.

Bezugspreise:

Deutschland: 220,– DM inkl. 7% MwSt.; Österreich: 1.610,– ATS inkl. 7% MwSt.; Schweiz: 220,– DM; andere Länder: 240,– DM inkl. 7% MwSt.

ISSN: 0946 – 8307

Das Impressum ist gerade in Arbeit...

Anhang A: Einführung in Ansichten und remote Daten

Ansichten

Im Grunde handelt es sich bei den Ansichten um SELCET...-Anweisungen, die im DBC für die spätere Verwendung gespeichert sind. Nachdem wir sie definiert haben, können wir sie wie ganz normale Tabellen verwenden. Anders als Cursor, die wir auch mit einer SQL-Anweisung erstellen können, sind die Ansichten nicht schreibgeschützt; Sie können die Werte ändern, ohne den Umweg über Konstruktionen wie SELECT ... INTO TABLE zu gehen.

Andere wichtige Informationen für das Verständnis der Ansichten:

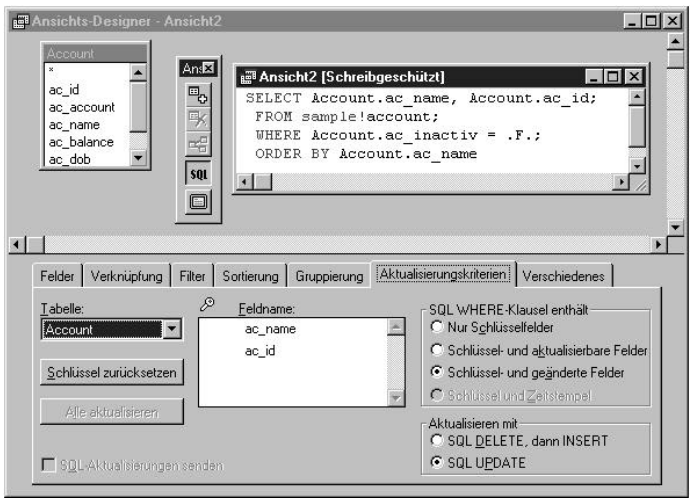


Abbildung A. Der Ansichts-Designer

- Sie sind als Datensätze im DBC gespeichert; weshalb Sie über einen aktiven DBC verfügen müssen, um Ansichten zu definieren und zu nutzen.
- Ansichten sind nicht schreibgeschützt. Sie können die Werte in der Ansicht ändern und anschließend die geänderten Werte in die Tabellen mit einem TABLEUPDATE() zurückschreiben.
- Ansichten sind immer optimistisch gesperrt. Als Vorgabewert ist die Datensatzsperre aktiviert, falls erforderlich können Sie aber auf Tabellensperre umschalten.

Lokale Ansichten

Lokale Ansichten beziehen sich auf VFP-Tabellen, unabhängig davon, ob es sich um Tabellen in einem Datenbankcontainer oder um freie Tabellen handelt. In VFP können wir Ansichten auf zwei unterschiedlichen Wegen erzeugen: wir können den Ansichten-Designer (in Abbildung A1 dargestellt) nutzen oder wir können sie per Programm erzeugen:

```
create database sample1
   create sql view account_lst_by_name_2 as ,
      SELECT Account.ac_name, Account.ac_id ;
      FROM account ;
      WHERE Account.ac_inactiv = .F. ;
      ORDER BY Account.ac_name
```

Die FoxPro-Syntax für die Erstellung von Ansichten kann sehr komplex werden, wenn viele DBSETPROP()-Anweisungen für das Setzen änderbarer Felder und ähnliches eingesetzt werden. Es ist erheblich einfacher, den Ansichts-Designer zu benutzen. Andererseits gibt es viele gültige und hilfreiche Ansichten, die durch den Ansichts-Designer nicht zu erstellen sind. Dazu aber später mehr.

Nachdem die SQL-Anweisung im DBC als Ansicht gespeichert ist, können wir sie mit FoxPros regulärem Befehl USE... nutzen.

```
set database to sample1
use account_lst_by_name
```

Werfen Sie nun einen Blick in das Fenster Datensitzung. Beachten Sie, daß VFP die Tabelle account zusammen mit der Ansicht account_lst_by_name geöffnet hat. Das war auch zu erwarten: Wenn Sie ein einfaches „SELECT * FROM account“ ausführen, gehen Sie auch davon aus, daß VFP die Quelldateien bei der Ausführung des Befehls öffnet.

Parametrisierte Ansichten

Aber Moment – da gibt es noch etwas (wie eigentlich immer bei Ihrer Arbeit mit FoxPro). Ansichten lassen sich auch flexibler gestalten. Statt für jede WHERE-Klausel eine eigene Ansicht zu speichern, können Sie mit Visual FoxPro auch parametrisierte Ansichten definieren. Dabei fügen Sie der Ansichten-Definition Variablen hinzu, die VFP auswertet, wenn Sie die Ansicht aufrufen. Sie definieren die Variable, indem Sie der Filterbedingung ein „?“ voranstellen (vgl. Abbildung A2).

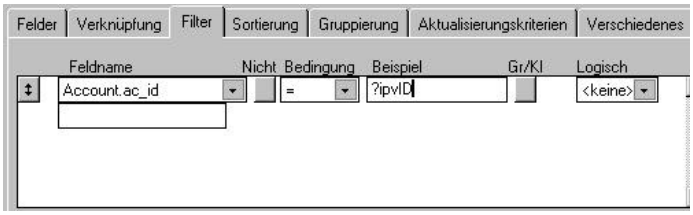


Abbildung B. Eine parametrisierte Filterbedingung einer Ansicht

Ansicht

Im Code würde diese Ansicht so aussehen:

```
Create sql view account_rec as ;
   Select * from account where account.ac_id = ?ipvID
```

Sie müssen lediglich dafür sorgen, daß die Variable, die Sie in der Definition der Ansicht angeben, FoxPro zur Laufzeit bekannt ist. Ansonsten fragt VFP Sie nach einem Wert. Sehen wir es uns einmal an:

```
use account_rec
```

Jetzt frag uns der Dialog, der in Abbildung C abgebildet ist, nach einem Wert für den Parameter der Ansicht. Lassen Sie uns aber an dieser Stelle abbrechen und die Parametervariable definieren, bevor wir die Ansicht nutzen:

```
IpvID = 45
Use account_rec
```

In der Statusleiste sehen Sie die Meldung „1 Datensatz in 0.01 Sekunden ausgewählt“ oder so ähnlich. Sehen wir uns nun den Inhalt der Ansicht an; anschließend weisen wir der Variablen einen neuen Wert zu und führen eine erneute Abfrage aus:

```
Browse
IpvID = 37
=requery()
```

Sie werden feststellen, daß die Ansicht jetzt den neuen Datensatz enthält.

Abbildung C. VFP fragt nach dem Parameter der Ansicht

Die Klauseln NODATA und NOREQUERY

Aber was ist, wenn Sie den Wert des Parameters zu dem Zeitpunkt, zu dem Sie die Ansicht öffnen, nicht kennen? VFP stellt uns mit NODATA die Möglichkeit zur Verfügung, Ansichten zur dem Zeitpunkt zu öffnen, zu dem wir auch die anderen Tabellen öffnen, beispielsweise im .Load() eines Formulars, und die Abfrage auf die Tabelle mit dem Befehl REQUERY() zu einem späteren Zeitpunkt durchzuführen.

```
Open data sample1
Use account_rec nodata
Browse && nix da???
IpvID = 23
= requery()
```

Wenn Sie die Ansicht erneut öffnen, wird ein implizites REQUERY() ausgeführt, es sei denn, Sie haben NOREQUERY angegeben:

```
ipvID = 13

use account_rec again in 0 alias acc2 norequery && 23 bleibt erhalten
```

Änderbare Ansichten

Ich habe bereits erwähnt, daß es zu den hervorragenden Eigenschaften von Ansichten gehört, daß wir mit ihrer Hilfe die Daten in den Tabellen der Datenbank ändern können. Lassen Sie uns die Ansicht account_rec änderbar machen:

```
set database to sample1
modify view account_rec
```

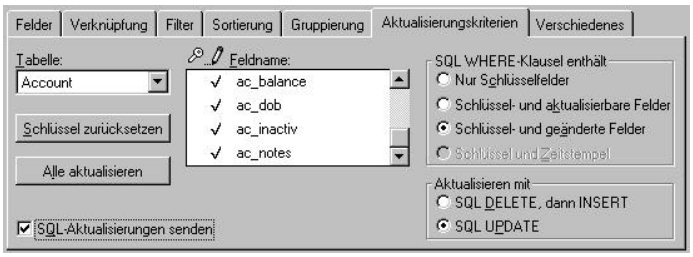


Abbildung D. Die Ansicht änderbar machen.

Mit Hilfe des Ansichten-Designers können wir den die Ansicht über den Tab „Aktualisierungskriterien“ änderbar machen. Wählen Sie zunächst den Primärschlüssel, indem Sie in die Spalte unterhalb des Schlüsselsymbols und anschließend auf die Schaltfläche „Alle aktualisieren“ klicken. Dadurch wird automatisch vor jedes Feld in die Spalte mit dem Stift ein Häkchen gesetzt (Ausnahme: das Feld mit dem Primärschlüssel). Eins haben wir noch nicht getan: Aktivieren Sie das Kontrollkästchen „SQL-Aktualisierungen senden“, um sicherzustellen, daß die Änderungen in den Backend-Tabellen gesichert werden. Jetzt können wir die Ansicht speichern.

Diese Aufgabe im Programm durchzuführen, ist eine echte Strafarbeit. Hier eine Kostprobe:

```
=DBSetProp('account_rec', 'View', 'SendUpdates', .T.)
=DBSetProp('account_rec.ac_id', 'Field', 'KeyField', .T.)
=DBSetProp('account_rec.ac_id', 'Field', 'Updatable', .T.)
=DBSetProp('account_rec.ac_account', 'Field', 'Updatetable', .T.)
=DBSetProp('account_rec.ac_name', 'Field', 'Updatable', .T.)
usw...
```

Lassen Sie uns die Updatefähigkeit der Ansicht prüfen:

```
ipvID = 24
use account_rec
replace AC_NAME with 'Keats, John'
? tableupdate(0,.F.,'account_rec'
```

Wenn Sie jetzt ein BROWSE auf die Tabelle account.dbf ausführen, werden Sie feststellen, daß der neue Wert in AC_NAME gespeichert wurde.

Nebenbei: Regeln und Trigger in der Quelldatenbank

Wenn Sie der Quelldatenbank eine Regel für die Feldgültigkeit hinzufügen, werden Sie feststellen, daß Regel erst überprüft wird, wenn der Befehl TABLEUPDATE() aufgerufen wird. Manche Entwickler benutzen gerade aus diesem Grund lokale Ansichten für die Eingabe von Daten: Die Überprüfung der Daten wird verschoben, so daß die Fehlerbehandlung an einer einzigen Stelle erfolgen kann. Hier ein Beispiel:

```
open database bank
alter table account ;
alter column ac_account ;
set check not empty(ac_account) ;
error „Kundennummer darf nicht leer sein“

open database sample1
use account_rec
replace AC_ACCOUNT with space(10)
if not tableupdate(0,.F.,'account_rec')
=aerrors(laError)
? laError(1,1), laError(1,2)
endif

1582            Kundennummer darf nicht leer sein
```

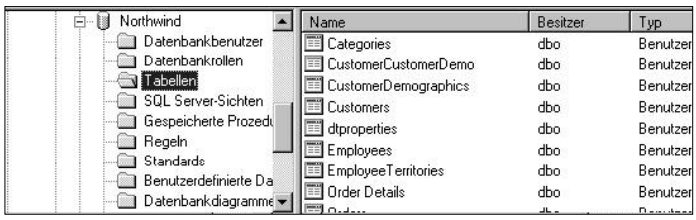


Abbildung E. Die Tabellen in der Datenbank Northwind

Abbildung E zeigt die in der Beispielsdatenbank Northwind des SQL Server enthaltenen Tabellen.

Ich habe das Erstellen der ODBC-Datenquelle an einer anderen Stelle dieses Artikels behandelt. Wir nehmen jetzt also an, daß wir wie in Abbildung F dargestellt eine Verbindung zur Datenbank Northwind hergestellt haben.

Anhang F. Die System-DSN der Datenbank Northwind

Nun ist es aber Zeit, zu Visual FoxPro zurückzukehren und festzustellen, wie wir via ODBC auf Daten zugreifen können.

SQL Passthrough

SQL Passthrough (SPT) bietet die Möglichkeit, den Prozeß der Übergabe einer SQL-Anweisung an die Backend-Datenbank zu übergeben, damit sie dort ausgeführt wird. Nachdem wir die ODBC-Datenquelle mittlerweile definiert haben, können wir folgenden Code verwenden:

```
H = SQLConnect('northwind_sql', 'sa', 'sauron')
=SQLExec(h, 'select * from customers')
=SQLDisconnect( h)
('sauron' ist das Paßwort der LoginID sa.)
```

Der SQL Server führt die Anweisung aus und gibt im Erfolgsfall einen Ergebniscursor (sqlresult ist der Vorgabename) im nächsten verfügbaren Arbeitsbereich zurück. Es handelt sich um einen sehr schnellen Weg, Daten vom Server zu empfangen. Der Nachteil dieser Methode ist – neben der Tatsache, daß Benutzername und Paßwort des SQL Server im Code angegeben werden müssen -, daß die SQL-Anweisung in einer Weise geschrieben werden muß, die der Server verarbeiten kann. Es können beispielsweise keine eingebetteten VFP-Funktionen verwendet werden. Ich werde aber später noch einmal auf SQL Passthrough eingehen.

Remote-Ansichten

Remote-Ansichten haben eine Menge Gemeinsamkeiten mit lokalen Ansichten. Der entscheidende Unterschied liegt im Zugriff auf remote Daten statt auf die nativen FoxPro-Tabellen.

Sie werden aber etwas unterschiedlich definiert und es gibt einige feine und nicht so feine Unterschiede.

Nachdem wir einen DBC geöffnet und ausgewählt haben, können wir eine Remote-Ansicht erstellen, indem wir dem Befehl CREATE VIEW die Klausel REMOTE hinzufügen:

```
open database <MeinBeispiel>
create view remote
```

Statt (wie bei einer lokalen Ansicht) nach den zu verarbeitenden Tabellen zu fragen, erwartet FoxPro die Angabe einer Verbindung oder einer entfernten Datenquelle:

Abbildung G. Auswahl einer Datenquelle für die remote

Ansicht

Lassen wir die Dinge einfach und wählen die Datenquelle northwind_sql, die wir bereits erstellt haben. Klicken Sie auf die Schaltfläche „OK“.

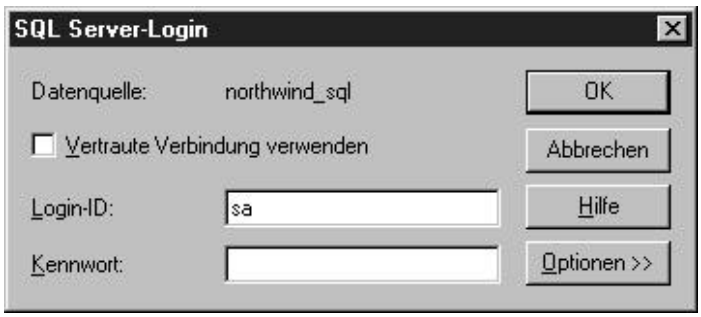


Abbildung H. Die Anmeldung des SQL Server

Wie Sie in Abbildung H sehen können, erfordert der SQL Server nach dem Zugriff auf die DSN, daß wir uns anmelden. Wir können unseren Benutzernamen und unser Paßwort eingeben und die Schaltfläche „OK“ wählen. (Beachten Sie das Kontrollfeld „Vertraute Verbindung verwenden“. Wenn wir die integrierten Sicherheitsmechanismen des SQL Servers konfiguriert haben, können wir uns mit unserem Netzwerk-Login anmelden.

Abbildung I . Auswahl einer Tabelle für die remote Ansicht

FoxPro präsentiert uns jetzt eine Liste von Tabellen in der Datenbank NorthWind, auf die wir unsere Abfrage ausführen können. Halten wir unser Beispiel einfach und wählen Felder aus der Tabelle Employees:

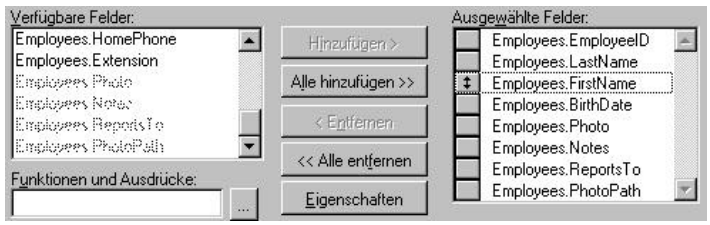


Abbildung J. Auswahl von Feldern aus der Tabelle NorthWind! Employee

In diesem Fall haben wir die Felder EmployeeID, LastName, FirstName, BirthDate, Photo, Notes, ReportsTo und PhotoPath ausgewählt. Nun können wir den Ansichten-Designer schließen und die Ansicht unter nw_employee_lst speichern.

Wenn wir nun versuchen, die Ansicht mit USE aus dem Befehlsfenster heraus zu öffnen, treten zwei interessante Ereignisse auf:

- Wir erhalten wieder den Anmeldedialog des SQL Server, der in Abbildung H dargestellt wird.
- Wenn wir erneut unser Paßwort eingeben, erhalten wir die Fehlermeldung: „Datatype property for field Notes ist invalid“.

Datenzuordnung

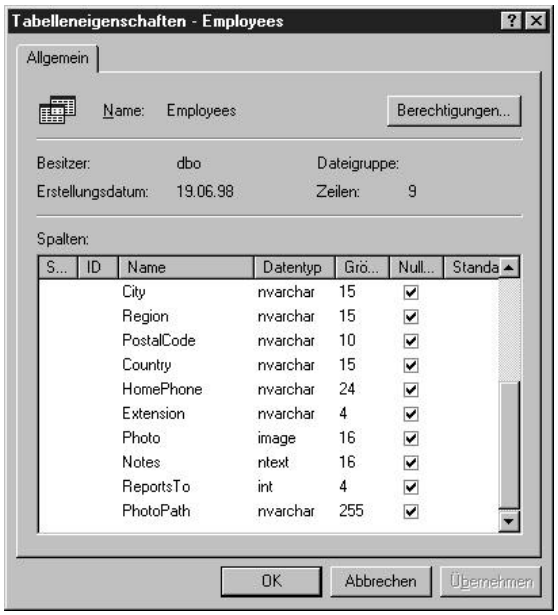
Remote-Ansichten unterstützen die Datenzuordnung. Damit ist gemeint, daß der Datentyp eines Feldes in der entfernten Quelldatei in einen anderen Datentyp im entsprechenden Feld der Ansicht konvertiert wird. Sie erreichen dieses Feature, indem, Sie wie in Abbildung A10 dargestellt, im Ansichten-Designer auf die Schaltfläche „Eigenschaften“ klicken. VFP öffnet dann den Dialog, den Abbildung K zeigt.

Abbildung K. Ändern der Datenzuordnung für das Feld „Notes“

Andern wir beispielsweise den Datentyp des Feldes Notes in der Ansicht von char(20) in Memo, erhalten wir bei der Ausführung der Ansicht die Fehlermeldung: „Die durch die Eigenschaft ‘Data Type‘ für Feld ‘Notes‘ erforderliche Typkonvertierung ist ungültig.“ Es ist zwar möglich, daß wir die Ansicht mit dem Datentyp char(254) speichern können, aber das ist nicht sehr befriedigend. Auch beim Feld PhotoPath tritt dieses Problem auf.

Wenn Sie den Enterprise Manager des SQL Server ausführen und sich die Struktur der Tabelle Employee anzeigen lassen (vgl. Abbildung A12), stellen Sie fest, daß Notes und PhotoPath den Datentyp nText bzw. nVarChar(255) besitzen. Die Dokumentation des SQL Server beschreibt diese Datentypen zusammen mit nchar als „Unicode-Datentypen“. Mit dem SQL Server 6.5 ist dieses Problem nicht aufgetreten; vielleicht stand ich da aber auch nicht vor diesem Problem.

Abbildung L. Die Tabelleneigenschaften der Tabelle „Employee“ im SQL Server



Nachdem wir die Datentypen der Felder Notes und PhotoPath in der Definition der Ansicht nw_employee_lst geändert haben, können wir können wir die Ansicht speichern und versuchen, sie uns in FoxPro anzeigen zu lassen:

```
use nw_employee_lst
browse
```

Vermutlich habe Sie bemerkt, daß der Anmeldedialog des SQL Server jedes Mal aufgerufen wird, wenn wir die Ansicht benutzen oder ändern. Da muß es doch eine bessere Lösung geben!

Verbindungen

Statt die Remote-Ansicht auf einer ODBC-Datenquelle aufzubauen, können wir auch eine benannte Verbindung erstellen. Die benannte Verbindung wird im DBC gespeichert und enthält zusätzliche Informationen über die Verbindung zur Backend-Datenbank. Wenn wir unsere Remote-Ansicht statt auf der Datenquelle auf einer benannten Verbindung aufbauen, umgehen wir das Problem mit dem ständigen Anmelden.

```
open database sample
create connection
```

Damit öffnen Sie den Verbindungs-Designer, der in Abbildung A13 dargestellt wird. Wir geben die Datenquelle, den Benutzernamen, das Kennwort und die Datenbank an. Anschließend können wir die Verbindung unter einem beliebigen Namen abspeichern, beispielsweise unter con_NorthWindSQL.

Abbildung M. Der Verbindungs-Designer

Lassen Sie uns jetzt eine Remote-Ansicht erstellen, die diese benannte Verbindung nutzt:

```
open database sample
create sql view nw_customers ;
    remote ;
    connection con_NorthWind_sql share ;
    as ;
    select * from customers
use nw_customers
```

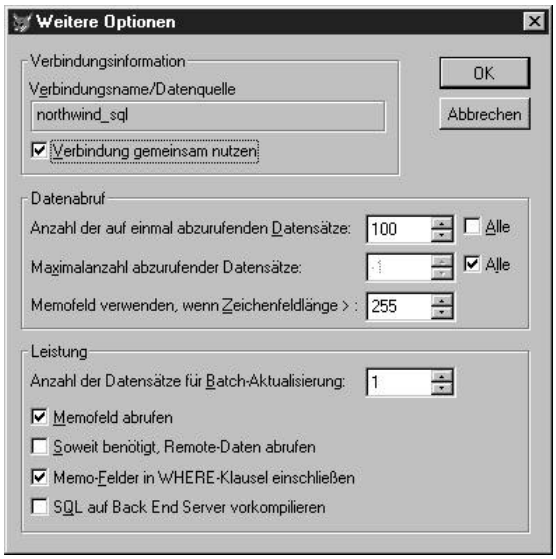
Sie sehen, daß wir nun den Anmeldedialog nicht mehr erhalten, wenn wir die Ansicht benutzen oder ändern. Wir sollten also unsere Ansichten auf den benannten Verbindungen aufbauen.

Wir können die Verbindung genauso gut mit SQL Passthrough nutzen:

```
open database sample
h = SQLConnect('con_northwind_sql')
=SQLEXEC( h, 'select * from customers')
=SQLDisconnect( h )
```

Beachten Sie, daß eine benannte Verbindung nicht das gleiche ist wie eine Verbindung zu einer Backend-Datenbank. VFP nutzt die in der benannten Verbindung gespeicherten Informationen, um eine Datenbank-Verbindung aufzubauen.

Abbildung N. Die erweiterten Optionen des Ansichts-Designers



Gemeinsam genutzte Verbindungen

Sie werden wahrscheinlich bemerkt haben, daß der Befehl CREATE SQL VIEW, den wir oben benutzt haben, die Klausel SHARE benutzt. Als Vorgabewert nutzt jede Remote-Ansicht ihren eigenen Kanal zum SQL Server. Jeder dieser Kanäle braucht Ressourcen auf der Maschine, auf der der SQL Server läuft. In unserer Anwendung war es denkbar, daß zehn Anwender gleichzeitig eine Instanz eines modalen Formulars öffnen, die ihrerseits sechs Ansichten öffnet. Das bedeutet, daß der Server gleichzeitig 180 Verbindungen verwalten muß! Es gab auch keine vernünftige Möglichkeit, die Anzahl der Benutzer und/oder der geöffneten Formulare zu begrenzen. Daher haben wir Versuche unternommen, eine Möglichkeit zu finden, daß sich alle Ansichten wenn möglich bestehende Verbindungen teilen.

Eine Warnung

Nehmen wir an, daß Sie auf der Seite „Remote-Daten“ des Optionen-Dialogs von VFP angegeben haben, daß VFP gemeinsam genutzte Verbindungen verwenden soll. Trotzdem erstellt FoxPro Ansichten, die dieses Feature nicht nutzen, unabhängig davon, ob Sie die Ansichten über den Ansichten-Designer oder mit dem Befehl CREATE VIEW REMOTE erstellen. Sie müssen entweder im Code die Klausel SHARE angeben oder im Dialog „Erweiterte Optionen...“ des Menüs „Abfrage“ explizit die gemeinsam genutzte Verbindung angeben.

Es handelt sich hierbei um einen sehr wichtigen Dialog, da er viele Optionen enthält, die unsere Anwendungen für den bestmöglichen Datenzugriff benötigen.

Weitere Informationen

Ich habe hier die Offline-Ansichten nicht erwähnt. Wenn diese für Sie interessant sind, lesen Sie in der Hilfedatei von VFP und der Dokumentation weiter.

Anhang B: Referenz der Gespeicherten Prozeduren

Hier sind die 14 wichtigsten Gespeicherten Prozeduren sowohl für den lokalen als auch für den remoten DBC. In den meisten Fällen besteht die lokale Version der Prozedur lediglich aus einem Dummy oder Platzhalter, damit beim Aufruf der Prozedur durch die Anwendung nicht eine Fehlermeldung „Prozedur nicht gefunden“ (oder so ähnlich) erhalten.

sp_GetNewID(cEntity)

Alle Werte neuer maschinell oder automatisch generierter Primärschlüssel werden durch diese Funktion erzeugt. Die Methode gibt den nächsten verfügbaren Integerwert, der mit cEntity angegeben wird, zurück.

Lokal: ⊙	Durch ein reguläre von VFP durchgeführte Satzsperre in der Tabelle UNIQUEID wird die Sicherheit im Mehrbenutzerbetrieb gewährleistet.
Remote: ⊙⊙	Diese Methode sucht durch sp_GetAvailableConnectionHandle() eine Referenz auf eine geöffnete Verbindung zum SQL Server. Wird eine Referenz gefunden, nutzt sie SQLExec() um die remote Gespeicherte Prozedur sp_getnewid aufzurufen und den Wert des neuen Feldes zurückzugeben. Wird keine Verbindung gefunden, wird sp_GetAvailableConnectionHandle() aufgerufen, um eine neue Verbindung aufzubauen.

sp_SQLExecute(cSQL, cResultCursor, iFlags)

Diese Methode führt eine SQL-Anweisung aus und erstellt den in cResultCursor spezifizierten Cursor, der die Ergebnisse enthält. Beachten Sie, daß die Remote-Version SQL-Anweisungen von VFP in eine SQL-Version umsetzt, die vom SQL Server ausgeführt werden kann.

Lokal: ⊙	Diese Methode fügt der SQL-Anweisung cSQL “‘INTO-CURSOR’ +cResultError” hinzu und fügt es als Makro aus. Im Erfolgsfall wird 1 zurückgegeben. Der Parameter iFlags wird ignoriert.
Remote: ⊙⊙	Diese Methode prüft die Anweisung in cSQL und führt folgende Ersetzungen aus: “==” in “=”, “.F.” in “0”, “.T.” in “1” und doppelte Anführungszeichen in einfache. Anschließend wird, wenn notwendig, eine Referenz auf eine geteilte Verbindung erstellt und SQLExec() aufgerufen. Optional wird geprüft, ob die Verbindung aktiv ist. Mögliche Flags: SQLEXEC_FORCE_NEW_CONNECTION und SQLEXEC_NO_BUSY_CHECK.

sp_Pack(cTable)

Diese Methode gibt .T. zurück, wenn sie alle als gelöscht markierten Datensätze aus einer angegebenen Tabelle entfernen konnte.

Lokal: ⊙	Ruft sp_ExclusiveAction() auf, um ein PACK auf die Tabelle auszuführen.
Remote: ⊙⊙	Diese Methode gibt .T. zurück, ohne eine Aktion auszuführen.

sp_Zap(cTable)

Diese Methode gibt .T. zurück, wenn sie alle Datensätze aus einer angegebenen Tabelle entfernen konnte.

Lokal: ⊙	Diese Methode ruft sp_ExclusiveAction() auf, um ein ZAP auf die Tabelle auszuführen.
Remote: ⊙⊙	Diese Methode gibt sp_Delete(cTable) zurück.

sp_ExclusiveAction(cTable, cActionCode)

Diese Methode ist nur in den Gespeicherten Prozeduren des lokalen DBC enthalten. Sie stellt vor jedem PACK, ZAP und REINDEX sicher, daß die angegebene Tabelle exklusiv geöffnet werden kann. Wenn notwendig schaltet sie die Einstellung von SET(‘DATABASE‘) vom DBC mit den lokalen Ansichten auf den DBC mit den Xbase-Tabellen, um die Operation durchzuführen. Im Erfolgsfall wird .T. zurückgegeben.

sp_Delete(cTable, cForClause)

Diese Methode markiert alle auf die FOR...-Klausel, die in cForClause angegeben wird, passenden Datensätze als gelöscht. Wenn der zweite Parameter nicht übergeben wird, werden alle Datensätze gelöscht. Bei Erfolg wird .T. zurückgegeben.

Lokal: ⊙	Diese Methode führt einen sp_ExclusiveAction() entsprechenden Code aus, um die angegebene Tabelle zu öffnen. Anschließend wird “DELETE ALL FOR&cForClause” ausgeführt.
Remote: ⊙⊙	Diese Methode ruft sp_SQLExecute() auf, um eine DELETE FROM ... FOR ...-Anweisung auf dem Server auszuführen.

sp_GetLocalDate(vDateTime)

Diese Methode wird meist beim Umgang mit Cursorn eingesetzt, die mit SQL Passthrough erzeugt wurden und die DATETIME-Felder remoter Daten enthalten, die lokal als DATE gespeichert sind. Dadurch wird sichergestellt, daß der Datentyp DATE unterstützt wird.

Lokal: ⊙	Die Methode gibt den Parameter unverändert zurück.
Remote: ⊙-⊙	Diese Methode gibt TTOD(vDateTime) zurück.

sp_GetDBType()

Diese Methode benötigen Sie in den seltenen Fällen, in denen Ihre Anwendung prüfen muß, ob sie mit remoten oder mit lokalen Daten arbeitet. Im Idealfall sollten Sie sie nie einsetzen müssen. Der Gedanke bei der Entwicklung der Methode war die Möglichkeit zu eröffnen, innerhalb eines Framework arbeiten zu können, das Ihnen die Möglichkeit bietet, unabhängig von der Plattform des Backend zu entwickeln.

Lokal: ⊙	Die Methode gibt den String “FOXPRO” zurück.
Remote: ⊙-⊙	Die Methode gibt den String “SQLSERVER” zurück.

sp_GetAvailableConnectionHandle()

Diese Methode wird meist von anderen Gespeicherten Prozeduren aufgerufen.

Lokal: ⊙	Die Methode gibt 1 zurück, ohne eine andere Aktion auszuführen.
Remote: ⊙-⊙	Die Methode führt auf der aktuell geöffneten remoten Ansicht CURSORGETPROP(‘ConnectHandle’) aus, um einen positiven numerischen Wert zurückzugeben, der eine geöffnete Verbindung zum SQL Server repräsentiert. Wird -1 zurückgegeben, konnte die Methode keine Verbindung finden.

sp_GetNewConnectionHandle()

Diese Methode wird meist von anderen Gespeicherten Prozeduren aufgerufen.

Lokal: ⊙	Die Methode gibt 1 zurück, ohne eine andere Aktion auszuführen.
Remote: ⊙-⊙	Die Methode findet mit Hilfe von ADBOBJECTS() den Namen der Verbindung im aktuellen DBC und gibt den Wert an SQLConnect() zurück.

sp_IsConnectionBusy(cViewAlias)

Diese Methode kann innerhalb der gesamten Anwendung immer dann eingesetzt werden, wenn Ihr Code eine Aktion ausführt, die zum Fehler 1541 (Verbindung aktiv) führt.

Lokal: ⊙	Die Methode gibt .F. zurück, ohne eine andere Aktion auszuführen.
Remote: ⊙-⊙	Diese Methode erhält die Verbindung von der Ansicht, die in cVieAlias angegeben ist oder sie ruft sp_GetAvailableConnectionHandle() auf. Anschließend wird SQLGetProp(‘ConnectBusy’) auf, um festzustellen, ob .T. zurückgegeben wird. Die Methode enthält eine Schleife, in der die Verbindung auf ihren Status geprüft wird, während ein WAIT WINDOW “Verbindung ist beschäftigt. Bitte warten oder Abbruch mit ESC” angezeigt wird.

sp_WaitWhileConnectionBusy(cAlias)

Lokal: ⊙	Die Methode macht nichts.
Remote: ⊙-⊙	Die Methode ruft sp_IsConnectionBusy(cAlias) auf, gibt aber keinen Wert zurück.

sp_SQLCancel(cView | hConnection)

Lokal: ⊙	Die Methode gibt 1 zurück, ohne eine andere Aktion auszuführen.
Remote: ⊙-⊙	Der Methode wird entweder eine Verbindung oder den Alias einer Ansicht (mit dem sie die Verbindung mit SQLGetProp() überwacht) übergeben. Sie gibt das Ergebnis von SQLCancel() an die Verbindung zurück.

sp_IsFlagSet(iFlags, iSpecificFlag)

Diese Methode ist in den remoten und lokalen Gespeicherten Prozeduren identisch. Sie benutzt BITAND(), um festzustellen, ob das in iSpecifiedFlag angegebene Bit in iFlags gesetzt ist.

Notizen